МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни «Крос-платформне програмування»

для студентів 4 курсу
рівень вищої освіти – «бакалавр»
спеціальності 122 «Комп’ютерні науки»

ЗАТВЕРДЖЕНО
на засіданні групи забезпечення
спеціальності 122 Комп'ютерні науки
від «_14_»_серпня____ 2023 року
протокол №_1_ Голова групи
_____Кузніченко С.Д.

Затверджено на засіданні
кафедри Інформаційних технологій
протокол № 1 від «14» серпня 2023 р.
Зав. кафедри_____Казакова Н.Ф.

ОДЕСА – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт
з дисципліни «Крос-платформне програмування»

для студентів 4 курсу
рівень вищої освіти – «бакалавр»
спеціальності 122 «Комп’ютерні науки»

ОДЕСА – 2023

Методичні вказівки до виконання лабораторних робіт з дисципліни «Кросс-платформне програмування» для студентів 4 курсу спеціальності 122 «Комп’ютерні науки» / Укладачі: к.геогр.н., доц. Кузніченко С.Д., ас. Молчанова А.Ю.

Зміст

Передмова 4
Лабораторна робота №1 Створення графічного інтерфейсу користувача 5
Лабораторна робота №2 Обробка семантичних подій в Java 24
Лабораторна робота №3 Використання потоків і створення анімації в Java 29
Лабораторна робота №4 Програмування потоків вводу/виводу 42
Лабораторна робота №5 Програмування колекцій в Java 67
Лабораторна робота №6 Управління мережевими з'єднаннями. Використання сокетів у розподілених додатках 84
Список літератури 111

Передмова

Методичні вказівки призначені для студентів IV курсу спеціальності «Комп’ютерні науки». Мета виконання лабораторних робіт – засвоєння необхідних знань з основ розробки крос-платформних компонентів, а також формування практичних навичок щодо розроблення додатків з використанням компонентного підходу при розробці розподілених систем.

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

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

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

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

За кожну лабораторну роботу студент отримує дві оцінки: за виконання та за захист роботи. Згідно з цих пунктів студенту зараховується відповідна кількість балів. Максимальні бали з кожної лабораторної роботи встановлюються згідно силлабусу: [тут буде посилання на силабус 2023 року, який ще в процесі затвердження].

Лабораторна робота №1
Створення графічного інтерфейсу користувача

1. Мета роботи

Метою роботи є придбання навичок створення інтерфейсу користувача на мові Java з використанням графічних об'єктів і зображень.

Після виконання лабораторної роботи студенти мають оволодіти уміннями створювати, налаштовувати та запускати застосунки з графічним інтерфейсом.

2. Теоретичні відомості до виконання лабораторної роботи

Бібліотеки AWT, Swing та JavaFX

Для створення графічного інтерфейсу користувача (GUI) в Java найпоширенішими підходами є використання бібліотек AWT, Swing та JavaFX.

AWT (Abstract Window Toolkit) була першою бібліотекою для створення графічного інтерфейсу в мові програмування Java, розробленою в 1995 році. Вона надавала базовий набір компонентів і засобів для відображення елементів інтерфейсу на екрані. Однак, AWT використовувала нативні компоненти платформи для відображення, що робило її залежною від платформи та недостатньо крос-платформенною. Це призводило до проблем із сумісністю та виглядом застосунків на різних операційних системах.

Swing був створений у 1998 році для заміни AWT та боротьби з його недоліками. Він є розширенням AWT і надає високорівневі компоненти GUI, які не залежать від платформи і виглядають однаково на різних операційних системах. Swing пропонує великий набір компонентів, можливостей стилізації за допомогою платформонезалежного механізму малювання та можливості анімації. Однак завантаження та відображення великої кількості компонентів Swing вимагає значних ресурсів, що часто призводить до зниження продуктивності додатка.

JavaFX була розроблена в 2008 році компанією Sun Microsystems (тепер належить Oracle) з метою заміни старих і менш продуктивних бібліотек Swing та AWT. JavaFX враховує всі недоліки попередників та є міцною, гнучкою і сучасною технологією. На даний час JavaFX є стандартною технологією для розробки GUI в Java.

Установка та початок роботи з JavaFX

Серед найпопулярніших інтегрованих середовищ розробки (IDE) для роботи з JavaFX - Eclipse, IntelliJ IDEA, Apache NetBeans тощо. Кожне з цих середовищ має свої особливості, недоліки та переваги. В даних методичних вказівках буде розглянутий процес встановлення JavaFX в Eclipse та IntelliJ IDEA. Студенти можуть за власним бажанням обрати одне з цих середовищ або будь-яке інше, в якому є підтримка JavaFX.

Для роботи з JavaFX необхідно мати встановлений JDK (Java Development Kit, комплект розробника застосунків). Щоб перевірити, чи встановлено JDK (Java Development Kit) на комп'ютері, в командному рядку введіть команду:

java -version

Якщо JDK встановлено на комп'ютері, буде виведена версія Java, якщо ні - ви побачите повідомлення про те, що команда не впізнана або не знайдена. В такому випадку необхідно встановити останню версію JDK с офіційного сайту Oracle: https://www.oracle.com/java/technologies/downloads.

