ЗМІСТ

ГОЛОСАРІЙ

ІС – інформаційна система

СРС – самостійна робота студентів

ЗМ-Л – змістовний модуль лекційний

ЗМ-П – змістовний модуль практичний

ІЗ – індивідуальне завдання

ОЗЕ – заліково-екзаменаційної сесії

ПО – накопичена підсумкова оцінка

ОЗЕ – кількісна оцінка заходів контролю СРС під час проведення аудиторних занять

ОМ – кількісна оцінка заходів контролю СРС у міжсесійний період

ОЗКР - оцінка залікової контрольної роботи

ЗКР – залікова контрольна робота

ВСТУП

Мета методичних вказівок до виконання самостійної роботи студентів (СРС) полягає в допомозі студентам заочної форми навчання в самостійній роботі при вивчені курсу «Крос-платформне програмування», та виконанні індивідуального завдання в вигляді контрольної роботи.

При виконанні індивідуального завдання закріплюються знання, отримані на лекціях та лабораторних роботах, де студенти вивчають особливості розробки крос-платформних програмних систем та використання засобів крос-платформного програмування. Окремо розглядаються класифікація крос-платформних мов програмування; середовища розробки крос-платформного програмного забезпечення та бібліотеки для створення платформо-незалежного програмного забезпечення. В якості мови програмування для виконання лабораторних робіт пропонується сучасна крос-платформна мова програмування Java.

Самостійна робота студента з дисципліни «Крос-платформне програмування» включає:

- підготовку до лекційних та лабораторних занять;
- підготовку до заходів поточного та підсумкового контролю;
- написання індивідуальної роботи;
- підготовка до залікової контрольної роботи.

В загальній частині методичних вказівок наведені мета і задачі курсу, які відповідають робочій програмі, перелік тем лекцій та лабораторних робіт. Дається перелік основної та додаткової навчальної літератури, а також базова компонента – перелік знань та вмінь якими повинен володіти студент після засвоєння даної дисципліни.

В розділі «Організація самостійної роботи по виконанню завдань на СРС» міститься:

- перелік завдань на самостійну роботу, які передбачені навчальним планом та програмою курсу;
- повчання по послідовному вивченню теоретичного матеріалу даної дисципліни;
- вказівки до виконання індивідуальної роботи.

1 ЗАГАЛЬНА ЧАСТИНА

Навчальна дисципліна «Крос-платформне програмування» є обов’язковою дисципліною у бакалаврській підготовці за напрямком "Комп’ютерні науки", належить до циклу професійної та практичної підготовки і базується на знанні дисциплін: «Алгоритмізація та програмування», «Об’єктно-орієнтоване програмування», «Операційні системи», «Комп’ютерні мережі».

Метою дисципліни «Крос-платформне програмування» є теоретичне вивчення та отримання практичних навичок розробки крос-платформних програмних систем та використання засобів крос-платформного програмування.

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.

У методичних вказівках розглядаються питання, які відповідають навчальній програмі дисципліни.Методичні вказівки містять рекомендації по вивченню розділів дисципліни, контрольні запитання та завдання. Окремі завдання контрольної роботи підкріплені прикладами розв’язання типових задач на ПЕОМ.

Отримані студентами знання та вміння використовуються при виконанні розрахунково-графічних робіт, курсовому та дипломному проектуванні, у низці магістерських дисциплін.

1.1 Програма лекційного модулю

ЗМ-Л1. РОЗРОБКА ДОДАТКІВ З GUI МОВОЮ JAVA

Компонентна ідеологія [1, ст.8-17]

Визначення та властивості компонентів [1, ст.18-21]

Специфікації інтерфейсу як контракту [1, ст.22-38]

Модель посилань [3, ст.275-287]

Стратегії інтеграції програмного забезпечення [1, ст.48-57]

Багатопотокове програмування [3, ст.328-350]

Використання потоків і створення анімації в Java [3, ст.88-97]

Синхронізація потоків [3, ст.328-350]

Переривання потоків [3, ст.328-350]

ЗМ-Л2. АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ КОМПОНЕНТНИХ СИСТЕМ

Архітектура та проектування компонентних систем [1, ст.18-27]

Розробка та збирання компонентів [1, ст.218-224]

Розподілена архітектура компонентних систем [1, ст.225-239]

Компонентно-орієнтоване проектування [1, ст.168-179]

Потоки введення/виведення і колекції в Java [3, ст.205-210]

Програмування потоків введення/виведення [3, ст.211-217]

Шаблони Adapter і Decorator [3, ст.220-229]

Програмування колекцій в Java [1, ст.229-259]

Розробка мережевих програм [3, ст.350-355]

Використання сокетів у розподілених додатках [3, ст.356-359]

Датаграми і протокол UDP [3, ст.360-364]

1.2 Програма практичного модулю

ЗМ-П1:

1. Програмування апплетів з елементами мультимедіа для Web сторинок Internet.
2. Використання потоків і створення анімації в Java.

ЗМ-П2:

1. Створення графічних додатків. Використання системи Swing в Java
2. Створення графічних застосувань. Програмування потоків введення/виведення.

ЗМ-П3:

1. Програмування колекцій в Java
2. Використання сокетів у розподілених додатках

Для виконання самостійної роботи потрібено встановити на ПЕОМ мову програмування Java 2 SDK версії 1.5 та вище та інтегроване середовище розробки IDE Eclipse.

1.3 Знання та вміння

Базові знання та вміння, якими має оволодіти студент для написання залікової контрольної роботи.

За результатами вивчення ЗМ-Л студент повинен ЗНАТИ:

- базові принципи створення крос-платформних програмних систем;
- класифікація крос-платформних мов програмування;
- середовища розробки крос-платформного програмного забезпечення;
- бібліотеки для створення платформо-незалежного програмного забезпечення;
- визначення та властивості компонентів;
- специфікації інтерфейсу як контракту;
- модель посилань;
- стратегії інтеграції програмного забезпечення;
- використання потоків і створення анімації в Java;
- синхронізація потоків;
- переривання потоків;
- розробка та збирання компонентів;
- розподілена архітектура компонентних систем;
- компонентно-орієнтоване проектування;
- програмування потоків введення/виведення;
- ієрархія класів бібліотеки java.io;
- шаблони Adapter і Decorator;
- ієрархія класів бібліотеки java.util;
- компоненти і інтерфейси колекцій;
- програмування колекцій в Java;
- використання сокетів у розподілених додатках;
- датаграми і протокол UDP.

