Зміст

Передмова

Методичні вказівки призначені для студентів IV курсу денної форми навчання. Мета виконання лабораторних робіт – засвоєння необхідних знань з основ розробки крос-платформних компонентів, а також формування практичних навичок щодо розроблення додатків з використанням компонентного підходу при розробки розподілених систем.

Дисципліна «Крос-платформне програмування» є нормативною дисципліною у напрямі бакалаврської підготовки «Комп'ютерні науки». Внаслідок вивчення даної дисципліни студенти повинні знати: основи багатопотокової моделі Java: подієву модель делегування: джерело події, слухач події, об'єкт події; етапи створення GUI застосувань з обробкою подій, основи багатопоточного програмування; основи мережної взаємодії, розробка сокетів; компонентну ідеологію.

За результатами навчання студенти повинні вміти: створювати крос-платформні компоненти та розробляти застосування з використанням компонентного підходу при розробки розподілених систем.

Методичні вказівки містять рекомендації по вивченню розділів дисципліни, контрольні запитання та завдання. Окремі лабораторні роботи підкріплені прикладами розв'язання типових задач на ПЕОМ.

Під час підготовки до лабораторної роботи студент повинен вивчити відповідний теоретичний матеріал за конспектом лекцій і літературою, розібрати приклади розв'язання задач, та відповісти на контрольні питання. На початку лабораторної роботи викладач проводить співбесіду за результатами якої студент отримує, або не отримує допуск до виконання лабораторної роботи. Якщо студент не отримав допуску, він залишається на заняттях, але не виконує лабораторної роботи на комп'ютері. Замість цього він вивчає теоретичний матеріал за даною темою, щоб відповісти на питання викладача та отримати допуск до виконання роботи.

За кожну лабораторну роботу студент отримує дві оцінки: за виконання та за захист роботи. Згідно з цих пунктів студенту зараховується відповідна кількість балів. Максимальні бали з кожної лабораторної роботи встановлюються згідно робочої програми дисципліни. На першому занятті студенти отримують графік контролюючих заходів: перелік контролюючих заходів, терміни виконання, бали за кожний вид робіт.

Список літератури

1. *Верлань А.Ф., Чмырь И.А., Кузниченко С.Д., Коваленко Л.Б.* Императивное программирование и объектно-ориентированное моделирование: JAVA, UML, OCL: Учебное пособие для студентов ВУЗ. Одесса: Экология, 2013.– 432 с.
2. *Джошуа Блох.* Java. Эффективное программирование. – М.: Лори, 2002. – 224 с.
3. *Герберт Шилдт.* Java. Полное руководство. Java 7.– 8-е изд. – М.: Вильямс, 2012. – 1104 с.
4. *Иванова Е.Б., Вершинин М.М.* Java 2, Enterprise Edition. Технологии проектирования и разработки. – СПб.: БХВ-Петербург, 2003. –1088 с:
5. *Иван Портянкин* Swing: Эффектные пользовательские интерфейсы, 2-е издание, Swing: Эффектные пользовательские интерфейсы, 2-е издание. – 2-е. – Санкт-Петербург: «Лори», 2011. – 600 с.
6. *Ноутон П., Шилдт Г.* Java 2: Пер. с англ. – СПб: БХВ – Петербург, 2007. – 1072 с.: ил.
7. *Эккель Б.* Философия Java. :BHV., СПб – 2001. – 850с.
8. *Кьоу Дж., Джеанини М.* Объектно-ориентированное программирование. Учебный курс – СПб: "Питер", 2005.– 238 с.
9. *Васильев А.Н.* Java. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2011 – 400 с.
10. *Хорстманн, Кей С., Корнелл, Гари.* Java2. Библиотека профессионала, том 1. Основы, 7-е изд.: Пер. с англ. – М.:Издательский дом «Вильямс», 2007. − 896 с.
11. *Хорстманн Кей С., Корнелл Гари* Java 2. Библиотека профессионала, том 2. Тонкости программирования, 7-е изд.: Пер. с англ. – М.:Издательский дом «Вильямс», 2007.
12. *Гранд М.* Шаблоны проектирования в Java – М.*:* Новое знание, 2004 – 559 с.: ил.
13. *Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес* Приемы объектно-ориентированного проектирования. – СПб.: Питер, 2010 – 412 с.
14. *С. Макконнелл* Совершенный код. Практическое руководство по разработке программного обеспечения М.: Издательство «Русская редакция», 2010.

Лабораторна робота 1

Програмування аплетів з елементами мультимедіа для Web сторинок Internet

1 Мета роботи

Метою лабораторної роботи є придбання навичок програмування аплетів на мові Java з використанням графічних об'єктів і зображень.

Після вконання лабораторної роботи студенти мають оволодіти вміннями створювати, запускати, виконувати та передавати параметри аплету.

2 Завдання на підготовку до лабораторної роботи

2.1 Перелік знань та вмінь, на які необхідно спиратися при виконанні роботи

Знання:

- технологія побудови Java – аплетів;
- можливості класу Applet. - тег <APPLET>;
- передача параметрів в аплет;
- інтерфейси Appletcontext, Audioclip, Appletstub.

Вміння:

- запускати, виконувати та передавати параметри аплету;
- використовувати мультимедіа в аплетах – розміщення графічних об'єктів і зображень.

2.2 Теоретичні відомості

2.2.1 Аплети Java

Запуск аплетів. Аплети Java, на відміну від застосувань, не є самостійними програмами, а вбудовуються в Web-сторінки і виконуються під управлінням Web-браузера.

Програма - аплет запускається в документі HTML в контейнері <applet> ... </applet>.

У контейнері <applet> ... </applet> можна також помістити текст, який буде виведений на Web-сторінці, якщо Web-браузер користувача не підтримує роботу з апплетами Java.

Атрибути, які можна поставити у дескрипторі <applet> наведені в табл.1.

Результат роботи аплету Java можна переглянути або за допомогою Web-браузера, або за допомогою програми appletviewer, що входить до складу SDK

(як параметр для цієї програми задається ім'я файлу HTML, що містить виклик аплету).

Таблиця 1 – Атрибути дескриптора <applet>

Атрибут	Значення	Чи є обов'язковим
code	Ім'я файла скомпільованого аплету (це повинен бути файл з розширенням **.class**)	Так
width	Ширина в пікселях того простору, який аплет займатиме на Web-сторінці	Так
height	Висота в пікселях того простору, який аплет займатиме на Web-сторінці	Так
codebase	Каталог на Web-сервері, де зберігаються **.class**-файли, на які посилається атрибут **code**	Ні
alt	Дозволяє вказувати альтернативний, текст, який буде виведений на місці аплету в тому випадку, коли Web-браузер розпізнає дескриптор **<applet>**, але не підтримує мову Java.	Ні
name	Дозволяє задати ім'я для аплету. Після цього інші аплети на сторінці можуть звертатися до цього аплету по імені і обмінюватися з ним даними	Ні
align	Дозволяє вибрати режим вирівнювання аплета на страниці. Можливі значення параметра – ті ж, що і для атрибута **align** в дескрипторі **<img>**: **top**, **texttop**, **middle**, **absmiddle**, **baseline**, **bottom**, **absbottom**, **left**, **right**.	Ні
vspace	Дозволяє задати величину в пікселях верхнього і нижнього полів навколо аплету	Ні
hspace	Дозволяє задати величину в пікселях правого і лівого полів навколо аплету	Ні

Виконання аплетів. Аплети є розширенням класу Applet, тому оголошення первинного класу аплета повинно мати наступний вигляд:

модифікаторы **class** *ідентифікатор-аплету* **extends Applet**
{
тіло-аплету
}

Олошення класу Applet знаходиться в пакеті java.applet, який автоматично не може з'єднатися, тому в програмі має бути заданий оператор import для цього пакету, тобто оператор

import java.applet.*;

Так як аплет може виконуватися на інших комп'ютерах в мережі, йому, в цілях безпеки, віртуальною машиною Java (JVM) забороняється виконувати багато операцій, наприклад, перегляд і читання вмісту каталогів і файлів на комп'ютері, а також зміни вмісту існуючих файлів і запис нових файлів.

Життєвий цикл аплета містить наступні чотири етапи:

- етап ініціалізації (initialization stage).
- етап запуску (start stage).
- етап зупину (stop stage).
- етап знищення (destroy stage).

На етапі ініціалізації створюється і завантажується об'єкт аплету. У цей момент зручно створювати об'єкти для аплету, а також ініціалізувати значення, необхідні при роботі аплету. Протягом життєвого циклу ініціалізація виконується тільки один раз. Можна втрутитися в процес ініціалізації, перевизначивши метод **init()** класу Applet.

На етапі запуску система починає виконання аплету. Етап запуску може слідувати відразу ж після етапу ініціалізації або після повторного запуску аплету. Зазвичай це відбувається тоді, коли користувач, працюючи з Web - браузером, повертається до сторінки, яка містить аплет, після перегляду якої-небудь іншої сторінки. На відміну від етапу ініціалізації, етап запуску протягом життєвого циклу може виконуватися безліч разів. Для того щоб виконувався власний код запуску, необхідно перевизначити метод **start ()** класу Applet.

Етап зупину є протилежністю етапу запуску. Інтерпретатор виконує фазу зупину, коли аплет більше не видно на екрані, наприклад, коли користувач звертається до іншої Web-сторінці. За замовчанням на цьому етапі аплет продовжує роботу у фоновому режимі. Якщо потрібно виконувати на етапі зупину інші дії, необхідно перевизначити метод **stop()** класу Applet.

Етап знищення за призначенням протилежний етапу ініціалізації і починається тоді, коли система збирається видалити аплет з пам'яті. Подібно фазі ініціалізації, етап знищення виконується тільки один раз. Якщо аплет використовував ресурси, які перед знищенням аплету необхідно звільнити, то це потрібно робити на етапі знищення. Цю фазу можна змінити, перевизначивши метод **destroy()** класу Applet .

Можна вважати також, що між етапами зупину і запуску існує етап прорисовки (paint stage). Ця фаза виконується кожен раз, коли область відображення аплета повинна проектуватися на екран, тобто відразу ж після етапу запуску аплета, а також щоразу, коли зображення аплету відновлюється або змінюється. Це відбувається, коли вікно аплета звільняється від іншого вікна, що перекривало його, або коли програма змінює область відображення аплета і явно перемальовує його вікно.

Передача параметрів аплету. Середовищем виконання програми Java є операційна система, і параметри до програми передаються за допомогою завдання аргументів командного рядка при запуску програми.

Аналогічний механізм існує і для передачі даних з документа HTML аплету Java.

Середовищем виконання аплету Java є Web-браузер, тому параметри передаються аплету в документі HTML, що викликає його, за допомогою дескриптора **<param>**. Цей дескриптор задається для кожного параметра аплету і містить два обов'язкових атрибути: name – вказує ім'я параметра і value – задає значення параметра.

Для того, щоб зробити доступним значення параметра, заданого в атрибуті value, в аплеті використовується метод класу Applet

public String getParameter (String name),

де name – ім'я параметра, що задане в атрибуті name дескриптора <param>. Метод повертає рядок, що містить значення параметра, яке задане в атрибуті value .

Якщо в дескрипторі <applet> немає дескриптора <param> з ім'ям, заданим як параметр, то метод getParameter () повертає рядок зі значенням null.

При множинної передачі параметрів в аплет дуже зручно використовувати клас **StringTokenizer** з пакету java.util. Наведемо коротку інформацію про нього.

Класс StringTokenizer

Конструктори:

– **StringTokenizer (String str)** – створює об'єкт, готовий розбити рядок str на слова, розділені пробілами, символами табуляцій '\t' , переведення рядка '\n' і повернення каретки '\ r'. Роздільники не включаються до числа слів.

– **StringTokenizer(String str, String delimeters)** – задає роздільники другим параметром delimeters, наприклад:

StringTokenizer("Казнить,нельзя:пробелов-нет", " \t\n\r,:-");

Тут перший роздільник – пробіл. Потім йдуть символ табуляції, символ переведення рядка, символ повернення каретки, кома, двокрапка, дефіс. Порядок розташування роздільників в рядку delimeters не має значення. Роздільники не включаються до числа слів.

– **StringTokenizer(String str, String delimeters, boolean flag)** – конструктор дозволяє включити роздільники до числа слів. Якщо параметр flag дорівнює true, то роздільники включаються до числа слів, якщо false – ні.

Приклад. Розбиття рядка на слова:

```
String s = "Строка, которую мы хотим разобрать на слова";
StringTokenizer st = new StringTokenizer(s, " \t\n\r,.");
while(st.hasMoreTokens()){
// Отримуємо слово і що-небудь робимо з ним, наприклад,
// просто виводимо на екран
System.out.println(st.nextToken());  }
```

Таблиця 2 – Основні методи класу StringTokenizer

Методи	Опис
String nextToken()	Повертає у вигляді рядка наступне слово
boolean hasMoreTokens()	Повертає true, якщо в рядку ще є слова, і false, якщо слів більше немає
int countTokens()	Повертає число слів, що залишилися в рядку
String nextToken (String newDelimeters)	Дозволяє "на ходу" міняти роздільники. Наступне слово буде виділено за новими розділювачами newDelimeters, нові роздільники діють далі замість старих роздільників, визначених у конструкторі або попередньому методі nextToken().

2.2.2 Виведення графічного контексту

При створенні компонента, тобто об'єкта класу Component, автоматично формується його графічний контекст (graphics context). У контексті розміщується область малювання і виведення тексту і зображень.

Управляє контекстом клас **Graphics** або новий клас **Graphics2D**, введений в Java 2. Оскільки графічний контекст сильно залежить від конкретної графічної платформи, ці класи зроблені абстрактними. Тому не можна безпосередньо створити екземпляри класу Graphics або Graphics2D.

Однак кожна віртуальна машина Java реалізує методи цих класів, створює їх екземпляри для компонента і надає об'єкт класу Graphics за допомогою методу класу Component: **public Graphics getGraphics()** , а також як аргумент методів **paint()** і **update()**.

Методи класу Component: paint(), update(), а також repaint() використовуються для відображення (малювання на екрані) компонента.

Метод **public void paint (Graphics g)** малює компонент на екрані. Як параметр використовується графічний контекст g – об'єкт класу Graphics. Цей метод повинен перевизначатися аплетом або графічним додатком.

Метод **public void repaint()** вказує, що компонент при першій можливості повинен бути перемальований. Це рано чи пізно (але не відразу) призведе до виклику методу update ().

Методи малювання класу Graphics для роботи з контурними малюнками наведені в табл.3.

Клас **Polygon** забезпечує більш гнучкий спосіб завдання багатокутників. Можна створити об'єкт класу Polygon, передавши конструктору **Polygon (int x [], int y [], int pointsNumber)** масиви координат і кількість точок.

Таблиця 3 – Методи малювання класу Graphics

Метод	Опис
drawLine (int x1, int y1, int x2, int y2)	Виводить лінію від позиції, заданої першими двома цілими числами (координати x1 і y1), до позиції, позначеної координатами x2 і y2
drawRect (int x, int y, int width, int height)	Виводить прямокутник. Координати x і y вказують верхній лівий кут прямокутника, width і height – відповідно ширину і висоту
draw3DRect (int x, int y, int width, int height, boolean raised)	Виводить підсвічений тривимірний прямокутник. Сам прямокутник задається як і в методі drawRect, а булевська змінна raised вказує, чи повинен прямокутник бути піднятий над фоном
drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	Виводить прямокутник з округленими кутами, вписаний в нормальний прямокутник, заданий першими чотирма параметрами. Параметри arcWidth і arcHeight вказують ширину і висоту дуги для кутів
drawOval (int x, int y, int width, int height)	Виводить овал, вписаний в прямокутник, визначений як у методі drawRect
drawArc (int x, int y, int width, int height, int angleBegin, int angleEnd)	Виводить дугу, вписану в прямокутник, визначений як у методі drawRect. Параметри angleBegin і angleEnd вказують початкові і кінцеві кути, вимірювані в градусах. Відлік кутів починається від центру правого боку графічної області. Додатні значення вказують напрямок обертання проти годинникової стрілки, а від'ємні – за годинниковою стрілкою
drawPolygon (int[] xPoints, int[] yPoints, int nPoints)	Виводить багатокутник. Цілочисельні масиви містять координати x і y для точок, складових багатокутник, а параметр nPoints вказує загальна кількість точок
drawPolygon (Polygon name)	Виводить багатокутник. Багатокутник заданий об'єктом класу Polygon
drawPolyline(int[] xPoints, int[] yPoints, int nPoints)	Виводить послідовність з'єднаних між собою ліній, кінці яких задаються координатами x і y, а параметр nPoints вказує загальну кількість точок (кількість ліній на 1 менше)

2.2.3 Використання кольорів

Колірна модель, що використовувана в Java, називається RGB (за початковими літерами слів Red – червоний, Green – зелений, Blue – синій). Це означає, що колір задається кількістю червоного, зеленого і синього кольору в ньому.

Крім того, може бути задана прозорість кольору (компонента alpha) – від повністю непрозорого до повністю прозорого (невидимого) кольору.

Ця складова проявляє себе при накладенні одного кольору на інший. Якщо alpha має максимальне значення, то колір абсолютно непрозорий, попередній колір не просвічує крізь нього. Якщо alpha дорівнює свого мінімального значення, то колір абсолютно прозорий, для кожного пікселя видно тільки попередній колір.

Колір задається як об'єкт класу **Color** за допомогою конструкторів цього класу.

Два конструктора:

public Color (int r, int g, int b, int a)

public Color (int r, int g, int b)

задають компоненти червоного (r) , зеленого (g) і синього (b) кольору і alpha (a) у вигляді цілих чисел з діапазону від 0 до 255 (у другому конструкторі alpha дорівнює 255).

Наступна пара конструкторів:

public Color (float r, float g, float b, float a)

public Color (float r, float g, float b)

задають компоненти кольору і alpha у вигляді дійсних чисел в діапазоні від 0.0f до 1.0f (у другому конструкторі alpha дорівнює 1.0f) .

І, нарешті, пара конструкторів

public Color (int rgba, boolean hasalpha)

public Color (int rgb)

задає компоненти кольору і alpha у вигляді одного шістнадцятирічного числа. У першому конструкторі значення alpha задається в бітах 24-31 , якщо параметр hasalpha дорівнює true. Якщо ж hasalpha дорівнює false, то складова альфа вважається рівною 0xFF, незалежно від того, що записано в старших бітах параметра rgba. У бітах 16-23 першого і другого конструктора записується червона складова, в бітах 8-15 – зелена, а в бітах 0-7 – синя складова кольору.

Кольори не завжди необхідно створювати, оскільки в класі Color є набір визначених кольорів, що задаються статичними змінними типу Color:

Color.blue (синій колір),

Color.white (білий колір),

Color.lightGray (світло-сірий колі)

Color.gray (сірий колір),

Color.darkGray (темно-сірий колір),

Color.black (чорний колір),

Color.red (червоний колір),
Color.pink (рожевий колір),
Color.orange (помаранчевий колір),
Color.yellow (жовтий колір),
Color.green (зелений колір),
Color.magenta (пурпурний колір),
Color.cyan (синьо-зелений, або ціановий, колір).

Наступні методи класу Color дозволяють отримати складові поточного кольору :

public int getRed () – значення червоної компоненти (від 0 до 255);
public int getGreen () – значення зеленої компоненти (від 0 до 255);
public int getBlue () – значення синьою компоненти (від 0 до 255);
public int getAlpha () – значення компоненти alpha (від 0 до 255);
public int getRGB () – значення синьою компонент в бітах 31-0;
public float [] getComponents (float [] compArray) – масив, що містить значення червоної, зеленої, синьої компонент і компоненти alpha (значення в діапазоні від 0.0f до 1.0f).

Створивши колір, можна задати його поточним кольором для малювання за допомогою методу

public abstract void setColor (Color c)

класу Graphics.

Нестандартний колір можна вивести на екран у більш темному або більш світлому відображенні. Для цього необхідно використовувати методи darker () і brighter (). Наприклад ,

```
Color my = new Color (156,112,108);
g.setColor (my.darker ());
g.setColor (my.brighter ());
```

Методи корисні в тих випадках, коли треба виділити активний компонент або, навпаки, показати неактивний компонент блідіше інших компонентів.

Поточний колір малювання можна отримати за допомогою методу

public abstract Color getColor ()

класу Graphics.

За допомогою методів

public void setForeground (Color c)
public void setBackground (Color c)

класу Component можна встановити (зазвичай така установка виконується в методі init ()) колір переднього плану (той колір, яким будуть малюватися лінії і виводиться текст) і колір фону.

Поточні значення кольорів переднього плану і фону можна отримати за допомогою таких методів класу Component:

public Color getForeground ()
public Color getBackground ()

Всі методи малювання фігур, крім drawArc () , drawLine () і drawPolyline(), мають відповідні методи заповнення виведеної замкнутої фігури в класі Graphics: fillRect () , fill3Drect () , fillRoundRect () , fillOval () , fillPolygon ().

Ці методи мають ті ж параметри, що і відповідні їм методи малювання. Колір заповнення фігури попередньо задається за допомогою методу setColor ().

2.2.4. Виведення тексту

Підтримка виводу тексту реалізується в Java за допомогою методів класів Graphics, **Font** і **FontMetrics**.

Для виведення рядка у вікно аплета використовується метод

public abstract void drawString (String str, int x, int y)

класу Graphics, якому в якості параметрів передається рядок str, а також координати x і y початку базової лінії. Слід зазначити, що на відміну від виведення фігур, в яких значення x і y задаються для лівого верхнього кута, для виведення тексту в якості початкової точки задається фактично лівий нижній кут, що слід мати на увазі при розміщенні тексту на екрані.

Для виведення тексту в програмі на Java використовується один з шрифтів, званий стандартним шрифтом або шрифтом за замовчуванням.

Для того, щоб змінити шрифт тексту його спочатку треба визначити як об'єкт класу Font за допомогою конструктора:

Font (String name, int style, int size)

У параметрі name задається найменування, або гарнітура, шрифту (typeface). Ім'я шрифту може бути рядком з фізичним ім'ям шрифту, наприклад, **"Courier New"**, або один з рядків **"Dialog"**, **"DialogInput"**, **"Monospaced"**, **"Serif"**, **"sansSerif"**, **"Symbol"**. Це так звані логічні імена шрифтів (logical font names). Якщо name дорівнює null, то задається шрифт за замовчуванням.

При виведенні тексту логічним іменам шрифтів і стилям зіставляються фізичні імена шрифтів або імена сімейств шрифтів. Це імена реальних шрифтів, наявних в графічній підсистемі операційної системи.

Наприклад, логічному імені **"serif"** може бути зіставлено ім'я сімейства (family) шрифтів Times New Roman, а в поєднанні зі стилями – конкретні фізичні імена Times New Roman Bold, Times New Roman Italic. Ці шрифти повинні перебувати у складі шрифтів графічної системи тієї машини, на якій виконується додаток.

У параметрі **style** вказується стиль шрифту (font style): **Font.PLAIN** (звичайний), **Font.BOLD** (напівжирний) і **Font.ITALIC** (курсив), **Font.BOLD | Font.ITALIC** (напівжирний курсивний), а в параметрі **size** задається розмір, або кегль, шрифту (point size). Кеглем в поліграфії називають розмір шрифту в друкарських пунктах. При виведенні на принтер один пункт відповідає 1/72 дюйма, але при виведенні на екран це співвідношення не завжди дотримується.

Установка нового шрифту при виведенні тексту в методі paint() виконується за допомогою методу класу Graphics

public abstract void setFont (Font font)

За допомогою методів

public String getName()

public int getStyle()

public int getSize()

класу Font можна отримати характеристики поточного шрифту.

2.2.5 Позиціонування тексту за допомогою клас FontMetrics

Методи класу FontMetrics дозволяють точно задавати положення тексту. Щоб використовувати методи даного класу необхідно створити екземпляр класу:

FontMetrics f = g.getFontMetrics ();

При роботі з шрифтами використовується наступна термінологія :

– Висота (height) – розмір від верхньої до нижньої точки найвищого символу в шрифті.

– Базова лінія (baseline) – лінія, по якій вирівнюються нижні межі символів (не рахуючи зниження (descent)).

– Підйом (ascent) – відстань від базової лінії до верхньої точки символу.

– Зниження (descent) – відстань від базової лінії до нижньої точки символу.

Нижче наведені деякі методи класу FontMetrics:

GetHeight() – повертає висоту поточного шрифту, тобто відстань від верхньої до нижньої межі найвищого символу шрифту.

GetAscent(), getDescent() – повертають підйом і зниження шрифту. Сума підйому та зниження дають повну висоту шрифту. Висота шрифту – це не просто відстань від найнижчої точки букв g і y до самої верхньої точки великої літери T і символів на зразок дужок. Висота включає підкреслення і т.п.

GetMaxAscent() і **getMaxDescent()** – методи служать для отримання максимальних підйому та зниження всіх символів в шрифті.

StringWidth(String str) – повертає ширину заданого рядка для поточного шрифту.

CharWidth (char ch) – повертає ширину символу ch .

charsWidth (char[] c, int off, int len) – повертає ширину зазначеного масиву символів, починаючи з індексу off на len позицій для поточного шрифту.

2.2.6 Вставка зображень в аплет

Істотним достоїнством мови Java є наявність стандартних засобів роботи з графікою в програмах.

Мова Java має готові класи, що здатні завантажувати файли зображень, однак ці класи забезпечують роботу тільки з зображеннями у графічних форматах GIF і JPEG (проте саме ці формати використовуються для вставки зображень в Web- сторінки).

Для вставки зображення в аплет необхідно виконати наступні кроки:

– створити об'єкт URL (Universal Locator Resource – універсальний покажчик ресурсів) вказує на місце розташування графічного файлу;

– створити об'єкт на базі класу Image і завантажити його в програму;

– вивести зображення на екран.

Хоча можна просто вписати URL зображення у вихідний текст програми, але в цьому випадку доведеться змінювати і перекомпілювати програму щоразу при переміщенні графічного файлу в інший каталог на диску. Більш зручний спосіб створення URL зображення – викликати метод

public URL getDocumentBase() або метод **public URL getCodeBase()** класу Applet .

Перший метод повертає URL каталог, з якого був завантажений поточний файл HTML, а другий дозволяє отримати URL каталог, з якого була запущена програма. Метод getDocumentBase() зручніше використовувати тоді, коли зображення зберігаються в тому ж каталозі (або підкаталозі цього каталогу), де знаходяться HTML - файли, а метод getCodeBase() – у тому випадку, коли зображення зберігаються в тому ж каталозі (або підкаталозі цього каталогу), де знаходяться файли класів (з розширенням . class) для даної програми.

Результат виклику методу привласнюється об'єкту класу URL (цей клас визначений у пакеті java.net), наприклад:

URL codeBase = getCodeBase ();

Створення та його завантаження в пам'ять комп'ютера можна виконати одночасно за допомогою методу **public Image getImage (URL url, String name)** класу Applet ,

де **url** – URL, повернутий методом getCodeBase() або getDocumentBase(), а **name** – відносна адреса зображення, наприклад:

Image myImage = getImage (CodeBase, "images / myImage.gif");

Для виведення зображення в аплет досить викликати один з перевантажуються методів **drawImage()** класу Graphics. Найбільш часто використовуються методи

public abstract boolean drawImage(Image img, int x, int y,
ImageObserver observer)
public abstract boolean drawImage(Image img, int x, int y,
int width, int height, ImageObserver observer)

де параметр img задає зображення, що виводиться
параметри x і y – координати зображення у вікні аплету, параметри width і height – ширину і висоту зображення, що виводиться (у першому методі вони беруться з графічного файлу), а параметр observer – об'єкт, який повідомляє про

перетворення зображення (зазвичай в якості цього параметра використовується ключове слово this), наприклад:

g.drawImage (myImage, 10, 20, this);

Клас Image має два методи,

public abstract int getWidth (ImageObserver observer)

public abstract int getHeight (ImageObserver observer),

які повертають ширину і висоту зображення, наприклад:

int width = myImage.getWidth (this);

int height = myImage.getHeight (this);

g.drawImage (myImage, 10, 20, width, height, this);

Збільшити або зменшити розмір зображення, яке виводиться на екран, можна, якщо змінити значення параметрів width і height при виклику методу drawImage () . Наприклад, задавши в попередньому прикладі значення width/2 і height/2, можна вивести на екран зменшене в два рази в порівнянні з оригіналом зображення.

2.3 Контрольні питання

1. Які атрибути дескриптора <applet> є обов'язковими для запуску аплета?
2. Які інструментальні засоби можна використовувати для перегляду аплета Java?
3. Які методи класу Applet дозволяють отримати інформацію про аплет?
4. Яка система координат використовується для малювання і виведення тексту в графічному режимі?
5. Які геометричні фігури можна малювати за допомогою методів класу Graphics?
6. Які методи маніпуляції з об'єктами визначені в Java?
7. Як можна задавати кольори у Java?
8. Які методи заповнення геометричних фігур, що замкнуті визначені в Java?
9. Як задаються характеристики шрифту, що виводиться, в Java?
10. Як виводиться напис в графічному режимі, і які параметри можна задати для напису?
11. Як в Java можна визначити ширину рядка, що виводиться, в пікселях?
12. Як можна вставити зображення в аплет?

3 Опис приладів, устаткування та інструментів

– Обладнання: IBM-сумісний персональний комп’ютер (ПК).

– Програмне забезпечення: операційна система Windows, Java 2 SDK версії 1.5 та вище. Інтегроване середовище розробки IDE Eclipse.

4 Правила техніки безпеки та охорони праці

Правила техніки безпеки при виконанні лабораторної роботи регламентуються «Правилами техніки безпеки при роботі в комп'ютерній лабораторії».

5 Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

– контроль рівня підготовленості студентів до виконання роботи у формі усного опитування;

– інструктаж по правилах охорони праці перед початком лабораторної роботи та оформлення його підсумків в журналі проведення лабораторних робіт, який ведеться у навчальній лабораторії;

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java у інтегрованому середовищі розробки Eclipse.

5.1 Завдання до лабораторної роботи

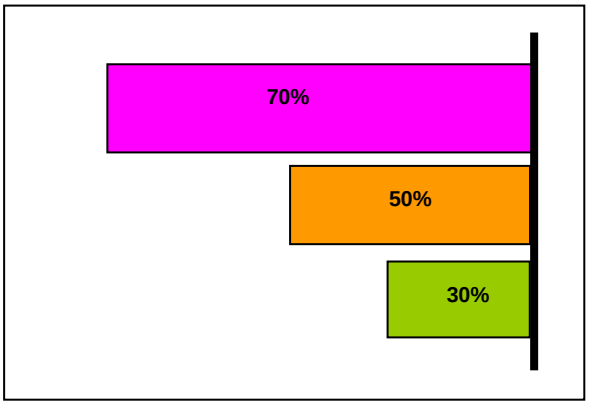
– Вивчити IDE і програму appletviewer . Создать простий аплет, в якому вивести інформацію про себе (прізвище, ім'я, по батькові, група). Використовувати різні кольори фону та шрифту, стилі тексту, вирівнювання! По можливості використовувати всі основні атрибути тега <APPLET>. Переглянути створений * . html файл. Запустити його з браузера.

– Підготувати свою фотографію у форматі gif або jpg і додати це зображення в аплет. Підключити до аплету аудіо файл з розширенням .au. Додати в аплет поточну дату і час (клас Calendar з пакету java.util).