Установка JavaFX в Eclipse IDE

Eclipse IDE можна завантажити з офіційного сайту https://www.eclipse.org/downloads. Процес встановлення стандартний.

В Eclipse IDE перейдіть в Help > Eclipse Marketplace. В пошуковому рядку впишіть “fx”. Встановіть набір плагінів e(fx)clipse (рис. 1.1).

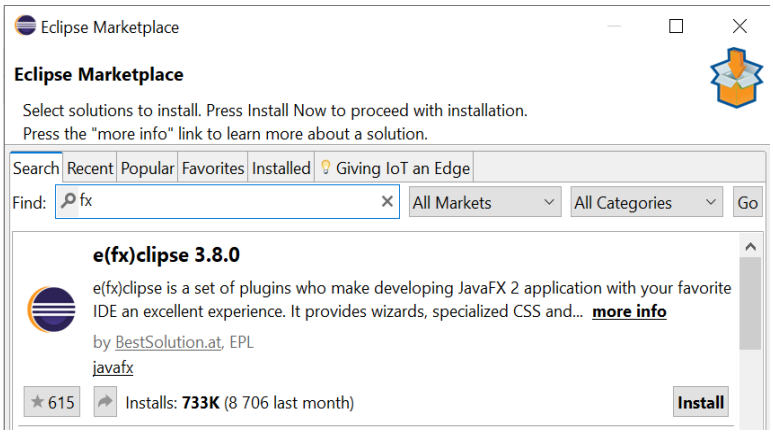

Рисунок 1.1 - Встановлення набору плагінів для роботи з JavaFX з Eclipse Marketplace

Після встановлення цього набору плагінів можна створювати застосунки JavaFX. Для цього натисніть File > New > Project… та у вікні, що відкрилося, оберіть JavaFX Project (рис. 1.2). Вкажіть його ім’я у наступному вікні та натисніть Finish.

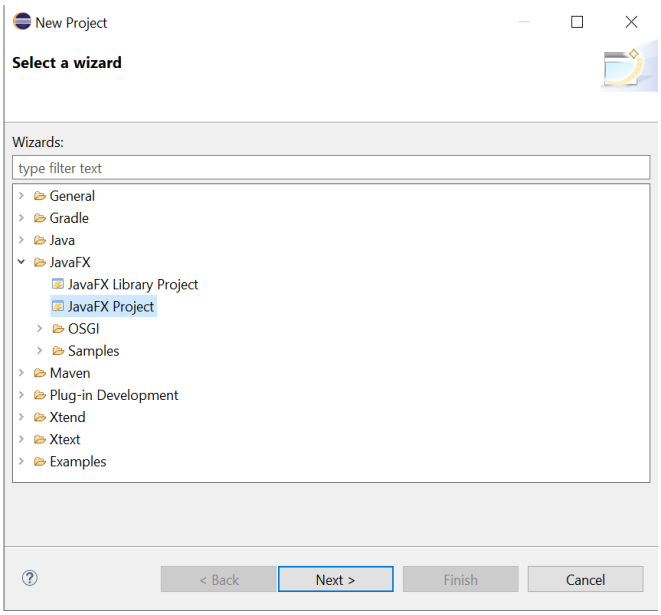

Рисунок 1.2 - Створення проекту JavaFX

За замовчуванням Eclipse створює певну ієрархію проекту, в якому є пакет src, де міститься основний код застосунку JavaFX (рис. 1.3). Клас Main.java - це головний клас застосунку, що містить метод main, який є входом до програми. В цьому класі і необхідно писати код застосунку.

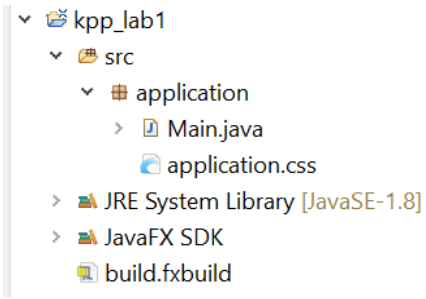

Рисунок 1.3 - Структура проекту JavaFX в Eclipse IDE

Установка JavaFX в IntelliJ IDEA

IntelliJ IDEA Community Edition можна завантажити з офіційного сайту JetBrains https://www.jetbrains.com/ru-ru/idea/download. Враховуйте, що середовище займає близько 2.9 Гб на диску.

Переконайтеся, що плагін JavaFX завантажений та активований. Для цього перейдіть у File > Settings > Plugins та введіть в пошуковому рядку “fx” (рис. 1.4).