За результатами вивчення ЗМ-Л студент повинен ВМІТИ:

- проектувати компоненти програмного забезпечення;
- проектувати крос-платформного графічного інтерфейсу користувача;
- реалізовувати прототипи архітектури програмного забезпечення;
- інтегрувати компоненти в систему;
- установлювати, налаштовувати та обслуговувати системне, інструментальне і прикладне програмне забезпечення та інформацій-

ні системи.

- запускати, виконувати та передавати параметри аплету;
- використовувати мультимедіа в аплетах – розміщення графічних об'єктів і зображень.
- створювати потоки в Java;
- проектувати графічний інтерфейс користувача інформаційних систем;
- обробляти семантичні події;
- використовувати для розміщення компонентів менеджери компонування.
- організовувати взаємодію з файловою системою комп’ютера;
- організовувати потоки вводу/виводу в Java.
- вибирати і реалізовувати класи-колекції для вирішення завдань;
- використовувати колекцій при розробці java-додатків;
- розробляти застосування з архітектурою «клієнт-сервер» і управляти мережевими з'єднаннями на рівні сокетів;
- управляти з'єднаннями на рівні дейтаграм;
- створювати багатопотоковий сервер.

ЗМ-П. В результаті засвоєння матеріалу змістовного модуля студент повинен ВМІТИ:

- запускати, виконувати та передавати параметри аплету;
- виконувати обробку низькорівневих подій;
- створювати потоки використовуючи розширення класу Thread та за допомогою інтерфейсу Runnable;
- створювати анімацію з використанням подвоєної буферизації зображення;
- проектувати графічний інтерфейс користувача інформаційних систем;
- обробляти семантичні події;
- використовувати для розміщення компонентів менеджери компонування;
- розробляти вимоги та специфікації компонентів інформаційних систем і об’єктів професійної діяльності;
- організовувати взаємодію з файловою системою комп’ютера;
- організовувати потоки вводу/виводу в Java;

− реалізовувати взаємодію клієнта та сервера, використовуючи сокети TCP і UDP.

Інструментальні компетенції:

– Професійне володіння комп'ютером та інформаційними технологіями.

Загальнопрофесійні компетенції:

– Ґрунтовна підготовка в області програмування, володіння алгоритмічним мисленням, методами програмної інженерії для реалізації програмного забезпечення з урахуванням вимог до його якості, надійності, виробничих характеристик.

Спеціалізовано-професійні компетенції:

– Знання архітектури та стандартів компонентних моделей, комунікаційних засобів і розподілених обчислень, здатність вирішувати проблеми масштабованості, підтримки віддалених компонентів і взаємодії різних програмних платформ в розподілених інформаційних системах рівня підприємства.

1.4 Перелік навчально-методичної літератури

Перелік навчально-методичної літератури з дисципліни «Крос-платформне програмування», що є в наявності у бібліотеці університету, або в електронному виді, або на кафедрі інформаційних технологій:

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. Методичні вказівки для виконання лабораторних робіт по дисципліні Крос-платформне програмування. Одеса: ОДЕКУ, 2015. – 153 с.

3. *Блинов И. Н.* Java. Промышленное программирование: практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн: УниверсалПресс, 2007. – 768 с.

4. *Портянкин И.*, Swing эффективные пользовательские интерфейсы.– Питер, 2005. – 523 с.

5. *Официальный сайт Oracle.* The Java Tutorials [электронный ресурс]. Режим доступа: http://docs.oracle.com/javase/tutorial/index.html

6. *Хорстманн, Кей* С., Корнелл, Гари. Java2. Библиотека профессионала, том1. Основы, 7-е изд.: Пер. с англ. – М.:Издательский дом «Вильямс», 2007. − 896 с.

7. *Хорстманн, Кей* С., Корнелл, Гари. Java2. Библиотека профессионала, том2. Тонкости программирования, 7-е изд.: Пер. с англ. – М.:Издательский дом «Вильямс», 2007. − 626 с.

8. Електрона бібліотека ОДЕКУ www.library-odeku.16mb.com

Додаткова література

1. *Васильев А. Н.* Java. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2011. – 400 с.

2. *Джошуа Блох.* Java.Эффективное программирование =Effective Java. – М.: Лори, 2002. – 224 с.

3. *Гранд М.* Шаблоны проектирования в Java / пер. с англ. С. Беликовой. – М.: Новое знание, 2004.–. 559 с.

4. *Гамма Э.*, Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. - СПб.: Питер, 2001.-368с.

5. *Ноутон П.*, Шилдт Г. Java 2 в подлиннике. – СПб.: БХВ-Петербург, 2007.– 1072 с.: ил.

6. *Макконнелл С.* Совершенный код. Практическое руководство по разработке программного обеспечения. 2-е изд. / Пер. с англ.- М.: Русская редакция, СПб.: Питер, 2005.- 896 с.: ил.

7. *Дейтел Х. М.* Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.

8. *Дейтел Х. М.* Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.

9. *Дэвид М. Гери*, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М.: Издательский дом "Вильямс", 2011. – 544 с.

10. *Эккель Б.* Философия Java. Библиотека программиста. / Б. Эк-кель. – 4-е изд. – СПб. : Питер, 2009. – 640 с.: ил.

2 ОРГАНІЗАЦІЯ САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ

Основною формою навчання студента є самостійна робота (СРС) над навчальним матеріалом. СРС повинна сприяти активізації творчого мислення студентів, формуванню високої культури розумової праці, підвищенню самостійності студентів та індивідуалізації процесу навчання. СРС складається з наступних елементів: вивчення матеріалу по підручниках, конспектах лекцій, виконання лабораторних та контрольних робіт. Студент може звертатися до викладача з питаннями для одержання письмової чи усної консультації. Однак студент повинен пам'ятати, що тільки при систематичній і завзятій самостійній роботі допомога викладача виявиться досить ефективною.

Загальні поради:

– зміст кожної теми курсу вивчається за допомогою наведеного у розділі І переліку навчальної та методичної літератури, в першу чергу використовується основна та методична література.

– після засвоєння змісту кожної теми курсу треба відповісти на "Питання до самоконтролю", які наведені у цих методичних вказівках.