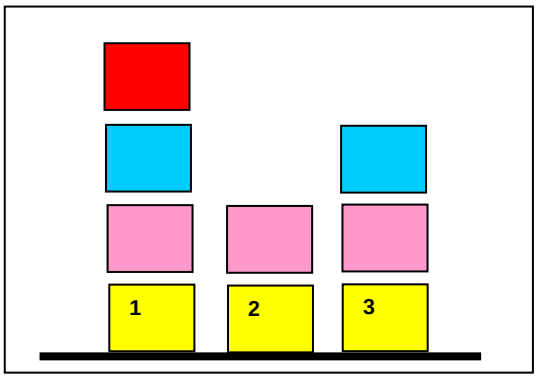
– Змінити аплет таким чином, щоб імена файлів, які підключаються, передавалися як параметри. В області аплету намалювати картинку згідно виданого викладачем варіанту (використовувати передачу множинних параметрів в аплет!).

5.2 Варіанти

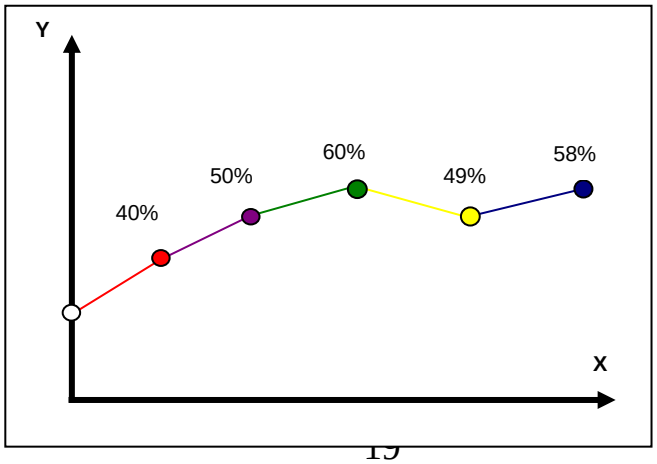
1. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від ширини вікна аплета. Колір прямокутників задати у вигляді масиву. Ширіну кожного прямокутника прийняти постійною, розрахованою так, щоб всі прямокутники розмістилися по висоті у вікні аплета. Вивести на малюнку ширину кожного прямокутника %.



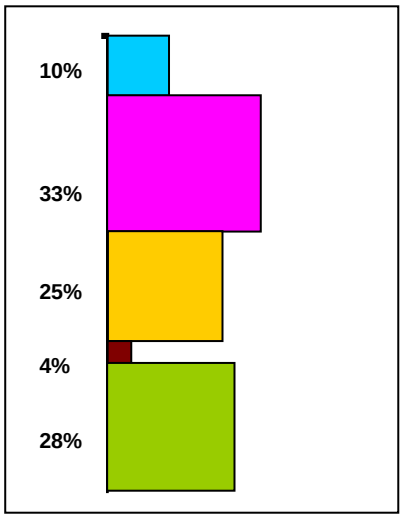
2. Використовуючи передачу множинних параметрів в аплет, намалюйте вказані фігури (див. рис). Кількість квадратів, складових фігури задати як параметри у вигляді цілочисельних значень. Колір квадратів задати у вигляді масиву. Розмір кожного квадрата прийняти постійним, розрахованим так, щоб навіть найвища фігура уміщалися по висоті у вікні аплета.



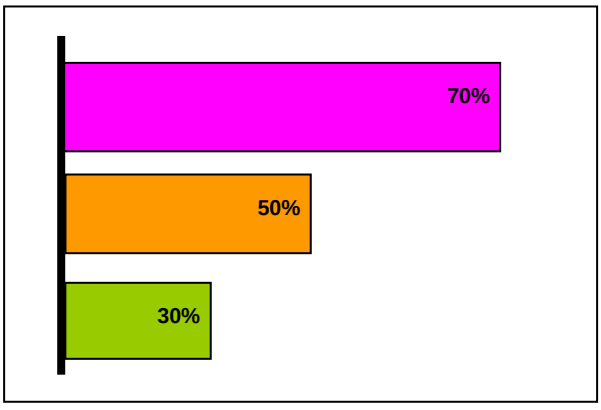
3. Використовуючи передачу множинних параметрів в аплет, намалюйте графік (див. рис). Координати точок по осі Y у відсотках від висоти області побудови графіка задати як параметри у вигляді цілочисельних значень. Колір точок на графіці задати у вигляді масиву. Крок побудови графіка прийняти постійним, розрахованим так, щоб всі точки уміщалися по ширині області побудови.



4. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного квадрата передати як параметр - цілочисельне значення, що позначає % від висоти області побудови (їх сума повинна складати 100%). Колір квадратів задати у вигляді масиву. Вивести на малюнку висоту кожного квадрата %.

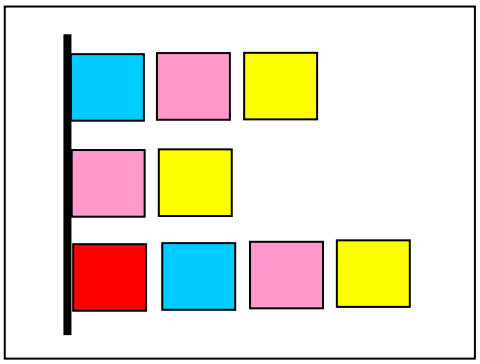


5. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від ширини вікна аплета. Колір прямокутників задати у вигляді масиву. Ширіну кожного прямокутника прийняти постійною, розрахованою так, щоб всі прямокутники розмістилися по висоті у вікні аплета. Вивести на малюнку ширину кожного прямокутника %.

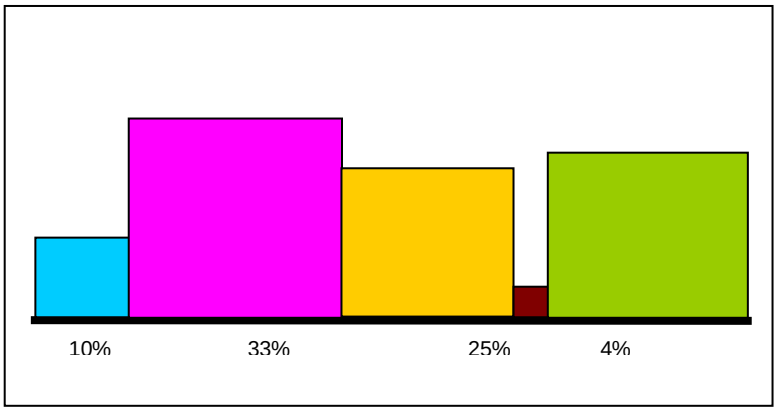


6. Використовуючи передачу множинних параметрів в аплет, намалюйте вказані фігури (див. рис). Кількість квадратів, складових фігури, задати як параметри у вигляді цілочисельних значень. Колір квадратів задати у вигляді

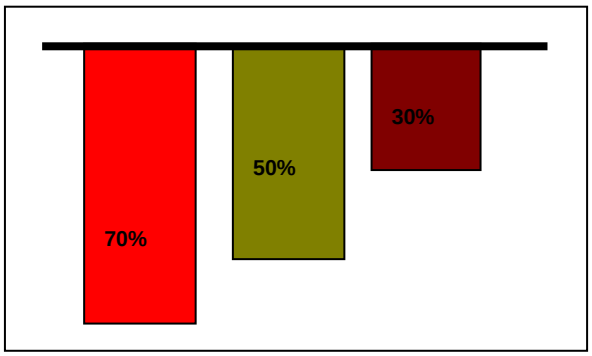
масиву. Розмір кожного квадрата прийняти постійним, розрахованим так, щоб навіть найвища фігура уміщалися по ширині у вікні аплета.



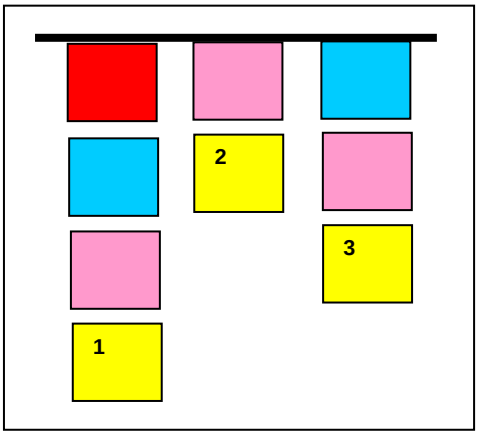
7. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Ширіну кожного квадрата передати як параметр - цілочисельне значення, що позначає % від ширини області побудови (їх сума повинна складати 100%). Колір квадратів задати у вигляді масиву. Вивести на малюнку ширину кожного квадрата %.



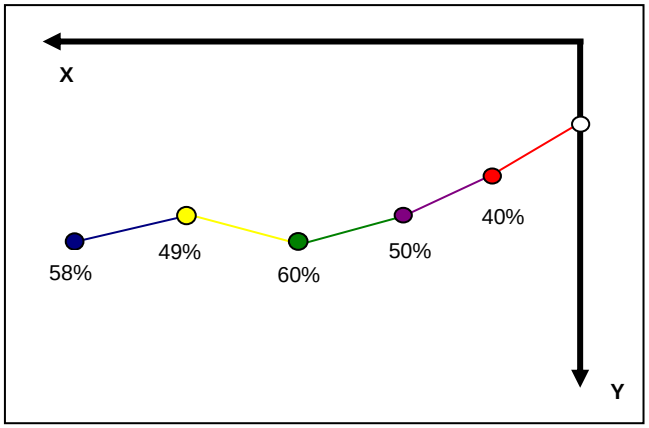
8. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від висоти вікна аплета. Колір прямокутників задати у вигляді масиву. Ширіну кожного прямокутника прийняти постійною, розрахованою так, щоб всі прямокутники розмістилися у вікні аплета. Вивести на малюнку висоту кожного прямокутника %.



9. Використовуючи передачу множинних параметрів в аплет, намалюйте вказані фігури (див. рис). Кількість квадратів, складових фігури, задати як параметри у вигляді цілочисельних значень. Колір квадратів задати у вигляді масиву. Розмір кожного квадрата прийняти постійним, розрахованим так, щоб навіть найвища фігура уміщалися по висоті у вікні аплета.

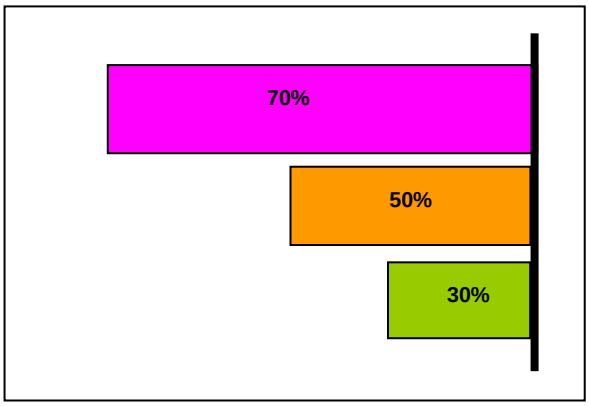


10. Використовуючи передачу множинних параметрів в аплет, намалюйте графік (див. рис). Координати точок по осі Y у відсотках від висоти області побудови графіка задати як параметри у вигляді цілочисельних значень. Колір точок на графіці задати у вигляді масиву. Крок побудови графіка прийняти постійним, розрахованим так, щоб всі точки уміщалися по ширині області побудови.

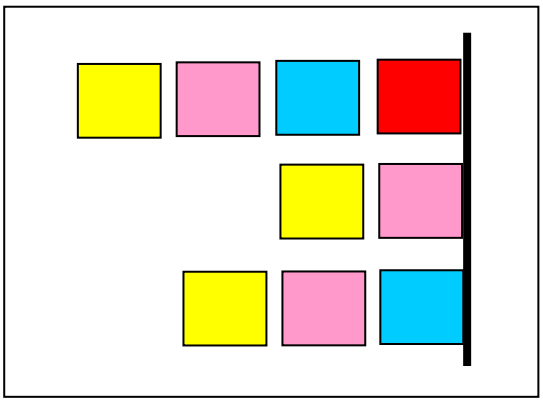


11. Використовуючи передачу множинних параметрів в аплет, намалюйте гистограмму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від ширини вікна аплету. Колір прямокутників задати у вигляді масиву. Ширину кожного прямокутника прийняти постійною, розрахованої таким чином, щоб всі прямокутники

розмістилися по висоті у вікні аплету. Вивести на малюнку ширину кожного прямокутника в %.



12. Використовуючи передачу множинних параметрів в аплет, намалюйте зазначені фігури (див. рис). Кількість квадратів, що складають фігури, задати як параметри у вигляді цілочисельних значень. Колір квадратів задати у вигляді масиву. Розмір кожного квадрата прийняти постійним, розрахованим таким чином, щоб навіть найвища фігура вмістилися по ширині у вікні аплету.



6 Порядок обробки результатів експериментів та вироблення висновків

На підставі отриманих результатів лабораторної роботи зробить висновки щодо особливостей створення, запуску та виконання апплетів Java. Нижче наведений приклад виконання лабораторної роботи.

6.1 Приклад виконання лабораторної роботи

```
/*
<applet code=”MyApplet” width=300   height=300>
<PARAM NAME = "data"   VALUE = "10|56|78|14|75|23|59">
```

```
<PARAM NAME = "image"   VALUE = "picture.jpg">
<PARAM NAME = "sound"   VALUE = "sound.au">
</applet>
*/

import java.awt.*;
import java.applet.*;
import java.util.*;

public class MyApplet extends Applet {
Calendar c=Calendar.getInstance();
int my[];//массив параметров для построения гистограммы
int x,y,size;
Color curColor[]={Color.cyan, Color.orange,
Color.blue, Color.pink, Color.green,
Color.magenta, Color.red};
    String ImageParam;
    AudioClip auClip; //для звука
    Image  myImage; //для вывода изображения

public void init(){
resize(250,250);
setBackground(Color.pink);
setForeground(Color.blue);
StringTokenizer myTokenizer =
                new StringTokenizer(getParameter("data"),"|");
  my=new int [myTokenizer.countTokens()];
  int i=0;
  while (myTokenizer.hasMoreTokens())
  {   my[i]=new Integer(myTokenizer.nextToken()).intValue();
      i++;}
      ImageParam = getParameter("image");
      myImage=getImage(getDocumentBase(),ImageParam);
      String sound = getParameter("sound");
      auClip = this.getAudioClip(getDocumentBase(),sound);}

 public void start(){ auClip.loop();}
 public void stop() { auClip.stop();}

 public void paint(Graphics g){
 g.drawString("Date"+c.get(Calendar.DATE)+"."
+c.get(Calendar.MONTH)+"."+ c.get(Calendar.YEAR), 160, 28);
 g.setColor(Color.white);
 g.drawImage(myImage,20,20,this);
 Font mf=new Font("Serif",Font.BOLD,(int)getSize().width/40);
     g.setFont(mf);
     FontMetrics f= g.getFontMetrics();
     String s="ИВАНОВ ИВАН ИВАНОВИЧ";
     String s1="группа К-42";
     g.setColor(Color.blue);
 g.drawString(s,((int)getSize().width-f.stringWidth(s))/2,(int)
(getSize().height*0.1));
  g.drawString(s1,((int)getSize().width-f.stringWidth(s1))/2,(int)
(getSize().height*0.15));
    int step=(int)getSize().width/(my.length+2);
```

```
        int h=(int) getSize().height-50;
        int i1=0;
        for(int i=0;i<my.length;i++,i1++){
          g.setColor(curColor[i1]);
          int hh=h*my[i]/100;
          g.fillRect(step*(i+1),h-hh,step,h*my[i]/100);
          g.setColor(Color.white);
          Font mf1=new Font("Serif",Font.BOLD,(int)(step/4));
          g.setFont(mf1);
          g.drawString(new                    Integer(my[i]).toString()+"
%",(step*(i+1)+(step-f.stringWidth(my[i]+"                    %"))/2),(h-
hh+(hh+mf.getSize())/2));
          if (i1==6) i1=0;
         }}

     }
```

7 Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

- титульний лист, де вказані номер і назва лабораторної роботи, відомості про виконавця;
- номер варіанта роботи та текст завдання;
- відповіді на контрольні запитання до лабораторної роботи;
- листинг програмного коду;
- результати виконання програми;
- висновки по підсумках роботи.

Підготовлений звіт надається викладачу на перевірку та захищається студентом на останьому занняті з даної лабораторної роботи.

Лабораторна робота 2

Використання потоків і створення анімації в Java

1 Мета роботи

Метою роботи є отримання практичних навичок роботи з потоками і створення анімації в Java.

Після вконання лабораторної роботи студенти мають оволодіти вміннями створення багатопотокових программ з графічним інтерфейсом користувача і елементами анімації.

2 Завдання на підготовку до лабораторної роботи

2.1 Перелік знань та вмінь, на які необхідно спиратися при виконанні роботи

Знання:

- потокова модель в Java;
- пріоритети і групи потоків;
- синхронізація потоків;
- подвійна буферизація;

Вміння:

- створювати потоки в Java;
- робота з анімацією;
- розроблення графічних додатків.

2.2 Теоретичні відомості

2.2.1 Реалізація потоків в Java

Мова Java є одною з небагатьох мов програмування, які містять засоби підтримки потоків. У мові Java потоки зазвичай використовуються для того, щоб аплети могли виконувати якісь дії в той час, як Web-браузер продовжує свою роботу, однак потоки можна застосувати в будь-якій програмі при необхідності паралельного виконання декількох завдань.

Так, наприклад, при створенні колективом програмістів великого і складного програмного продукту, як правило, окремі модулі програми розробляються паралельно окремими програмістами або групами програмістів. У цьому випадку процес розробки кожного модуля програми можна представити як окремий потік.

Реалізація використання потоків в програмах мовою може виконуватися двома способами:

розширенням класу **Thread**;

реалізацією інтерфейсу **Runnable**.

При першому способі клас стає потоковим, якщо він створений як розширення класу Thread , який визначений в пакеті java.lang, наприклад:

public class GreatRace extends Thread

При цьому стають доступними всі методи потоків.

Зазвичай, коли необхідно, щоб даний клас був розширенням деякого іншого класу і в ньому необхідно реалізувати потоки, попередній підхід не можна використовувати, оскільки в Java немає множинного спадкоємства. Для вирішення цієї проблеми для даного класу потрібно реалізувати інтерфейс Runnable, наприклад:

public class GreatRace extends Applet implements Runnable

Інтерфейс Runnable має тільки один метод **public void run()**. Насправді клас Thread також реалізує цей інтерфейс, проте стандартна реалізація run() у класі Thread не виконує ніяких операцій. Необхідно або розширити клас Thread, щоб включити в нього новий метод run(), або створити об'єкт Runnnable і передати його конструктору потока. Коли створюється клас, який реалізує інтерфейс Runnable, цей клас повинен перевизначити метод run(). Саме цей метод виконує фактичну роботу, покладену на конкретний потік.

Створити потік можна за допомогою одного з наступних конструкторів:

public Thread()
public Thread(String name)
public Thread(Runnable target)
public Thread(Runnable target, String name)
public Thread(ThreadGroup group, String name)
public Thread(ThreadGroup group, Runnable target)
public Thread(ThreadGroup group, Runnable target, String name)

У першому конструкторі створюється потік, який використовує самого себе в якості такого інтерфейсу Runnable. В інших конструкторах використовувані параметри мають наступний зміст:

name – ім'я, яке присвоюється новому потоку;

target – визначення цільового об'єкта, який буде використовуватися новим об'єктом Thread при запуску потоків. Якщо опустити цей параметр або привласнити йому значення null, новий об'єкт Thread буде запускати потоки за допомогою виклику методу run() поточного об'єкта Thread. Якщо при створенні нового об'єкта Thread вказується цільовий об'єкт, то для запуску нових процесів він буде викликати метод run() зазначеного цільового об'єкта;

group – призначений для розміщення нового об'єкта Thread в дерево об'єктів даного класу. Якщо опустити даний параметр або привласнити йому значення null, новий об'єкт класу Thread стане членом поточної групи потоків ThreadGroup.

Метод **public String toString()** повертає строкове представлення потоку, включаючи ім'я потоку, пріоритет і ім'я групи, а методи **public final String getName()** і **public final void setName(String name)** дозволяють отримати ім'я потоку або встановити ім'я потоку.

Запуск потоку виконує метод **public void start() throws IllegalThreadStateException** (виняток кидається, якщо робиться спроба запуску вже запущеного потоку).

Для зупинки потоку рекомендується привласнити потоку значення null, наприклад :

```
Thread myThread;
…
myThread.start(); // Запуск потоку
…
myThread = null; // Зупинка або завершення потоку
```

Припустимо, що на якомусь етапі роботи над програмним проектом необхідні два модуля, над якими працюють дві групи програмістів. Етап може початися, тільки якщо обидві групи закінчили роботу над своїми модулями, тобто однієї з груп програмістів доведеться чекати закінчення роботи над модулем іншої групи. Для такого узгодження дій використовується **public final void join() throws InterruptedException**.

Методу **join()** можна також передати значення тайм-ауту, використовуючи його варіанти **public final syncronized void join(long millis) throws InterruptedException** і **public final syncronized void join(long millis, int nanos) throws InterruptedException**. Ці методи чекають протягом millis мілісекунд або nanos наносекунд.

Якщо потрібно, щоб перед продовженням роботи потік чекав визначений час, можна використовувати метод **public static void sleep(long millis) throws InterruptedException** або **public static void sleep(long millis , int nanos) throws InterruptedException**, де параметри millis і nanos мають той же сенс, що і для методу join. Метод sleep() дуже часто використовується в циклах, керуючих анімацією.

Якщо в програмі є потік, який захоплює процесор для великої кількості обчислень, може з'явитися необхідність змусивати його час від часу "відпускати" процесор, даючи можливість виконуватися іншим потокам. Це досягається за допомогою методу **public static void yield()**.

Метод public void destroy() знищує потік без будь-якої очистки даних, що відносяться до нього, а метод **public final boolean isAlive()** дозволяє визначити, чи запущений потік і ще «живий».

Потоки в Java можуть переривати один одного. Механізм переривань реалізується за допомогою таких методів:

public void interrupt () – перериває даний потік;

public static boolean interrupted () – перевіряє, чи був перерваний даний потік (цей метод очищає ознаку переривання для потоку, тобто при повторному виклику методу для цього ж перерваного потоку він поверне значення false);

public boolean isInterrupted() – аналогічний попередньому методу, але не очищає ознаки переривання для потоку.

У класі Thread є ряд статичних методів для вивчення поточного потоку та інших потоків з тієї ж групи. Метод **public static Thread currentThread()**

повертає об'єкт, відповідний виконуваному в момент виклику потоку, метод **public static void dumpStack()** виводить трасування стека для поточного потоку.

Метод **public final void checkAccess()** перевіряє, чи має право поточний потік модифікувати даний потік

Звичайно програма на Java працює до завершення всіх потоків, що входять до неї. Але іноді, зустрічаються потоки, що працюють у фоновому режимі, виконуючи допоміжні дії, які ніколи не закінчуються. Можна помітити такий потік як потік - демон (daemon thread), що говорить JVM про те, що цей потік не треба приймати в розрахунок при визначенні, чи всі потоки даної програми завершилися. Іншими словами, додаток на Java виконується до тих пір, поки не завершиться останній потік, який не є демоном. Потоки, що не помічені як демони, називаються призначеними для користувача потоками (user threads).

Щоб потік вважався демоном, треба скористатися методом **public final void setDaemon(boolean on) throws IllegalThreadStateException**. Якщо параметр on дорівнює true, потік отримує статус демона, якщо false – статус користувальницького потоку. Статус потоку може бути змінений в процесі його виконання. Метод **public final boolean isDaemon()** повертає true, якщо потік є демоном, і false, якщо це користувальницький потік.

Метод **public static int enumerate(Thread[] threadArray)** класу Thread заповнює масив об'єктами Thread, що представляють потоки в групі, до якої належить поточний потік. Оскільки перед таким викликом необхідно створити масив threadArray, треба знати, скільки елементів буде отримано. Метод **public static int activeCount()** класу Thread повідомляє, скільки активних потоків в групі, до якої належить цей потік.

Пріоритети та групи потоків. Розподіл процесорного часу між потоками в Java виконується за такими правилами: коли потік блокується, тобто припинений, переходить в стан очікування або повинен дочекатися якоїсь події, Java вибирає інший потік з тих, які готові до виконання. Вибирається потік, що має найбільший пріоритет. Якщо таких кілька, вибирається будь-який з них. Пріоритет потоку можна встановити методом **public final void setPriority(int newPriority) throws IllegalArgumentException**.

Пріоритет потоку повинен бути числом в діапазоні від Thread.MIN_PRIORITY до Thread.MAX_PRIORITY. Будь-яке значення поза цими межами викликає виключення IllegalArgumentException. За замовчуванням потоку приписується пріоритет Thread.NORM_PRIORITY. Значення пріоритету потоку можна визначити за допомогою методу **public final int getPriority()**.

Клас ThreadGroup реалізує стратегію забезпечення безпеки, яка дозволяє впливати один на одного тільки потокам з однієї групи. Наприклад, потік може змінити пріоритет іншого потоку з тієї ж групи або перевести його в стан очікування. Якби не було розбиття потоків на групи, один потік міг би

викликати хаос в середовищі Java, перевівши в стан очікування всі інші потоки, або, що ще гірше, завершивши їх.

Групи потоків організовані в ієрархічну структуру, де у кожної групи є батьківська група. Потоки можуть впливати на потоки зі своєї групи і з дочірніх груп. Групу потоків можна створити, просто задавши її ім'я, за допомогою конструктора **public ThreadGroup (String groupName)**.

Можна також створити групу потоків, дочірню по відношенню до існуючої, використовуючи конструктор **public ThreadGroup(ThreadGroup existingGroup, String groupName) throws NullPointerException**.

Метод **public String toString()** повертає строкове представлення даної групи, а методи **public final String getName()** і **public final void setName(String name)** дозволяють отримати ім'я групи або встановити ім'я групи.

Інші методи класу ThreadGroup є аналогами відповідних методів класу Thread, але застосовуються до всієї групи:

public int activeCount() – повертає число активних потоків в даній групі;

public int activeGroupCount() – повертає число активних груп в даній групі;

public final void checkAccess() – перевіряє, чи може потік, що виконується в даний час, модифікувати дану групу;

public final void destroy() throws IllegalThreadStateException, **SecurityException** і **public boolean isDestroyed()** – відповідно знищує всі потоки даної групи та її підгруп (всі потоки в групі повинні бути попередньо зупинені) або перевіряє чи «жива» дана група;

public final void interrupt() – перериває всі потоки в даній групі;

public final boolean isDaemon() і **public final void setDaemon(boolean daemon)** – відповідно перевіряє і встановлює ознаку потоку-демона для всієї групи.

Можна обмежити пріоритет потоків з групи за допомогою виклику методу **public final synchronized void setMaxPriority(int priority)** , а визначити максимальний пріоритет потоку в групі можна за допомогою **методу public final int getMaxPriority()**.

Батьківська група потоків доступна за допомогою виклику методу **public final ThreadGroup getParent()**, а за допомогою методу **public final boolean parentOf (ThreadGroup g)** можна перевірити чи є група g батьківської для даної групи.

Синхронізація потоків. Як вже вказувалося вище, основна відмінність потоків від процесів полягає в тому, що потоки не захищені один від одного засобами операційної системи. Тому будь-який з потоків може отримати доступ і навіть внести зміни в дані, які інший потік вважає своїми. Вирішення цієї проблеми полягає в синхронізації потоків.

Синхронізація потоків полягає в гарантії надання доступу до даних тільки до одного потоку в кожен момент часу. Для цього використовується наступний оператор:

synchronized (*вираз*)
оператор

Взятий в дужки *вираз* повинен вказувати на дані, що блокуються, – зазвичай він є посиланням на об'єкт. Найчастіше при блокуванні об'єкта необхідно виконати відразу декілька операторів, тому оператор, як правило, являє собою блок.

Якщо виявляється, що критична ділянка поширюється на весь метод, а ресурсом, що розділяється, є весь об'єкт в цілому, то можна просто вказати модифікатор **synchronized** в оголошенні методу, наприклад:

```
synchronized void myMethod()
{
// Тіло методу
}
```

Більш гнучкий і ефективний спосіб координації виконання потоків забезпечують методи **wait()** і **notify()** класу **Object**.

Метод **wait()** переводить потік в стан очікування виконання певної умови і викликається за допомогою однієї з наступних форм:

public final void wait() throws InterruptedException, IllegalMonitorStateException – зупинення виконання поточного потоку до отримання повідомлення;

public final void wait(long timeout) throws InterruptedException, IllegalMonitorStateException, IllegalArgumentException – зупинення виконання поточного потоку до отримання повідомлення або до закінчення заданого інтервалу часу timeout (в мілісекундах);

public final void wait(long timeout, int nanos) throws InterruptedException, IllegalMonitorStateException, IllegalArgumentException – зупинення виконання поточного потоку до отримання повідомлення або до закінчення заданого інтервалу часу timeout (в мілісекундах) і, додатково, в наносекундах (значення в діапазоні 0-999999).

Метод **public final void notify()** переводить в активний стан один з потоків, встановлених в стан очікування за допомогою методу wait (). Критерій вибору потоку є довільним і залежить від конкретної операційної системи і реалізації віртуальної машини Java. Якщо необхідно перевести всі потоки, що очікують, в активний стан, можна скористатися **методом public final void notifyAll()**.

Методи можуть викликатися тільки потоком, який є власником монітора даного об'єкта. Якщо до виклику методів wait() або notify() не виконати захоплення монітора даного об'єкта, генерується виключення **IllegalMonitorStateException**. Потік стає власником даного об'єкта одним з трьох способів:

– викликом методу, оголошеного як **synchronized**, який здійснює доступ до необхідного екземпляра об'єкта класу **Object**;
– виконанням тіла оператора **synchronized**, призначеного для синхронізації доступу до необхідного екземпляра об'єкта класу Object;
– виконанням, для об'єктів типу Class, синхронізованого статичного методу цього класу.

2.2.2 Створення анімації

MediaTracker

Потокові додатки зручно використовувати для створення анімації. Для цього в програмі необхідно виконати коректне завантаження зображень – кадрів (особливо це важливо, якщо завантаження здійснюється по мережі).

У Java 2 був доданий клас MediaTracker, визначений у пакеті AWT, який дозволяє створити спостерігач завантаження цілої групи зображень. Об'єкт цього класу оголошується на базі будь-якого компонента

MediaTracker all = new MediaTracker (this);

Щоб додати зображення в спостерігач **all** слід використовувати метод **addImage()**

addImage (Image image, int id);

де **image** – об'єкт зображення, **id** – його ідентифікаційний номер у групі зображень об'єкту MediaTracker. Він може бути унікальним для кожного зображення або загальним для певної групи зображень.

Розглянемо методи класу MediaTracker.

Метод	Опис
checkID(int id) или checkAll()*	Перевірка на закінчення завантаження зображення або групи зображень з ідентифікаційним номером id.
isErrorID(int id) или isErrorAny()*	Перевірка на наявність помилки в процесі завантаження зображення або групи зображень з ідентифікаційним номером id.
removeImage(Image image, int id)	Видалення зображення image з ідентифікаційним номером id з групи зображень об'єкту MediaTracker
waitForID(int id) или waitForAll()*	Чекає завантаження зображення з ідентифікаційним номером id. Даний метод слід використовувати в блоці try-catch зважаючи на можливість виникнення виключення InterruptedException

***** - дані методи використовуються для реалізації описаних дій над всією групою зображень об'єкту MediaTracker.

Подвійна буферизація зображень. Часто в процесі роботи із зображеннями може виникнути ефект мерехтіння. Це відбувається із-за невеликої затримки між запуском методу paint() і моментом, коли зображення дійсно завантажено. Ефект мерехтіння можна усунути використовуючи технологію подвійної буферизації зображень. Для реалізації цієї технології зображення спочатку малюється в пам'яті комп'ютера, після чого воно виводиться на екран. Для цього слід виконати такі дії:

1) Проініціалізувати об'єкт зображення в тілі основного класу.