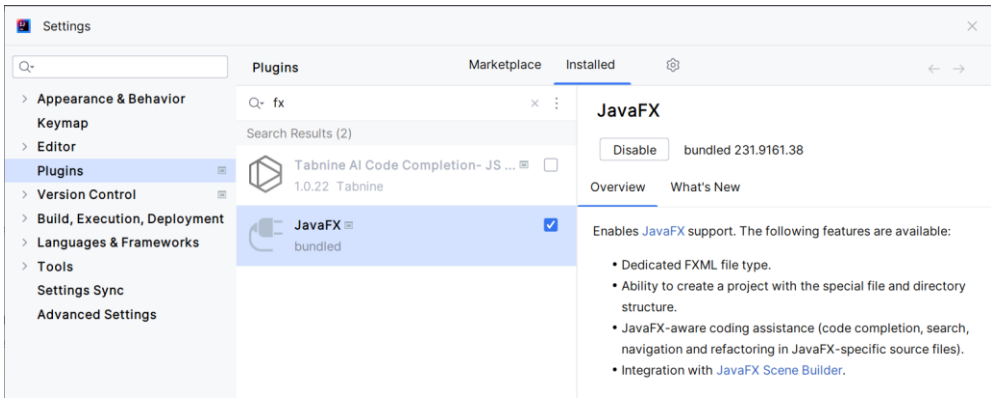

Рисунок 1.4 - Встановлення плагіну JavaFX в IntelliJ IDEA

Тепер можна створити застосунок JavaFX через File > New > Project…. У списку Generators ліворуч оберіть JavaFX. Введіть ім’я проекту та групу для проекту та натисніть Next (рис. 1.5).

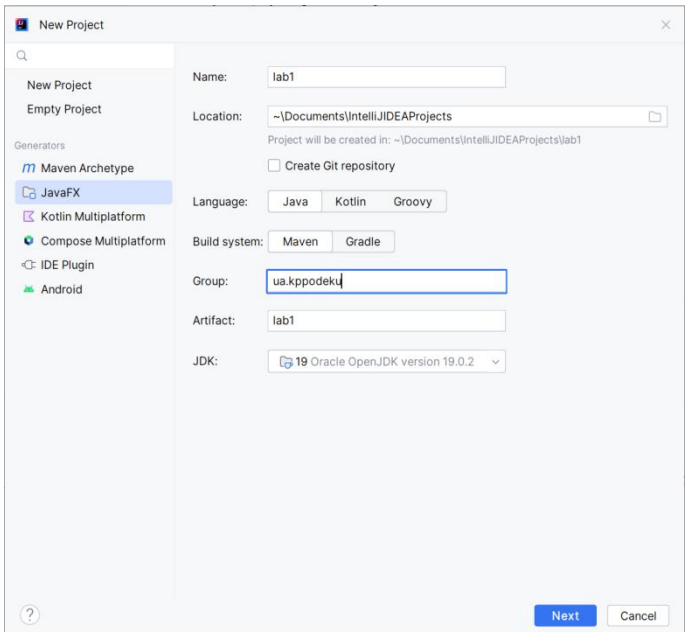

Рисунок 1.5 - Створення проекту JavaFX в IntelliJ IDEA

У наступному вікні можна за необхідністю підключити додаткові бібліотеки, наприклад бібліотеку ControlsFX, яка містить високоякісні елементи управління UI та інші інструменти, що доповнюють основний дистрибутив JavaFX.

IntelliJ IDEA також за замовчуванням створює певну структуру проекту (рис. 1.6). Подібно до Eclipse, в пакеті src міститься основний код застосунку JavaFX. Клас HelloApplication.java - це головний клас застосунку, що містить метод main, який є входом до програми. В цьому класі і необхідно писати код застосунку.

Також IntelliJ IDEA створює файли hello-view.fxml та HelloController.java. Файл hello-view.fxml є файлом опису інтерфейсу користувача. FXML (FXML Markup Language) - це мова розмітки, яка використовується для опису графічного інтерфейсу JavaFX. Він використовує синтаксис XML та дозволяє відділити код інтерфейсу від Java-коду. Цей файл можна редагувати в Intellij IDEA за допомогою візуального редактора або вручну за допомогою редактора тексту. Після створення файлу hello-view.fxml, його зазвичай використовують разом з класом контролера (наприклад, HelloController.java), який забезпечує логіку обробки подій та інтерактивності між елементами інтерфейсу і кодом Java.

Використання FXML та контролерів спрощує розробку та підтримку інтерфейсу у великих проектах. Однак ці файли не є обов'язковими для створення застосунків JavaFX і в даних лабораторних роботах не будуть використовуватися. Це передбачено в тому числі для простоти та швидкості створення застосунків, контролю над кодом і його легшим розумінням, зменшенням кількості файлів проекту та залежностей від додаткових бібліотек та фреймворків, пов'язаних з обробкою FXML-файлів та контролерів. Тому ці файли можна видалити.