– завдання з контрольної роботи виконувати згідно з наведених у розділі 3 вимог.

– якщо виникли питання при вивченні теоретичного матеріалу або при виконанні контрольної роботи, можна звернутися до викладача, який читав установчі лекції, письмово на адресу університету звичайною або електронною поштою: kaf-inf@ogmi.farlep.odessa.ua або dean-comp@ogmi.farlep.odessa.ua.

2.1 Рекомендації по вивченню теоретичного матеріалу

При вивченні теоретичного матеріалу студент повинен здобути знання по темах:

ТЕМИ ЗМ-Л1. *«Розробка додатків з GUI мовою Java»*
При вивченні тем ЗМ-Л1 студент повинен звернути увагу на питання.
Питання до самоконтролю:

1. Принципи створення аплетів Java. Запуск і виконання аплетів. Передача параметрів аплету [1, ст.8-12].

2. Виведення графічного контексту. Використання кольорів [4, ст.212-232].
3. Виведення тексту. Класс StringTokenizer [4, ст.237-245].
4. Стратегії інтеграції програмного забезпечення [1, ст.48-57].
5. Створення графічних додатків. Вставка зображення в графічний додаток.Вставка зображень в аплет [4, ст. 317-320].
6. Багатопотокове програмування Реалізація потоків в Java [3, ст.328-350].
7. Пріоритети та групи потоків [3, ст.332-338].
8. Синхронізація потоків. Концепція Monitor [3, ст.338-347].
9. Способи переривання потоків [3, ст.347-350]
10. Використання потоків і створення анімації в Java. Клас MediaTracker [4, ст.308-315].
11. Подвійна буферизація зображень [4, ст.318-322].

ТЕМИ ЗМ-Л2. *«Архітектура та проектування компонентних систем»*
При вивченні тем ЗМ-Л2 студент повинен звернути увагу на питання.
Питання до самоконтролю:

1. Модель делегування подій в Java [3, ст.275-287].
2. Обробка низькорівневих та семантичних подій [3, ст.278-284].
3. Блоки прослуховування. Класи-адаптери [3, ст.275-278].
4. Основні компоненти AWT і Swing [4, ст.201-226].
5. Контейнери верхнього рівня і спеціалізовані контейнери [4, ст.308-317].
6. Програмування потоків введення/виведення [3, ст.205-210].
7. Робота з файлами в Java. Клас File [1, ст.8-17].
8. Байтові і символьні потоки вводу/виводу [3, ст.118-127].
9. Компоненти і інтерфейси колекцій [1, ст.229-259].
10. Реалізації колекцій і алгоритми [1, ст.229-235].
11. Розробка мережевих програм [3, ст.350-355].
12. Класи бібліотеки java.net [1, ст.8-17].
13. Використання сокетів у розподілених додатках [3, ст.360-364].
14. Програмування дейтаграм. Протокол UDP [3, ст.360-364].

.

2.2 Рекомендації по розробці Індивідуального завдання

Основною формою індивідуальної роботи по дисципліні «Крос-платформне програмування» є виконання індивідуального завдання (ІЗ) з теоретичного та практичного модулю, яке студенти виконують самостійно під керівництвом викладача.

ІЗ виконується з метою систематизації закріплення, поглиблення та узагальнення знань, одержаних студентом за час навчання та придбання практичних навичок їх застосування при розробці крос-платформних систем.

Мета ІЗ: теоретичне вивчення та отримання практичних навичок розробки крос-платформних програмних систем з графічним інтерфейсом користувача мовою програмування Java.

У процесі виконання ІЗ студент повинен застосувати набуті теоретичні знанням і практичні навики та продемонструвати здібності до науково-дослідної роботи і вміння творчо мислити, навчитися вирішувати науково-прикладні актуальні задачі.

Самостійне вивчення дисципліни «Крос-платформне програмування» у міжсесійний період складається з:

- придбання теоретичних знань по дисципліні по темах, зазначених у завданнях №1 – №6;
- виконання контрольної роботи з варіантів завдань, зазначених у тематичних завданнях (відповідно до варіанту).

Відповідно до програми вивчення дисципліни в контрольній роботі з кожного тематичного завдання потрібно:

- відповісти на запитання з теоретичної частини завдань;
- написати програмний код мовою Java відповідно до завдання практичної частини варіанту;
- оформити отриманий програмний код і результати його виконання у контрольній роботі.

Кожен студент для виконання контрольної роботи визначає свій варіант завданя. Номер варіанту визначається за номером залікової книжки.

Контрольна робота складається з 6-х завдань, які студент повинен виконати дистанційно до початку сесії. Кожне виконане завдання присилається викладачеві для перевірки на електронну адресу: kaf-inf@ogmi.farlep.odessa.ua або dean-comp@ogmi.farlep.odessa.ua. Кожне завдання перевіряється та оцінюється викладачем згідно з табл. 2.1:

Максимальна сума балів яку студент може отримати за виконання ІЗ дорівнює 60 балам.

Якщо на початок сесії студент виконав всі завдання, та отримав позитивну оцінку, то він роздруковує контрольну роботу та здає її викладачеві.

Таблиця 2.1– Етапи виконання контрольної роботи

Номер завдання	Дати виконання завдання	Максимальна сума балів, яку можна отримати за виконане завдання
1	1.01 –25.01	10
2	26.01 –20.03	10
3	21.03–20.04	20
4	20.04–10.05	20

3 ЗАВДАННЯ ДЛЯ ВИКОНАННЯ КОНТРОЛЬНОЇ РОБОТИ

Завдання №1

Програмування аплетів з елементами мультимедіа для Web сторинок Internet

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

Після виконання завдання студенти мають оволодіти вміннями створювати, запускати, виконувати та передавати параметри аплету.

Постановка завдання

– 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!).

– Створити аплет згідно варіанту завдання.

Для виконання завдання №1 необхідно:

- вивчити теоретичний матеріал, наведений в [3, ст. 43-58].
- відповісти на контрольні питання [3, ст. 44-58; 2, ЛР1];
- вивчити IDE і програму appletviewer [5];
- створити аплет згідно варіанту завдання;
- оформити відповіді на питання, програмний код, результати виконання програми у контрольній роботі згідно до стандартів оформлення.

Короткі теоретичні відомості за темою завдання

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>.

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`**,	Ні

	bottom, **absbottom**, **left**, **right**.	
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.*;

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.