2) При ініціалізації аплету створити пусте зображення, використовуючи метод createImage(), виходячи з розмірів віконного об'єкта.

3) Визначити новий графічний контекст на підставі створеного зображення.

4) Намалювати в новоствореному графічному контексті необхідне зображення.

5) Намалювати в графічному контексті вже самого аплету створене зображення.

Нижче наведений приклад виконаної подвійної буферизації:

```
public void  update (Graphics g) {
paint (g);
}

public void paint (Graphics g) {
Image buffer = createImage(getWidth(), getHeight());
Graphics2D g2d = (Graphics2D) buffer.getGraphics();
// малюємо об’єкт g2d
//…
g.drawImage(buffer, 0, 0, null);
}
```

2.2.3 Створення графічних додатків

Для створення графічних додатків в Java використовуються фрейми.

Фрейм (невидимий в початковий момент) можна створити або конструктором без параметрів

public Frame ()

(без заголовка) або конструктором

public Frame (String frameTitle)[1]

(з заголовком).

Після створення, перед відображенням фрейма на екрані, необхідно задати його розміри і положення на екрані, використовуючи методи

public void setSize(int x, int y)

public void setLocation(int x, int y),

[1] Тут розглянемо клас Frame бібліотеки AWT, а у наступній лабораторній роботі буде розглянутий клас JFrame бібліотеки Swing

Щоб фрейм став видимим або невидимим, необхідно звернутися до методу

public void is Visible(boolean visibility).

Стан фрейма (згорнутий або розгорнутий) визначається за допомогою властивостей

public static final int Frame.ICONIFIED

public static final int Frame.NORMAL

Поки фрейм існує (видимий чи ні), він займає частину ресурсів віконної системи, в якій виконується додаток або аплет. Якщо фрейм більше не потрібен, його слід знищити, використовуючи метод

public void dispose(),

визначений у класі **Windows**.

У класі **Frame** визначено такі основні методи:

public String getTitle() и **public void setTitle(String newTitle**) – визначення та завдання заголовка, відображуваного у верхній частині фрейму;

public void setResizable(boolean allowResizing) і **public boolean isResizable()** – установка або відміна заборони зміни розмірів фрейма і перевірка значення **allowResizing**;

public Image getIconImage() і **public void setIconImage(Image image)** – визначення та встановлення значка-зображення для даного фрейму;

public int getState() і **public void setState(int state)** – визначення та встановлення стану даного фрейма (стан дорівнює одній з двох констант, наведених вище).

Для створення графічного додатка необхідно виконати наступні дії:

1. Оголосити клас у додатку як клас, який розширює клас Frame.

2. Визначити в класі конструктор, в якому мають бути задані наступні оператори:

2.1. **super** ("ім'я"); – для виклику конструктора класу Frame.

2.2. **setSize** (ширина, висота); – установка розмірів вікна.

2.3. **setLocation** (координата-x, координата-y); – встановлення положення вікна на екрані.

2.4. **addWindowListener(this);** – додати блок прослуховування вікна з обробкою закриття вікна за допомогою методу **windowClosing()**, а також (при необхідності) інших методів інтерфейсу **WindowListener** або класу **WindowAdapter;**

2.5. **setVisible(true);** – зробити вікно видимим.

У конструкторі можуть бути визначені й інші необхідні дії.

3. Створити в методі **main()** за допомогою конструктора новий об'єкт класу, що розширює клас **Frame** і (за необхідності) виконати інші дії.

4. При необхідності виконати первісне промальовування вікна за допомогою методу **paint()**.

Нижче наведений скелет програми графічного додатку:

```
import java.awt.* ;
import java.awt.event.*;
public class GraphicApplication extends Frame
{
...
      // Конструктор класса GraphicApplication
public GraphicApplication (String frameName)
{
      // Вызов конструктора класса Frame
      super(frameName);
      // Добавление блока прослушивания окна
      addWindowListener(
            new WindowAdapter()
            {
                  // Обработка закрытия окна
                  public void windowClosing
                        (WindowEvent e)
                  {
                        …
                        // Завершение программы
                        System.exit(0);
                  }
                  // Другие методы WindowAdapter
            }
      );
      // Установка размера фрейма
      setSize(ширина, высота);
      // Установка положения фрейма
      setLocation(x, y);
      // Сделать фрейм видимым
      setVisible(true);
}
// Метод main()
public static void main(String args[])
{
      // Создание нового объекта
            // класса ApplicationExample
      new GraphicApplication("имя_окна");

}
// Метод paint() (выполняется при прорисовке окна)
public void paint(Graphics g)
{
      // Вызов методов рисования
}
}
```

Можна також створити програму, яка буде запускатися **і як додаток і як аплет**. Для цього необхідно виконати наступні дії:

1. Крім класу, що розширює клас Applet, який запускає аплет, створити в тому ж або окремому файлі клас, який розширює клас Frame.

2. У класі, який розширює клас Applet, задати метод main(), в якому створюється об'єкт класу, що розширює клас Frame. Для цього об'єкту в методі

main() також задається за допомогою методів setSize() і setLocation() розмір вікна програми та за допомогою методу isVisible() це вікно робиться видимим.

2. Оголосити в класі, що розширює клас Frame, об'єкт класу, що розширює клас Applet.

3. Створити в класі, що розширює клас Frame, конструктор, що виконує наступні дії:

3.1. Виклик конструктора класу Frame.

3.2. Створення нового об'єкта класу, що розширює клас Applet.

3.3. Запустити для створеного об'єкту методи init() і start().

3.4. Додати створений об'єкт за допомогою методу add().

4. Для методу windowClosing() інтерфейсу WindowListener включити методи stop() і destroy() для створеного об'єкта класу, що розширює клас Applet.

Нижче наводиться скелет програми, яка може виконуватися і як додаток, і як аплет:

```
import java.applet.*;
import java.awt.* ;
import java.awt.event.*;
public class AppletAndApplication extends Applet
{
...

// Метод main()
public static void main(String args[])
{
      // Объявление нового объекта
            // класса ApplicationFrame
      ApplicationFrame windowForApplication =
            new ApplicationFrame("имя");
      // Установка размера фрейма
      windowForApplication.setSize(ширина, высота);
      // Установка положения фрейма
      windowForApplication.setLocation(x, y);
      // Сделать фрейм видимым
      windowForApplication.setVisible(true);
      ...
}
// Метод init()
public void init()
{
      // Инициализация апплета
}


// Метод paint()
public void paint(Graphics g)
{
      // Первоначальная прорисовка окна апплета
            // или приложения
}
}
class ApplicationFrame extends Frame
```

```
{
// Объявление переменной класса AppletAndApplication
private AppletAndApplication appletFrame;
// Конструктор класса ApplicationFrame
public ApplicationFrame(String frameName)
{
      // Вызов конструктора класса Frame
      super(frameName);

// Добавление блока прослушивания окна
      addWindowListener(
            new WindowAdapter()
            {
                  // Обработка закрытия окна
                  public void windowClosing
                        (WindowEvent e)
                  {
                        ...
                              // Остановка апплета
                        appletFrame.stop();
                        // Уничтожение апплета
                        appletFrame.destroy();
                        // Завершение программы
                        System.exit(0);
                  }
            }
      );
      // Создание нового объекта
            // класса AppletAndApplication
      appletFrame = new AppletAndApplication ();
      // Инициализация апплета
      appletFrame.init();
      // Запуск апплета
      appletFrame.start();
      // Добавление апплета в фрейм
      add(appletFrame);
}
}
```

Вставка зображення в графічний додаток. Щоб вставити зображення у графічний додаток використовуються методи

public abstract Image getImage (String filename)
public abstract Image getImage (URL url)
класу **Toolkit** з пакету **java.awt**.
Звернення до цих методів з компонента виконується через метод
public Toolkit getToolkit ()
класу **Component**, наприклад:
Image img =
getToolkit (). getImage ("D: \ \ images \ \ myimage.gif");
У загальному випадку звернення можна зробити через статичний метод
public static Toolkit getDefaultToolkit ()
класу **Toolkit**, наприклад:
Image img =

Toolkit.getDefaultToolkit (). GetImage ("D: \ \ images \ \ myimage.gif");

Але, крім цих методів, клас Toolkit містить п'ять методів **createImage()**, які повертають посилання на об'єкт типу Image. Зазвичай використовуються методи:

public abstract Image createImage (String fileName) – створює зображення з вмісту графічного файлу filename;

public abstract Image createImage (url address) – створює зображення з вмісту графічного файлу за адресою address.

Створене зображення так само, як і в аплете, виводиться на екран одним з методів **drawImage ()** класу **Graphics**.

2.3 Контрольні питання

1. Які два способи використовуються для реалізації потоків в Java?
2. Як виконується зупинка або завершення потоку?
3. Які операції над потоками визначені в Java?
4. Як виконується переривання потоку і перевірка стану переривання?
5. Що таке потоки-демони і чим вони відрізняються від звичайних потоків?
6. Як задати і отримати значення пріоритету для потоку?
7. Як можна створити групу потоків і які операції визначені для групи потоків в Java?
8. Які засоби синхронізація потоків є в Java?
9. Як виконується синхронізація потоків за допомогою оператора synchronized?
10. Як виконується синхронізація потоків за допомогою методів wait() і notify()?
11. Як функціонує модель делегування подій в Java?
12. Які методи визначені в Java для обробки подій миші і клавіатури?
13. Які дві групи подій визначено в Java, і чим вони відрізняються один від одного?
14. Які блоки прослухування визначені для низькорівневих подій в Java?
15. Як реалізується включення блоку прослухування в програмах Java?
16. Які дії необхідно виконати в програмі, для того, щоб вона могла обробляти події?
17. Як реалізується обробка подій за допомогою адаптерів?

3 Опис приладів, устаткування та інструментів

– Обладнання: IBM-сумісний персональний комп’ютер (ПК).

– Програмне забезпечення: операційна система Windows, Java 2 SDK версії 1.5 та вище. Інтегроване середовище розробки IDE Eclipse.

4 Правила техніки безпеки та охорони праці

Правила техніки безпеки при виконанні лабораторної роботи регламентуються «Правилами техніки безпеки при роботі в комп'ютерній лабораторії».

5 Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

– контроль рівня підготовленості студентів до виконання роботи у формі усного опитування;

– інструктаж по правилах охорони праці перед початком лабораторної роботи та оформлення його підсумків в журналі проведення лабораторних робіт, який ведеться у навчальній лабораторії;

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java у інтегрованому середовищі розробки Eclipse.

5.1 Завдання до лабораторної роботи

1) Використовуючи масив зображень, що формують кадри, створіть аплет з анімацією. В аплеті застосувати подвійну буферизацію. Додати у вікно стану браузера біжучий рядок, що містить інформацію про автора аплету (П.І.Б. та номер групи).
2) Напишіть програму по одному з наведених нижче варіантів.

5.2 Варіанти

1. Створити фрейм з використанням потоків: рядок рухається горизонтально, відбиваючись від кордонів фрейму і змінюючи при цьому випадковим чином свій колір.

2. Створити фрейм з правильними трикутниками, що обертаються в площині екрана навколо свого центру. Кожному об'єкту відповідає потік із заданим пріоритетом.

3. Створити фрейм з використанням потоків: рядок рухається по діагоналі. При досягненні кордонів аплету всі символи рядка випадковим чином змінюють регістр.

4. Створити фрейм з точкою, що рухається по колу з постійною кутовою швидкістю. Згортання фрейму повинно приводити до зміни кутової швидкості руху точки для наступного запуску потоку.

5. Створити фрейм з правильними трикутниками, що обертаються в площині екрану таким чином, що центр обертання переміщається від одного краю фрейму до іншого з постійною швидкістю паралельно горизонтальній осі.

6. Створити фрейм, в якому змоделювати сонячну систему (Сонце, планети і Місяць). Пропорції відстаней від Сонця до планет і від Землі до Місяця можна не дотримуватися. Головне - щоб все було чітко намальовано. Кожна планета (і Місяць теж) керуються окремою ниткою. Одна з планет (Земля) рухається самостійно із заздалегідь заданою швидкістю, рух інших планет здійснюється через синхронізацію з рухом Землі. Всі орбіти вважати круговими.

7. Створити фрейм, в якому зобразити точку, що перетинає з постійною швидкістю вікно зліва направо (справа наліво) паралельно горизонтальній осі. Як тільки точка доходить до межі вікна, в цей момент від лівого (правого) краю з вертикальною координатою Y, обраної за допомогою датчика випадкових чисел, починає свій рух інша точка і т.д. Колір точки також можна вибирати за допомогою датчика випадкових чисел. Для кожної точки створюється власний потік.

8. Створити фрейм з трьома кульками, одночасно літаючими у вікні. С кожною кулькою пов'язаний свій потік зі своїм пріоритетом.

9. Два зображення виводяться у вікно. Потім вони поступово зникають з різною швидкістю в різних потоках (випадковим чином вибираються точки зображення, і їх колір встановлюється в колір фону).

10. Створити фрейм з двома потоками. Один потік породжує випадкове кількість (від 3 до 12) випадкових чисел, які можуть бути довжинами сторін відповідного багатокутника; другий потік отрісовує у вікні у випадковому місці багатокутник, довжини сторін якого породжені першим потоком. Передача чисел відбувається з використанням синхронізації.

11. Гра "камінь-ножиці-папір". Три потоку генерують випадково свої ходи. Провести 10000 ходів. Дані про ходи вивести у файл. Результат у вигляді рахунку вивести у вікні. Взаємодія потоків через синхронізацію.

12. Три потока. Перший генерує три точки на площині в області (0 .. 10, 0 .. 10), другий одну точку в цій же області. Третій приймає дані від перших двох потоків і визначає, чи лежить точка всередині трикутника. Дані про координати всіх точок і результат роботи третього потоку записувати у файл. Провести 10000 ітерацій. Кількість вдалих ітерацій (точка знаходиться всередині трикутника) вивести у вікні.

6 Порядок обробки результатів експериментів та вироблення висновків

На підставі отриманих результатів лабораторної роботи зробить висновки щодо особливостей роботи з анімацією в Java. Нижче наведений приклад виконання першого завдання лабораторної роботи.

6.1 Приклад виконання першого завдання лабораторної роботи

<Вставити параметри>

```
import java.applet.*;
import java.awt.*;

public class zad1 extends Applet implements Runnable {

   String msg,ImageParam;
    Thread t=null;
    boolean stopFlag;
    Image  myIm[];
    int myIndex = 0;

public void init()
{ setBackground(Color.white);
      myIm = new Image[5];
      String ImParam;
      for(int i=0; i<5; i++){
      ImParam = getParameter("s"+i);
      myIm[i]=getImage(getDocumentBase(),ImParam);}
    }

public void start()
    { msg=getParameter("message");
      if(msg==null) msg="Message not found.";
      msg =" "+msg;
      t=new Thread(this);
      stopFlag=false;
      t.start();
}
    public void run(){
        char ch;
        for(;;)
        { try{
            repaint();
            Thread.sleep(150);
            ch=msg.charAt(0);
            msg=msg.substring(1,msg.length());
            msg+=ch;
            myIndex++;
            if(myIndex==5) myIndex=0;
            if(stopFlag) break;}
          catch(InterruptedException e) {}
        } }

    public void stop()
    { stopFlag = true; t=null;}

 public void  update (Graphics g) {
 paint (g);}

public void paint(Graphics g)
{    showStatus(msg);
     Image current=myIm[myIndex];
     Image buffer = createImage(getWidth(), getHeight());
```

```
            Graphics2D g2d = (Graphics2D) buffer.getGraphics();
            g2d.drawImage(current,0,0,this);
            g.drawImage(buffer,0,0,this);
          }
    }
```

7 Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

- титульний лист, де вказані номер і назва лабораторної роботи, відомості про виконавця;
- номер варіанта роботи та текст завдання;
- відповіді на контрольні запитання до лабораторної роботи;
- листинг програмного коду;
- результати виконання програми;
- висновки по підсумках роботи.

Підготовлений звіт надається викладачу на перевірку та захищається студентом на останьому занняті з даної лабораторної роботи.

Лабораторна робота 3

Створення графічних застосувань. Використання системи Swing в Java

1 Мета роботи

Метою роботи є придбання навичок програмування графічних додатків Java з використанням системи Swing.

Після виконання лабораторної роботи студенти мають оволодіти вміннями проектувати графічний інтерфейс користувача; обробляти семантичні події; використовувати для розміщення компонентів менеджери компонування.

2 Завдання на підготовку до лабораторної роботи

2.1 Перелік знань та вмінь, на які необхідно спиратися при виконанні роботи

Знання:

- модель делегування подій;
- класи та інтерфейси подій;
- класи-адаптери;
- менеджери компонування;
- компоненти бібліотек AWT і Swing.

Вміння:

- проектувати графічний інтерфейс користувача інформаційних систем;
- обробляти семантичні події;
- використовувати для розміщення компонентів менеджери компонування.

2.2 Теоретичні відомості

2.2.1 Модель делегування подій в Java

Модель делегування подій в Java працює наступним чином. Кожен елемент взаємодії між інтерфейсом користувача і програмою визначається як **подія**. Класи додатків з'ясовують свою зацікавленість в деякому очікуваному певним компонентом події шляхом опитування компонента і видачі йому пропозиції помістити в список відомості про **блок прослуховування** даного компонента (listener). Коли відбувається деяка подія, джерело події повідомляє про нього зареєстровані блоки прослуховування.

Події, як і виключення, утворюють ієрархію класів. Коренем цієї ієрархії є клас **EventObject**, що розширює клас Object. Події, пов'язані з

графічним інтерфейсом користувача, в свою чергу, утворюють гілку в ієрархії класів подій, коренем якої є клас **AWTEvent**, що розширює клас EventObject.

Обробка низькорівневих подій. Існують два різних типи подій – низькорівневі і семантичні.

Низькорівневі події пов'язані з фізичними аспектами інтерфейсу користувача – клацаннями миші, натисканнями клавіш клавіатури і т.п.

Семантичні події будуються на базі низькорівневих подій. Наприклад, для вибору команди меню може знадобитися розкрити список команд першого клацанням миші, а потім другим клацанням вибрати в ньому необхідну команду. Ця послідовність низькорівневих подій може бути об'єднана в одну семантичну подію.

Класи низькорівневих і високорівневих подій, пов'язаних з GUI і AWT, містяться в пакеті **java.awt.event**.

У кожному класі подій визначено відповідні властивості і методи для обробки подій. У таблиці наведені основні методи для класів KeyEvent (обробка подій, пов'язаних з клавіатурою) і MouseEvent (обробка подій, пов'язаних з мишею):

Клас	Метод	Результат
KeyEvent	public char getKeyChar()	Повертає символ, відповідний натиснутій клавіші
	public int getKeyCode()	Повертає ціле число – код натиснутої клавіші
	public String getKeyText()	Повертає ім'я натиснутої клавіші, наприклад "Shift", "Ctrl" або "S"
MouseEvent	public int getClickCount()	Повертає число клацань миші, пов'язаних з цією подією
	public Point getPoint()	Повертає об'єкт Point, що містить координати курсора миші в момент, коли відбулася подія
	public int getX()	Повертає координату **x** курсора миші в момент, коли відбулася подія
	public int getY()	Повертає координату **y** курсора миші в момент, коли відбулася подія

Блоки прослуховування. Майже кожен тип події, має пов'язаний з ним інтерфейс блоку прослуховування, в якому оголошені методи обробки даного типу події. Ці інтерфейси в свою чергу утворюють ієрархію інтерфейсів, коренем якої є інтерфейс EventListener.

Кожен блок прослуховування містить оголошення методів, що обробляють дану подію. Список методів блоків прослуховування для низькорівневих подій KeyListener, MouseListener, MouseMotionListener і WindowListener наведено нижче.

Интерфейс	Методи
KeyListener	public void keyPressed(KeyEvent e) public void keyReleased(KeyEvent e) public void keyTyped(KeyEvent e)
MouseListener	public void mouseClicked(MouseEvent e) public void mouseEntered(MouseEvent e) public void mouseExited(MouseEvent e) public void mousePressed(MouseEvent e) public void mouseReleased(MouseEvent e)
MouseMotionListener	public void mouseDragged(MouseEvent e) public void mouseMoved(MouseEvent e)
WindowListener	public void windowActivated(WindowEvent e) public void windowClosed(WindowEvent e) public void windowClosing(WindowEvent e) public void windowDeactivated (WindowEvent e) public void windowDeiconified (WindowEvent e) public void windowIconified(WindowEvent e) public void windowOpened(WindowEvent e)

Для того, щоб блок прослуховування міг фіксувати і обробляти події він повинен бути включений за допомогою методів додавання блоків прослуховування в клас (ці методи визначені в класі Component).

Методи включення блоків прослуховування мають наступний загальний формат:

public void addXXXX (XXXX l)

де **XXXX** – ім'я відповідного блоку прослуховування, наприклад включення в клас блоку прослуховування для подій клавіатури реалізується за допомогою методу

public void addKeyListener (KeyListener l)

Таким чином, для обробки події необхідно:

Зробити доступними методи, визначені у відповідному інтерфейсі або інтерфейсах, оголосивши клас як реалізацію (implementation) відповідного інтерфейсу або інтерфейсів.

Включити необхідні блоки прослуховування за допомогою відповідного методу **addXXXX**.

Описати в програмі всі методи всіх реалізованих інтерфейсів (якщо який-небудь метод не використовується в класі, він оголошується з порожнім

тілом – {}). У методах інтерфейсів зазвичай використовуються методи і властивості відповідного класу обробки подій.

Скелет програми (для обробки подій клавіатури) представлений нижче:

```
class KeyTest implements KeyListener
{
      ...
      addKeyListener(имя-объекта-KeyListener);
      ...
      public void keyPressed(KeyEvent e)
      {
      Обработка события нажатия клавиши или пусто
      }
      public void keyReleased(KeyEvent e)
      {
      Обработка события отпускания клавиши или пусто
      }
      public void keyTyped(KeyEvent e)
      {
            Обработка ввода символа или пусто
      }
}
```

Для того, щоб у класі можна було описувати тільки ті методи, які необхідно в JDK були введені абстрактні спеціальні класи – **адаптери**, що містять описи тих же методів, що і відповідні блоки прослуховування. Імена цих класів формуються так само, як і блоків прослуховування, тільки замість суфікса **Listener** в них використовується суфікс **Adapter**, наприклад, **KeyAdapter** або **MouseAdapter**.

Якщо створити всередині даного класу підклас, що розширює клас відповідного адаптера (наприклад, підклас myKeyAdapter, що розширює клас KeyAdapter), то всередині нього можна перевизначити тільки ті методи адаптера, які необхідно використовувати. У цьому випадку у відповідному методі addXXXX() як параметр можна задати екземпляр створеного підкласу, наприклад,

addKeyListener (new myKeyAdapter ());

Компактнішим записом, який часто використовується програмістами на Java, є безпосереднє визначення підкласу в параметрі відповідного методу addXXXX (), наприклад:

```
addKeyListener(
                  new KeyAdapter()
            {
                  // Описание подкласса класса KeyAdapter
            }
);
```

2.2.2 Компоненти управління AWT і обробка подій

Поряд із засобами малювання, використання кольорів і шрифтів в пакеті java.awt визначені також елементи управління.

Елементи управління (controls) – це компоненти, які надають користувачеві різні способи взаємодії з додатком. Ці елементи реалізовані в наступних класах пакета java.awt:

Клас **Button** – кнопки;

Клас **Label** – текст;

Клас **Checkbox** – прапорці та перемикачі;

Клас **Choice** – розкривні списки;

Клас **List** – списки;

Клас **TextField** - рядка введення;

Клас **TextArea** – поля введення;

Клас **ScrollBar** – смуги прокрутки;

Клас **Canvas** – малюнки.

На відміну від низькорівневих подій, семантичні події для компонент AWT визначаються для класів. Оскільки в інтерфейсах для семантичних подій визначений тільки один метод, тому для елементів управління немає необхідності у використанні абстрактних класів-адаптерів, як для низькорівневих подій.

Интерфейс ActionListener і клас ActionEvent. Для класів Button, List і TextField визначений блок прослуховування ActionListener з єдиним методом

public void actionPerformed (ActionEvent e),

викликається, коли відбувається дія для заданого об'єкта класу Button, List або TextField.

Клас **ActionEvent** обробляє події, пов'язані з натисканням кнопки, вибором елемента зі списку або з натисканням клавіші Enter в текстовому полі.

Для класу **ActionEvent** визначені наступні статичні final змінні типу int:

ACTION_PERFORMED – дія відбулася;

ALT_MASK – натиснута клавіша Alt;

SHIFT_MASK – натиснута клавіша Shift;

CTRL_MASK – натиснута клавіша Ctrl;

ACTION_FIRST – перший номер у списку ідентифікаторів, використовуваних для подій дії;

ACTION_LAST – останній номер у списку ідентифікаторів, використовуваних для подій дії.

Методи

public String getActionCommand()

public int getModifiers()

public long getWhen()

public String paramString()

дозволяють отримати відповідно командний рядок, пов'язаний з даною дією; ключ-модифікатор для даної дії; час лічильника (у мілісекундах) для даної дії і рядок параметра, що ідентифікує дану дію (зазвичай використовується для налагодження).

Інтерфейси ItemListener, ItemSelectable і клас ItemEvent. Для класів Choice, Checkbox і List визначений блок прослуховування ItemListener з єдиним методом

public void itemStateChanged (ItemEvent e),

викликається, коли відбувається вибір або скасування вибору елементів в об'єктах класу Choice, Checkbox або List.

Клас **ItemEvent** містить змінні і методи для обробки подій, пов'язаних з вибором елемента або скасуванням вибору елемента.

Для класу **ItemEvent** визначені наступні статичні final змінні типу int:

SELECTED – елемент вибраний;

DESELECTED – вибірка елемента скасована;

ITEM_STATE_CHANGED – стан елемента змінено;

ITEM_FIRST – перший номер у списку ідентифікаторів, використовуваних для подій вибору елемента;

ITEM_LAST – останній номер у списку ідентифікаторів, використовуваних для подій вибору елемента.

Методи

public Object getItem ()

public int getStateChange ()

public String paramString ()

дозволяють отримати відповідно елемент, який викликав подію; зміну стану елемента (обраний або не вибрано) і рядок параметра, що ідентифікує вибір елемента (зазвичай використовується для налагодження).

Метод

public ItemSelectable getItemSelectable ()

повертає об'єкт інтерфейсу **ItemSelectable**. Цей інтерфейс призначений для об'єктів, в яких допустимий вибір одного або більше елементів, і містить оголошення методів

public void addItemListener (ItemListener l)

public void removeItemListener (ItemListener l)

додавання та видалення блоків прослуховування вибору або скасування вибору елементів зі списку, а також метод

public Object [] getSelectedObjects (),

який повинен повертати масив вибраних об'єктів або null, якщо не вибрано жодного об'єкта.

Обробка семантичних подій. Включення блоку прослуховування, як і для низькорівневих подій, виконується за допомогою відповідного методу addXXXX(), наприклад, addActionListener(). Для операції включення необхідно вказати, для якого об'єкта AWT додається даний блок прослуховування, причому зазвичай, як і для адаптера, метод блоку прослуховування перевизначається безпосередньо в параметрі відповідного методу addXXXX (), наприклад:

```
Button myButton;
…
myButton.addActionListener
(
      new ActionListener()
      {
      public void actionPerformed(ActionEvent e)
      {
            // Операторы, реализующие обработку нажатия
            // кнопки myButton
      }
      }
);
```

Для завдання кольору переднього плану (кольори тексту), кольори фону та шрифту написів для елементів управління AWT можна використовувати відповідно методи

public void setForeground(Color c)
public void setBackground(Color c)
public void setFont(Font f)

Отримати ці значення для елемента керування можна за допомогою методів

public Color getForeground()
public Color getBackground()
public Font getFont()

2.2.3 Основні компоненти Swing

Компоненти Swing можна розділити на наступні типи:

контейнери верхнього рівня (класи **JWindow**, **JFrame**, **JDialog** і **JApplet**);

спеціалізовані контейнери (класи **JInternalFrame**, **JLayeredPane**, **JRootPane** і **JOptionPane**);

загальноцільові контейнери (класи **JPanel**, **JScrollPane**, **JSplitPane**, **JTabbedPane** і **JToolBar**);

компоненти управління (класи **JButton**, **JCheckBox**, **JRadioButton**, **JToggleButton**, **JComboBox**, **JList**, **JMenuBar**, **JMenu**, **JMenuItem**, **JCheckboxMenuItem**, **JRadioButtonMenuItem**, **JSeparator** і **JSlider**);

нередаговані інформаційні компоненти (класи **JLabel**, **JProgressBar** і **JToolTip**);

редаговані інформаційні компоненти (класи **JColorChooser**, **JFileChooser**, **JTable**, **JTree**, **JTextField**, **JPasswordField**, **JTextArea**, **JEditorPane** і **JTextPane**).

На відміну від компонентів AWT, компоненти системи Swing здатні працювати тільки за моделлю делегування подій.

2.2.4 Контейнери верхнього рівня і спеціалізовані контейнери

Так само, як і для AWT, для створення вікон графічних додатків використовується не клас **JWindow**, а клас **JFrame** (вікна, створювані класом JWindow не містять найменування вікна і кнопок управління вікном). Додатки з графічним інтерфейсом використовує, принаймні, один фрейм. Аплети також можуть використовувати фрейми.

Для створення вікон, які залежать від іншого вікна (наприклад, зникають, коли згортається вікно, в якому вони використовуються) застосовуються діалогові вікна класу **JDialog.**

Аплети, що використовують компоненти Swing, повинні бути підкласами класу **JApplet.**

Будь-яка програма, яка використовує компоненти Swing, містить, принаймні, один контейнер верхнього рівня. Цей контейнер є коренем ієрархії контейнерів, що містять всі компоненти Swing.

Як правило, окремий графічний додаток має, принаймні, одну ієрархію контейнерів, в якій коренем є JFrame. Діалогове вікно або аплет також утворюють ієрархію контейнерів, коренем якої є JDialog або JApplet. Наприклад, якщо додаток містить одне головне вікно і два діалогових вікна, то він містить три ієрархії контейнерів.

Коренева панель. Кожен контейнер верхнього рівня базується на проміжному, прихованому, контейнері, званому кореневої панеллю (root pane). Коренева панель визначена в класі JRootPane.

Сама коренева панель зазвичай не використовується, а використовуються її компоненти, які коренева панель (клас) надає фрейму (або іншьому контейнеру верхнього рівня). Коренева панель містить такі компоненти:

- багатошарова панель (layered pane);
- панель вмісту (content pane);
- рядок меню (menu bar) – необов'язковий компонент;
- скляна панель (glass bar).

Єдиним обов'язковим контейнером верхнього рівня є панель вмісту.

Панель вмісту містить всі компоненти Swing (кнопки, написи, текстові поля тощо). Оскільки для контейнерів верхнього рівня вміст вікна визначається за допомогою **JRootPanel** і повинно, на відміну від вікон AWT, визначатися вручну, для додавання компонент або установки менеджера компоновки використовуються не методи **add()** і **setLayout()**, а методи отримання та установки панелі вмісту:

public Container getContentPane()

і

public void setContentPane (Container contentPane),

які визначені в класах **JFrame**, **JDialog**, **JApplet** і **JInternalFrame**.