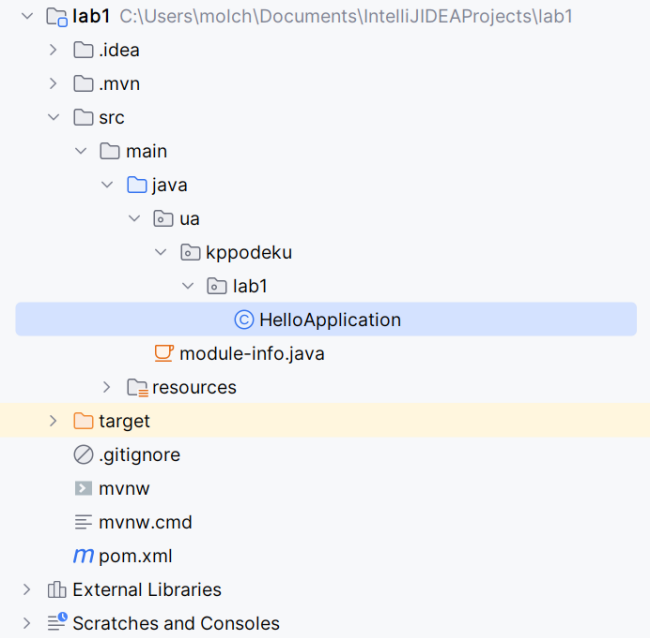

Рисунок 1.6 - Структура проекту JavaFX в Eclipse IDE

Створення графічного інтерфейсу

Основні поняття в JavaFX для створення GUI включають Stage, Scene та Node (рис. 1.7).

Stage (Вікно). Вікно є головним контейнером в JavaFX, воно представляє основне вікно застосунку. У кожному застосунку JavaFX має бути принаймні один об'єкт Stage. Вікно має свою сцену (Scene) і містить всі елементи GUI, які відображаються на екрані.

Scene (Сцена). Сцена представляє контейнер, в якому розміщені всі елементи інтерфейсу. Вікно (Stage) може містити одну або кілька сцен (Scene). Сцена може містити будь-які контроли, контейнери та інші елементи, що будуть відображені на вікні.

Node (Вузол): Node є базовим класом для всіх елементів інтерфейсу в JavaFX. Всі елементи, такі як кнопки, текстові поля, мітки, картинки, форми тощо, є підкласами Node.

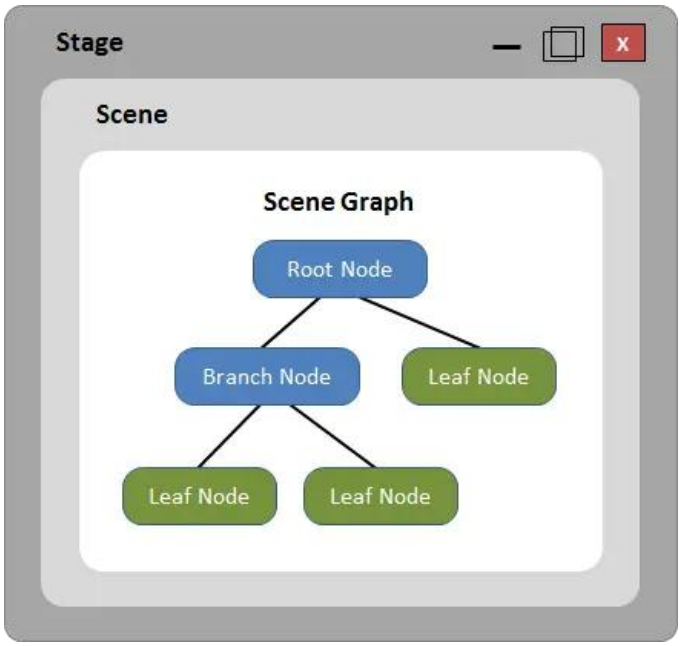

Рисунок 1.7 - Структура GUI на JavaFX

Застосунок JavaFX повинен мати клас, що розширює (extends) Application, оскільки Application є основним класом для запуску застосунків та представляє точку входу для платформи JavaFX. JavaFX використовує стандартний метод main() для запуску додатків, і він вимагає наявності класу, який успадковує Application. При успадкуванні Application, потрібно також перевизначити його абстрактний метод start(), який буде викликаний платформою JavaFX під час запуску застосунку. Ви можете використовувати цей метод для ініціалізації графічного інтерфейсу та відображення його на екрані.

Таким чином, основна структура класу застосунку JavaFX має вигляд:

```
public class Main extends Application {

     public static void main(String[] args) {
          launch(args);
     }
```

```
    @Override
    public void start(Stage stage) {
        //код GUI та обробки подій застосунку
    }
}
```

Найбільш поширені методи, які відносяться до Stage, тобто головного вікна застосунку, наведені в таблиці 1.1.

Таблиця 1.1 - Основні методи налаштування Stage

setTitle(String title)	Встановлює заголовок (назву) вікна
setWidth(double width)	Встановлює ширину вікна
setHeight(double height)	Встановлює висоту вікна
setScene(Scene scene)	Встановлює сцену, яка буде відображена у вікні. Сцена містить всі елементи GUI, які будуть видимі на вікні.
setResizable(boolean resizable)	Встановлює, чи можна змінювати розмір вікна користувачем
show()	Відображає вікно на екрані. Цей метод слід викликати після налаштування всіх властивостей вікна
close()	Закриває вікно та зупиняє додаток. Це припинить виконання програми.