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().

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());  }
```

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

Метод	Опис
	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)

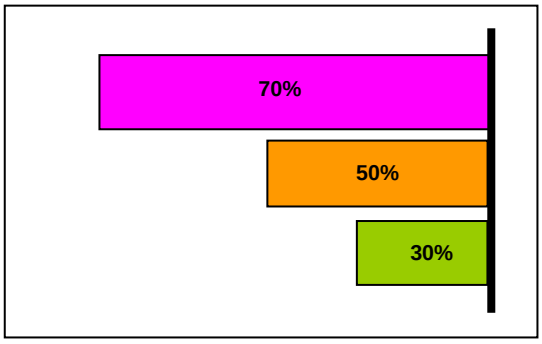
Контрольні питання

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?

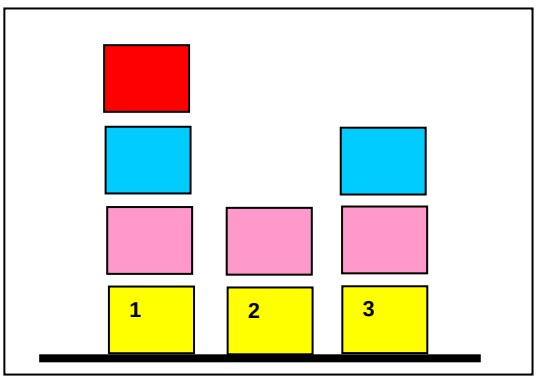
6. Які методи маніпуляції з об'єктами визначені в Java?
7. Як можна задавати кольори у Java?
8. Які методи заповнення геометричних фігур, що замкнуті визначені в Java?
9. Як задаються характеристики шрифту, що виводиться, в Java?
10. Як виводиться напис в графічному режимі, і які параметри можна задати для напису?
11. Як в Java можна визначити ширину рядка, що виводиться, в пікселях?
12. Як можна вставити зображення в аплет?

Варіанти завдання

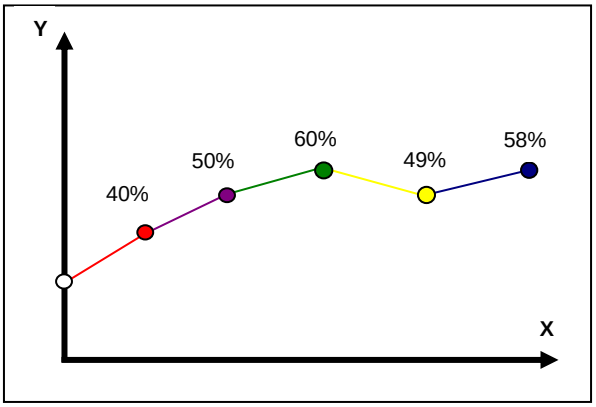
1. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від ширини вікна аплета. Колір прямокутників задати у вигляді масиву. Ширіну кожного прямокутника прийняти постійною, розрахованою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



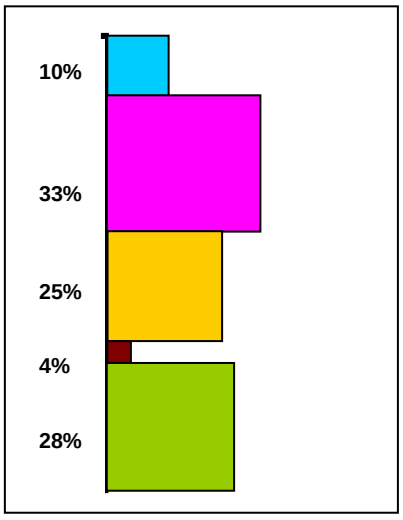
2. Використовуючи передачу множинних параметрів в аплет, намалюйте вказані фігури (див. рис). Кількість квадратів, складових фігури задати як параметри у вигляді цілочисельних значень. Колір квадратів задати у вигляді масиву. Розмір кожного квадрата прийняти постійним, розрахованим так, щоб навіть найвища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



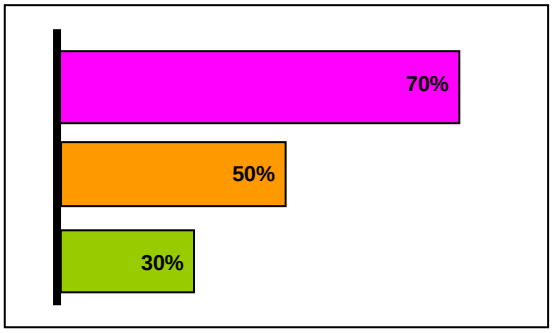
3. Використовуючи передачу множинних параметрів в аплет, намалюйте графік (див. рис). Координати точок по осі Y у відсотках від висоти області побудови графіка задати як параметри у вигляді цілочисельних значень. Колір точок на графіці задати у вигляді масиву. Крок побудови графіка прийняти постійним,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



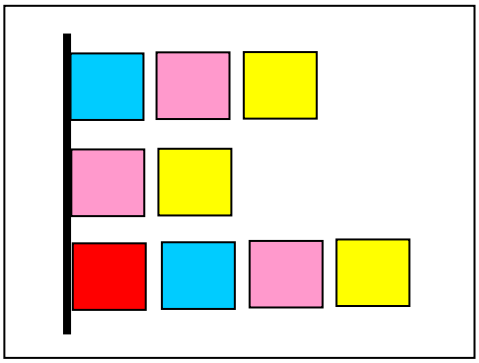
4. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного квадрата передати як параметр - цілочисельне значення, що позначає % від висоти області побудови (їх сума повинна складати 100%). Колір квадратів задати у вигляді масиву. Вивести на малюнку висоту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



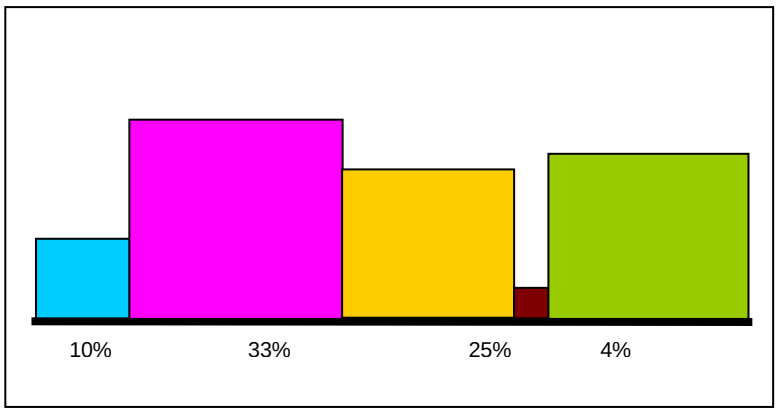
5. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від ширини вікна аплета. Колір прямокутників задати у вигляді масиву. Ширіну кожного прямокутника прийняти постійною, розрахованою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



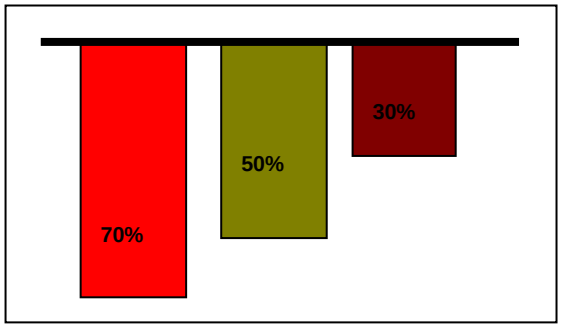
6. Використовуючи передачу множинних параметрів в аплет, намалюйте вказані фігури (див. рис). Кількість квадратів, складових фігури, задати як параметри у вигляді цілочисельних значень. Колір квадратів задати у вигляді масиву. Розмір кожного квадрата прийняти постійним, розрахованим так, щоб навіть найвища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



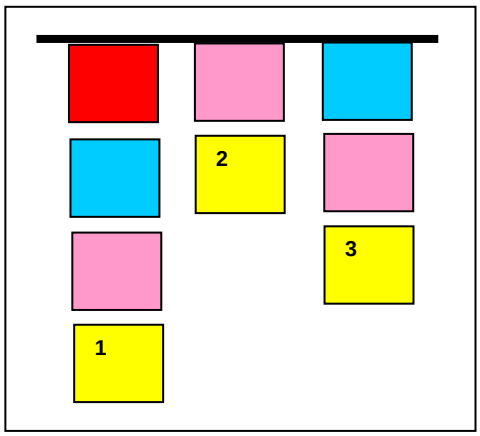
7. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Ширіну кожного квадрата передати як параметр - цілочисельне значення, що позначає % від ширини області побудови (їх сума повинна складати 100%). Колір квадратів задати у вигляді масиву. Вивести на малюнку ширину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



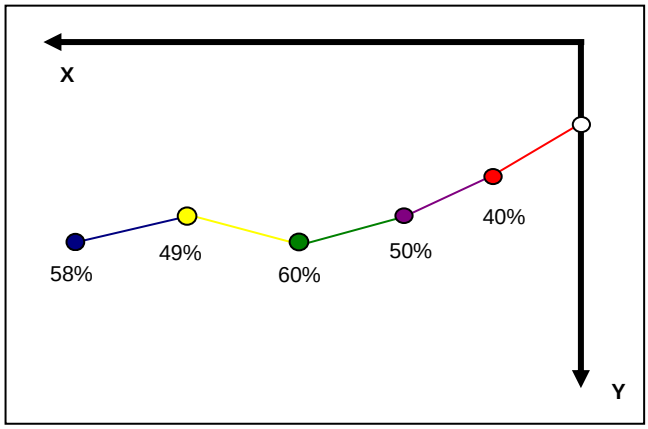
8. Використовуючи передачу множинних параметрів в аплет, намалюйте гістограму (див. рис). Висоту кожного прямокутника передати як параметр - цілочисельне значення, що позначає % від висоти вікна аплета. Колір прямокутників задати у вигляді масиву. Ширіну кожного прямокутника прийняти постійною, розрахованою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



Приклад виконання завдання 1