Так, додавання текстового поля у фрейм, діалогове вікно, аплет або внутрішній фрейм з установкою менеджера компоновки FlowLayout виглядає наступним чином:

```
Container contentPane = getContentPane();
JTextField inputField = new JTextField(15);
contentPane.setLayout (new FlowLayout());
contentPane.add(inputField);
```

Слід зазначити, що для всіх контейнерів верхнього рівня (включаючи JApplet) менеджером компонування за умовчанням є BorderLayout.

До контейнера верхнього рівня може бути додано рядок меню (menu bar). Рядок меню також розташовується всередині головного контейнера, але за межами панелі вмісту.

Діалогові вікна можна створювати або за допомогою класу **JDialog**, або використовуючи клас **JOptionPane**.

Основними конструкторами класу **JDialog** є:

JDialog();
JDialog(Dialog owner)
JDialog(Dialog owner, boolean modal)
JDialog(Dialog owner, String title)
JDialog(Dialog owner, String title, boolean modal)
JDialog(Frame owner)
JDialog(Frame owner, boolean modal)
JDialog(Frame owner, String title)
JDialog(Frame owner, String title, boolean modal)

Перший конструктор створює немодальне діалогове вікно без імені. Параметри owner, title і modal в інших конструкторах задають відповідно батька даного діалогового вікна, ім'я вікна і вікна (true – модальне вікно або false – немодальне вікно).

Вікно класу **JDialog** має ті ж риси, що **JFrame** і пряме використання об'єктів класу **JDialog** аналогічно прямому використанню об'єкта **JFrame**.

Клас **JOptionPane** полегшує висновок стандартних діалогових або інформаційних вікон.

Для класу **JOptionPane** визначено такі основні конструктори:

JOptionPane();

JOptionPane (Object message, int messageType, int optionType, Icon icon, Object [] options, Object initialValue);

Крім цього визначено конструктори, що містять відповідно тільки перший параметр, тільки перші два параметри, тільки перші три параметри, тільки перші чотири параметри, тільки перші п'ять параметрів другого конструктора.

Всі використання цього класу зазвичай зводяться до виклику одного з його методів:

static int showConfirmDialog (Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) – задає питання на який потрібує відповідь типу Yes, No або Cancel. Існують також наступні три методи **showConfirmDialog():** метод, що містить перші два параметри, метод, що містить перші чотири параметри і метод, що містить перші п'ять параметрів, наведеного вище методу;

static Object showInputDialog (Component parentComponent, Object message, String title, int messageType, Icon icon, Object [] selectionValues, Object initialValue) – дозволяє ввести деякий текст. Існують також наступні п'ять методів **showInputDialog()**: метод, що містить другий параметр, метод, що містить другий і сьомий параметри, метод, що містить перші два параметри, метод, що містить перший, другий і сьомий параметри, і метод, що містить перші три параметри і сьомий параметр;

static void showMessageDialog (Component parentComponent, Object message, String title, int messageType, Icon icon) – дозволяє вивести деяке повідомлення. Існують також наступні два методи **showMessageDialog**: метод, що містить перші два параметри і метод, що містить перші чотири параметра;

static int showOptionDialog (Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object [] options, Object initialValue) – об'єднує функції перерахованих вище діалогових вікон.

Параметри в наведених конструкторах і методах мають наступний зміст:

parentComponent – визначає компонент, який є батьком даного діалогового вікна (якщо цей параметр дорівнює null, як батька передбачається фрейм за умовчанням);

message – описову повідомлення, що поміщається в діалогове вікно (слід мати на увазі, що це об'єкт класу Object);

selectionValues – масив об'єктів, що розглядається як набір повідомлень;

icon – зображення, що виводиться в діалозі (значення за замовчуванням визначається значенням параметра **messageType**);

messageType – визначає тип повідомлення. Допустимі наступні значення: ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE і PLAIN_MESSAGE;

optionType – визначає набір кнопок, які будуть виведені в нижній частині діалогового вікна: DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION і OK_CANCEL_OPTION

options – більш детальний опис набору кнопок, які будуть виведені в нижній частині діалогового вікна (якщо задано об'єкт класу Component, він прямо додається в нижню частину діалогового вікна, якщо об'єкт типу Icon, створюється кнопка класу JButton, для об'єктів інших класів проводиться перетворення об'єкта в рядок, який виводиться на кнопці класу JButton);

title – ім'я діалогового вікна;

initialValue – вводиться значення за замовчуванням.

Для методів припустимі наступні можливі значення значення, що повертається: YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION і CLOSED_OPTION.

2.2.5 Загальноцільові контейнери

Клас JPanel є варіантом класу Panel в AWT. Чотири конструктора класу:

public JPanel()
public JPanel(LayoutManager layout)
public JPanel(boolean isDoubleBuffered)
public JPanel(LayoutManager layout, boolean isDoubleBuffered)

забезпечують створення об'єкта **JPanel** відповідно під управлінням менеджера компоновки **FlowLayout**, під управлінням заданого менеджера компоновки, з вбудованою підтримкою подвійний буферизації і, нарешті, під управлінням заданого менеджера компоновки і з вбудованою підтримкою подвійної буферизації.

Робота з об'єктами класу **JPanel** не відрізняється від роботи з об'єктами класу Panel, за винятком того, що панель додаються не у фрейм, а в панель вмісту фрейма.

Клас JTabbedPane панель з вкладками (tabbed pane) введена в Swing замість менеджера CardLayout, що використовувався в AWT, тобто менеджер компоновки CardLayout в Swing не реалізований. Кожна вкладка має заголовок. Коли користувач вибирає вкладку, її вміст стає видимим. Тільки одна з вкладок може бути обрана одночасно. Панель з вкладками зазвичай використовується для установки параметрів конфігурації.

Панель з вкладками реалізована класом **JTabbedPane**, який має такі конструктори

JTabbedPane()

JTabbedPane(int tabPlacement)

JTabbedPane(int tabPlacement, int tabLayoutPolicy)

Перший конструктор створює нову панель з вкладками, в якій заголовки вкладок розташовані зверху, а другий дозволяє визначити розташування заголовків вкладок (**JTabbedPane.TOP** – зверху, **JTabbedPane.BOTTOM** – знизу, **JTabbedPane.RIGHT** – вправо або **JTabbedPane.LEFT** – вліво). Третій конструктор додатково задає правило розташування вкладок (**JTabbedPane.WRAP_TAB_LAYOUT** – вкладки розташовуються в кілька рядів, якщо вони не поміщаються в одному ряду – значення за замовчуванням і **JTabbedPane.SCROLL_TAB_LAYOUT** – вкладки прокручуються, якщо вони не поміщаються в одному ряду).

Формування в програмі панелі з вкладками виконується за допомогою наступної процедури:

1. Створити об'єкт **JTabbedPane**.

2. Для додавання кожної вкладки в панель викликати один з методів **addTab()**:

public void addTab(String title, Icon icon, Component component, String tip)

public void addTab(String title, Icon icon, Component component)

public void addTab(String title, Component component)

Аргументи цих методів визначають заголовок вкладки **title**, компоненту **component**, який вона містить, зображення **icon** і спливаючу підказку **tip**.

3. Додати панель з вкладками в панель змісту вікна.

Для відстеження вибору тієї чи іншої вкладки використовується блок прослуховування **ChangeListener**, який додається або видаляється за допомогою методів

public void addChangeListener (ChangeListener l)

і

public void removeChangeListener (ChangeListener l)

класу **JTabbedPane**. В інтерфейсі **ChangeListener** визначений єдиний метод

public void stateChanged (ChangeEvent e),

в якому за допомогою методів

public Component getSelectedComponent ()

або

public int getSelectedIndex ()

класу **JTabbedPane** можна отримати вибраний компонент або індекс обраного компонента.

Клас JSplitPane розщеплені панелі (клас JSplitPane) використовуються для розділення виведеної панелі на дві компоненти. конструктори классу

public JSplitPane()

public JSplitPane(int newOrientation)

public JSplitPane(int newOrientation, boolean newContinuousLayout)

public JSplitPane(int newOrientation, Component newLeftComponent, Component newRightComponent)

public JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent,Component newRightComponent)

задають у параметрі **newOrientation** вирівнювання зліва направо з використанням константи **JSplitPane.HORIZONTAL_SPLIT** або зверху вниз з використанням константи **JSplitPane.VERTICAL_SPLIT**, в параметрі **newContinuousLayout** безперервну перерисовку компонент при переміщенні роздільника (значення true) або перемальовування компонент після закінчення переміщення роздільника (значення false), а в параметрах **newLeftComponent** і **newRightComponent** – компоненти розщепленої панелі.

Положення роздільника в панелі задається за допомогою методу

public void setDividerLocation (int location),

який задає в пікселях позицію **x** або **y** роздільника (залежно від вертикальної або горизонтальної орієнтації панелей). Метод

public void setOneTouchExpandable (boolean newValue)

дозволяє задати на розділовій смузі кнопки (зі стрілками) для швидкого згортання і розгортання однієї з панелей розщепленої панелі.

2.2.6 Значки, написи і кнопки

Интерфейс Icon і клас ImageIcon

Значки (icons) реалізуються в Swing за допомогою інтерфейсу **Icon** (насправді вони не є компонентами і їх можна використовувати практично з усіма компонентами Swing). Реалізацією інтерфейсу **Icon**, що створює значок із зображення, є клас **ImageIcon**. Двома основними конструкторами цього класу є:

ImageIcon (String filename)

ImageIcon (URL url)

Перша форма використовує зображення у файлі з ім'ям **filename** (використовується в графічних додатках) а друга форма – в ресурсі, розташованому за URL-адресою url (використовується в апплетах Swing).

На відміну від класу **Image**, в класі **ImageIcon** зображення завантажується попередньо з використанням методів класу **MediaTracker**.

Для створення власних зображень можна безпосередньо реалізовувати інтерфейс **Icon**.

Клас JLabel

Написи Swing є об'єктами класу **JLabel**. Цей об'єкт може відображати тексти та/або значки. Основними його конструкторами є:

JLabel(Icon image)

JLabel(String text)

JLabel (String text, Icon image, int horizontalAlignment)

У цих конструкторах **text** і **image** – текст і значок, використовуваний в напису. Параметр **horizontalAlignment** визначає вирівнювання і має значення **LEFT**, **RIGHT** або **CENTER**. Ці константи визначені в інтерфейсі **SwingConstants**, поряд з кількома іншими, використовуваними класами системи Swing.

Значок і текст, пов'язаний з написом, можна зчитувати і записувати такими методами:

public Icon getIcon();
public String getText();
public void setIcon(Icon image);
public void setText(String text);

Клас AbstractButton

Кнопки Swing є підкласами класу **AbstractButton**. Цей клас містить багато методів, які дозволяють керувати поведінкою кнопок, прапорців і перемикачів. Наприклад, можна визначати різні значки для відображення компонента, коли він віджатий (disabled), натиснутий (pressed), або обраний (selected). Окремий значок можна використовувати як значок "наїзду" (rollover), який відображається, коли курсор миші встановлений поверх цього компонента ("наїхав" на нього). Нижче наведені описи методів, які керують цією поведінкою:

public void setDisabledIcon(Icon image)
public void setPressedIcon (Icon image)
public void setSelectedIcon(Icon image)
public void setRolloverIcon(Icon image)

Текст, зв'язаний з кнопкою, можна читати і записувати за допомогою методів:

public String getText()
public void setText(String text)

У класі **AbstractButton** визначені також методи

public int getMnemonic ()

і

public void setMnemonic (int mnemonic)

відповідно для отримання і установки «гарячих» клавіш. Як параметр при установці «гарячої» клавіші задається одне з статичних властивостей класу **KeyEvent**. Ці властивості мають загальне ім'я **VK_XXX**, де **XXX** – ім'я однієї з клавіш, наприклад, **VK_D**.

При натисканні кнопки конкретні підкласи **AbstractButton** генерують події **Action**. Блоки прослуховування реєструють і скасовують регістрацію для цих подій за допомогою таких методів:

void addActionListener(ActionListener l)
void removeActionListener(ActionListener l)

Клас JButton

Клас **JButton** є підкласом класу **AbstractButton** і забезпечує функціональні можливості кнопки. Цей клас дозволяє пов'язати з кнопкою зображення, текст або і те й інше. Деякі з його конструкторів:

JButton(Icon image)
JButton(String text)
JButton(String text, Icon image)

Тут **text** і **image** – напис і зображення, які використовуються для кнопки.

Клас **JButton** успадковує властивості інтерфейсу **SwingConstants**, а також властивості й методи класу **AbstractButton**.

2.2.7 Прапорці та перемикачі

Клас **JCheckBox**, який забезпечує функціональні можливості прапорця, є конкретною реалізацією класу **AbstractButton**. Деякі з його конструкторів:

JCheckBox(Icon image)
JCheckBox(Icon image, boolean state)
JCheckBox(String text)
JCheckBox(String text, boolean state)
JCheckBox (String text, Icon image)
JCheckBox(String text, Icon image, boolean state)

У цих конструкторах використовуються наступні параметри: **image** – зображення для прапорця, **text** – напис. Якщо **state** має значення **true**, то прапорець спочатку вибраний.

Стан прапорця може бути змінено за допомогою методу
void public void setSelected (boolean b),
а отримати стан прапорця можна за допомогою методу
public boolean isSelected().

Перемикачі підтримуються класом **JRadioButton**, який також є конкретною реалізацією класу **AbstractButton**. Нижче наведені деякі з його конструкторів:

JRadioButton(Icon image)
JRadioButton(Icon image, boolean state)
JRadioButton(String text)
JRadioButton(String text, boolean state)
JRadioButton(String text, Icon image)
JRadioButton(String text, Icon image, boolean state)

де параметри мають той же зміст, що і для класу **JCheckBox**.

Перемикачі повинні бути об'єднані в групу, де в кожен момент може бути вибраний тільки один елемент. Наприклад, якщо користувач клацає по перемикачу, який знаходиться в групі, будь-який перемикач в цій групі, що був попередньо натиснутий, автоматично скидається. Щоб створити групу

кнопок, створюється об'єкт класу **ButtonGroup**. Для цієї мети використовується єдиний конструктор класу

public ButtonGroup().

Елементи до групи перемикачів додаються за допомогою методу

public void add (AbstractButton a),

а кількість перемикачів в групі можна отримати за допомогою методу

public int getButtonCount().

Батьківським класом і для **JCheckBox** і для J**RadioButton** є клас **JToggleButton**, який не має еквівалента в AWT. Він веде себе як об'єкт класу Button, який після натискання на нього залишається в натиснутому стані.

2.2.8 Списки

Клас JComboBox

Замість класу Choice в AWT, Swing забезпечує комбіноване поле (combo box) – комбінація текстів поля і списку, що розкривається (клас JComboBox). Комбіноване поле зазвичай відображає один вхід (елемент) списку. Однак воно може також відображати і список, що розкривається, який дає можливість користувачеві вибирати різні елементи. Можна також ввести з клавіатури своє значення елементу списку в текстове поле. Два найпоширеніші конструктора:

JComboBox ()

JComboBox (Object [] items)

дозволяють створити порожній список з вибором або задати масив об'єктів даного списку.

Елементи додаються до списку вибору за допомогою методу

void addItem (Object obj)

де **obj** – об'єкт, який буде додано до комбінованого полю.

Решта методів класу аналогічні відповідним методам класу Choice, однак додані методи

public void setEditable (boolean aFlag)

і

public boolean isEditable (),

дозволяють встановити можливість введення значення в список і перевірити прапорець можливості введення свого значення в список.

Для обробки подій, пов'язаних з комбінованим списком, можна використовувати або блоки прослуховування **ItemListener**, або блоки прослуховування **ActionListener**, описані раніше для компонент AWT.

Клас JList

Аналогом класу List в Swing є клас **JList**. Два найпоширеніші конструктора цього класу

JList ()

JList (Object [] items)

дозволяють задати або порожній список, або список, що містить задані об'єкти (зверніть уваги, що елементами списків в Swing можуть бути не тільки рядки, але довільні об'єкти).

Прокрутка списку не виконується в **JList** автоматично, а виконується за допомогою приміщення списку в об'єкт класу **JScrollPane.**

Набір методів у класі **JList** істотно розширений у порівнянні з класом List (так, наприклад, є можливість задавати колір тексту і фону для виділених елементів) і, крім того, замість інтерфейсу **ItemListener** для списків **JList** використовується інтерфейс **ListSelectionListener**, блок прослуховування якого додається і видаляється з допомогою методів

public void addListSelectionListener (ListSelectionListener listener)

і

public void removeListSelectionListener (ListSelectionListener listener)

класу **JList**.

Метод

public void valueChanged (ListSelectionEvent e)

інтерфейсу обробляє події, пов'язані з вибором елементів списку, а метод

public Object [] getSelectedValues ()

класу **JList** дозволяє отримати список обраних об'єктів.

2.2.9 Текстові компоненти

Клас JTextComponent

Батьківським класом текстових компонент в Swing є клас **JTextComponent**. Він забезпечує функціональні можливості, які є загальними для всіх текстових компонент.

Можливості компонента **JTextComponent** бібліотеки Swing набагато перевершують вимоги, що пред'являються до звичайного текстового поля або напису. Даний компонент надає розробникам великий набір методів роботи з текстом, наприклад:

public void copy () – копіювати в буфер обміну;

public void cut () – вирізати в буфер обміну;

public void paste () – вставити з буфера обміну;

public String getSelectedText () – отримати виділений текст;

public int getSelectionStart () – визначити точку початку виділеного тексту;

public int getSelectionEnd () – визначити точку кінця виділеного тексту;

public String getText () – ввести текст з поля;

public void setText (String text) – помістити текст в поле;

public void setEditable (boolean b) – дозволити редагування;

public void setCaretPosition (int position) – помістити курсор в зазначену позицію.

Класи JTextField, JPasswordField і JTextArea

Підкласи класу **JTextComponent** – **JTextField** і **JTextArea** практично аналогічні класам TextField і TextArea пакета AWT. Між ними існують лише такі відмінності:

рядки введення паролів задаються в Swing за допомогою окремого класу **JPasswordField** (конструктори цього класу аналогічні конструкторам класу **JTextField**, метод **public char[] getPassword()** дозволяє отримати значення введеного пароля, а методи **public void setEchoChar (char c)** і **public char getEchoChar ()** – встановити і отримати значення луна-символу);

прокрутка об'єкта класу **JTextArea** (так само, як і об'єкта класу JList) виконується за допомогою приміщення списку в об'єкт класу **JScrollPane**.

2.2.10 Меню

Компоненти **JMenuBar**, **JMenu**, **JMenuItem** і **JCheckBoxMenuItem** в Swing за своїми функціональними можливостями і використанню аналогічні компонентам MenuBar, Menu, MenuItem і CheckboxMenuItem, використовуваним в AWT.

Новий компонент **JRadioButtonMenuItem** дозволяє організувати в меню перемикачі. Група альтернативних перемикачів формується таким чином: спочатку створюється об'єкт класу ButtonGroup, а потім в цей об'єкт за допомогою методу add() додаються об'єкти класу **JRadioButtonMenuItem**.

Класи, що реалізують меню в Swing, відрізняються від відповідних класів AWT наступними основними особливостями:

оскільки всі класи, крім **JMenuBar**, є підкласами класу **JAbsractButton**, для меню та пунктів меню можна задавати не тільки текст, але й зображення (для цього в конструктори цих класів введений параметр Icon image і додані методи, пов'язані із зображеннями);

рядок меню встановлюється за допомогою методу **public void setJMenuBar (JMenuBar menubar)** класу **JFrame**;

доданий інтерфейс **MenuListener** для обробки подій пов'язаних з меню (об'єктами класу **JMenu**). Методи цього інтерфейсу

public void menuSelected(MenuEvent e)

public void menuDeselected(MenuEvent e)

public void menuCanceled(MenuEvent e)

дозволяють обробляти події, пов'язані з вибором меню, скасуванням вибору меню і скасуванням меню. Блоки прослуховування для меню додаються і видаляються за допомогою методів

public void addMenuListener (MenuListener l)

і

public void removeMenuListener (MenuListener l);

мнемонічні клавіші («гарячі» клавіші) для меню та пунктів меню на відміну від AWT діють тільки тоді, коли відповідні меню або пункти меню виведені на екрані. Для того, щоб клавіші діяли незалежно від того, чи видно відповідний елемент меню на екрані, необхідно задати такі клавіші за допомогою методу **public void setAccelerator (KeyStroke keyStroke);**

для пунктів меню введений інтерфейс **MenuDragMouseListener**, методи якого

public void menuDragMouseEntered(MenuDragMouseEvent e)
public void menuDragMouseExited(MenuDragMouseEvent e)
public void menuDragMouseDragged(MenuDragMouseEvent e)
public void menuDragMouseReleased(MenuDragMouseEvent e)

дозволяють організувати обробку подій, пов'язаних з переміщенням курсора миші над пунктами меню (наприклад, зміни виду пункту меню при наведенні на нього курсора миші). Блоки прослуховування для меню додаються і видаляються за допомогою методів

public void addMenuDragMouseListener(MenuDragMouseListener l)
і
public void removeMenuDragMouseListener(MenuDragMouseListener l);

крім цього, для пунктів меню введений інтерфейс **MenuKeyListener**, методи якого

public void menuKeyTyped (MenuKeyEvent e)
public void menuKeyPressed (MenuKeyEvent e)
public void menuKeyReleased (MenuKeyEvent e)

дозволяють організувати обробку подій, пов'язаних з натисканням і відпусканням, а також окремо з натисканням і окремо з відпусканням клавіш, пов'язаних з пунктами меню.

2.2.11 Бігунки і підказки

Класи JScrollBar і JSlider

Полоса прокрутки (клас **JScrollBar**) в Swing практично не відрізняється від полоси прокрутки (клас Scrollbar) в AWT.

Бігунки реалізуються в Swing за допомогою класу **JSlider**. Цей компонент, як і **JScrollBar**, дозволяє вибрати значення за допомогою переміщення бігунка. Основними конструкторами класу **JSlider** є

JSlider()
JSlider(int orientation)
JSlider(int min, int max)
JSlider(int min, int max, int value)
JSlider(int orientation, int min, int max, int value)

де параметр **orientation** задає орієнтацію, а параметри **min**, **max** і **value** відповідно мінімальне, максимальне і поточне значення бігунка.

Для бігунків можна задати лінійки з насічками великої або маленької висоти з використанням методів

public void setMajorTickSpacing (int n)

і

public void setMinorTickSpacing (int n),

де параметр **n** задає відстань між насічками в пікселях.

Для відстеження подій бігунка використовується інтерфейс **ChangeListener**, описаний вище для індикатора.

Клас JToolTip

«Спливаючі» підказки (текст, що з'являється, якщо навести на деякий час покажчик миші на який-небудь об'єкт) реалізуються в Swing за допомогою класу **JToolTip**. Об'єкт цього класу створюється за допомогою конструктора

JToolTip().

Прикріпити підказки до компоненту і отримати компонент, до якого прикріплена дана підказка, можна за допомогою методів

void setComponent (JComponent comp)

і

JComponent getComponent (),

а установка та отримання тексту підказки виконується за допомогою методів

void setTipText (String tipText)

і

String getTipText ().

Методи класу JComponent

public void setToolTipText (String text)

і

public String getToolTipText ()

дозволяють встановити і отримати підказку безпосередньо для будь-якого графічного компонента.

2.2.12 Оформлення рамок

Кожен стандартний графічний компонент в Java міститься в прямокутної області на екрані зі сторонами, паралельними сторонам екрана. Для деяких графічних компонент сторони цього прямокутника виділяються якимось чином. Так, наприклад, сторони кнопки класу JButton намальовані так, що створюють враження її опуклості. При натисканні кнопки миші на ній графічне оформлення сторін кнопки JButton змінюється, створюючи враження її «вдавленности».

Бібліотека Swing дозволяє змінити оформлення кордонів будь-якого компонента, в тому числі і контейнера за допомогою рамок різного виду.

Самі загальні характеристики всіх рамок описані інтерфейсом **Border** з пакету **javax.swing.border**. Викреслювання рамки виконується методом

public void paintBorder (Component c, Graphics g, int x, int y, int width, int height);

Тут задається компонент **c**, що обводиться рамкою, екземпляр класу Graphics **g**, що володіє методами малювання, і розміри рамки, які зазвичай збігаються з розмірами компонента, або трохи більше або трохи менше їх.

Рамка може бути прозорою або непрозорою. Це відзначається логічним методом

public boolean isBorderOpaque().

Третій і останній метод інтерфейсу

public Insets getBorderInsets (Component c)

повертає простір, зайнятий рамкою даного компонента **c**, у вигляді екземляра класу **Insets**. У класі **Insets** цей простір визначається товщиною рамки зверху **top**, ліворуч **left**, праворуч **right** і знизу **bottom**. Всі чотири поля класу **Insets** є цілочисельними змінними, які можна використовувати для визначення розмірів самого компонента без рамки.

Клас **AbstractBorder** розширюють близько двадцяти класів, викреслюється найрізноманітніші рамки. Для зручності роботи з ними в пакеті javax.swing є клас **BorderFactory**, в якому зібрані статичні методи виду **createXxxBorder()** для різних типів рамок з різними параметрами. Найчастіше для створення рамки досить скористатися одним з цих методів, а потім встановити отриману рамку в компонент методом

public void setBorder(Border border)

класа **JComponent**, наприклад:

```
JLabel myLabel = new JLabel("Надпись с толстой синей рамкой");
myLabel.setBorder(BorderFactory.createLineBorder(Color.blue, 3));
```

Нижче наведені основні статичні методи класу **BorderFactory** для створення рамок різних типів:

метод **public static Border createEmptyBorder()** створює порожню рамку з нульовими розмірами, а статичний метод **public static Border createEmptyBorder (int top, int left, int bottom, int right)** – порожню рамку із заданими розмірами;

методи **public static Border createLineBorder(Color color)** і **public static Border createLineBorder(Color color, int thickness)** створюють відповідно рамку із заданим кольором і рамку із заданим кольором заданої товщини;

метод **public static Border createBevelBorder (int type)** створює рамку заданого типу **type** (опуклого – **BevelBorder.RAISED** або увігнутого – **BevelBorder.LOWERED**), метод **public static Border createBevelBorder(int type, Color highlight, Color shadow)** створює рамку заданого типу із заданим світлим (**highlight**) і темним (**shadow**) кольором, а метод **public static Border createBevelBorder (int type, Color highlightOuter, Color highlightInner,**

Color shadowOuter, Color shadowInner) створює об'ємну двокольорову рамку заданого типу (внутрішні лінії мають кольору **highlightInner** і **shadowInner**, а зовнішні – кольору **highlightOuter** і **shadowOuter**);

метод **public static Border createEtchedBorder()** створює стандартну «врізану» рамку з квітами трохи світліше і трохи темніше кольору фону, метод **public static Border createEtchedBorder (int type)** створює «врізану» рамку (якщо **type** дорівнює **EtchedBorder.RAISED**) або «вдавлену» (якщо **type** дорівнює **EtchedBorder.LOWERED**), метод **public static Border createEtchedBorder (Color highlight, Color shadow)** задає кольори «врізаної рамки», а метод **public static Border createEtchedBorder (int type, Color highlight, Color shadow)** об'єднує можливості двох попередніх методів;

рамка одного кольору, але з ліній різної товщини задається за допомогою методу **public static MatteBorder createMatteBorder (int top, int left, int bottom, int right, Color color)**, де **color** – колір рамки, **top**, **left**, **bottom** і **right** – товщина рамки (у пікселях) відповідно зверху, ліворуч, знизу і справа, а метод **public static MatteBorder createMatteBorder (int top, int left, int bottom, int right, Icon tileIcon)** задає в якості рамки повторюване зображення **titleIcon**;

метод **public static TitledBorder createTitledBorder (Border border, String title, int titleJustification, int titlePosition, Font titleFont, Color titleColor)** задає для рамки **border** напис **title** шрифтом **titleFont** і кольору **titleColor**, вирівняну по горизонталі у відповідності зі значенням **titleJustification** (одна з статичних констант класу **TitledBorder**: LEFT, CENTER, RIGHT, LEADING, TRAILING або DEFAULT_JUSTIFICATION – значення LEADING) і по вертикалі відповідно зі значеннями **titlePosition** (одна з статичних констант класу **TitledBorder**: ABOVE_TOP, TOP, BELOW_TOP, ABOVE_BOTTOM, BOTTOM, BELOW_BOTTOM або DEFAULT_POSITION – значення TOP). Інші п'ять методів **createTitledBorder()** містять окремі параметри або частину параметрів наведеного вище методу;

метод **public static CompoundBorder createCompoundBorder (Border outsideBorder, Border insideBorder)** використовується для створення рамки, що складається з двох вкладених рамок будь-яких типів.

Хоча бібліотека Swing надає безліч готових рамок і клас BorderFactory для їх швидкого створення, часто виникає необхідність сконструювати оригінальну рамку.

Для її створення можна розширити абстрактний клас AbstractBorder, визначивши хоча б один конструктор, і перевизначивши методи **paintBorder()** і **getBorderInsets()**. Якщо рамка непрозора, то треба перевизначити метод **isBorderOpaque()** так, щоб він повертав **true**.

2.3 Контрольні питання

1. Які типи компонент визначені в системі Swing?
2. Які компоненти містить коренева панель в додатках і аплетах Swing?
3. Як задається панель вмісту в Swing? Який менеджер компоновки за умовчанням використовується в додатках і аплетах Swing?
4. Які операції можна визначити при закритті вікна JFrame?
5. Як створюється і використовуються діалогові вікна за допомогою класу JDialog?
6. Які типи діалогових вікон можна задати за допомогою класу JOptionPane?
7. Як реалізується панель з вкладками в Swing?
8. Як реалізується розщеплена панель в Swing?
9. Як можна задати зображення в компонентах Swing?
10. Які методи управління поведінкою кнопок, прапорців і перемикачів визначені в класі AbstractButton?
11. Які зміни в реалізації прапорців і перемикачів внесені до Swing в порівнянні з AWT?
12. Які відмінності в реалізації класу JComboBox в Swing в порівнянні з класом Choice в AWT?
13. Як виконується прокрутка списку в класі JList?
14. Які відмінності в реалізації текстових компонент в Swing в порівнянні з реалізацією відповідних компонент в AWT?
15. Як виконується обробка подій меню в Swing?
16. Як реалізовані бігунки в Swing?
17. Як задаються «спливаючі» підказки для компонент Swing?
18. Якими способами можна задати рамку для компонента Swing?
19. Які види рамок можна задати для компонент Swing?
20. Як можна задати власний вигляд рамки для компонента Swing?

3 Опис приладів, устаткування та інструментів

– Обладнання: IBM-сумісний персональний комп’ютер (ПК).

– Програмне забезпечення: операційна система Windows, Java 2 SDK версії 1.5 та вище. Інтегроване середовище розробки IDE Eclipse.

4 Правила техніки безпеки та охорони праці

Правила техніки безпеки при виконанні лабораторної роботи регламентуються «Правилами техніки безпеки при роботі в комп'ютерній лабораторії».

5 Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

– контроль рівня підготовленості студентів до виконання роботи у формі усного опитування;

– інструктаж по правилах охорони праці перед початком лабораторної роботи та оформлення його підсумків в журналі проведення лабораторних робіт, який ведеться у навчальній лабораторії;

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java у інтегрованому середовищі розробки Eclipse.

5.1 Завдання до лабораторної роботи

Створити графічний додаток згідно одного з наведених нижче варіантів. Для кожного використовуваного компонента має бути задана «спливаюча» підказка.

5.2 Варіанти

1. Малювання ліній в графічному вікні. Лінії починаються при натисненні кнопки миші і продовжуються при перетяганні миші до тих пір, поки кнопка миші не буде відпущена. Колір лінії вибирається за допомогою радіокнопок.