Основні елементи GUI

Кнопки (Button)

```
Button button = new Button("Натисни на мене!");
```

Зміна кольору фону:

```
button.setStyle("-fx-background-color: #3498db;");
```

Зміна розміру:

```
button.setPrefSize(100, 50);
```

Додавання іконки:

```
Image image = new Image("шлях_до_файлу_зображення.png");
ImageView imageView = new ImageView(image);
```

```
button.setGraphic(imageView);
```
Вирівнювання тексту та іконки:
```
button.setContentDisplay(ContentDisplay.TOP);
//або ContentDisplay.LEFT, ContentDisplay.RIGHT, ContentDisplay.BOTTOM
```
Активація / деактивація кнопки:
```
button.setDisable(true);
button.setDisable(false);
```

Текст (Label)
```
Label label = new Label("Привіт, світ!");
```
або
```
label.setText("Привіт, світ!");
```
Зміна кольору тексту:
```
label.setTextFill(Color.BLUE);
```
Зміна розміру тексту:
```
label.setFont(new Font("Arial", 20));
```
Вирівнювання тексту:
```
label.setTextAlignment(TextAlignment.CENTER);
```
Автоматичне перенесення тексту:
```
label.setWrapText(true);
```
Затінення тексту:
```
DropShadow dropShadow = new DropShadow();
dropShadow.setColor(Color.GRAY);
dropShadow.setOffsetX(2);
dropShadow.setOffsetY(2);
label.setEffect(dropShadow);
```
Міжрядковий інтервал:
```
label.setLineSpacing(5);
```

Однорядкові текстові поля (TextField)
```
TextField textField = new TextField();
```
Отримання тексту з поля:
```
String text = textField.getText();
```
Встановлення тексту за замовчуванням:
```
TextField textField = new TextField("Введіть текст тут");
```
або
```
textField.setText("Введіть текст тут");
```
Обмеження довжини введеного тексту:
```
int maxLength = 10;
```

```
    textField.textProperty().addListener((observable,
oldValue, newValue) -> {
        if (newValue.length() > maxLength) {
            textField.setText(oldValue);
        }
    });
```

Зміна розміру:

```
textField.setPrefSize(200, 30);
```

Багаторядкові текстові поля (TextArea)

```
TextArea textArea = new TextArea();
```

Встановлення тексту за замовчуванням:

```
textArea.setText("Це текст за замовчуванням.");
```

Отримання введеного тексту:

```
String inputText = textArea.getText();
```

Зміна розміру:

```
textArea.setPrefWidth(300);
textArea.setPrefHeight(200);
```

Зміна кількості рядків:

```
textArea.setPrefRowCount(3);
```

Встановлення режиму автоматичного перенесення тексту:

```
textArea.setWrapText(true);
```

Зміна шрифту та стилю тексту:

```
textArea.setFont(Font.font("Arial",FontWeight.BOLD, 14));
```

Зображення (ImageView)

```
Image image = new Image("шлях_до_файлу_зображення.png");
ImageView imageView = new ImageView(image);
```

Налаштування розміру зображення:

```
imageView.setFitWidth(200);
imageView.setFitHeight(150);
```

Зміна вирівнювання зображення:

```
imageView.setPreserveRatio(true);
```

Зміщення та обрізання зображення:

```
Rectangle2D viewportRect = new Rectangle2D(10, 10, 100,
100);
imageView.setViewport(viewportRect);
```

Поля для введення паролю (PasswordField)

```
PasswordField passwordField = new PasswordField();
```

Отримання тексту з поля:

```
String password = passwordField.getText();
```

Списки (ListView)

```
ListView<String> listView = new ListView<>();
```

Додавання елементів до списку:

```
listView.getItems().addAll("Елемент 1", "Елемент 2",
"Елемент 3");
```

Комбінований список (ComboBox)

```
ComboBox<String> comboBox = new ComboBox<>();
comboBox.getItems().addAll("Варіант 1", "Варіант 2",
"Варіант 3");
```

Чекбокси (CheckBox)

```
CheckBox checkBox = new CheckBox("Варіант 1");
CheckBox checkBox = new CheckBox("Варіант 2");
```

Отримання стану чекбоксу (вибрано / невибрано):

```
boolean isSelected = checkBox.isSelected();
```

Перемикач (RadioButton)

```
RadioButton radioButton1 = new RadioButton("Варіант 1");
RadioButton radioButton2 = new RadioButton("Варіант 2");
```

Додавання до групи (група дозволяє вибирати тільки один перемикач з групи):

```
ToggleGroup toggleGroup = new ToggleGroup();
radioButton1.setToggleGroup(toggleGroup);
radioButton2.setToggleGroup(toggleGroup);
```

Сповіщення (Alert)

```
Alert alert = new Alert(Alert.AlertType.INFORMATION);
alert.setTitle("Інформація");
alert.setHeaderText(null);
alert.setContentText("Це повідомлення з інформацією.");
alert.showAndWait();
```

Основні графічні елементи

Використовуючи бібліотеку JavaFX, можна малювати попередньо визначені форми, такі як лінія, прямокутник, коло, еліпс, багатокутник, ламана лінія, кубічна крива, чотирикутник, дуга, а також такі елементи шляху, як елемент шляху MoveTO, лінія, горизонтальна лінія, вертикальна лінія, кубічна крива, квадратична крива, дуга.