```
/*
<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;
         }}

     }
```

Звіт про виконання завдання №1

Звіт про виконання завдання повинен відповідати змісту:

1. Постановка задачі.
2. Основні теоретичні положення (відповіді на контрольні питання).
3. Листинг програмного коду.
4. Результати виконання програми (скріншоти вікон додатка).
5. Висновки по підсумках роботи.

Завдання 2

Використання потоків і створення анімації в Java

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

Після виконання завдання студенти мають оволодіти вміннями створення багатопотокових программ з графічним інтерфейсом користувача і елементами анімації.

Постановка завдання

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.

Для виконання завдання №2 необхідно:

- вивчити теоретичний матеріал, наведений в [1, ст. 88-92, 2 ст.328-350].
- відповісти на контрольні питання [2, ст. 328-350; 2, ЛР2];
- створити програму – аплет згідно варіанту завдання;
- оформити відповіді на питання, програмний код, результати виконання програми у контрольній роботі згідно до стандартів оформлення.

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?

Короткі теоретичні відомості за темою завдання

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.

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.

Варіанти до завдання

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.

Приклад виконання частини 1 завдання 2

```
<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);
    }
}
```

Звіт про виконання завдання №2

Звіт про виконання завдання повинен відповідати змісту:

1. Постановка задачі.
2. Основні теоретичні положення (відповіді на контрольні питання).
3. Листинг програмного коду.
4. Результати виконання програми (скріншоти вікон додатка).
5. Висновки по підсумках роботи.

Завдання 3

Створення графічних застосувань. Використання системи Swing в Java

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

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

Постановка завдання

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

Для виконання завдання №3 необхідно:

- вивчити теоретичний матеріал, наведений в [1, ст. 112-122; 3, ст. 259-328].
- відповісти на контрольні питання [3, ст. 259-328; 2, ЛР3];
- створити графічний додаток згідно варіанту завдання;
- оформити відповіді на питання, програмний код, результати виконання програми у контрольній роботі згідно до стандартів оформлення.

Короткі теоретичні відомості за темою завдання

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
          }
);
```

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.

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.