2. Малювання прямокутників в графічному вікні. При натисненні кнопки миші фіксується лівий верхній кут прямокутника. При відпуску миші фіксується правий нижній кут прямокутника, і прямокутник промальовувався на екрані. Колір зафарбовування прямокутника задається за допомогою списку, що розкривається.

3. Переміщення зображення в графічному вікні. Спочатку зображення знаходиться в центрі вікна. При натисненні однієї з чотирьох кнопок із стрілками вгору, вниз, вліво або вправо зображення стрибкоподібно переміщається в цьому напрямі.

4. Малювання багатокутників в графічному вікні. При клацанні мишею фіксується чергова точка багатокутника. При подвійному клацанні мишею ця точка з'єднується з першою точкою багатокутника і малювання

багатокутника закінчується. Колір зафарбовування прямокутника задається за допомогою списку, що розкривається.

5. Виведення послідовності символів в заданій точці графічного вікна. Координати введення x і в (у пікселях) задаються за допомогою текстових рядків, а послідовність символів – за допомогою текстового поля.

6. Вивід в двох текстових рядках в графічному вікні поточних координат миші (x,y) при її переміщенні.

7. Зміна фігури в графічному вікні. При натисненні кнопок "Трикутник", "Квадрат" або "Еліпс" у вікні додатка повинні з'являтися відповідно закрашений трикутник, квадрат або еліпс. При натисненні кнопки "Видалити" фігура повинна віддалятися з вікна.

8. Зміна кольору закрашеного прямокутника із закругленими кутами в графічному вікні. При виборі пунктів "Червоний", "Зелений" або "Синій" в списку, що розкривається, фігура повинна міняти колір зафарбовування відповідно на червоний, зелений або синій колір. При виборі пункту "Фон" фігура має бути кольору фону.

9. Перетягання фігури закрашеного прямокутника в графічному вікні. Колір закрашеного прямокутника задається за допомогою радіокнопок. При натисненні кнопки миші на фігурі і переміщенні миші фігура переміщається услід за мишею. При відпусканні кнопки миші позиція фігури фіксується. Координати x і y лівого верхнього кута прямокутника виводяться в двох текстових рядках.

10. Переміщення фігури закрашеного трикутника в графічному вікні. Спочатку фігура знаходиться в лівому верхньому кутку. Колір закрашеного прямокутника задається за допомогою розкриваючого списку. При клацанні мишею в якій-небудь точці вікна додатка фігура поміщається в цю крапку. Координати x і y лівого верхнього кута прямокутника виводяться в двох текстових рядках.

11. Зміна кольору закрашеного трикутника і кольору фону в графічному вікні. Зміна кольору задається циклічно за допомогою кнопок "Фігура" і "Фон". Кнопки укладені в рамку з написом зверху "Зміна кольору".

12. При клацанні мишею по зафарбованому прямокутнику з закругленими кутами в текстовому рядку графічного вікна з'являється напис "Включено", а при повторному клацанні – напис "Виключено". Напис має бути укладений в рамку з написом "Зміна стану".

6 Порядок обробки результатів експериментів та вироблення висновків

На підставі отриманих результатів лабораторної роботи зробить висновки щодо особливостей створення графічного інтерфейсу користувача бібліотеками AWT і Swing в Java. У висновку до лабораторної роботи

наведіть відмінності використання цих бібліотек, їх недоліки і переваги стосовно крос-платформної реалізації GUI.

7 Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

- титульний лист, де вказані номер і назва лабораторної роботи, відомості про виконавця;
- номер варіанта роботи та текст завдання;
- відповіді на контрольні запитання до лабораторної роботи;
- листинг програмного коду;
- результати виконання програми;
- висновки по підсумках роботи.

Підготовлений звіт надається викладачу на перевірку та захищається студентом на останньому занятті з даної лабораторної роботи.

Лабораторна робота №4

Створення графічних застосувань.
Програмування потоків вводу/виводу

1 Мета роботи

Метою роботи є придбання навичок використання потоків вводу-виводу і програмування графічних додатків Java з використанням системи Swing.

Після вконання лабораторної роботи студенти мають оволодіти вміннями роботи з файловою системою та потоками вводу/виводу в Java.

2 Завдання на підготовку до лабораторної роботи

2.1 Перелік знань та вмінь, на які необхідно спиратися при виконанні роботи

Знання:

- класи пакету java.io;
- потоки вводу/виводу в Java;
- консольний ввод даних;
- менеджери компонування;
- компоненти бібліотек AWT і Swing.

Вміння:

- проектувати графічний інтерфейс користувача інформаційних систем;
- організовувати взаємодію з файловою системою комп'ютера;
- організовувати потоки вводу/виводу в Java.

2.2 Теоретичні відомості

2.2.1 Робота з файлами в Java

Інформація на пристроях зовнішньої пам'яті зберігається у **файлах** – іменованих областях на диску, дискеті або іншому машинному носії. Структура (як правило, ієрархічна) розміщення файлів на носіях інформації, а також види та правила завдання атрибутів файлу (імені, типу, дати створення та/або модифікації та інш.) називається **файловою системою.** Файлові системи в різних операційних системах, як правило, відрізняються.

Клас File

Клас **File** дозволяє отримати від системи всі дані, починаючи з імені файлу і закінчуючи часом останньої його модифікації. Клас **File** можна використовувати для створення нових каталогів, а також для видалення і

перейменування файлів. Для створення об'єкта **File** потрібно викликати один з трьох конструкторів класу:

File(String path)
File(String path, String name)
File(File dir, String name)

Перший конструктор створює об'єкт **File** із зазначеним повним ім'ям файлу. Другий конструктор створює цей об'єкт, використовуючи окремо шлях і ім'я файлу, а третій створює об'єкт, використовуючи шлях і ім'я файлу, при цьому шлях визначається іншим об'єктом **File.**

Методи для роботи з файлами і каталогами класу **File** наведені в таблиці 2.1.

Таблиця 2.1 – Методи для роботи з файлами і каталогами класу File

Метод	Опис
public String getName()	Визначає ім'я файлу.
public String getPath()	Визначає відносний шлях до файлу (наприклад з даного каталогу).
public String getAbsolutePath()	Визначає повне ім'я файлу (шлях до каталогу з кореневого каталогу). Цей шлях називається також абсолютним шляхом до файлу.
public String getParent()	Визначає батьківський каталог файлу.
public boolean exists()	Повертає значення true, якщо файл існує.
public boolean canWrite()	Повертає значення true, якщо в файл можна записувати.
public boolean canRead()	Повертає значення true, якщо файл можна читати.
public boolean isFile()	Повертає значення true, якщо об’єкт є файлом.
public boolean isDirectory()	Повертає значення true, якщо об’єкт є каталогом.
public boolean isAbsolute()	Повертає значення true, якщо ім'я файлу повне.
public long lastModified()	Повертає час останньої модифікації файлу.
public long length()	Повертає довжину файлу.
public boolean mkdir()	Створює каталог.
public boolean renameTo (File newName)	Перейменовує файл.
public boolean delete()	Видаляє файл.
public boolean mkdirs()	Створює дерево каталогів.

Метод	Опис
public String[] list()	Створює строковий масив імен файлів і підкаталогів в каталозі.
public String[] list (FilenameFilter filter)	Створює строковий масив імен всіх файлів, що задовольняють фільтру імен по інтерфейсу FilenameFilter.
public File[] listFiles()	Створює масив об'єктів класу File в каталозі.
public File[] listFiles (FilenameFilter filter)	Створює масив об'єктів класу File, що задовольняють фільтру імен по інтерфейсу FilenameFilter.
public File[] listFiles (FileFilter filter)	Створює строковий масив об'єктів класу File, що задовольняють фільтру імен по інтерфейсу FileFilter.

Часто потрібно обмежити кількість файлів, що повертаються методами **list()** або **listFiles()**, щоб включати в список тільки ті з них, імена яких відповідають деякому зразку або фільтру. Для цього можна використовувати перевантажену форму методів **list()** і **listFiles()**:

public String[] list(FilenameFilter filter)

public File[] listFiles(FilenameFilter filter)

У цих формах як параметр задається об'єкт класу, який реалізує інтерфейс типу **FilenameFilter**.

Інтерфейс **FilenameFilter** визначає єдиний метод **accept()**, який має наступний вигляд:

public abstract boolean accept(File dir, String name)

Цей метод повертає true для файлів каталогу **dir**, які повинні бути включені в відфільтрований список (з іменами name), і повертає false для тих файлів, які повинні бути виключені зі списку (імена яких не збігаються з **name**).

При використанні методу

public File[] listFiles(FileFilter filter)

файли фільтруються не просто за іменами, а за їх повними іменам (зі шляхами в ієрархії каталогів). При цьому повертаються файли з тими іменами шляхи, які задовольняють файловому фільтру **filter**. Інтерфейс **FileFilter** визначає тільки один метод, **accept()**, який викликається одного разу для кожного файлу у списку. Його загальна форма:

boolean accept(File path)

Метод повертає true для файлів, які повинні бути включені в список (тобто тих файлів, які вказані в **path**), і false – для тих файлів, які повинні бути виключені (не вказані в **path**).

Змінна **public final static char separatorChar** класа **File** повертає символ поділення імен каталогів і файлів ("\" в Windows і "/" – в Unix), а

змінна **public final static char pathSeparatorChar** містить символ-роздільник шляхів у списку ("**;**" в Windows і "**:**" – в Unix).

Вибір файлів в Swing

Клас **JFileChooser** в Swing забезпечує діалогове вікно для вибору каталогів і файлів. Основні конструктори цього класу:

JFileChooser()
JFileChooser(File currentDirectory)
JFileChooser(String currentDirectoryPath)

дозволяють відкрити вікно вибору файлів як для всіх файлів, так і для конкретного каталогу.

Методи класу **JFileChooser** наведені у таблиці:

Метод	Опис
public File getSelectedFile()	Отримання вибраного файлу.
public File[] getSelectedFiles()	Отримання вибраних файлів.
public void setSelectedFile(File file)	Установка вибраного файлу.
public void setSelectedFiles(File[] selectedFiles))	Установка вибраних файлів.
public File getCurrentDirectory()	Отримати вибраний каталог.
public void setCurrentDirectory(File dir)	Встановити вибраний каталог.
public String getDescription(File f)	Отримання характеристик файлу.
public void setFileFilter(FileFilter filter)	Установка фільтра для імен виведених файлів.
public FileFilter getFileFilter()	Отримання фільтра для імен виведених файлів.

Для відстеження подій, пов'язаних з кнопками в діалоговому вікні вибору файлів, необхідно реалізувати інтерфейс і додати або видалити блок прослуховування для кнопок діалогового вікна вибору файлів за допомогою методів

public void addActionListener(ActionListener l)

і

public void removeActionListener(ActionListener l)

класу **JFileChooser**.

2.2.2 Організація вводу-виводу в Java

Всі дані в комп'ютерній системі проходять від пристроїв введення через комп'ютер до пристроїв виводу. Аналогія з перетікаючими даними викликала термін «потоки» (streams). Два терміна в Java, що означають різні поняття: thread і stream, переводяться одним і тим же словом – потік. Щоб уникнути плутанини, будемо використовувати для першого терміна слова «потік обчислень», а для другого – просто «потік».

Потоком називається канал обміну інформацією між її джерелом і одержувачем. На одному кінці потоку завжди знаходиться програма на Java. Якщо вона буде служити джерелом даних, то даний потік буде вихідним, якщо програма буде перебувати на приймаючій стороні – то вхідним.

Починаючи з версії JDK 1.1, в Java визначені два типи потоку: байтовий і символьний потоки.

Байтовий потік або оперує з байтами і використовується при читанні (*input stream*) або запису (*output stream*) даних в двійковому коді.

Символьний потік (*reader* або *writer*) використовується для вводу і виводу символів. У більш старих мовах, наприклад C або C ++, поняття символ і байт є еквівалентними, оскільки символи мали однобайтових кодування. Однак в Java символ не є еквівалентом байта, тому що являє собою 16-бітовий елемент, призначений для розміщення кодів Unicode.

У мові Java потоки представляються класами. Найпростіші з цих класів працюють з базовими потоками вводу і виводу, що мають основні засоби роботи з потоками. Від базових класів породжуються інші класи, більшою мірою орієнтовані на конкретний тип вводу або виводу.

Всі класи вводу/виводу Java описані в пакеті **java.io**.

Слід зауважити, що практично кожен метод будь-якого класу в пакеті java.io здатний генерувати ту чи іншу форму виняткової ситуації **IOException**. Тому для поліпшення стилю програмування слід завжди поміщати виклики операцій вводу/виводу до блоку **try** і **catch**.

2.2.3 Байтові потоки вводу/виводу

Класи InputStream і OutputStream

Абстрактний клас **InputStream** являє собою базовий потік вводу. Клас містить єдиний конструктор **InputStream()**.

Методи класу **InputStream** наведені у таблиці:

Метод	Опис
public int read() throws IOException	Зчитує з вхідного потоку окремі байти як цілі числа, повертаючи значення -1, коли більше нічого читати.

public int read(byte b[]) throws IOException	Зчитує безліч байтів в байтовий масив, повертаючи кількість реально введених байтів.
public int read(byte b[], int off, int len) throws IOException	Також читає дані в байтовий масив, проте дозволяє вказати зсув (off) в масиві, з якого почнеться запис символів, а також задати максимальне число зчитувальних байтів (len).
public long skip(long n) throws IOException	Пропускає байти в потоці.
public int available() throws IOException	Повертає кількість байтів, що є в даний момент в потоці.
public void mark(int readlimit)	Позначає позицію readlimit в потоці.
void reset() throws IOException	Повертається до зазначеної позиції потоку.
public boolean markSupported()	Повертає булевське значення, яке вказує на те, чи можна в даному потоці відзначати позиції і повертатися до них.
public void close() throws IOException	Закриває потік.

Абстрактний клас **OutputStream**, забезпечує базові функції для всіх вихідних потоків і має єдиний конструктор **OutputStream()**.

Методи класу **OutputStream** наведені в таблиці.

Метод	Опис
public void write(int b) throws IOException	Записує байт в потік.
public void write(byte b[]) throws IOException	Записує в потік всі байти, що містяться в заданому байтовому масиві.
public void write(byte b[], int off, int len) throws IOException	Записує дані з байтового масиву, вказуючи початковий зсув (off) і кількість виведених байтів (len).
public void flush() throws IOException	Виконує примусовий запис всіх буферізованих вихідних даних.

Класи FileInputStream і FileOutputStream

Клас **FileInputStream** створює (відкриває) об'єкт, який можна використовувати для читання байтів з файлу за допомогою одного з наступних конструкторів:

FileInputStream (String filepath) throws FileNotFoundException

FileInputSream(File file) throws FileNotFoundException

Клас **FileInputStream** реалізує методи **available()**, **close()**, **skip()** і всі форми методу **read()**.Крім того, в цьому класі доданий метод

public final FileDescriptor getFD() throws IOException,

який повертає об'єкт **FileDescriptor** для відкритого файлу і метод

protected void finalize() throws IOException

для очищення зв'язку з файлом і виклику методу **close()**.

Клас **FileOutputStream** створює об'єкт **OutputStream**, який можна застосовувати для запису байтів в файл. Зазвичай використовуються такі конструктори цього класу:

FileOutputStream(String filePath)

FileOutputStream(File fileObj)

FileOutputStream(String filePath, boolean append)

Параметри в цих конструкторах мають той же зміст, що і в конструкторах класу **FileInputStream**. Якщо параметр **append** в останньому конструкторі дорівнює true, файл (якщо він вже існує) відкривається в режимі додавання, інакше файл записується заново.

Клас **FileOutputStream** реалізує метод **close()** і всі форми методу **write()** класу **OutputStream**, а також використовує свої власні методи **getFD()** і **finalize()**, аналогічні однойменним методам класу **FileInputStream**.

Класи ByteArrayInputStream і ByteArrayOutputStream

Клас **ByteArrayInputStream** є реалізацією вхідного потоку, яка використовує байтовий масив як джерело. Цей клас має два конструктора, кожен з яких використовує байтовий масив в якості джерела даних:

ByteArrayInputStream(byte array[])

ByteArrayInputStream (byte array [], int off, int len)

Тут байтовий масив **array** – це джерело вводу. Другий конструктор створює об'єкт, що складається з байтового масиву, який починається з позиції **off** і має довжину **len** байтів.

Клас **ByteArrayInputStream** реалізує методи **read()** класу **InputStream** для читання оного байта і частини масиву. Реалізація методу **reset()** в цьому класі має таку особливість: якщо метод **mark()** не викликало, то **reset()** встановлює потоковий покажчик на початок потоку, який в цьому випадку є початком масиву байт, переданого конструктору. Цю особливість можна використовувати, наприклад, для читання одного і того ж потоку введення двічі.

При виконанні операції виводу масив байтів можна переслати в локальну пам'ять комп'ютера. Конкретним класом вивідного потоку **OutputStream**, здатним вирішити це завдання, є **ByteArrayOutputStream**.

Клас **ByteArrayOutputStream** має наступні два конструктора:

ByteArrayOutputStream()

ByteArrayOutputStream(int len)

Першому з них не передається ніяких параметрів, тому він створює буфер розміром 32 байт. При необхідності розмір буфера може бути збільшений. Однак якщо збільшення буде виконуватися на кілька байтів за кожне звернення, то в деяких системах це може викликати сильну фрагментацію пам'яті. Якщо заздалегідь відомо, що буде потрібно буфер розміром, наприклад, в 1024 байта, слід використовувати другу версію конструктора, якому передається один параметр.

До методів, успадкованих від базового класу **OutputStream**, клас **ByteArrayOutputStream** ще підтримує такі методи:

Метод	Опис
public int size()	Повертає поточний розмір буфера.
public void reset()	Скидає значення лічильника вивідного потоку в 0, так, що вже накопичене вміст буфера вивідного потоку втрачається.
public byte[] toByteArray()	Перетворює дані вивідного потоку в новий масив байтів.
public String toString()	Перетворює вміст буфера в рядок відповідно до правил кодування символів за замовчуванням.
public void writeTo(OutputStream out) throws IOException	Записує весь вміст потоку в інший потік.

Клас **ByteArrayOutputStream** можна використовувати як один з елементів для побудови більш складних об'єктів, включаючи процедури взаємодії між процесами, або для заміни інших потоків (наприклад, потоку мережевих даних) в процесі тестування.

Клас SequenceInputStream

Клас **SequenceInputStream** дозволяє зчіплювати безліч об'єктів вхідного потоку. Основна форма конструктора цього класу використовує в якості параметрів два об'єкти класу **InputStream** або його підкласів:

SequenceInputStream(InputStream firstStream,

InputStream secondStream)

Клас виконує запити читання першого об'єкта типу **InputStream** (параметр **firstStream**), поки він не закінчиться, і потім перемикається на

другий (параметр **secondStream**). У свою чергу, використовуючи як одного з об'єктів об'єкт класу **SequenceInputStream**, можна організувати введення більш ніж двох об'єктів вхідного потоку.

Клас **SequenceInputStream** реалізує методи **available()** і **close()**, а також методи **read()** для читання байта і частини масиву класу **InputStream**.

Класи BufferedInputStream і BufferedOutputStream

Байтовий **буферизований потік** розширює класи фільтрованого потоку, приєднуючи буфер пам'яті до потоків введення/виводу. Такий буфер дозволяє виконувати операції вводу-виводу (використовуючи внутрішній буфер) не з одним, а з декількома байтами одночасно, і, отже, збільшує ефективність роботи програми. При наявності буфера стає можливим такі операції, як пропуск байтів, маркування та перевстановлення потоку. Буферизовані потокові класи – це класи **BufferedInputStream** і **BufferedOutputStream**.

Клас **BufferedInputStream** містить два конструктора:

BufferedInputStream (InputStream in)

BufferedInputStream(InputStream in, int size)

Перша форма створює буферизований потік, використовуючи розмір буфера, заданий за замовчуванням. У другій формі розмір буфера передається в параметрі **size**. Змінна

protected byte[] buf

класу **BufferedInputStream** містить внутрішній буфер вхідного потоку, а змінна

protected int count

містить лічильник кількості введених байт (ця величина змінюється від нуля до величини масиву **size**).

Клас **BufferedInputStream** підтримує всі методи класу **InputStream**, за винятком методу **read()** для читання масиву байт.

Клас **BufferedOutputStream** є ефективним варіантом класу **OutputStream**, виконуючим запис байтів у внутрішній буфер, який потім може бути виведений у потік нижнього рівня. Конструктори цього классу

BufferedOutputStream(OutputStream out)

BufferedOutputStream(OutputStream out, int size)

створюють об'єкт **BufferedOutputStream** з буфером за замовчуванням (512 байт) або з буфером заданого розміру.

Властивість цього классу

protected byte[] buf

повертає вміст внутрішнього буфера, а властивість

protected int count

– кількість виведених у буфер байт. Клас **BufferedOutputStream** реалізує методи **write()** запису одного байта і частини масиву байт, а також методи **close()** і **flush()** класу **OutputStream**.

Клас PushbackInputStream

Одне із застосувань буферизації – виконання повернення зчитаного байта назад у вхідний потік (операція pushback). Цю можливість в Java здійснює клас **PushbackInputStream**. Цей клас має наступні конструктори:

PushbackInputStream (InputStream inputStream)

PushbackInputStream(InputStream inputStream, int size)

Перша форма створює потоковий об'єкт, який дозволяє повернути один байт у вхідний потік. Друга форма створює потік, який має спеціальний pushback-буфер довжиною **size** байт. Він дає можливість виконувати повернення декількох байтів у вхідний потік.

Крім методів **available()**, **close()**, **markSupported()** і **read()** для читання частині масиву байт класу **InputStream**, в **PushbackInputStream** визначений метод **unread()** в наступних формах:

void unread(int b)

void unread(byte b[])

void unread(byte b [], int off, int len)

Перша форма повертає молодший байт параметра **b**. Повернений байт буде прочитаний наступним за **unread()** викликом **read()**. Друга форма повертає групу байтів – весь буферний масив **b[]**. Третя форма забезпечує отримання частини масиву **b[]** (**len** байт, починаючи з індексу **off**). Якщо здійснюється спроба повернути байт при заповненому буфері повернення, буде кинуто виключення класу **IOException.**

Клас PrintStream

Клас **PrintStream** дозволяє вивести в потік нижнього рівня подання даних примітивного, строкового або об'єктного типу, яке вони матимуть при друку.

Два конструктора класу **PrintStream**:

PrintStream(OutputStream out)

PrintStream(OutputStream out, boolean autoFlush)

створюють новий потік для друку. Якщо другий параметр в другому конструкторі дорівнює true, то вивідний буфер буде очищатися в одній з наступних ситуацій: записаний масив байт, викликаний один з методів **println()** або в вивідний потік записується символ "\ n"**.**

Крім реалізації методів **write()** запису масиву байт і частини масиву байт, а також методів **flush()** і **close()** класу **OutputStream**, в класі **PrintStream** визначені також наступні методи:

– **protected void setError()** – установка стану помилки для потоку в **true**;

public boolean checkError() – очистка буфера потоку з перевіркою його стану помилки.

Крім цього, в класі визначені два вже відомих методів:

public void print(*аргумент***)**

і **public void println(***аргумент***).**

В якості аргументів при виклику цих методів можуть бути задані: **boolean b**, **char c**, **char[] s**, **float f**, **double d**, **int i**, **long l**, **String s** і **Object obj**.

2.2.4 Символьні потоки ввода-виводу

Класи Reader і Writer

Функції класів **InputStream** і **OutputStream** для символьних потоків виконують абстрактні класи **Reader** і **Writer**.

Клас **Reader** визначає модель символьного вхідного потоку. Конструктори класу

protected Reader()

protected Reader(Object lock)

визначають новий вхідний символьний потік. Критичні секції потоку в першому конструкторі синхронізуються з самим вхідним потоком, а в другому – з вказаним об'єктом **lock**.

Короткий опис методів класу **Reader** наведено в таблиці.

Метод	Опис
public int read() throws IOException	Зчитує з вхідного потоку окремі символи як цілі числа, повертаючи значення -1, коли більше нічого читати.
public int read(char c[]) throws IOException	Зчитує безліч символів в символьний масив, повертаючи кількість реально введених символів.
public int read(char c[], int off, int len) throws IOException	Також читає дані в символьний масив, проте дозволяє вказати зсув (off) в масиві, з якого почнеться запис символів, а також задати максимальне число зчитувальних символів (len).
public long skip(long n) throws IOException	Пропускає символи в потоці.
public boolean ready() throws IOException	Повертає true, якщо потік готовий до читання, і false - в іншому випадку.
public void mark(int readlimit)	Позначає позицію readlimit в потоці.
public boolean markSupported()	Повертає булевское значення, яке вказує на те, чи можна в даному потоці відзначати позиції і повертатися до них.

void reset() throws IOException	Повертається до зазначеної позиції потоку.
public void close() throws IOException	Закриває потік.

Клас **Writer** визначає модель поточного символьного виводу. Конструктори класу

protected Writer()

protected Writer (Object lock)

визначають новий вивідний символьний потік аналогічно конструкторам класу **Reader**.

Методи класу **Writer** представлені в наступній таблиці:

Метод	Опис
public void write(int c) throws IOException	Записує символ у вихідний потік.
public void write(char c[]) throws IOException	Записує у вихідний потік всі символи, що містяться в заданому символьному масиві.
public void write(char c[], int off, int len) throws IOException	Записує дані з символьного масиву, вказуючи початковий зсув (off) і кількість виведених символів (len).
public void write(String str) throws IOException	Записує рядок у вихідний потік.
public void write(String str, int off, int len) throws IOException	Записує дані з рядка, вказуючи початковий зсув (off) і кількість виведених символів (len) рядка.
public void flush() throws IOException	Виконує примусовий запис всіх буферізованих вихідних даних.
public void close() throws IOException	Закриває вихідний потік.

Класи InputStreamReader і OutputStreamWriter

Класи **InputStreamReader** і **OutputStreamWriter** є перехідниками між байтовими і символьними потоками.

Клас **InputStreamReader** перетворює байтовий потік в символьний потік. Основний конструктор цього класу:

InputStreamReader (InputStream in)

Цей конструктор створює з байтового потоку **in** символьний потік відповідно до кодуванням за замовчуванням (кодування за замовчуванням

залежить від реалізації віртуальної машини Java і встановлених для неї локальних значень).

Для класу **InputStreamReader** визначені наступні методи: **read()** для читання одного символу або частини масиву символів, методи **ready()** і **close()** класу **Reader**.

Клас **OutputStreamWriter** перетворює символьний потік в байтовий потік. Основний конструктор цього класу:

OutputStreamWriter (OutputStream out)

Цей конструктор створює з символьного потоку **out** байтовий потік відповідно до кодуванням за замовчуванням.

Для виведення символів в класі **OutputStreamWriter** використовуються перші три форми методу **write()**, метод **flush()** і метод **close()** класу **Writer**.

Класи FileReader і FileWriter

Клас **FileReader** створює об'єкт, який можна використовувати для читання вмісту файлу. Для створення об'єкта класу **FileReader** можна використовувати один з наступних конструкторів:

FileReader (String fileName) throws FileNotFoundException

FileReader (File file) throws FileNotFoundException

У першому конструкторі як параметр задається повне ім'я файлу, а в другому – об'єкт класу **File.**

Клас **FileReader** успадковує методи **read()** для читання символу і частини масиву символів, **close**() і **ready**() класу **InputStreamReader**, а також метод **read**() для читання масиву символів, **mark**(), **markSupported**(), **reset**() і **skip**() класу **Reader**.

Клас **FileWriter** створює об'єкт, який можна використовувати для запису у файл. Конструктори цього класу:

FileWriter (String fileName) throws IOException

FileWriter (File file) throws IOException

FileWriter (String fileName, boolean append) throws IOException

задають в параметрах або повне ім'я файлу (параметр **fileName**), або об'єкт класу **File** (параметр **file**). Другий параметр **append** останнього конструктора задає режим запису у файл: якщо true, то вивод додається в кінець файлу, якщо false – файл перезаписується.

При створенні об'єкта класу **FileWriter**, якщо файл існує, він відкривається, якщо файл не існує, він буде створений і відкритий.

Клас **FileWriter** успадковує перші три форми методу **write**(), метод **flush**() і метод **close**() класу **OutputStreamWriter**.

Класи CharArrayReader і CharArrayWriter

Клас **CharArrayReader** є реалізацією вхідного потоку, яка використовує символьний масив як джерело. Для цього класу визначені наступні конструктори:

CharArrayReader (char з [])

CharArrayReader (char з [], int off, int len)

Перший конструктор створює з масиву символів **c** вхідний потік, а другий конструктор читає **len** символів масиву **c**, починаючи з зміщення **off.**

Клас реалізує наступні методи класу **Reader**: **skip**(), **ready**(), **reset**(), **mark**(), **markSupported**(), **close**(), а також методи rea**d**() для читання одного символу і частини масиву символів.

Клас **CharArrayWriter** реалізує вихідний потік, що записує дані в символьний масив. Цей клас має два конструктора:

CharArrayWriter ()

CharArrayWriter (int initialSize)

У першій формі створюється буфер з розміром, заданим за замовчуванням. У другій формі – буфер з розміром, зазначеним у параметрі **initialSize**. Буфер міститься в поле **buf** класу **CharArrayWriter**. Якщо необхідно, розмір буфера буде збільшено автоматично. Число символів, що зберігаються в буфері, визначається полем **count** класу **CharArrayWriter**. І **buf** і **count** оголошені з модифікатором **protected**, тобто є захищеними полями.

Поряд з реалізацією методів **write**() для запису одного символу, частини масиву символів і частини рядки, **flush**() і **close**() класу **Writer**, в класі **CharArrayWriter** реалізовані наступні власні методи:

Метод	Опис
public char[] toCharArray()	Повертає копію даних, що вводяться.
public String toString()	Перетворює дані, що вводяться в рядок.
public int size()	Повертає поточний розмір буфера.
public void writeTo(Writer out) throws IOException	Записує вміст буфера в інший символьний потік.

Клас PrintWriter

Клас **PrintWriter** виводить форматоване представлення об'єктів в текстовий вивідний потік.

Для класу **PrintWriter** визначені наступні конструктори:

PrintWriter (OutputStream out)

PrintWriter (OutputStream out, boolean autoFlush)

PrintWriter (Writer out)

PrintWriter (Writer out, boolean autoFlush)

Перший і другий конструктори створюють новий об'єкт **PrintWriter** з існуючого вивідного байтового потоку **out**, третій і четвертий конструктори

створюють новий об'єкт **PrintWriter** з існуючого вивідного символьного потоку **out** (при цьому створюється проміжний об'єкт **OutputStreamWriter**, що перетворює символи в байти з використанням кодування символів за замовчуванням). Перший і третій конструктори виводять символи у вихідний потік без автоматичного звільнення буфера при переході на новий рядок. Завдання для **autoFlush** значення true в другому і четвертому конструкторах викликає очистку буферів при використанні методів **println**().

Методи цього класу не кидають виключення вводу-виводу. Замість них використовуються методи даного класу

protected void setError ()

і

public boolean checkError ()

відповідно встановлює стан помилки в true і перевіряє стан помилки.

Методи **print**() і **println**() класу **PrintWriter** мають ті ж аргументи, що і відповідні методи класу **PrintStream**, і діють аналогічно.

Крім того, для класу **PrintWriter** визначені всі методи класу **Writer**.

2.2.5 Консольний ввод

Вводити дані з клавіатури, не створюючи власних потоків можна за допомогою об'єкта **System.in**, що є екземпляром класу **InputStream**. Оскільки об'єкт є екземпляром класу, йому доступні всі методи цього класу.