Line (Лінія)

```
Line line = new Line(x1, y1, x2, y2);
```

або

```
line.setStartX(x1);
line.setStartY(y1);
line.setEndX(x2);
line.setEndY(y2);
```

де (x1, y1) - координати початкової точки, а (x2, y2) - координати кінцевої точки.

Rectangle (Прямокутник)

```
Rectangle rectangle = new Rectangle(x, y, width, height);
```

де (x, y) - координати верхнього лівого кута, width - ширина, height - висота.

Circle (Коло)

```
Circle circle = new Circle(centerX, centerY, radius);
```

де (centerX, centerY) - координати центра кола, radius - радіус.

Ellipse (Еліпс)

```
Ellipse ellipse = new Ellipse(centerX, centerY, radiusX, radiusY);
```

де (centerX, centerY) - координати центра еліпса, radiusX - радіус великої осі, radiusY - радіус малої осі.

Polygon (Многокутник)

```
Polygon polygon = new Polygon();
polygon.getPoints().addAll(x1, y1, x2, y2, x3, y3, ...);
```

де (x1, y1), (x2, y2), (x3, y3), ... - координати вершин многоугольника.

Polyline (Ламана)

```
Polyline polyline = new Polyline();
polyline.getPoints().addAll(x1, y1, x2, y2, x3, y3, ...);
```

де (x1, y1), (x2, y2), (x3, y3), ... - координати точок ламаної.

Arc (Дуга)

```
Arc arc = new Arc(centerX, centerY, radiusX, radiusY, startAngle, length);
```

де (centerX, centerY) - координати центра кола, radiusX - радіус великої осі, radiusY - радіус малої осі, startAngle - початковий кут дуги, length - довжина дуги.

Для кожної фігури можна встановити (на прикладі прямокутника):

- колір заповнення:
  rectangle.setFill(Color.BLUE);
- колір границі:
  rectangle.setStroke(Color.BLACK);
- товщину границі:
  rectangle.setStrokeWidth(1);

Організація елементів у вікні

Розміщувати елементи у вікні JavaFX можна, вказуючи координати X та Y у вікні. Зверніть увагу, у Java початок координат знаходиться у верхньому лівому куті вікна. Це правило є актуальним для більшості віконних інтерфейсів, включаючи JavaFX, Swing та багато інших графічних бібліотек для розробки інтерфейсу користувача в Java. Початок координат у верхньому лівому куті є стандартом, який використовується в багатьох платформах та інструментах розробки.

Координата (0, 0) відповідає верхньому лівому куту вікна. Вісь X збільшується вправо, а вісь Y збільшується вниз (рис. 1.8).

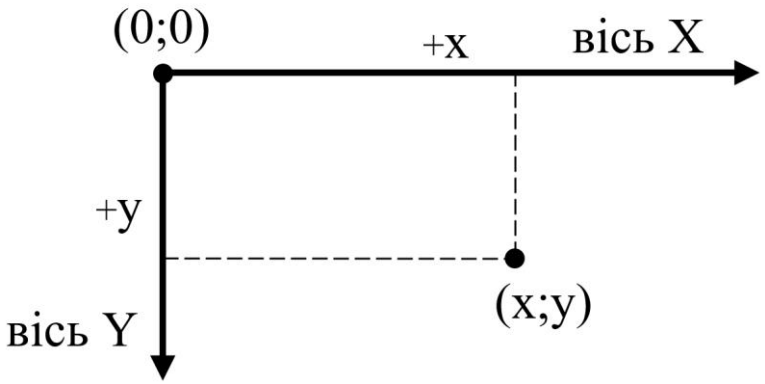

Рисунок 1.8 - Система координат у вікнах JavaFX

Для розміщення та організації графічних елементів у вікні у JavaFX також існують спеціальні класи-контейнери, які є підкласами класу javafx.scene.layout.Pane (табл. 1.2).