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
     }
     }
);
```

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?

Варіанти до завдання

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.

Звіт про виконання завдання №3

Звіт про виконання завдання повинен відповідати змісту:

1 Постановка задачі.
2 Основні теоретичні положення (відповіді на контрольні питання).
3 Листинг програмного коду.
4 Результати виконання програми (скріншоти вікон додатка).
5 Висновки по підсумках роботи.

Завдання №4

Створення графічних застосувань. Програмування потоків вводу/виводу

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

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

Постановка авдання

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

Для виконання завдання №4 необхідно:

- вивчити теоретичний матеріал, наведений в [1, ст. 158-174; 3, ст. 205-229].
- відповісти на контрольні питання [3, ст. 205-229; 2, ЛР4];
- створити графічний додаток згідно варіанту завдання;
- оформити відповіді на питання, програмний код, результати виконання програми у контрольній роботі згідно до стандартів оформлення.

Короткі теоретичні відомості за темою завдання

Робота з файлами. 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 1.

Таблиця 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.
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.

Метод	Опис
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.

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**.

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 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**.

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?

Варіанти до завдання

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.

Звіт про виконання завдання №4

Звіт про виконання завдання повинен відповідати змісту:

1 Постановка задачі.
2 Основні теоретичні положення (відповіді на контрольні питання).
3 Листинг програмного коду.
4 Результати виконання програми (скріншоти вікон додатка).
5 Висновки по підсумках роботи.

Завдання № 5

Програмування колекцій в JAVA

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

Після виконання завдання студенти мають оволодіти вміннями вибирати і реалізовувати класи-колекції для вирішення завдань.

Постановка завдання

Напишіть графічний додаток Swing для виконання операцій з колекціями по одному з наведених нижче варіантів.

На підставі отриманих результатів зробить висновки щодо особливостей роботи з колекціями в Java та порівняльний аналіз реалізацій колекцій.

Для виконання завдання №5 необхідно:

- вивчити теоретичний матеріал, наведений в [1, ст. 185-209; 3, ст.229-259].
- відповісти на контрольні питання [3, ст.229-259; 2, ЛР5];
- створити графічний додаток згідно варіанту завдання;
- оформити відповіді на питання, програмний код, результати виконання програми у контрольній роботі згідно до стандартів оформлення.

Короткі відомості за темою завдання

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.

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**.

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?

Варіанти до завдання

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.

Звіт про виконання завдання №5

Звіт про виконання завдання повинен відповідати змісту:

1. Постановка задачі.
2. Основні теоретичні положення (відповіді на контрольні питання).
3. Листинг програмного коду.
4. Результати виконання програми (скріншоти вікон додатка).
5. Висновки по підсумках роботи.

Завдання № 6

Управління мережевими з'єднаннями. Використання сокетів у розподілених додатках

Метою завдання є придбання навиків створення мережевих додатків на мові Java з використанням бібліотеки java.net.

Після виконання завдання студенти мають оволодіти вміннями реалізовувати взаємодію клієнта та сервера, використовуючи сокети TCP і UDP.

Постановка завдання

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.

Для виконання завдання №6 необхідно:

- вивчити теоретичний матеріал, наведений в [1, ст. 212-224; 3, ст. 350-364].
- відповісти на контрольні питання [3, ст. 350-364; 2, ЛР6];
- створити клієнт-серверний додаток згідно варіанту завдання;
- оформити відповіді на питання, програмний код, результати виконання програми у контрольній роботі згідно до стандартів оформлення.

Короткі теоретичні відомості за темою завдання

У пакеті **java.net** визначена група класів та інтерфейсів, що дозволяють управляти мережевими з'єднаннями в Java-програмах.

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

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**.

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.

Сокети і сокетне 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**[1]
ServerSocket(int port, int maxQueue, InetAddress localAddress)	Створює сокет сервера на зазначеному порту з максимальною довжиною черги **maxQueue.** На груповому[2] хост-комп'ютері localAddress визначає IP-адресу, з якою цей сокет пов'язаний.

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

[1] Довжина черги повідомляє системі, скільки підключень клієнта вона може залишати затриманими перш, ніж повинна буде просто відкинути з'єднання. За замовчуванням - 50

[2] Груповий хост – комп'ютер, який має декілька мережевих адаптерів або налаштований з декількома IP-адресами для одиночного мережевого адаптера.

	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.

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?

Варіанти до завдання

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.

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----------

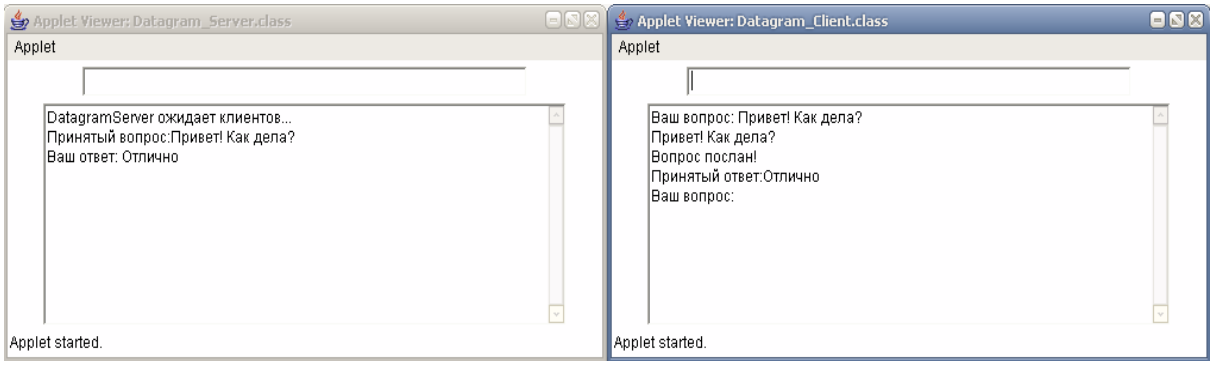
```
import java.net.*;
import java.io.IOException;
/* Класс серверного приложения, реализующего диалог с
множеством клиентов в режиме "вопрос/ответ".
*/
public class DatagramServer
{
```