Ознакою закінчення введення з клавіатури може служити введення заданого символу або послідовності символів, а також натискання клавіш **Ctrl + Z**.

Хоча можна обробляти вхідний потік з клавіатури як байтовий, в Java 2 рекомендується перетворити вхідний байтовий потік в символьний, використовуючи об'єкт класу **InputStreamReader**, а потім перетворити його в буферний ввод з використанням класу **BufferedReader**.

2.2.6 Клас StreamTokenizer

При читанні символьних потоків дуже часто потрібно розбивати дані на окремі слова. Слово – це послідовність символів, відокремлена від інших слів пробілом або деяким символом-роздільником. Для вирішення такого завдання в Java існує спеціальний клас **StreamTokenizer**, конструктору якого як параметр передається посилання на вхідних символьний потік.

Властивості цього класу визначають наступні типи слів:

TT_EOF – кінець файлу;

TT_EOL – кінець рядка;

TT_NUMBER – число;

TT_WORD – текстове поле.

Крім, того, властивість

int nval

містить значення поточного слова, якщо воно число, а властивість

String sval

містить значення поточного слова, якщо воно – рядок. Властивість

int ttype

містить тип тільки що прочитаного слова.

Клас **StreamTokenizer** включає безліч різних методів, основні з яких наведені в таблиці:

Метод	Опис
void commentChar (int char)	Визначає символ однострочного коментаря.
void eolIsSignificant(boolean flag)	Дозволяє вказати, чи слід вважати ознаку кінця рядка роздільником слів (якщо так, то flag - true, інакше - false).
int lineno()	Повертає номер поточного рядка.
void lowerCaseMode(boolean flag)	Дозволяє перевести екземпляр об'єкта в режим примусового переведення всіх виділених слів в нижній регістр.
int nextToken()	Читає наступне слово і повертає його тип. Крім того, тип слова встановлюється в ttype.
void ordinaryChar()	Дозволяє встановити, які символи слід вважати простими. Прості символи обробляються як Односимвольна лексема незалежно від того, чи є вона частиною роздільника рядків, використовується в коментарі або входить до складу звичайного слова.
void ordinaryChars(int low, int high)	Встановлює, що символи з кодами в діапазоні від low до high є простими.
void parseNumbers()	Дозволяє визначити, чи слід робити спроби розпізнавання чисел і перетворення їх у відповідну числову форму.
void pushBack()	Змушує клас при наступному виклику методу nextToken() виконати повернення останнього виділеного слова назад у вхідний потік.
void quoteChar(int char)	Визначає символ, що

Метод	Опис
	використовується для виділення текстових рядків.
void resetSyntax()	Після виклику цього методу даний екземпляр об'єкта буде використовувати синтаксичну таблицю, прийняту за замовчуванням, в якій всі символи є "простими".
void slashSlashComments(boolean flag)	Задає режим обробки однорядкових коментарів, що виконані в стилі C ++.
void slashStarComments(boolean flag)	Задає режим обробки багаторядкових коментарів, що виконані у стилі C.
String toString()	Повертає представлення поточної лексеми у вигляді об'єкта класу String.
void whitespaceChars (int low, int high)	Задає символи, які сприймаються як пробіли.
void wordChars (int low, int high)	Встановлює, які символи можуть входити до складу слів.

Клас **StreamTokenizer** можна використовувати як основу для створення підпрограм обробки текстів, які вводяться з вхідного потоку.

2.3 Контрольні питання

1. Для яких цілей використовується клас File? Які операції над файлами визначені в класі File?
2. Як в класі File реалізована фільтрація файлів?
3. Які засоби для навігації по файлам і каталогам забезпечує клас JFileChooser?
4. Що таке потік даних? Які види потоків визначені в Java?
5. Які методи читання і запису даних визначені в класах InputStream і OutputStream?
6. Як в Java реалізований байтовий обмін даними з файлами?
7. Для яких цілей використовуються класи ByteArrayInputStream і ByteArrayOutputStream?
8. Як в Java виконується об'єднання двох потоків вводу в один потік?
9. Які засоби буферизації потоків введення-виведення визначені в Java?
10. Як в Java виконується виведення байтових і символьних потоків на друк?
11. Які методи читання і запису даних визначені в класах Reader і Writer?

12.Як в Java реалізований перехід між байтовими і символьними потоками?

13.Як в Java реалізований символьний обмін даними з файлами?

14.Як в Java реалізований введення-виведення в символьні масиви?

15.Для яких цілей використовується клас StreamTokenizer?

16.Які методи аналізу даних визначено в класі StreamTokenizer?

3 Опис приладів, устаткування та інструментів

– Обладнання: IBM-сумісний персональний комп’ютер (ПК).

– Програмне забезпечення: операційна система Windows, Java 2 SDK версії 1.5 та вище. Інтегроване середовище розробки IDE Eclipse.

4 Правила техніки безпеки та охорони праці

Правила техніки безпеки при виконанні лабораторної роботи регламентуються «Правилами техніки безпеки при роботі в комп'ютерній лабораторії».

5 Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

– контроль рівня підготовленості студентів до виконання роботи у формі усного опитування;

– інструктаж по правилах охорони праці перед початком лабораторної роботи та оформлення його підсумків в журналі проведення лабораторних робіт, який ведеться у навчальній лабораторії;

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java у інтегрованому середовищі розробки Eclipse.

5.1 Завдання до лабораторної роботи

Створити графічний додаток Swing для виконання операцій вводу-виводу по одному з наведених нижче варіантів.

5.2 Варіанти

1. Створіть програму копіювання файлу з одного каталогу в інший. Компоненти графічного вікна: напис "Копіювання файлу" в області North, розщеплена панель в області Center і кнопка "Копіювати" в області South. У лівій частині розщепленої панелі розміщений об'єкт класу JFileChooser для

вибору файлу копіювання, а в правій частині – другий об'єкт класу JFileChooser, в якому вибирається каталог для копіювання. Копіювання файлу виконується при натисканні кнопки.

2. Створіть програму перейменування файлу або каталогу. Компоненти графічного вікна: напис "Перейменування файлу або каталогу" в області North, в області Center розміщений об'єкт класу JFileChooser, в області South розміщено напис "Введіть ім'я файлу / каталогу:", текстове поле для введення нового імені та кнопка "Перейменувати". Перейменування файлу або каталогу виконується при натисканні кнопки.

3. Створіть програму видалення файлу або каталогу. Компоненти графічного вікна: напис "Видалення файлу або каталогу" в області North, в області Center розміщений об'єкт класу JFileChooser, в області South розміщено напис "Введіть ім'я файлу / каталогу:", текстове поле для введення імені файлу, що видаляється / каталогу і кнопка "Видалити". Видалення файлу або каталогу виконується при натисканні кнопки після виведення діалогового вікна для підтвердження або скасування видалення файлу.

4. Створіть програму для перегляду даних в текстовому файлі з використанням класу StreamTokenizer. Кожен рядок файлу містить ім'я користувача, пароль користувача і довільні відомості про користувача (облікову інформацію користувача). Компоненти графічного вікна: напис "Отримання даних користувача" в області North, в області Center розміщено напис "Введіть ім'я:" і текстове поле для введення імені користувача, в області South розміщені кнопка "Виведення", напис "Пароль:", текстове поле для виводу пароля користувача, напис "Облікова інформація:", текстове поле для виводу облікової інформації користувача, напис "повідомлення:" і текстове поле для виведення повідомлення про результат операції. При натисканні кнопки файл проглядається і, якщо користувач знайдений, його пароль і облікова інформація виводяться у відповідних полях, інакше виводиться повідомлення "Користувач не знайдений".

5. Створіть програму виведення файлів каталогу з заданим розширенням. Компоненти графічного вікна: напис "Виведення файлів заданих типів" в області North, в області Center розміщений об'єкт класу JFileChooser, в області South розміщено напис "Введіть тип файлу:", текстове поле для введення типу файлу (наприклад, ".doc" або "всі файли ") і кнопка" Виведення". Виведення файлів каталогу з заданим розширенням виконується при натисканні кнопки.

6. Створіть програму для заміни слів у текстовому файлі з використанням класу StreamTokenizer. Компоненти графічного вікна: напис "Заміна слів у текстовому файлі" в області North, в області Center розміщений об'єкт класу JFileChooser, в області South розміщено напис "Слово пошуку:", текстове поле для введення слова пошуку, напис "Слово заміни:", текстове поле для введення слова заміни, прапорець з написом "Облік регістра", кнопка "Замінити", напис "Кількість замін:" і текстове поле для виводу кількості

замін слова. При натисканні кнопки виконується введення рядків файлу, їх зміну і збереження в тимчасовому файлі. Після перегляду файлу вміст вихідного файлу перезаписується з тимчасового файлу, а потім тимчасовий файл видаляється.

7. Створіть програму для перегляду даних в текстовому файлі з використанням класу StreamTokenizer. Кожен рядок файлу містить ім'я товару, ціну і довільні відомості про товар. Компоненти графічного вікна: напис "Відомості про товар" в області North, в області South розміщені спадне меню (JComboBox) з іменами товарів, напис "Характеристики товару:", текстове поле для виводу відомостей про товар, напис "Ціна:" і текстове поле для виводу ціни товару. На початку роботи програми проглядається файл і з імен товарів формується спадне меню. При виборі товару виводяться характеристики товару і його ціна.

8. Створіть програму переміщення файлу з одного каталогу в інший. Компоненти графічного вікна: напис "Переміщення файлу" в області North, розщеплена панель в області Center і кнопка "Перемістити" в області South. У лівій частині розщепленої панелі розміщений об'єкт класу JFileChooser для вибору переміщуваного файлу, а в правій частині - другий об'єкт класу JFileChooser, в якому вибирається каталог для переміщення. Переміщення файлу виконується при натисканні кнопки.

9. Створіть програму невеликого редактора (до 100 рядків) з використанням класу BufferedReader. Компоненти графічного вікна: напис "Введення тексту в файл" в області North, в області South розміщено напис "Введіть ім'я файлу:", текстове поле для введення імені файлу і кнопка "Зберегти". Цей редактор вводить дані з клавіатури в масив рядків (розмір масиву - 100 елементів). Ознакою закінчення введення та закриття вхідного потоку є введення Ctrl + Z. При натисканні кнопки введені рядки запам'ятовуються в заданому файлі.

10. Створіть програму виведення кількості файлів каталогу з заданим розширенням у виділеному каталозі. Компоненти графічного вікна: напис "Виведення кількості файлів заданих типів" в області North, в області Center розміщений об'єкт класу JFileChooser, в області South розміщено напис "Введіть тип файлу:", текстове поле для введення типу файлу (наприклад, ".doc" або "Всі файли"), напис "Кількість файлів:" і текстове поле для виводу кількості файлів заданого типу. Виведення кількості файлів каталогу з заданим розширенням виконується при натисканні клавіші Enter в полі введення типу файлу.

11. Створіть програму злиття вмісту виділених файлів в каталозі. Компоненти графічного вікна: напис "Злиття виділених файлів" в області North, в області Center розміщений об'єкт класу JFileChooser, в області South розміщені кнопка "Виділити", текстове поле для введення імені створюваного файлу і кнопка "Злити". Файли зливаються при натисканні

кнопки "Злити" в тому порядку, в якому вони були виділені у вікні, а потім відзначені за допомогою кнопки "Виділити".

12. Створіть програму для отримання характеристик списку файлів в каталозі з використанням методів класу File. Компоненти графічного вікна: напис "Введіть ім'я каталогу:" і текстове поле для введення шляху до каталогу в області North, а в області Center розміщена текстова область для виводу списку файлів в заданому каталозі. Виведення списку файлів виконується при натисканні клавіші Enter в текстовому полі. В кожному рядку списку виводиться ім'я файлу, його тип, розмір і дата останньої модифікації.

6 Порядок обробки результатів експериментів та вироблення висновків

На підставі отриманих результатів лабораторної роботи зробить висновки щодо особливостей роботи з файловою системою і потоками вводу/виводу в Java. В висновках до лабораторної роботи наведіть ієрархію символьних і байтових потоків. Опишіть призначення класів адаптеров і декораторів.

7 Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

- титульний лист, де вказані номер і назва лабораторної роботи, відомості про виконавця;
- номер варіанта роботи та текст завдання;
- відповіді на контрольні запитання до лабораторної роботи;
- листинг програмного коду;
- результати виконання програми;
- висновки по підсумках роботи.

Підготовлений звіт надається викладачу на перевірку та захищається студентом на останьому занняті з даної лабораторної роботи.

Лабораторна робота № 5

Програмування колекцій в JAVA

1 Мета роботи

Метою роботи є придбання навичок використання колекцій в програмах на мові Java.

Після вконання лабораторної роботи студенти мають оволодіти вміннями вибирати і реалізовувати класи-колекції для вирішення завдань.

2 Завдання на підготовку до лабораторної роботи

2.1 Перелік знань та вмінь, на які необхідно спиратися при виконанні роботи

Знання:

- компоненти колекцій;
- інтерфейси колекцій;
- реалізації колекцій і алгоритми;
- способи використання колекцій при розробці java-додатків;
- компоненти бібліотек AWT і Swing.

Вміння:

- вибирати і реалізовувати класи-колекції для вирішення завдань;
- використовувати колекцій при розробці java-додатків;
- проектувати графічний інтерфейс користувача інформаційних систем.

2.2 Теоретичні відомості

2.2.1 Компоненти колекцій

Колекція, або контейнер – це об'єкт, який об'єднує декілька елементів в один об'єкт. Колекції використовуються для зберігання даних, доступу і маніпуляцій з даними, а також для передачі даних від одного методу до іншого. Колекція зазвичай містить дані, які представляють природну групу, наприклад телефонний довідник (колекція відповідностей між ім'ям і телефонним номером). Масив також можна розглядати як колекцію, що об'єднує дані одного типу, елементи якої розташовані послідовно, в порядку зростання індексу.

Всі інтерфейси і класи, що відносяться до колекцій, знаходяться в пакеті **java.util**.

Схема колекцій (collections framework) – це уніфікована архітектура для представлення і маніпулювання колекціями. Всі схеми колекцій містять наступні три компоненти:

 Інтерфейси – абстрактні типи даних, що представляють колекції. Інтерфейси дозволяють маніпулювати колекціями незалежно від деталей їх подання. В Java, як і в інших об'єктно-орієнтованих мовах, ці інтерфейси зазвичай утворюють ієрархію.

 Реалізації – конкретні реалізації інтерфейсів колекції. За своєю суттю вони є повторно використовуваними структурами даних.

 Алгоритми – методи, що виконують деякі корисні дії, наприклад, пошук або сортування, над об'єктами, що реалізують інтерфейси колекцій. Ці методи називають поліморфними, так як один і той же метод може використовуватися в багатьох різних реалізаціях відповідного інтерфейсу колекцій. По суті алгоритми забезпечують повторно використовувану функціональність.

2.2.2 Интерфейси колекцій

Ключові інтерфейси колекцій введені, починаючи з JDK 1.2, і використовуються для маніпулювання колекціями, а також для передачі їх від одного методу до іншого. Головною метою цих інтерфейсів є можливість маніпулювання колекціями незалежно від деталей їх реалізації.

Інтерфейси утворюють дві ієрархії.

Інтерфейс Collection є коренем першої ієрархії. Цей інтерфейс представляє групу об'єктів, відомих як його елементи. Деякі реалізації колекцій дозволяють дублювання елементів, інші – ні. Пакет SDK не забезпечує прямий реалізації цього інтерфейсу і він використовується для передачі колекцій і маніпулювання ними, коли потрібна максимальне узагальнення.

Інтерфейс **Collection** оголошує методи, що виконують такі операції:

– визначення кількості елементів в колекції (методи int **size()**);

– перевірка, чи містить колекція елементи (метод **boolean isEmpty()**);

– перевірка, чи перебуває даний елемент в колекції (метод **boolean contains (Object element)**);

– порівняння заданого об'єкта з даною колекцією на рівність (метод **public boolean equals (Object o)**);

– додавання елемента до колекції (метод **boolean add (Object element)**) і видалення елемента з колекції (метод **boolean remove (Object element)**), а також очищення колекції (метод **public void clear ()**);

– забезпечення ітераційних операцій над колекцією (метод **Iterator iterator ()**);

– забезпечення моста між колекціями і старими інтерфейсами прикладних програм, які припускали передачу їм параметрів об'єкта у вигляді масиву (методи **Object [] toArray ()** і **Object [] toArray (Object a [])**).

Крім цього, для колекцій визначені наступні групові операції

boolean containsAll (Collection c)

boolean addAll (Collection c)

boolean removeAll (Collection c)
boolean retainAll (Collection c),

які дозволяють перевірити, чи містяться всі елементи колекції **c** в даній колекції; додати всі елементи колекції **c** до даної колекції; видалити з даної колекції всі елементи, що містяться в колекції **c** або залишити в даній колекції тільки ті елементи, які містяться в колекції **c**.

Метод **iterator** повертає об'єкт інтерфейсу **Iterator**. Інтерфейс **Iterator** оголошує наступні методи для колекцій:

boolean hasNext (),
перевіряє, чи є наступний елемент,
Object next (),
повертає наступний елемент і
void remove (),
видаляє даний елемент.

Нижче наводиться приклад, як об'єкт **Iterator** використовується для фільтрації колекції, тобто видалення з колекції елементів, що відповідають певним умовам:

```
static void filter(Collection c)
{
        for (Iterator i = c.iterator(); i.hasNext(); )
            if (!cond(i.next()))
                i.remove();
}
```

Даний код є поліморфним, тобто буде працювати для будь-якої колекції, що підтримує видалення елементів, незалежно від реалізації.

Інтерфейс Set є колекцією, яка не повинна містити елементів, що повторюються. Інтерфейс розширює інтерфейс **Collection** і містить ті ж методи, що і батьківський інтерфейс. Проте методи в цьому інтерфейсі мають більш конкретний математичний сенс. Так, наприклад:

– вираз **s1.containsAll (s2)** повертає true, якщо s2 є підмножиною s1;

– вираз **s1.addAll (s2)** перетворює s1 в об'єднання s1 і s2;

– вираз **s1.retainAll (s2)** перетворює s1 в перетин s1 і s2 (перетин двох множин містить елементи, загальні для обох множин);

– вираз **s1.removeAll (s2)** видаляє з s1 всі елементи, що містяться в s2.

Інтерфейс List є впорядкованою колекцією (іноді званою послідовністю). Колекція **List** може містити впорядковані елементи. На додаток до операцій, успадкованих від колекції, інтерфейс оголошує наступні операції:

– отримання значення елемента із заданим індексом (метод **public Object get (int index)**) і установка значення елемента із заданим індексом (метод **public Object set (int index, Object element)**);

– пошук елемента у списку і повернення значення його індексу (методи **public int indexOf (Object o)** і **public int lastIndexOf (Object o)**);

– додавання (метод **public void add (int index, Object element)**) або видалення (метод **public Object remove (int index)**) елемента по заданому індексу, а також очищення списку (метод **public void clear ()**);

– операції над частиною списку (метод **public List subList (int fromIndex, int toIndex)**);

– ітераційні операції у списку (метод **public ListIterator listIterator()** і **public ListIterator listIterator (int index)**).

Інтерфейс ListIterator розширює інтерфейс **Iterator** і містить наступні методи перегляду списку:

– **Object next()** – повертає наступний елемент (якщо наступного елемента немає, то викидається виключення типу NoSuchElementException);

– **Object previous()** – повертає попередній елемент (якщо попереднього елемента немає, то викидається виключення типу NoSuchElementException);

– **boolean hasNext()** – повертає true, якщо існує наступний елемент, інакше повертає false;

– **boolean hasPrevious()** – повертає true, якщо існує попередній елемент, інакше повертає false;

– **int nextIndex()** – повертає індекс наступного елемента (якщо наступного елемента немає, повертає розмір списку);

– **int previousIndex()** – повертає індекс попереднього елемента (якщо попереднього елемента немає, повертає -1);

– **void add (Object obj)** – вставляє obj в список перед елементом, який буде повернутий наступним викликом next();

– **void remove()** – видаляє поточний елемент із списку (викидається виключення типу IllegalStateException, якщо метод remove() викликається, перш ніж викликаний метод next() або previous());

– **void set (Object obj)** – призначає obj на поточний елемент (це останній елемент, повернутий викликом методу next() або previous()).

Об'єкти в програмах Java упорядковано за допомогою методу

public int compareTo (Object obj),

оголошеного в інтерфейсі **Comparable** (це єдиний метод даного інтерфейсу). Метод повинен повертати значення 0, якщо порівнювані об'єкти дорівнюють один одному; значення, менше нуля, якщо приймаючий об'єкт менше obj; значення, більше нуля, якщо приймаючий об'єкт більше obj. Конкретні реалізації цього методу в об'єктних розширеннях числових класів (наприклад, Integer або Float) дозволяють порівнювати числа за значенням, для рядків – порівнювати рядки в лексикографічному порядку, для дат – в хронологічному порядку. Сортування, засноване на такому порівнянні, називається природним порядком порівняння.

Але часто необхідно відсортувати елементи колекції в порядку, відмінному від природного або відсортувати елементи без реалізації інтерфейсу **Comparable**. У цьому випадку необхідно реалізувати інтерфейс **Comparator** і забезпечити конкретну реалізацію його методів

public int compare (Object o1, Object o2).

і

public boolean equals (Object obj).

Перший метод повертає від'ємне значення, якщо об'єкт **o1** менше об'єкта **o2**, значення 0 у разі рівності об'єктів і додатне значення, якщо об'єкт **o1** більше об'єкта **o2**. Другий метод перевизначає відповідний метод класу **Object** і перевіряє рівності даного об'єкта інтерфейсу **Comparator** з об'єктом **obj**.

Інтерфейс SortedSet є розширенням інтерфейсу **Set**, елементи якого відсортовані в природному порядку або згідно з порядком, що задається методом інтерфейсу **Comparator**.

Інтерфейс реалізує наступні операції (додатково до операцій інтерфейсу **Set**):

– операції над частиною набору (метод **public SortedSet subSet (Object fromElement, Object toElement)**), головною частиною набору (метод **public SortedSet headSet (Object toElement)**) і хвостовою частиною набору (метод **public SortedSet tailSet (Object fromElement)**);

– повернення першого (метод **public Object first ()**) і останнього (метод **public Object last ()**) елемента відсортованого набору;

– доступ до інтерфейсу **Comparator** (метод **public Comparator comparator()**).

Інтерфейс Map є об'єктом, який ставить у відповідність ключам значення. Ключі повинні бути унікальними і їм повинно відповідати єдине значення. Таку колекцію називають відображенням (map), словником (dictionary) або асоціативним масивом (associative array).

В інтерфейсі **Map** визначені наступні основні операції:

– отримання розміру відображення (метод **public int size()**);

– перевірка відображення на наявність елементів (метод **public boolean isEmpty()**);

– приміщення елемента в відображення (метод **public Object put (Object key, Object value)**) та отримання значення елемента із заданим ключем (метод **public Object get (Object key)**);

– порівняння відображення з заданим об'єктом на рівність (метод **public boolean equals (Object o)**);

– представлення ключів колекції у вигляді безлічі (метод **public Set keySet ()**);

– видалення елемента із заданим ключем (метод **public Object remove (Object key)**) і очистка відображення (метод **public void clear()**);

– перевірка наявності елемента із заданим ключем (метод **public boolean containsKey (Object key)**) або значенням (метод **public boolean containsValue (Object value)**);

– додавання до даного відображенню всіх пар із заданого відображення (метод **public void putAll (Map t)**);

– представлення всіх значень даного відображення у вигляді колекції (метод **public Collection values()**);

– представлення колекції у вигляді безлічі, кожен елемент якого – пара з даного відображення, з якою можна працювати методами вкладеного інтерфейсу Map.Entry (метод **public Set entrySet()**).

Інтерфейс Map.Entry описує методи роботи з парами «ключ-значення», отриманими методом **entrySet()**:

– отримання ключа (метод **public Object getKey()**) і значення (метод **public Object getValue()**);

– зміна значення (метод **public Object setValue (Object value)**);

– порівняння заданого об'єкта з даною парою «ключ-значення» на рівність (метод **public boolean equals (Object o)**).

Можливо також використання об'єктів **Map**, в яких ключам відповідає кілька значень. Це досягається зазвичай, якщо в якості значень задати об'єкти **List**.

Інтерфейс SortedMap є розширенням **Map**. Елементи для цього інтерфейсу відсортовані в природному порядку або згідно з порядком, що задається методами інтерфейсу **Comparator**.

Інтерфейс реалізує наступні операції (додатково до операцій інтерфейсу **Map**):

– виділення частини відображення (метод **public SortedMap subMap (Object fromKey, Object toKey)**), головної частини відображення (метод **public SortedMap subMap (Object fromKey, Object toKey)**) або хвостовій частині відображення (метод **public SortedMap tailMap (Object fromKey)**);

– отримання першого (метод **public Object firstKey()**) і останнього (метод **public Object lastKey()**) елемента відображення;

– доступ до інтерфейсу **Comparator** (метод **public Comparator comparator()**).

В цілому, інтерфейс **SortedMap** є аналогом інтерфейсу **SortedSet**.

2.2.3 Реалізації колекцій і алгоритми

Реалізації є дійсними об'єктами даних, які втілюють у життя ключові інтерфейси колекцій, описані в попередньому розділі.

Існують три типи реалізацій:

– загальноцільові реалізації;

– пакувальники;

– альтернативні (convenience) реалізації.

Загальноцільові реалізації та їх зв'язок з інтерфейсами і структурами даних представлені в таблиці:

Інтерфейс	Реалізації			
	Хеш-таблиця	Змінюваний масив	Збалансоване дерево	Зв'язаний список
Set	**HashSet**		**TreeSet**	**LinkedHashSet**
List		**ArrayList**		**LinkedList**
Map	**HashMap** **WeekHashMap**		**TreeMap**	**LinkedHashMap**

Всі реалізації мають не тільки схожі імена, але й подібну поведінку. Всі вони реалізують кожну з операцій, визначених у своєму інтерфейсі. Всі дозволяють використовувати елементи, ключі і значення з константою null.

Основним при реалізації структур даних є вибір інтерфейсу. У більшості випадків вибір реалізації впливає тільки на продуктивність програми. Тому переважним стилем програмування є: спочатку створення колекції, потім вибір реалізації і присвоювання нової колекції змінної відповідного інтерфейсного типу (або передача колекції методу, що очікує аргумент интерфейсного типу). Таким чином, програма стає незалежною від будь-яких нових методів, що додаються у даній реалізації, залишаючи за програмістом право зміни реалізації, якщо це необхідно для ефективності програми.

Класи AbstractSet, AbstractList, AbstractSequentialList і AbstractMap

Клас **AbstractSet** є суперкласом для класів **HashSet** і **TreeSet** і містить реалізації методів **equals()** і **removeAll()** інтерфейсу **Set**.

Клас **AbstractList** є суперкласом для класів **AbstractSequentialList**, **ArrayList**, **Vector** і містить реалізації двох методів **add()**, методів **addAll()**, **clear**(), **equals**(), **get**(), **indexOf**(), **iterator**(), **lastIndexOf**(), **listIterator**(), **remove**(), **set**(), **sublist**() інтерфейсу **List**.

Клас **AbstractList** містить також метод

protected void removeRange (int fromIndex, int toIndex)

який дозволяє видалити елементи списку в заданому діапазоні.

Клас **AbstractSequentialList** є суперкласом для класу **LinkedList** і містить реалізації методів **add**(), **addAll**(), **get**(), **iterator**(), **listIterator**(), **remove**() і **set**() інтерфейсу **List**.

Клас **AbstractMap** є суперкласом для класів **HashMap**, **TreeMap** і **WeakHashMap** і містить реалізації методів **clear**(), **containsKey**(), **containsValue**(), **entrySet**(), **equals**(), **get**(), **isEmpty**(), **keySet**(), **put**(), **putAll**(), **remove**(), **size**() і **values**() інтерфейсу **Map**. Крім того, клас **AbstractMap** містить метод

protected Object clone()

дозволяє отримати порожню копію відображення (ключі і значення не копіюються), а також метод

public String toString()
дозволяє перетворити відображення в строкове представлення.

Клас HashSet

Для інтерфейсу **Set** двома загальноцільовими реалізаціями є **HashSet** і **TreeSet**.

Клас **HashSet** реалізує інтерфейс **Set** с підтримкою хеш-таблиці (зазвичай є реалізацією **HashMap**) і має наступні конструктори:

– **HashSet()** – створює новий, порожній набір з ємністю за замовчуванням і фактором завантаження, рівним 0.75;

– **HashSet(Collection c)** – створює новий набір, що містить елементи заданої колекції;

– **HashSet(int initialCapacity)** – створює новий порожній набір із заданою ємністю і фактором завантаження, рівним 0.75;

– **HashSet(int initialCapacity, float loadFactor)** – створює новий порожній набір із заданою ємністю за замовчуванням і заданим чинником завантаження.

У класі **HashSet** реалізовані наступні методи інтерфейсу **Set**: **add**(), **clear**(), **contains**(), **isEmpty**(), **iterator**(), **remove**() і **size**(). Метод

public Object clone()
дозволяє отримати порожню копію об'єкта **HashSet** (без елементів).

Клас ArrayList

Клас **ArrayList** забезпечує реалізацію інтерфейсу **Set** для масивів із змінними розмірами. Так само, як об'єкти класу **StringBuffer**, об'єкти **ArrayList** мають дві характеристики – ємність і розмір масиву. Якщо розмір списку перевищить ємність, ємність автоматично збільшується на деяку кількість елементів.

Клас **ArrayList** має наступні конструктори:

– **ArrayList()** – створює порожній список;

– **ArrayList(Collection c)** – створює список із заданої колекції в порядку, заданому ітератором даної колекції;

– **ArrayList(int initialCapacity)** – створює порожній список із заданою ємністю.

Клас **ArrayList** реалізує наступні методи інтерфейсу **List**: два методи **add**(), два методи **addAll**(), методи **clear**(), **contains**(), **get**(), **indexOf**(), **isEmpty**(), **lastIndexOf**(), **remove**(), **set**(), size() і два методи **toArray**().

Крім того, клас містить наступні власні методи:

– **public Object clone()** – отримання порожній копії об'єкта **ArrayList** (без елементів);

– **public void ensureCapacity (int minCapacity)** – збільшує ємність екземпляру **ArrayList**, якщо це необхідно, для того, щоб він міг містити число елементів, задане в **minCapacity**;

– **protected void removeRange (int fromIndex, int toIndex)** – видаляє елементи в заданому діапазоні індексів;

– **public void trimToSize()** – зменшує ємність об'єкта **ArrayList** до його попереднього значення.

Клас LinkedList

Клас **LinkedList**, на додаток до реалізації методів інтерфейсу **List** забезпечує методи для отримання, видалення і вставки елементів в список, що дозволяє використовувати зв'язані списки як стеки, черги або черги з двома кінцями.

Клас **LinkedList** має два конструктора:

– **LinkedList ()** – створює порожній зв'язаний список;

– **LinkedList (Collection c)** – створює зв'язаний список із заданої колекції в порядку, заданому ітератором даної колекції.

На додаток до двох методів **add**(), двох методів **addAll**(), методів **clear**(), **contains**(), **get**(), **indexOf**(), **isEmpty**(), **lastIndexOf**(), **listIterator**(), двох методів **remove**(), методів **set**(), **size**() і двох методів **toArray**() інтерфейсу **List**, клас **LinkedList** забезпечує наступні методи:

– **public void addFirst (Object o)** і **public void addLast (Object o)** – додавання елемента в початок або кінець зв'язаного списку;

– **public Object getFirst()** і **public Object getLast()** – отримання першого або останнього елемента зв'язаного списку;

– **public Object removeFirst()** і public **Object removeLast()** – видалення першого або останнього елемента зв'язаного списку;

– **public Object clone()** – отримання порожній копії об'єкта **LinkedList** (без елементів).