Таблиця 1.2 - Основні види контейнерів JavaFX

StackPane	Контейнер, який використовується для розміщення елементів один на одному. Всі елементи розташовуються один на одному у стековому порядку
HBox	Горизонтальний контейнер, в якому елементи розташовуються в одному рядку
VBox	Вертикальний контейнер, в якому елементи розташовуються в одному стовпчику
BorderPane	Контейнер, який розділяє вікно на п'ять областей: верхню, нижню, ліву, праву та центральну.
GridPane	Контейнер, який організовує елементи у вигляді сітки з рядками та стовпцями
FlowPane	Контейнер, який розміщує елементи один за одним у рядок або стовпчик, автоматично переходячи до нового рядка або стовпця при необхідності
TilePane	Контейнер, який розміщує елементи один за одним у рядок або стовпчик з фіксованим розміщенням

Схематично зазначені типи контейнерів представлені на рис. 1.9.

а б в г

ґ д е

Рисунок 1.9 - Основні типи контейнерів Pane:
а) - StackPane б) - HBox в) - VBox г) - BorderPane
ґ) - GridPane д) - FlowPane е) - TilePane

Додавати елементи до більшості контейнерів (StackPane, HBox, VBox, FlowPane, TilePane) можна за допомогою методу:

```
getChildren().add(Node),
```

де Node - елемент, який треба додати.

У деяких контейнерах метод add має більше аргументів, наприклад, GridPane:

```
grid.add(Node, columnIndex, rowIndex),
```

де Node - елемент, який ви хочете додати, columnIndex - індекс стовпця, у якому потрібно розмістити елемент, rowIndex - індекс рядка, у якому потрібно розмістити елемент.

В BorderPane можна додати дочірні елементи за допомогою методів:

```
setTop(Node),
setBottom(Node),
setLeft(Node),
setRight(Node),
setCenter(Node).
```

Налаштувати вирівнювання дочірніх елементів можна за допомогою властивості alignment методом setAlignment. Наприклад:

```
grid.setAlignment(Position),
```

де Position - Pos.CENTER, Pos.TOP_LEFT, Pos.TOP_CENTER, Pos.TOP_RIGHT, Pos.BOTTOM_LEFT, Pos.BOTTOM_CENTER, Pos.BOTTOM_RIGHT, Pos.BASELINE_LEFT, Pos.BASELINE_CENTER або Pos.BASELINE_RIGHT.

В багатьох контейнерах є методи, які дозволяють налаштувати відстань між дочірніми елементами:

setSpacing(double value) - встановлює відстань між елементами в контейнері.

setHgap(double value) - встановлює горизонтальний проміжок між елементами в контейнері.

setVgap(double value) - встановлює вертикальний проміжок між елементами в контейнері.

Детальний опис та методи всіх класів контейнерів та інших елементів пакету javafx.scene.layout можна знайти в офіційній документації Oracle за посиланням:

https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/package-summary.html

Найчастіше використовуються HBox та VBox, а також BorderPane або GridPane. На практиці декілька контейнерів можуть використовуватися одночасно в одному вікні, вкладені один в одного.

Щоб створені елементи відобразилися у вікні, необхідно додати їх до контейнеру, а контейнер - до сцени. Потім необхідно встановити сцену у головне вікно та викликати метод головного вікна show(). Наприклад:

```
…
public void start(Stage stage) {
     stage.setTitle("Форма для відгуку");
     VBox vbox = new VBox();

     Label text1 = new Label("Залиште свій відгук:");
     text1.setFont(Font.font("Arial", FontWeight.BOLD, 12));
     vbox.getChildren().add(text1);

     GridPane grid = new GridPane();
     Label username = new Label("Ім'я:");
     grid.add(username, 0, 0);

     TextField text = new TextField();
     text.setPrefWidth(250);
     grid.add(text, 1, 0);
     Label comment = new Label("Відгук:");
     grid.add(comment, 0, 1);

     TextArea textArea = new TextArea();
     textArea.setPrefWidth(250);
     textArea.setPrefRowCount(3);
     textArea.setWrapText(true);
     grid.add(textArea, 1, 1);
     vbox.getChildren().add(grid);
```

```
        Button button = new Button("Відправити");
        vbox.getChildren().add(button);
        Scene scene = new Scene(vbox);
        stage.setScene(scene);
        stage.show();
}
```

Цей код створює простий графічний інтерфейс з заголовком "Форма для відгуку". Вікно містить контейнер VBox для вертикального розташування елементів, в якому є текстовий напис, сітка GridPane для розміщення елементів в рядки і стовпці, в якій розміщено поруч два написи на два текстових поля. Останній елемент в контейнері VBox - кнопка "Відправити".

Після створення кожного елемента вони додаються в контейнер VBox, який потім використовується для створення сцени (Scene). Сцена ініціалізується з контейнером VBox як вмістом. Потім сцена встановлюється для об'єкта Stage (вікна), і вікно показується (рис. 1.10).