```
    /* 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(); }
}
}
```

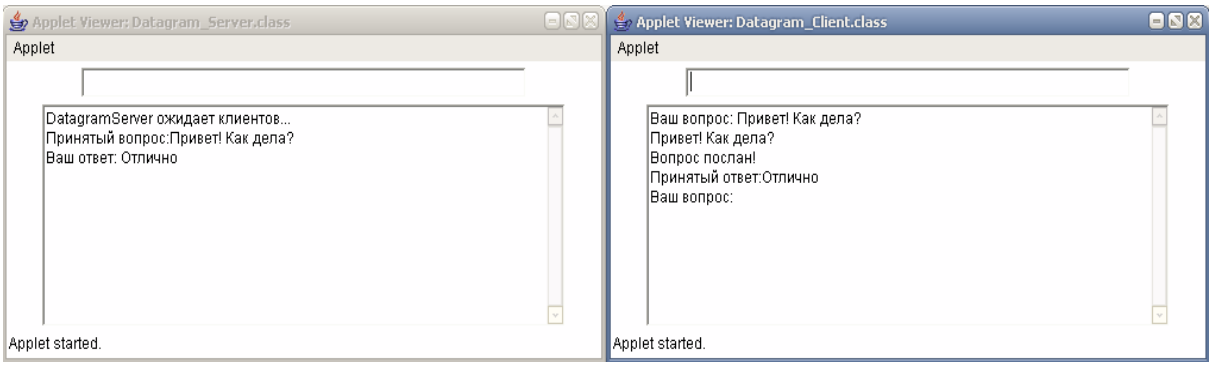
----------КЛІЄНТ----------