Клас **ArrayList** є кращим перед **LinkedList** у випадку, якщо необхідна висока продуктивність. Однак, якщо часто доводиться додавати елементи в початок списку або видаляти елементи з його середини, кращим є клас **LinkedList**. Цей клас повністю реалізує всі можливості класу **Stack.**

Клас HashMap

Клас **HashMap** є реалізацією інтерфейсу **Map** з використанням хеш-таблиць і за своїми можливостями відповідає класу **HashTable**.

Клас **HashMap** має наступні конструктори:

– **HashMap()** – створює нове, пусте відображення з ємністю за замовчуванням 16 і фактором завантаження, рівним 0.75;

– **HashMap (Map m)** – створює нове відображення, що містить елементи заданого відображення;

– **HashMap (int initialCapacity)** – створює нове, пусте відображення із заданою ємністю і фактором завантаження, рівним 0.75;

– **HashMap (int initialCapacity, float loadFactor)** – створює нове, пусте відображення із заданою ємністю і заданим фактором завантаження.

– Клас **HashMap** реалізує наступні методи інтерфейсу **Map**: **clear**(), **containsKey**(), **containsValue**(), **entrySet**(), **get**(), **isEmpty**(), **keySet**(), **put**(), **putAll**(), **remove**(), **size**() і **values**().

Клас Collections

Реалізації-оболонки делегують всю свою роботу колекціям, але додають в колекції деяку функціональність. Ці реалізації є анонімними, тобто вони не забезпечують загальнодоступний клас, а реалізуються за допомогою статичних методів в класі **Collections**.

Клас **Collections** оперує з колекціями або повертає колекції. Клас не містить конструктора, а тільки такі поля з модифікаторами **public static final**:

List EMPTY_LIST для порожніх списків;

Map EMPTY_MAP для порожніх відображень

Set EMPTY_SET для порожніх множин, а також наступні **public static** методи, в основному реалізують поліморфні алгоритми:

– **int binarySearch (List list, Object key)** – двійковий пошук в заданому списку заданого об'єкта;

– **int binarySearch (List list, Object key, Comparator c)** – двійковий пошук в заданому списку заданого об'єкта з використанням компаратора;

– **void copy (List dest, List src)** – копіювання списку-джерела в список призначення;

– **void fill (List dest, Object o)** – заповнення списку заданих об'єктом;

– **Object max (Collection c)** і **Object min (Collection c)** – повернення максимального або мінімального елемента колекції відповідно до природного порядку порівняння;

– **Object max (Collection c, Comparator com)** і **Object min (Collection c, Comparator com)** – повернення максимального або мінімального елемента колекції відповідно до порядку, заданих за допомогою компаратора;

– **void reverse (List l)** – переставляє елементи списку у зворотному порядку;

– **void shuffle (List l)** – перемішує елементи списку з використанням генератора випадкових чисел за замовчанням;

– **void shuffle (List l, Random rnd)** – перемішує елементи списку з використанням заданого генератора випадкових чисел;

– **List singletonList (Object o)** – повертає список, який містить лише даний об'єкт;

– **Map singletonMap (Object key, Object value)** – повертає відображення, що містить тільки даний ключ і дане значення;

– **void sort (List l)** і **void sort (List l, Comparator c)** – сортує список по зростанню відповідно до природнього порядку або з використанням компаратора;

– **Collection synchronizedCollection (Collection c)**, **List synchronizedList (List l)**, **Map synchronizedMap (Map m)**, **Set synchronizedSet (Set s)**, **SortedSet**

synchronizedSortedSet (SortedSet ss) і **SortedMap synchronizedSortedMap (SortedMap sm)** – створюють синхронізовані (які не залежать від потоків) екземпляри відповідних типів колекцій;

– **Collection unmodifiableCollection (Collection c)**, **List unmodifiableList (List l)**, **Map unmodifiableMap (Map m)**, **Set unmodifiableSet (Set s)**, **SortedSet unmodifiableSortedSet (SortedSet ss)** і **SortedMap unmodifiableSortedMap (SortedMap sm)** – створюють незмінні екземпляри відповідних типів колекцій.

2.3 Контрольні питання

1. Як визначаються колекції в мові Java? Які компоненти містить схема колекцій в Java?
2. Які інтерфейси колекцій визначені в мові Java?
3. Які операції над колекціями визначені за допомогою методів інтерфейсу Collection?
4. Які методи для перегляду елементів колекцій визначені в інтерфейсі Iterator?
5. Які додаткові методи для колекцій визначені в інтерфейсі List?
6. Які методи перегляду списку містить інтерфейс ListIterator?
7. Як упорядковано об'єкти з використанням інтерфейсу Comparable в програмах на мові Java?
8. Які методи (додатково до методів інтерфейсу Set) реалізує інтерфейс SortedSet?
9. Як визначається інтерфейс Map, і які операції визначені в цьому інтерфейсі?
10. Які методи (додатково до методів інтерфейсу Map) реалізує інтерфейс SortedMap?
11. Які типи реалізацій колекцій існують у мові Java? Дайте коротку характеристику кожного типу.
12. Які абстрактні класи реалізації колекцій визначені в Java? Дайте коротку характеристику кожного класу.
13. Як за допомогою класу ArrayList реалізуються масиви із змінними розмірами в програмах на мові Java?
14. Як в класі LinkedList реалізуються пов'язані списки?
15. Як в класі HashMap в Java реалізуються відображення з підтримкою хеш-таблиць?
16. Які операції над колекціями забезпечують методи класу Collections?

3 Опис приладів, устаткування та інструментів

– Обладнання: IBM-сумісний персональний комп’ютер (ПК).

– Програмне забезпечення: операційна система Windows, Java 2 SDK версії 1.5 та вище. Інтегроване середовище розробки IDE Eclipse.

4 Правила техніки безпеки та охорони праці

Правила техніки безпеки при виконанні лабораторної роботи регламентуються «Правилами техніки безпеки при роботі в комп'ютерній лабораторії».

5 Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

– контроль рівня підготовленості студентів до виконання роботи у формі усного опитування;

– інструктаж по правилах охорони праці перед початком лабораторної роботи та оформлення його підсумків в журналі проведення лабораторних робіт, який ведеться у навчальній лабораторії;

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java у інтегрованому середовищі розробки Eclipse.

5.1 Завдання до лабораторної роботи

Напишіть графічний додаток Swing для виконання операцій з колекціями по одному з наведених нижче варіантів.

5.2 Варіанти

1. Створіть додаток Swing для додавання абонента і перегляду списку абонентів телефонної мережі. Записи в списку є об'єктами класу HashMap, де ключем є номер телефону, а значенням – об'єкт, що містить чотири значення типу String: прізвище, ім'я, по батькові та адресу. Після введення записи сортуються за номером телефону. Вікно додатка містить п'ять текстових полів для введення значень номера телефону, прізвища, імені, по батькові та адреси, а також кнопки "Додати", "Сортувати", "Перегляд", "<" і ">". При натисканні кнопки "Перегляд" виводиться перший запис, при натисканні кнопки ">" виводиться наступний запис, а при натисканні кнопки "<" - попередній.

2. Створіть додаток Swing для перегляду списку книг і видалення книг у бібліотечному каталозі. Записи в списку є об'єктами класу HashMap, де ключем є індекс ISBN книги (ціле число), а значенням – об'єкт, що містить найменування книги, ПІБ автора, видавництво (типу String), рік видання (типу int) і ціну книги (типу float). Вікно додатка містить шість текстових полів для виводу значень індексу ISBN, найменування книги, ПІБ автора, видавництва, року видання і ціни. Крім цього, у вікні міститься текстове поля для введення індексу ISBN книги, що видаляється, і текстове поле для виведення

повідомлення "Книга видалена" або "Книга не знайдена", а також кнопку "Видалити".

3. Створіть додаток Swing для перегляду списку елементів в масиві цілих чисел, що є об'єктом класу ArrayList, і зміни елементів списку в заданому діапазоні індексів. При натисканні кнопки "Перегляд" виводиться індекс першого елемента і його значення, при натисканні кнопки ">" виводиться індекс наступного елемента і його значення, а при натисканні кнопки "<" – індекс попереднього елемента і його значення. Вікно додатка містить також три текстових поля для введення значення нижнього індексу, верхнього індексу і змінюваного елемента, а також кнопки "Змінити". При першому натисканні кнопки "Змінити" всі елементи в діапазоні від верхнього до нижнього індексу видаляються з масиву, і на їх місце вставляється ціле значення, введене в третьому текстовому полі. При введенні в третьому полі нового значення і натисканні кнопки "Змінити" введене значення поміщається після першого значення і т.д. Ознакою закінчення введення нових елементів є введення в третьому полі порожній рядки (якщо порожній рядок введений в самому початку, елементи в заданому діапазоні просто видаляються зі списку).

4. Створіть додаток Swing для пошуку абонента телефонної мережі. Записи в списку абонентів є об'єктами класу HashMap, де ключем є номер телефону, а значенням – об'єкт, що містить чотири значення типу String: прізвище, ім'я, по батькові та адресу. Вікно додатка містить список, що розкривається для введення критерію пошуку: за номером телефону, на прізвище або за адресою і текстове поле для введення значення пошуку, а також чотири текстових поля для виведення значень прізвища, імені, по батькові та адреси і кнопку "Пошук". Якщо абонент не знайдений, в текстовому полі для номера телефону виводиться символ "*".

5. Створіть додаток Swing для перегляду списку черговиків. Записи в списку є об'єктами класу LinkedList, де об'єкт містить поля прізвища, імені та по батькові черговика (типу String) і пріоритет черговика (типу int). Записи в черзі сортуються відповідно до пріоритету, і черговик додається останнім у черзі свого пріоритету. Вікно додатка містить текстове поле для введення номера пріоритету і три текстових поля для виведення значень прізвища, імені, по батькові, а також кнопку "Вибрати". При натисканні кнопки "Вибрати" виводиться перший запис, при натисканні кнопки ">" виводиться наступний запис, а при натисканні кнопки "<" – попередній. Якщо чергу заданого пріоритету порожня, або список черговиків заданого пріоритету вичерпано, в поле прізвища виводиться символ "*".

6. Створіть додаток Swing для додавання елементів і перегляду списку елементів в масиві цілих чисел, що є об'єктом класу ArrayList. Після введення елементи сортуються за зростанням. Вікно додатка містить два текстових поля для введення значення індексу, після якого вставляється число і значення числа, а також кнопки "Додати", "Сортувати", "Перегляд", "<" і ">". При натисканні кнопки "Перегляд" виводиться індекс першого елемента і його значення, при натисканні кнопки ">" виводиться індекс наступного елемента і

його значення, а при натисканні кнопки "<" – індекс попереднього елемента і його значення.

7. Створіть додаток Swing для перегляду списку зображень і видалення зображення зі списку зображень. Записи в списку є об'єктами класу HashMap, де ключем є найменування зображення (типу String), а значенням – зображення (об'єкт класу Image). При натисканні кнопки "Перегляд" в текстовому полі виводиться найменування першого зображення, а в заданому місці вікна (за допомогою методу drawImage ()) виводиться зображення. При натисканні кнопки ">" виводиться наступне найменування і зображення, а при натисканні кнопки "<" – попереднє найменування і зображення. При натисканні кнопки "Видалити" поточне зображення видаляється зі списку.

8. Створіть додаток Swing для зміни значення елементів і перегляду списку елементів в множині цілих чисел, що є об'єктом класу HashSet (елементи множини не повинні збігатися). При натисканні кнопки "Перегляд" в текстовому полі виводиться значення першого елемента множини, при натисканні кнопки ">" виводиться значення наступного елемента, а при натисканні кнопки "<" – значення попереднього елементу. Якщо змінити значення елемента в текстовому полі і натиснути кнопку "Редагувати", то буде зроблена спроба змінити значення поточного елемента. У другому текстовому полі виводиться повідомлення про підсумок операції: "Елемент змінено" або "Елемент вже є".

9. Створіть додаток Swing для зміни абонента телефонної мережі. Записи в списку абонентів є об'єктами класу HashMap, де ключем є номер телефону, а значенням – об'єкт, що містить чотири значення типу String: прізвище, ім'я, по батькові та адресу. Вікно додатка містить текстове поле для введення номера телефону і текстові поля для введення нових значень прізвища, імені, по батькові та адреси і кнопку "Замінити". Якщо номер телефону не знайдене в текстовому полі, для номера телефону виводиться символ "*".

10. Створіть додаток Swing для додавання книг і перегляду списку книг у бібліотечному каталозі. Записи в списку є об'єктами класу HashMap, де ключем є індекс ISBN книги (ціле число), а значенням – об'єкт, що містить найменування книги, ПІБ автора, видавництво (типу String), рік видання (типу int) і ціну книги (типу float). Після введення записи сортуються за зростанням значень індексу ISBN. Вікно додатка містить шість текстових полів для введення значень індексу ISBN, найменування книги, ПІБ автора, видавництва, року видання і ціни, а також кнопки "Додати", "Сортувати", "Перегляд", "<" і ">". При натисканні кнопки "Перегляд" виводиться перший запис, при натисканні кнопки ">" виводиться наступний запис, а при натисканні кнопки "<" – попередній.

11. Створіть додаток Swing для перегляду списку черговиків і зміни пріоритету черговиків в черзі. Записи в списку є об'єктами класу LinkedList, де об'єкт, містить поля прізвища, імені та по батькові черговика (типу String) і пріоритет черговика (типу int). Вікно додатка містить текстове поле для введення номера пріоритету і три текстових поля для виведення значень прізвища, імені, по

батькові, а також кнопку "Вибрати". При натисканні кнопки "Вибрати" виводиться перший запис, при натисканні кнопки ">" виводиться наступний запис, а при натисканні кнопки "<" – попередній. Якщо чергу заданого пріоритету порожня, або список черговиків заданого пріоритету вичерпано, в поле прізвища виводиться символ "*". При введенні в текстове поле нового значення пріоритету і натисканні кнопки "Змінити пріоритет", черговик ставиться в кінець черги нового пріоритету.

12. Створіть додаток Swing для покупки товарів в електронному магазині. Записи в списку є об'єктами класу HashMap, де ключем є артикул товару (ціле число), а значенням – об'єкт, що містить найменування товару (типу String) і ціну товару (типу float). При натисканні кнопки "Перегляд" виводиться перший запис (в текстових полях артикулу, найменування та ціни товару), при натисканні кнопки ">" виводиться наступний запис, а при натисканні кнопки "<" – попередній. Вікно додатка містить також текстове поле для введення кількості закуповуваного товару (у поточному виведеному запису) і текстове поле, в яке виводиться сумарна вартість купленого товару при натисканні кнопки "Замовити".

6 Порядок обробки результатів експериментів та вироблення висновків

На підставі отриманих результатів лабораторної роботи зробить висновки щодо особливостей роботи з колекціями в Java. В висновках до лабораторної роботи наведіть ієрархію інтерфейсів і класів-колекцій, зробіть порівняльний аналіз реалізацій колекцій.

Нижче наведений приклад використання класу HashMap для обчислення кількості повторень літер в тексті.

```
import java.util.HashMap;
import java.util.*;
public class Main {
public static void main(String[] args) {
String txt = " лабораторна робота " ;
HashMap<Character, Integer> map = new HashMap<Character, Integer>(40);
for (int i = 0; i < txt.length(); ++i) {
char c = txt.charAt(i);
// перевіряємо чи є символ літерою
if (Character.isLetter(c)) {
if (map.containsKey(c)) {
map.put(c, map.get(c) + 1);
} else {
map.put(c, 1); }
}
}
// виведення на екран букв з частотою їх появи
for (Entry<Character, Integer> entry : map.entrySet()) {
       System.out.println("буква: "+entry.getKey()+" кол - во:
"+entry.getValue());
       }
}}
```

7 Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

- титульний лист, де вказані номер і назва лабораторної роботи, відомості про виконавця;
- номер варіанта роботи та текст завдання;
- відповіді на контрольні запитання до лабораторної роботи;
- листинг програмного коду;
- результати виконання програми;
- висновки по підсумках роботи.

Підготовлений звіт надається викладачу на перевірку та захищається студентом на останьому занняті з даної лабораторної роботи.

Лабораторна робота № 6

Управління мережевими з'єднаннями. Використання сокетів у розподілених додатках

1 Мета роботи

Метою роботи є придбання навиків створення мережевих додатків на мові Java з використанням бібліотеки java.net.

Після вконання лабораторної роботи студенти мають оволодіти вміннями реалізовувати взаємодію клієнта та сервера, використовуючи сокети TCP і UDP.

2 Завдання на підготовку до лабораторної роботи

2.1 Перелік знань та вмінь, на які необхідно спиратися при виконанні роботи

Знання:

- класи пакету java.net;
- організація клієнтського і серверного сокетів;
- програмування дейтаграм;
- компоненти бібліотек AWT і Swing.

Вміння:

- розробка додатків клієнт-сервер і управління мережевими з'єднаннями на рівні сокетів;
- управління з'єднаннями на рівні дейтаграм;
- створення багато потокового серверу;
- проектувати графічний інтерфейс користувача інформаційних систем.

2.2 Теоретичні відомості

При роботі в мережі з застосуванням класів і методів пакету **java.net** будемо використовувати основні поняття, що характеризують мережеві з'єднання: IP - адресація, доменна служба імен, URL адреси, протоколи TCP і UDP, прикладні протоколи, сокети. У зв'язку з цим рекомендується повторити згадані теми за конспектом лекцій з дисципліни «Комп'ютерні мережі».

У пакеті **java.net** визначена група класів та інтерфейсів, що дозволяють управляти мережевими з'єднаннями в Java-програмах. Ось деякі з завдань, які можуть бути вирішені засобами **java.net**:

- Управління адресацією;
- Організація високорівневого програмування з використанням методів класу URL;

– Розробка додатків клієнт-сервер і управління мережевими з'єднаннями на рівні сокетів;

– Управління з'єднаннями на рівні дейтаграм.

Розглянемо ці завдання більш докладно.

2.2.1 Клас InetAddress

Клас **InetAddress** інкапсулює як числову IP - адресу, так і доменну адресу.

Клас **InetAddress** не має видимих конструкторів. Для створення об'єкта потрібно використовувати один з доступних фабричних методів.

Фабричні методи – просто угода, за допомогою якого статичні методи в класі повертають екземпляр даного класу. Це зроблено замість перевантаження конструктора різними списками параметрів, коли наявність унікальних імен методів призводить до більш ясних результатів.

Для створення об'єктів InetAddress можна використовувати три методи:

– **static InetAddress getLocalHost() trows UnknownHostException** – повертає об'єкт класу InetAddress для локального комп'ютера.

– **static InetAddress getByName (String hostName) trows UnknownHostException** – повертає об'єкт класу InetAddress для комп'ютера, DNS - ім'я якого передається в параметрі hostName.

– **static InetAddress[] getAllByName (String hostName) trows UnknownHostException** – повертає масив об'єктів класу InetAddress, що представляє всі адреси, пов'язані з зазначеним DNS-ім'ям hostName. Використовується в тому випадку, коли групі IP - адрес відповідає одне DNS-ім'я. Це дає можливість зменшити навантаження на найбільш часто відвідувані сайти.

Всі ці методи в разі неможливості розпізнавання імені комп'ютера викидають виключення типу **UnknownHostException**.

Розглянемо основні нестатичні методи класу **InetAddress**

Метод	Опис
boolean equals(Object other)	Повертає true, якщо повертаємий об'єкт має ту же IP - адресу, що і об'єкт, отриманий через параметр other
byte[] getAddress()	Повертає чотирьохелементний масив байтів, який представляє IP - адресу оброблюваного об'єкта
String getHostAddress()	Повертає рядок, який представляє собою адресу комп'ютера, пов'язаного з об'єктом класу InetAddress
String getHostName()	Повертає рядок, який представляє собою ім'я комп'ютера, пов'язаного з

	об'єктом класу InetAddress
int hashCode()	Повертає хеш-код викликаємого об'єкта
boolean isMulticastAddress()	Повертає true, якщо IP - адреса об'єкта цього класу – групова (multicast).
String toString()	Повертає рядок, який перераховує доменну і IP - адресу хост-комп'ютера (наприклад, "sterwave.com/192.147.170.6")
boolean equals(Object other)	Повертає true, якщо повертаємий об'єкт має ту же IP - адресу, що і об'єкт, отриманий через параметр other

Приклад. Розглянемо різні випадки застосування розглянутих раніше методів класу **InetAddress**, для отримання IP - адрес.

Зверніть увагу на одне розходження при використанні методів **getByName()** з параметром null і **getLocalHost()**. Якщо локальна мережа побудована на протоколі TCP/IP, то кожному комп'ютеру присвоюється IP - адреса. Тому метод **getLocalHost()** повертає дану IP - адресу цієї локальної мережі. З іншого боку метод **getByName()** з параметром null в будь-якому випадку повертає адресу 27.0.0.1.

```
import java.net.*;
public class My{

public static void main(String[] args)
throws UnknownHostException {

InetAddress localaddr = InetAddress.getLocalHost();
System.out.println("Local
                    address:"+localaddr.getHostAddress()+
                              " | "+ localaddr.getHostName());

localaddr = InetAddress.getByName(null);
System.out.println("Local
                    address:"+localaddr.getHostAddress()+
                              " | "+ localaddr.getHostName());

InetAddress addr = InetAddress.getByName("www.yandex.ru");
System.out.println("Yandex address:"+addr.getHostAddress()+
                              " | "+ addr.getHostName());

InetAddress multaddr[] =
InetAddress.getAllByName("www.google.com");
System.out.println("Google address:");
for(int i=0; i<multaddr.length; i++)
System.out.println((i+1)+": "+multaddr[i].getHostAddress()+
```

```
                                    " | "+ multaddr[i].getHostName());
}}
```

Результат роботи:

```
Local address:169.254.86.153 | SERVER
Local address:127.0.0.1   | localhost
Yandex address:77.88.21.3 | www.yandex.ru
Google address:
1: 209.85.129.104 | www.google.com
2: 209.85.129.147 | www.google.com
3: 209.85.129.99  | www.google.com
```

2.2.2 Класи URL і URLConnection

URL забезпечує зручну форму ідентифікації ресурсів мережі Internet і може складатися з 4-х складових: протоколу, доменного імені або IP - адреси комп'ютера, номера порту і шляху до файлу.

В якості протоколу можуть використовуватися http, ftp, gopher і file. Ім'я протоколу в URL завершується двокрапкою, потім після подвійного слеш (//) вказується DNS-ім'я комп'ютера або IP-адреса, наприклад http://www.yandex.ru/ або http://77.88.21.3/

Часто на комп'ютері в мережі відкривається група портів, через які відбуваються з'єднання. Для того, щоб вказати порт підключення, необхідно ввести його значення після імені комп'ютера, де як роздільник використовується двокрапка: http://somewhere.org:8888/

За ім'ям комп'ютера і номерів порту в URL адресі можна вказати шлях до файлу, для цього як роздільник використовують одинарний слеш (/): http://java.sun.com/docs/index.html

Для використання адрес URL в пакеті **java.net** визначено клас **URL**. Конструктори:

URL (String spec);

URL (String protocol, String host, String file);

URL (String protocol, String host, int port, String file);

де spec – строковий еквівалент адреси URL,

protocol – протокол,

host – ім'я комп'ютера даної URL - адреси,

port – номер порта підключення,

file – ім'я файлу, що завантажується.

Якщо вказана невірна URL – адреса, то буде згенеровано виключення **MalformedURLException**.

Методи класу **URL**

getProtocol(), getHost(), getPort(), getFile() – повертають значення протоколу, імені комп'ютера, номера порту і шляху до файлу відповідно.

openStream() – відкриває потік вхідних даних **InputStream** і дозволяє здійснити завантаження об'єкта класу **URL**.

Приклад. Завантажимо на екран вміст Web-сторінки, розташованої за адресою **http://www.yahoo.com/**

```
import java.net.*;
import java.io.*;

public class My{
public static void main(String[] args) {
try{
URL myURL = new URL("http://www.yahoo.com/");
// Создаем поток ввода
InputStream in = myURL.openStream();
// Читаем из потока и выводим на экран
int ch;
while((ch=in.read())!=-1)
System.out.print( (char)ch);
in.close();
}
// обработка исключений
catch(MalformedURLException me)
{System.out.println(me.getMessage());
 System.exit(0); }

catch(IOException e)
{System.out.println(e.getMessage());
 System.exit(0); }
}}
```

Для реалізації процесу завантаження інформації з мережі згідно об'єкту **URL** в пакеті **java.net** оголошений відповідний абстрактний клас **URLConnection**. У ньому зібрані методи, що керують процесом завантаження і його інформаційною підтримкою. Створити об'єкт **URLConnection** можна, використовуючи метод **openConnection()** класу **URL**.

URL myURL = new URL ("http://www.yahoo.com/");

URLConnection myCon = myURL.openConnection ();

Після цього буде доступний набір методів, які керують процесом завантаження. Ось деякі з них:

connect() – виконує з'єднання;

getContentLength() – повертає розмір завантаження об'єкта;

getContentType() – повертає тип вмісту об'єкта;

getDate() – повертає дату завантаження;

getExpiration() – повертає термін зберігання;

getPermission() – визначає параметри доступу;

getInputStream() – повертає потік введення InputStream завантажуваних даних.

2.2.3 Сокети і сокетне з’єднання

Сокети – це мережеві роз'єми, через які здійснюються двонаправлені потокові з'єднання між комп'ютерами. Сокет визначається номером порту та IP-адресою. При цьому IP-адреса використовується для ідентифікації комп'ютера, номер порту – для ідентифікації процесу, що працює на комп'ютері. Коли один додаток знає сокет іншого, створюється сокетне з'єднання. Клієнт намагається з'єднатися з сервером, ініціалізувавши сокетне з'єднання. Сервер чекає, поки клієнт зв'яжеться з ним. Перше повідомлення, що посилається клієнтом на сервер, містить сокет клієнта. Сервер в свою чергу створює сокет, який буде використовуватися для зв'язку з клієнтом, і посилає його клієнту з першим повідомленням. Після цього встановлюється комунікаційне з'єднання.

У Java в пакеті **java.net** визначені два класи **ServerSocket** і **Socket**, які реалізують програмування TCP/IP сокетів. Клас **ServerSocket** визначає серверне з'єднання і виконує функції прослуховування заданого порту, чекаючи з'єднання з клієнтом. З іншого боку, мережевий сокет **Socket** виконує функції клієнтського з'єднання, з його допомогою реалізується підключення до заданого адресою серверу, ініціалізація різних протоколів і передача або одержання даних.

При створенні об'єкта класу **Socket** вказується IP-адреса сервера і номер порту (80 для HTTP). Якщо вказано ім'я домену, то Java перетворить його за допомогою DNS-сервера до IP-адреси:

```
try {
Socket socket = new Socket("localhost", 8030);
} catch (IOException e) {
System.out.println("ошибка: " + e);
}
```

Сервер очікує повідомлення клієнта і повинен бути запущений із зазначенням певного порту. Об'єкт класу **ServerSocket** створюється конструктором із зазначенням номера порту і очікує повідомлення клієнта за допомогою методу **accept()**, який є блокуючим, тобто він буде чекати клієнта, щоб ініціалізувати зв'язок, і потім поверне Socket-об'єкт, який буде використовуватися для зв'язку з клієнтом:

```
try {
Socket s = null;
ServerSocket server = new ServerSocket(8030);
s = server.accept();
} catch (IOException e) {
System.out.println("ошибка: " + e);
}
```

Важливо розуміти, що клас **ServerSocket** визначає тільки процес прослуховування певного порту, а об'єкт Socket – процес передачі даних через нього.

Клієнт і сервер після встановлення сокетного зв’язку можуть отримувати дані з потоку введення і записувати дані в потік виводу за допомогою методів

getInputStream() і getOutputStream() або PrintStream для того, щоб програма могла трактувати потік як вихідні файли.

Для коректної роботи мережевих додатків, відкритих з використанням об'єктів **ServerSocket** і **Socket,** сокети в кінці роботи з ними необхідно закрити, використовуючи методи **close()**, оголошені в кожному з цих класів.

Організація серверного сокета

Конструктори класу **ServerSocket**

Конструктор	Опис
ServerSocket(int port)	Створює сокет сервера на зазначеному порту з довжиною черги за замовчуванням (50)
ServerSocket(int port, int maxQueue)	Створює сокет сервера на зазначеному порту з максимальною довжиною черги **maxQueue**[2]
ServerSocket(int port, int maxQueue, InetAddress localAddress)	Створює сокет сервера на зазначеному порту з максимальною довжиною черги **maxQueue.** На груповому[3] хост-комп'ютері localAddress визначає IP-адресу, з якою цей сокет пов'язаний.

Приклад. Сервер для прослуховування заданого порту і передачі через нього даних клієнта. У даному прикладі сервер посилає клієнтові рядок "привіт!", після чого розриває зв'язок.

```
import java.io.*;
import java.net.*;

public class MyServer{

public static void main(String[] args)throws Exception {
Socket s = null;
try { //посылка строки клиенту
ServerSocket server = new ServerSocket(2000);
System.out.println("Server running...");
s = server.accept();
PrintStream ps = new PrintStream(s.getOutputStream());
ps.println("привет!");
ps.flush();//Освобождает буффер
s.close(); // разрыв соединения
} catch (IOException e) {
System.out.println("ошибка: " + e);
```

[2] Довжина черги повідомляє системі, скільки підключень клієнта вона може залишати затриманими перш, ніж повинна буде просто відкинути з'єднання. За замовчуванням - 50

[3] Груповий хост – комп'ютер, який має декілька мережевих адаптерів або налаштований з декількома IP-адресами для одиночного мережевого адаптера.

```
}
}}
```

Організація клієнтського сокета

Методи і конструктори класу **Socket**

Конструктор/Метод	Опис
Socket(String hostName, int port)	Створює сокет, що з'єднує локальну хост-машину з іменованною хост-машиною і портом; може викидати виключення UnknownHostException або IOException
Socket(InetAddress ipAddress,int port)	Створює сокет, аналогічний попередньому, але використовується вже існуючий об'єкт класу InetAddress і порт; може викидати виключення IOException
InetAddress getInetAddress()	Повертає InetAddress-об'єкт, пов'язаний з Socket - об'єктом
int getPort()	Повертає віддалений порт, з яким з'єднаний даний Socket - об'єкт
int getLocalPort()	Повертає локальний порт, з яким з'єднаний даний Socket - об'єкт
InetAddress getLocalAddress()	Повертає адресу комп'ютера-клієнта.

При роботі з об'єктами **Socket** можуть виникати виняткові ситуації, для чого потрібно буде реалізувати відповідну обробку наступних виключень:

IOException – помилка вводу/виводу потоків;

SecurityException – помилка доступу до системи безпеки (якщо вона визначена на комп'ютері);

UnknownHostException – неправильно вказана адреса InetAddress.

Приклад. Програмування клієнта, який підключається до створеного раніше додатком серверу, зчитує дані і виводить їх на екран.

```
import java.io.*;
import java.net.*;

public class MyClient {

public static void main(String[] args) {

System.out.println("Client running...");
try {//получение строки клиентом

//Определяем адрес и порт подключения. Тестируем клиента и
сервера на одном //компьютере
InetAddress local = InetAddress.getLocalHost();
```

```
int port = 2000;

//Выполняем подключение
Socket socket = new Socket(local, port);

//Считываем поток данных по сети
BufferedReader dis = new BufferedReader(
new InputStreamReader(socket.getInputStream()));
String msg = dis.readLine();
System.out.println(msg);
socket.close();
dis.close();
} catch (IOException e) {
System.out.println("ошибка: " + e);
}
}
}
```

Аналогічно клієнт може послати дані серверу через потік виводу за допомогою методу **getOutputStream()**, а сервер може отримувати дані за допомогою методу **getInputStream()**.

Багатопоточність

Сервер повинен підтримувати багатопоточність, інакше він буде не в змозі обробляти кілька з'єднань одночасно. Сервер містить цикл, що очікує нового клієнтського з'єднання. Кожен раз, коли клієнт просить з'єднання, сервер створює новий потік. У наступному прикладі створюється клас **NetServerThread**, що розширює клас **Thread**. Сервер передає кожному новому підключеному клієнту його порядковий номер.

```
import java.net.*;
import java.io.*;
public class NetServerThread extends Thread {
Socket socket;
int nom; //номер подключения
PrintStream ps;
String msg;

public NetServerThread(Socket s, int nom) {
socket = s; this.nom=nom;
try {
ps = new PrintStream(s.getOutputStream());
} catch (IOException e) {
System.out.println("ошибка: " + e);
}
}
public static void main(String[] args) {
try {
ServerSocket server = new ServerSocket(2000);
System.out.println("Server running...");
int i=0;//счетчик подключений
while (true) {
```

```
Socket s = null;
s = server.accept(); i++;
NetServerThread nst = new NetServerThread(s, i);
nst.start();
}
} catch (IOException e) {
System.out.println("ошибка: " + e);
}
}
public void run() {
String msg = "сообщение: " + nom;
sendMessage(msg);
}

public void sendMessage(String msg) {
ps.println(msg);
System.out.println(msg + "<передача>");
ps.flush();}}
```

Для клієнтських додатків підтримка багатопоточності також необхідна. Наприклад, один потік очікує виконання операції вводу/виводу, а інші потоки виконують свої функції. Нижче наведено приклад програми, що реалізує отримання повідомлення клієнтом в потоці.

```
import java.net.*;
import java.io.*;

public class NetClientThread extends Thread {
BufferedReader br = null;
Socket s = null;

public NetClientThread() {
try {//соединение с кольцевым адресом
s = new Socket("127.0.0.1", 2000);
InputStreamReader isr =
new InputStreamReader (s.getInputStream());
br = new BufferedReader(isr);
} catch (IOException e) {
System.out.println("ошибка: " + e);
}}

public static void main(String[] args) {
NetClientThread nct = new NetClientThread();
nct.start();
}

public void run() {
while (true) {
try {
String msg = br.readLine();
if (msg == null) break;
else System.out.println(msg);
} catch (IOException e) {
```

```
System.out.println("ошибка: " + e);
}}}}
```

Не забудьте, що сервер повинен бути створений до того, як клієнт спробує здійснити сокетне з'єднання. При цьому може бути використана IP-адреса локального комп'ютера.

2.2.4 Програмування дейтаграм

Дейтаграми – невеликі за обсягом пакети даних, які передаються між комп'ютерами по мережі на основі протоколу UDP (User Datagram Protocol). Для роботи з дейтаграммами в пакеті java.net існують два класи **DatagramSocket** і **DatagramPacket**. Об'єкт **DatagramSocket** створює сам сокет, а дейтаграма являє собою об'єкт класу **DatagramPacket.**

У класі **DatagramSocket** є наступні конструктори:

DatagramSocket ();

DatagramSocket (int port);

DatagramSocket (int port, InetAddress addr);

де **port** – номер порту, **addr** – адреса підключення.

Методи класу **DatagramSocket**

Метод	Опис
getInetAddress()	Повертає адресу, до якої здійснюється підключення
getPort()	Повертає порт, до якого здійснюється підключення
getLocalAddress()	Повертає локальну адресу комп'ютера, з якої виконується підключення
getLocalPort()	Повертає локальний порт, через який здійснюється підключення
receive(DatagramPacket)	Чекає отримання дейтаграми і копіює дані в спеціальний DatagramPacket
send(DatagramPacket)	Посилає DatagramPacket
close()	Закриває сокет

Методами **send()** і **receive()**, можна виконувати відповідно відправку та отримання пакетів **DatagramPacket**. Однак попередньо потрібно оголосити екземпляри класу **DatagramPacket**, так як методи **send()** і **receive()** приймають параметр – об'єкт даного класу.

Конструктори класу **DatagramPacket**

для отримання пакета

DatagramPacket (byte [] buf, int length);

для відправки пакету

DatagramPacket (byte [] buf, int length, InetAddress address, int port);

де **buf** – масив байт, що представляють пакет дейтаграми,

length – розмір даного пакету,
address – Internet-адреса відправки,
port – порт комп'ютера, на який відправляється дейтаграмма.

Методи класу **DatagramPacket**

Метод	Опис
getData()	Повертає масив байт, що представляють собою вміст дейтаграми
setData()	Записує в пакет дані, отримуючи в якості параметрів байтові масиви
getAddress() і **setAddress()**	Повертає/встановлює адресу підключення
getPort() і **setPort()**	Повертає/встановлює порт підключення
getLength() і **setLength()**	Повертає/встановлює розмір дейтаграми

У процесі роботи з об'єктами дейтаграм можуть виникати такі виняткові ситуації:

SocketException – помилка протоколу при зверненні до сокета;
SecurityException – помилка доступу до системи безпеки;
UnknownHostException – невірно вказано адресу InetAddress.

2.2.5 Приклади деяких мережевих додатків

Програмна реалізація простої системи клієнт/сервер з прийомом/передачею дейтаграмм (за прикладом ISQ)

Нижче наводиться приклад, що демонструє діалог сервера з багатьма клієнтами в режимі "питання/відповідь", при якому всі сторони приймають і вводять повідомлення з терміналу. Роботу системи клієнт/сервер ініціалізував сервер, повідомляючи про свою готовність приймати дані. Серверний додаток DatagramServer створює свій сокет і пакет для прийому повідомлень від клієнтів.

Клієнтський додаток DatagramClient приймає від користувача повідомлення у вигляді рядка і передає його на сервер. Програма сервера відображує його на екрані, приймає і передає набрану на клавіатурі відповідь за зворотною адресою клієнта.

----------СЕРВЕР----------

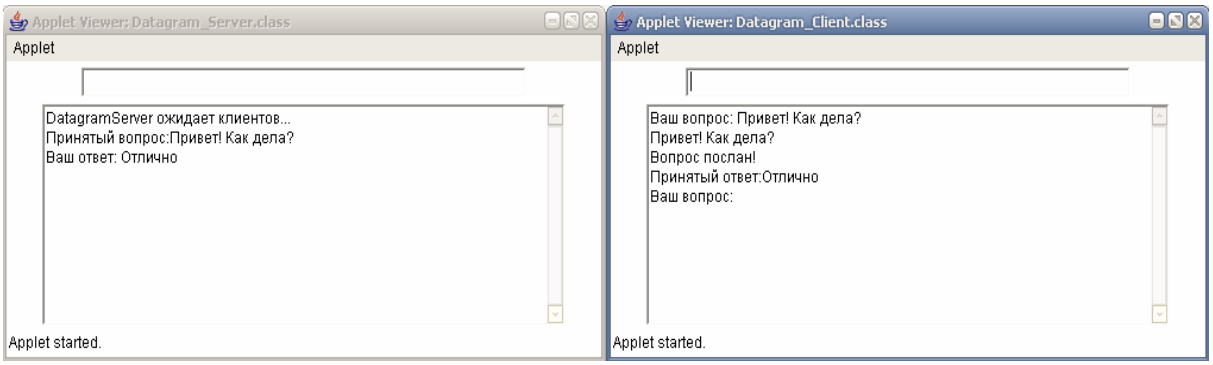
```
import java.net.*;
import java.io.IOException;
/* Класс серверного приложения, реализующего диалог с
множеством клиентов в режиме "вопрос/ответ".
*/
```

```
public class DatagramServer
{
/* Создает DatagramSocket с заданным номером порта для
получения вопросов от клиентов. Полученный вопрос выводит на
терминал для получения ответа, который тут же отсылается клиенту в
том же пакете, в котором был принят.
*/
public static void main(String[] args)
{
try {
DatagramSocket mysock = new DatagramSocket(1432);
byte[] buf = new byte[1024];
DatagramPacket p = new DatagramPacket(buf, buf.length);
System.out.println("DatagramServer ожидает клиентов...");
while(true)
{
int count;
mysock.receive(p);
System.out.print("Принятый вопрос:");
for(count=0; (char)buf[count]!='\n'; count++);
System.out.println(new String(buf,0,count));
System.out.print("Ваш ответ: ");
try{ count=System.in.read(buf);
} catch(IOException e)
{System.out.println("to-"+e.toString());}
mysock.send(p);
}
} catch ( Exception e ) { e.printStackTrace(); }
}
}
```

----------КЛІЕНТ----------

```
import java.io.IOException;
import java.net.*;
/* Класс DatagramClient клиентского приложения, задающего
вопросы серверу DatagramServer и получающего от него ответы.
*/
public class DatagramClient
{
/* Создает DatagramSocket, не указывая его порт. Для отправки
сообщений-вопросов и приема ответов использует один и тот же
DatagramPacket. Посылает вопрос и ждет ответа.
*/
public static void main(String[] args)
{
try {
DatagramSocket mysock = new DatagramSocket();
byte[] buf = new byte[1024];
int count=0;
DatagramPacket p = new DatagramPacket(buf, buf.length,
InetAddress.getLocalHost(), 1432);
while(true){
System.out.print("Ваш вопрос: ");
```

```
try{ count=System.in.read(buf);
} catch(IOException e)
{ System.out.println("This is "+e.toString()); }
mysock.send(p); // посылка дейтаграммы
System.out.println("Вопрос послан!");
mysock.receive(p);
for(count=0; (char)buf[count]!='\n'; count++);
 System.out.print("Принятый ответ:");
 System.out.println(new String(buf,0,count));
 }
}catch(Exception e) { e.printStackTrace(); }
}
}
```



Програмна реалізація простої системи клієнт/сервер з прийомом / передачею дейтаграм (за прикладом ISQ)

Нижче наводиться приклад, що демонструє діалог сервера з багатьма клієнтами в режимі "питання/відповідь", при якому всі сторони приймають і вводять повідомлення з терміналу.

Роботу системи клієнт/сервер инициализирует сервер, повідомляючи про свою готовність приймати дані. Серверний додаток DatagramServar створює свій сокет і пакет для прийому повідомлень від клієнтів.

Клієнтське додаток DatagramClient приймає від користувача повідомлення у вигляді рядка і передає його на сервер. Програма сервера відображає його на екрані, приймає і передає набраний на клавіатурі відповідь за зворотною адресою клієнта.

----------СЕРВЕР----------

```
import java.net.*;
import java.io.IOException;
/* Класс серверного приложения, реализующего диалог с
множеством клиентов в режиме "вопрос/ответ".
*/
public class DatagramServer
{
```

```
    /* Создает DatagramSocket с заданным номером порта для
получения вопросов от клиентов. Полученный вопрос выводит на
терминал для получения ответа, который тут же отсылается клиенту в
том же пакете, в котором был принят.
    */
    public static void main(String[] args)
    {
    try {
    DatagramSocket mysock = new DatagramSocket(1432);
    byte[] buf = new byte[1024];
    DatagramPacket p = new DatagramPacket(buf, buf.length);
    System.out.println("DatagramServer ожидает клиентов...");
    while(true)
    {
    int count;
    mysock.receive(p);
    System.out.print("Принятый вопрос:");
    for(count=0; (char)buf[count]!='\n'; count++);
    System.out.println(new String(buf,0,count));
    System.out.print("Ваш ответ: ");
    try{ count=System.in.read(buf);
    } catch(IOException e)
    {System.out.println("to-"+e.toString());}
    mysock.send(p);
    }
    } catch ( Exception e ) { e.printStackTrace(); }
    }
    }
```

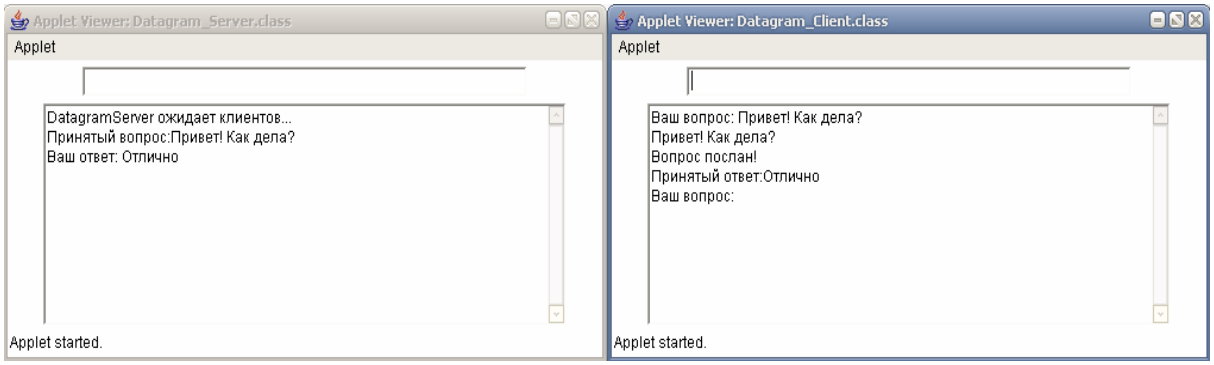
----------КЛІЄНТ----------

```
    import java.io.IOException;
    import java.net.*;

    /* Класс DatagramClient клиентского приложения, задающего
вопросы серверу DatagramServer и получающего от него ответы.
    */
    public class DatagramClient
    {
    /* Создает DatagramSocket, не указывая его порт. Для отправки
сообщений-вопросов и приема ответов использует один и тот же
DatagramPacket. Посылает вопрос и ждет ответа.
    */
    public static void main(String[] args)
    {
    try {
    DatagramSocket mysock = new DatagramSocket();
    byte[] buf = new byte[1024];
    int count=0;
```

```
DatagramPacket p = new DatagramPacket(buf, buf.length,
InetAddress.getLocalHost(), 1432);
while(true){
System.out.print("Ваш вопрос: ");
try{ count=System.in.read(buf);
} catch(IOException e)
{ System.out.println("This is "+e.toString()); }
mysock.send(p); // посылка дейтаграммы
System.out.println("Вопрос послан!");
mysock.receive(p);
for(count=0; (char)buf[count]!='\n'; count++);

System.out.print("Принятый ответ:");
System.out.println(new String(buf,0,count));
}
}catch(Exception e) { e.printStackTrace(); }
}
}
```



Приклад файлового сервера, обслуговуючого декілька клієнтів

Наведемо приклад файлового сервера, який посилає своїм клієнтам файли. Два класи, FileServer і StartServer, представляють сторону сервера. Сторона клієнта представлена додатком FileClient. Перший, FileServer, призначений для обслуговування одного клієнта через переданий йому сокет, другий – для очікування клієнтів і створення для кожного з них обслуговуючого FileServer. Втім, клас FileServer може запускатися автономно для обслуговування одного клієнта.

Протокол обміну інформацією наступний. Коли клієнт з'єднується з сервером, останній шле йому рядок зі своїм ім'ям і чекає запиту на файл. Потім клієнт повинен послати серверу або ім'я текстового файлу, яким він цікавиться, або повідомлення "QUIT", якщо він бажає закрити з'єднання. У першому випадку сервер намагається знайти і прочитати файл. Якщо ця операція закінчилася успішно, сервер шле клієнту рядок з його змістом і потім чекає новий запит; в іншому випадку клієнт отримає повідомлення "NOT FOUND". З отримання "QUIT" сервер посилає клієнтові, а той приймає прощальне повідомлення.

```
// Файл FileServer.java

import java.io.*;
import java.net.*;

/* Приложение FileServer, реализующее файловый сервер.
Получает связанный с клиентом сокет и обслуживает запросы этого
клиента на файлы сервера. Класс содержит статический метод main
для автономного запуска FileServer для одного клиента.
*/

public class FileServer extends Thread {
PrintStream ps=null;
DataInputStream dis=null;
ServerSocket servSock=null;
Socket sock=null;
String myName = "FILE_SERVER";
String outStr = null;
String file;
int clientNo;

/* Автономный запуск сервера для одного клиента. Создает
ServerSocket и ожидает соединения с клиентом. Как только клиент
соединяется, получает связанный с ним сокет и создает на нем
экземпляр класса для обслуживания этого клиента.
*/

public static void main(String[] args) {
try { ServerSocket sSoc = new ServerSocket(257);
Socket soc = sSoc.accept();
FileServer srv = new FileServer(soc,1);
srv.start();
} catch(IOException ex)
{ System.out.println("in start "+ex); }
}

/* Конструктор класса. Принимает и адаптирует связанный с
клиентом сокет.
*/
public FileServer(Socket soc,int n) {
sock=soc;
clientNo=n;
adaptConnection();
}

/* Главный цикл потока приложения FileServer. Ожидает запрос
на файл от своего клиента. Выходит из цикла и закрывает связь,
если вместо имени файла получает сообщение QUIT. Если находит файл
с данным именем, читает его и посылает клиенту.
*/
public void run() {
while((file = readRequest())!= null)
```

```
     { if(file.equalsIgnoreCase("QUIT"))
     { sendToClient("BYE"); break; }
     System.out.println("Запрос клиентом №"+clientNo+ " файла
'"+file+"'");
     if(loadFile()){ sendToClient(outStr);
     System.out.println("файл послан."); }
     else { System.out.println("файл не найден.");
     sendToClient("NOT FOUND"); }
     }
     closeConnection();
     }

     /* Связывает с принятым соединением потоки ввода/вывода.
Посылает клиенту первое сообщение - свое собственное имя.
     */
     public void adaptConnection() {
     try{ ps = new PrintStream(sock.getOutputStream());
     dis = new DataInputStream(sock.getInputStream());
     } catch(Exception e) {
     System.out.println("При открытии сервера: "+e);
     System.exit(1); }
     ps.println(myName); ps.flush();
     System.out.println("Сервер связался с клиентом №"+clientNo);
     }

     /* Закрывает связь с клиентом. */
     public void closeConnection() {
     try{ System.out.println("Закрытие связи с клиентом № "
     +clientNo); if(ps!=null) ps.close();
     if(dis!=null) dis.close();
     if(sock!=null) sock.close();
     } catch(Exception e) {
     System.out.println("При закрытии сервера: "+e); }
     }

     /* Посылает клиенту сообщение, которое может быть либо
приветствием или прощанием, либо содержанием текстового файла.
     */
     public void sendToClient(String outs) {
     try{ ps=new PrintStream(sock.getOutputStream());
     ps.println(outs); ps.flush(); }
     catch(Exception e)
     { System.out.println("При отправке данных: "+e);
     System.exit(1); }
     }

     /* Читает запрошенный текстовый файл в строку, отсылаемую
клиенту. */
     public boolean loadFile() {
     String line=null;
     try{ DataInputStream dis = new DataInputStream(new
FileInputStream(file));
```

```
     for(outStr=""; (line = dis.readLine())!=null; )
     outStr += line + "\n";
     dis.close();
     } catch(Exception e) {
     System.out.println("При чтении файла: "+e);
     return false; }
     return true;
     }

     /* Ожидание запроса на файл от своего клиента - чтение строки
из потока ввода сокета.
     */
     public String readRequest() {
     String req=null;
     System.out.println("Прием запроса... ");
     try{ req=dis.readLine();
     } catch(IOException e)
     { System.out.println("При приеме запроса: "+e);
     return null; }
     return req;
     }
     }

**********************************************************************
     // файл StartServer.java
     import java.awt.*;
     import java.awt.event.*;
     import java.net.*;
     import java.io.*;

     /* Класс для открытия гнездовых соединений с клиентами
приложения FileServer. Открывает ServerSocket и ждет клиентов. Как
только клиент соединяется, создает для него FileServer и ждет
новых клиентов.
     */

     public class StartServer extends Frame implements Runnable,
ActionListener
     {
     ServerSocket sSoc;
     int clientNo=0;
     int port=257;
     Button close;

     /* Метод автономного запуска потока ожидания клиентов.
     */
     public static void main(String args[])
     {
     new StartServer();
     }
```

```
     /* Конструктор класса StartServer. Запускает поток ожидания
клиентов.
     */
     public StartServer() {
     super("File Server 'FILE_SERVER'");
     System.out.println("FILE_SERVER ожидает клиентов...");
     close = new Button("close");
     add(close); close.addActionListener(this);
     resize(200,100);
     show();
     new Thread(this).start();
     }

     /* Обработчик событий. Служит для закрытия сервера через окно
приложения.
     */
     /*public boolean handleEvent(Event ev) {
     if(ev.id==Event.WINDOW_DESTROY || ev.arg=="close")
     { System.out.println("FILE_SERVER закрывается.");
     System.exit(0); return true;
     }
     return false;
     }
     */
     public void actionPerformed(ActionEvent e){
     if(e.getSource()==close)
     { System.out.println("FILE_SERVER
закрывается.");System.exit(0); }
     }

     /* Цикл ожидания и соединения с клиентами. Как только клиент
соединяется, принимает Socket и создает для него новый поток
FileServer.
     */
     public void run() {
     try {
     sSoc=new ServerSocket(port);
     while(true)
     {
     Socket soc = sSoc.accept();
     clientNo++;
     FileServer srv = new FileServer(soc,clientNo);
     srv.start();
     }
     } catch(IOException ex) { System.out.println("in start "+ex);
}
     }
     }

**********************************************************************

     // FileClient.java
```

```
import java.io.*;
import java.net.*;

/* Приложение клиентской стороны сервера файлов. Связывается с
сервером, называя его порт и адрес и принимая имя. Передает
серверу имя текстового файла и принимает его содержимое.
*/
public class FileClient {
static String srvName;
public Socket sock = null;
private DataOutputStream dos = null;
private DataInputStream dis = null;
private String file = null;
private StringBuffer servAnswer = null;
private int port=257;
private boolean connectOK=false;

/* Автономный запуск клиента файлового сервера.
*/
public static void main(String args[]) {
srvName=args[0];
FileClient client=new FileClient();
}

/* Конструктор клиента. Он же реализует весь процесс общения с
сервером. Посылает серверу либо QUIT, либо имя требуемого файла,
запрошенное с терминала клиента.
*/
public FileClient() {
String name=connectServer();
if(!connectOK) System.exit(1);
System.out.println("Connected with "+name);
do { file = keyBoard("Введите QUIT или имя файла: ");
send(file); // посылка запроса серверу
receiveData(); // прием ответа
System.out.println(servAnswer); // вывод ответа на экран
if(file.equalsIgnoreCase("QUIT")) break;
} while(true);
closeConnect(); // отключение от сервера
}

/* Устанавливает связь с сервером через сокет. Связывает с
сокетом потоки ввода/вывода. Читает из потока ввода имя сервера.
*/
public String connectServer() {
String name=null;
try{ sock = new Socket(srvName,port);
dis=new DataInputStream(sock.getInputStream());
dos=new DataOutputStream(sock.getOutputStream());

name=dis.readLine(); }
catch(IOException e) {
```

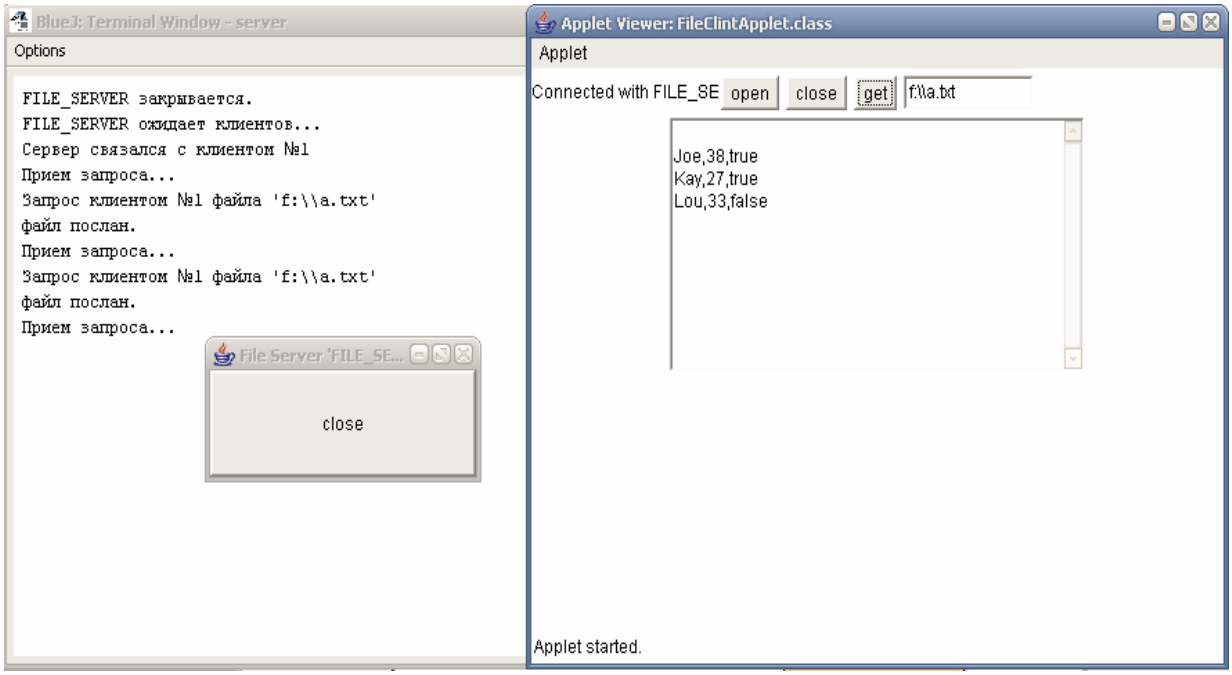
```
System.out.println("При попытке связаться с сервером: "+e);
System.exit(1); }
connectOK=true;
return name;
}

/* Шлет запрос серверу в виде текстовой строки.
*/
public void send(String str) {
try{ dos.writeBytes(str+"\n"); dos.flush(); }
catch(IOException e)
{ System.out.println("При отправке запроса: "+e);
System.exit(1); }
}

/* Получает сообщение от сервера в StringBuffer.
*/
public void receiveData() {
char r=' '; int c;
servAnswer = new StringBuffer();
try{ do { servAnswer.append(r);
c=dis.read(); r=(char) c; } while(r!='\r' ); }
catch(Exception e)
{ System.out.println("При приеме данных: "+e); }
}

/* Закрывает соединение с сервером.
*/
public void closeConnect() { String serverBye=null;
try{ dos.close(); dis.close(); sock.close(); }
catch(Exception e) { System.out.println("При выходе: "+e); }
}

/* Принимает с клавиатуры запросы клиента.
*/
public String keyBoard(String quest) {
System.out.print(quest);
System.out.flush();
String ans=null;
try{ DataInputStream dataIn = new DataInputStream(System.in);
ans=dataIn.readLine(); }
catch(IOException e)
{ System.out.println("При вводе запроса: "+e); return null; }
return ans;
}
}
```



2.3 Контрольні питання

1. Для чого призначений і яке значення повертає метод getAllByName() класу java.net.InetAddress?
2. Поясните, в чому полягає відмінність між методами getByName(null) і getLocalHost()?
3. Як отримати вміст сторінки, використовуючи її URL?
4. Які дії необхідно зробити для встановлення TCP з'єднання між двома java-додатками?
6. Які дії необхідно зробити для обміну даними по UDP протоколу?

3 Опис приладів, устаткування та інструментів

– Обладнання: IBM-сумісний персональний комп’ютер (ПК).
– Програмне забезпечення: операційна система Windows, Java 2 SDK версії 1.5 та вище. Інтегроване середовище розробки IDE Eclipse.

4 Правила техніки безпеки та охорони праці

Правила техніки безпеки при виконанні лабораторної роботи регламентуються «Правилами техніки безпеки при роботі в комп'ютерній лабораторії».

5 Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

– контроль рівня підготовленості студентів до виконання роботи у формі усного опитування;

– інструктаж по правилах охорони праці перед початком лабораторної роботи та оформлення його підсумків в журналі проведення лабораторних робіт, який ведеться у навчальній лабораторії;

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java у інтегрованому середовищі розробки Eclipse.

5.1 Завдання до лабораторної роботи

Розробити додаток, що дозволяє здійснювати взаємодію клієнта і сервера по спільному вирішенню завдань обробки інформації. Додаток повинен мати:

1) Призначений для користувача інтерфейс, що визначає можливості додатка;
2) Можливість передачі і модифікування даних;
3) Засоби діагностики і обробки виняткових ситуацій. Інформація на сервері, яка використовується в окремих варіантах завдання, повинна зберігатися на диску у вигляді текстових, гіпертекстових, графічних, табличних .java або .html файлів і/або в масивах.

5.2 Варіанти

1. «Chat». Два і більше клієнтів спілкуються через сервер. Сервер, чекає підключення клієнтів, а потім передає дані від одного клієнта іншому клієнтові.
2. Розробити програму організації простих розрахунків на сервері для клієнтських завдань (наприклад, вирішення квадратного рівняння). Використовувати TCP – сервіс.
3. Розробити програму організації простих розрахунків на сервері для клієнтських завдань (наприклад, вирішення квадратного рівняння). Використовувати UDP – сервіс.
4. Розробити програму, в якій організувати взаємодію, що реалізовує розсилку повідомлень від одного клієнта групі клієнтів. Використовувати TCP – сервіс.
5. Розробити програму, в якій організувати взаємодію, що реалізовує розсилку повідомлень від одного клієнта групі клієнтів. Використовувати UDP – сервіс.
6. Розробка HELP по Java на Java. Клієнт запрошує термін, сервер знаходить відповідь в help-файлі і пересилає його.
7. Організувати видалене виконання неінтерактивних програм. Клієнт пересилає серверу набрану на клавіатурі команду на виконання програми з параметрами. Сервер повинен виконати програму, помістити результати у файл і передати клієнтові, який виводить прийнятий файл на екран.
8. Веб-браузер: реалізувати у вигляді графічного додатка. Веб-браузер повинен відображувати html-сторінки з вказаного сервера.

9. Клієнт вибирає зображення із списку і пересилає його іншому клієнтові через сервер.
10. Сервер розсилає повідомлення в певний час вибраним із списку клієнтам. Список зберігається у файлі.
11. "Пакети Java": у відповідь на запит конкретного пакета (наприклад, java.io або java.util) сервер повинен передавати графічний файл (jpg або gif) з ієрархічною структурою пакета.
12. «ISQ». Сервер спілкується з багатьма клієнтами в режимі "питання/ відповідь". Використовувати UDP - сервіс.

6 Порядок обробки результатів експериментів та вироблення висновків

На підставі отриманих результатів лабораторної роботи зробить висновки щодо особливостей створення сокетного з'єднання в Java. В висновках до лабораторної роботи окремо опишіть етапи створення багатопотокового сервера.

7 Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:
- титульний лист, де вказані номер і назва лабораторної роботи, відомості про виконавця;
- номер варіанта роботи та текст завдання;
- відповіді на контрольні запитання до лабораторної роботи;
- листинг програмного коду;
- результати виконання програми;
- висновки по підсумках роботи.

Підготовлений звіт надається викладачу на перевірку та захищається студентом на останьому занняті з даної лабораторної роботи.

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни “Крос-платформне програмування”
для студентів IV курсу денної форми навчання
спеціальності 122 «Комп’ютерні науки»

Викладач: доц. Кузніченко С.Д.

Одеський державний екологічний університет,
65016, м. Одеса, вул. Львівська, 15