```
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.
     */
```

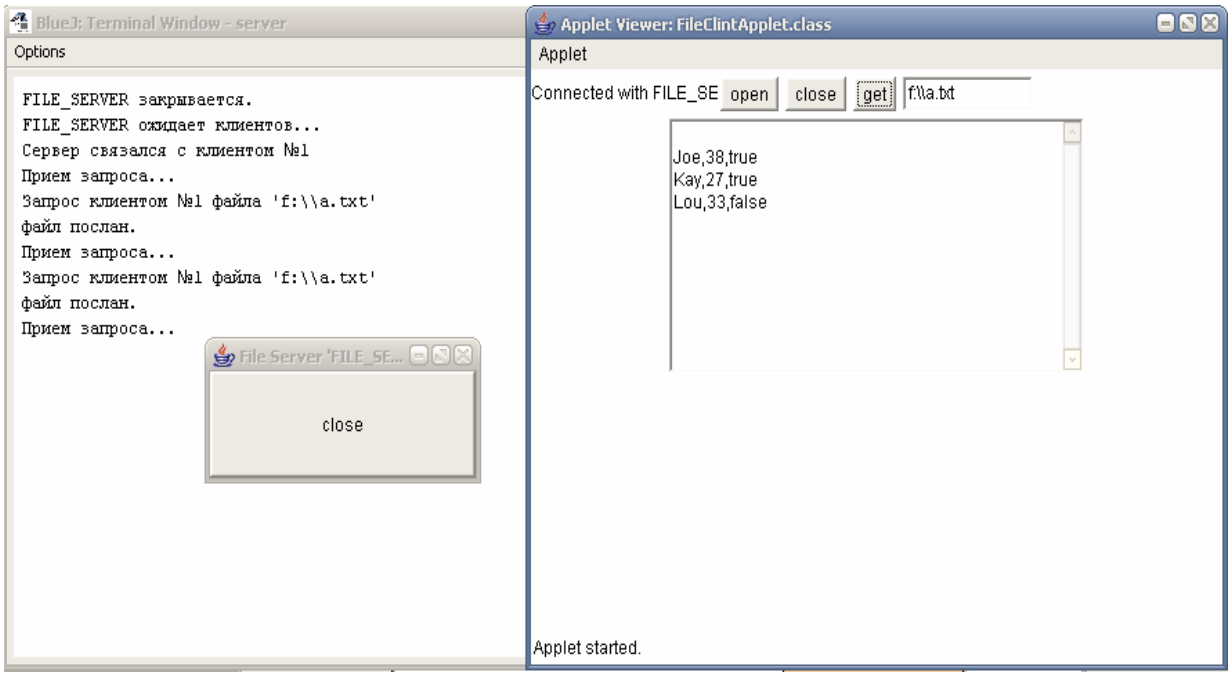
```
     public StartServer() {
     super("File Server 'FILE_SERVER'");
     System.out.println("FILE_SERVER ожидает клиентов...");
     close = new Button("close");
     add(close); close.addActionListener(this);
     resize(200,100);
     show();
     new Thread(this).start();
     }
     /* Обработчик 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;
    }}
```



Звіт про виконання завдання №6

Звіт про виконання завдання повинен відповідати змісту:

1. Постановка задачі.
2. Основні теоретичні положення (відповіді на контрольні питання).
3. Листинг програмного коду.
4. Результати виконання програми (скріншоти вікон додатка).
5. Висновки по підсумках роботи.

4 ОРГАНІЗАЦІЯ ПОТОЧНОГО ТА ПІДСУМКОВОГО КОНТРОЛЮ ЗНАНЬ ТА ВМІНЬ

Контроль знань та вмінь студентів, що навчаються за заочною формою, здійснюється за допомогою системи контролюючих заходів. Вони складаються з заходів поточного та підсумкового контролю. Поточний контроль здійснюється на протязі всього навчального року (семестру) та включає заходи контролю самостійної роботи студента під час вивчення навчальної дисципліни поза межами університету та роботи студента на лабораторних заняттях у період заліково-екзаменаційної сесії.

Підсумковий контроль здійснюється під час заліково-екзаменаційної сесії та має на меті встановлення рівня знань та вмінь, які опанував студент після вивчення навчальної дисципліни. Форма підсумкового контролю - залік - встановлюється навчальним планом дисципліни.

Поточна та підсумкова оцінка рівня знань студентів заочної форми навчання з дисципліни «Крос-платформне програмування» здійснюється за модульною системою у відповідності з «Положенні про організацію поточного та підсумкового контролю знань студентів заочної форми навчання ОДЕКУ». Навчальним планом передбачено проведення занять у вигляді лекцій, лабораторних занять та виконання контрольної роботи, а підсумкова атестація – залік.

4.1 Поточний контроль

Поточний контроль здійснюється на протязі навчального курсу за наступними формами:

- перевірка контрольної роботи, яка виконується у міжсесійний період;
- перевірка знань та вмінь студента під час аудиторних занять протягом заліково-екзаменаційної сесії.

Оцінка виконання СРС у міжсесійний період, визначається шляхом перевірки контрольної роботи, при визначенні якої враховується наступне:

- термін представлення контрольної роботи (на протязі семестру, перед початком заліково-екзаменаційної сесії, безпосередньо перед датою контролюючого заходу);
- відповідність змісту та кількості завдань з теоретичної та практичної частин навчальній програмі дисципліни;
- оформлення контрольної роботи згідно ДСТУ.

Кожне завдання (питання) контрольної роботи оцінюється за наступною шкалою:

90-100% від максимально можливої кількості балів – бездоганна вичерпна відповідь на всі завдання, оформлення контрольної роботи згідно ДСТУ, контрольна робота здана у встановлені терміни;

74-89,9% -//- – надані відповіді на всі завдання є правильними, але не є повними;

60-73,9% -//- – надані відповіді на 2/3 завдань є правильними, але не повними;

< 60% -//- – надані відповіді тільки на 1/3 завдань або відповіді на поставлені питання є помилковими, контрольна робота не оформлена згідно ДСТУ.

Контрольна робота вважається зарахованою, якщо студент одержав не менше 60% від максимальної суми балів. Для того щоб отримати 60% за контрольну роботу студент обов'язково повинен виконати 1–4 завдання.

Студенти, які не отримали за контрольну роботу мінімальної кількості балів (> 60%), повинні виконати інший варіант контрольної роботи або виправити помилки попереднього варіанту. Зарахована контрольна робота є допуском до залікової контрольної роботи.

До суми балів за аудиторні заняття (теоретичну та практичну частину дисципліни) під час заліково-екзаменаційної сесії (ОЗЕ) входять оцінки за виконання лабораторних робіт. При оцінці заходів контролю СРС під час проведення аудиторних занять за період сесії враховується:

- ритмічність роботи студента на протязі занять (присутність його на заняттях за розкладом);
- повнота та якість розкриття окремих питань;
- якість розрахунків та графічних побудов, достовірність отриманих висновків;
- оцінка захисту окремих розділів та завдань у цілому.

Якщо студент, який на дату контролюючого заходу не має заборгованості по виконанню міжсесійних та сесійних контролюючих заходів (набрав ≥ 50% по дисципліні), то допускається до залікової контрольної роботи, яка проводиться у письмовій формі за тестами. На написання залікової контрольної роботи студентові відводиться до 90 хвилин.

4.2 Підсумковий контроль

Накопичувальний підсумковий контроль проводиться на основі накопиченої (інтегральної) суми балів, яку отримав студент по підсумках поточного контролю та підсумкового контролю (залікової контрольної роботи).

Накопичена підсумкова оцінка (ПО) засвоєння студентом навчальної дисципліни складається з:

- системи оцінювання самостійної роботи студента у міжсесійний період (ОМ – оцінка міжсесійна);
- систему оцінювання СРС при проведенні аудиторних занять за дисципліною під час заліково-екзаменаційної сесії (ОЗЕ – оцінка сесійна);
- оцінювання заходу підсумкового контролю, який виконується в період заліково-екзаменаційної сесії (ОЗКР – оцінка залікової контрольної роботи).

Накопичувальний підсумковий контроль передбачає дві форми оцінювання успішності засвоєння студентом навчального матеріалу дисципліни:

- кількісна оцінка (бал успішності);
- якісна оцінка.

Оцінка за шкалою ECTS виставляється відповідно наведеної таблиці:

<table>
<tr><td rowspan="2">СУМА БАЛІВ</td><td rowspan="2">ОЦІНКА ECTS</td><td colspan="2">ОЦІНКА ЗА НАЦІОНАЛЬНОЮ ШКАЛОЮ</td></tr>
<tr><td>ЕКЗАМЕН</td><td>ЗАЛІК</td></tr>
<tr><td>90-100</td><td>A</td><td>ВІДМІННО</td><td rowspan="5">ЗАРАХОВАНО</td></tr>
<tr><td>82-89</td><td>B</td><td rowspan="2">ДОБРЕ</td></tr>
<tr><td>74-81</td><td>C</td></tr>
<tr><td>64-73</td><td>D</td><td rowspan="2">ЗАДОВІЛЬНО</td></tr>
<tr><td>60-63</td><td>E</td></tr>
<tr><td>35-59</td><td>FX</td><td rowspan="2">НЕЗАДОВІЛЬНО</td><td rowspan="2">НЕ ЗАРАХОВАНО</td></tr>
<tr><td>1-34</td><td>F</td></tr>
</table>

Накопичена підсумкова оцінка засвоєння студентами заочної форми навчальної дисципліни розраховується, як:

$$ПО = 0.75 \cdot (ОЗЕ + ОМ) + 0.25 \cdot ОЗКР,$$

де ОЗЕ – кількісна оцінка (у відсотках від максимально можливої) заходів контролю СРС під час проведення аудиторних занять;

ОМ – кількісна оцінка (у відсотках від максимально можливої) заходів контролю СРС у міжсесійний період;

ОЗКР - оцінка залікової контрольної роботи.

Одержана накопичена підсумкова оцінка виставляється викладачем у заліково-екзаменаційну відомість.

З урахуванням модульно-накопичувальної системи контроль СРС розподіляться наступним чином:

- у міжсесійний період

Номер змістовного модуля	Форма контролю	Максимальна сума балів, яку можна отримати у міжсесійний період
ІЗ	Виконання ІЗ	40

Максимальна сума міжсесійний період – 40 балів.

- сесійний період

Номер зміст. модуля	Форма контролю	Максимальна сума балів, яку можна отримати у сесійний період
ЗМ-Л	КР (ОМ),	20
1	Захист ЛР	10
2	Захист ЛР	10
3	Захист ЛР	10
4	Захист ЛР	10

Максимальна сума у сесійний період – 60 балів.

Студент вважається допущеним до підсумкового контролю (ОЗКР) з дисципліни, якщо він виконав всі види робіт поточного контролю (ОМ+ОЗЕ), передбачені робочою навчальною програмою дисципліни і набрав за накопичувальною системою суму балів за контрольну роботу не

менше 20 балів та сесійний період не менше 40 балів (50%) від максимально можливої за дисципліну, своєчасно виконав міжсесійні контрольні роботи.

МЕТОДИЧНІ ВКАЗІВКИ

до самостійної роботи студентів та
виконання контрольних робіт з дисципліни
“***Крос-платформне програмування***”
для студентів 5 курсу заочної форми навчання
Напрям підготовки – комп’ютерні науки

Підп. до друку Формат 60х84/16

Папір офс. Наклад прим.

Замовлення №

Надруковано з готового оригінал – макета.

Одеський державний екологічний університет,
65016, м. Одеса, вул. Львівська, 15