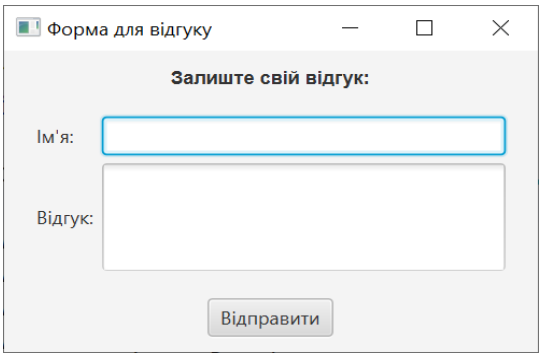

Рисунок 1.10 - Створене вікно з графічним інтерфейсом

3. Контрольні питання

1. З чого складається графічний інтерфейс вікна JavaFX?
2. Перерахуйте основні елементи GUI.
3. Де розташований початок координат у вікні JavaFX?
4. Які є основні види контейнерів JavaFX?
5. Опишіть алгоритм додавання елементу інтерфейсу до вікна.

4. Правила техніки безпеки та охорони праці

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

5. Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

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

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

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java в інтегрованому середовищі розробки Eclipse або IntelliJ IDEA.

6. Завдання до лабораторної роботи

Вивчити обране середовище розробки (Eclipse або IntelliJ IDEA). Створити вікно JavaFX з заголовком “Лаб. робота 1 | Ваше_Прізвище”, в якому повинні бути наступні елементи:

- зображення розміром 150x150 пікселів,
- написи з вашим ПІБ та групою,
- невелика форма з однорядковим текстовим полем, багаторядковим текстовим полем, групою з трьома перемикачами та кнопкою. Всі елементи повинні супроводжуватися написами, які їх пояснюють.
- Текстовий напис (місце для виведення повідомлень).

Всі елементи вікна повинні бути впорядковані за допомогою відповідних контейнерів.

Приклад створеного вікна наведено на рис. 1.11.

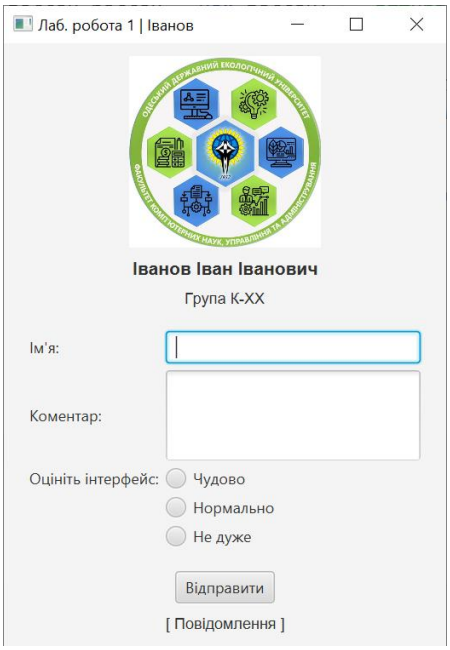

Рисунок 1.11 - Приклад інтерфейсу JavaFX

7. Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

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

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

Лабораторна робота №2
Обробка семантичних подій в Java

1. Мета роботи

Метою роботи є придбання навичок обробки семантичних подій в Java.

Після виконання лабораторної роботи студенти мають оволодіти уміннями обробляти події графічного інтерфейсу програми.

2. Теоретичні відомості до виконання лабораторної роботи

Семантичні події - це події, які мають спеціальне значення або семантику в контексті застосунку або системи. Вони називаються "семантичними", оскільки вони не просто вказують на факт виникнення події, але також надають інформацію про її сутність або контекст.

У контексті графічних інтерфейсів користувача, семантичні події можуть бути пов'язані зі змінами в стані або взаємодії користувача з елементами інтерфейсу. Наприклад, натискання кнопки, введення тексту у поле введення або переміщення курсору миші над певним елементом інтерфейсу можуть бути розглянуті як семантичні події.

У JavaFX обробка подій здійснюється за допомогою інтерфейсу EventHandler (JavaFX).

Інтерфейс EventHandler визначений у пакеті javafx.event і може бути прикріплений до елементів JavaFX для обробки специфічних подій, таких як натискання кнопки, клік мишею і т. д. Вимагає реалізації методу handle, який викликається при виникненні події.

Принцип використання інтерфейсу EventHandler в JavaFX:

1. Визначення об'єкту, який буде реагувати на події. Цей об'єкт повинен реалізовувати інтерфейс EventHandler<T>, де T - це тип події, на яку ви хочете реагувати.

2. Встановлення обробників подій. Після створення об'єкту слухача необхідно встановити його як обробник подій для відповідних елементів інтерфейсу за допомогою методів setOn<Event>(). Наприклад, setOnMouseClicked, setOnKeyPressed, тощо. Кожен із цих методів очікує об'єкт типу EventHandler<T>.

3. Реалізація методу handle. Об'єкт-обробник подій повинен містити реалізацію методу handle(T event). Цей метод буде викликаний автоматично при виникненні відповідної події.

Найчастіше одразу створюється анонімний клас, який імплементує інтерфейс EventHandler<ActionEvent>, і в ньому перевизначається метод handle().

Наприклад:

```
button.setOnAction(new EventHandler<ActionEvent>() {
     @Override
     public void handle(ActionEvent event) {
          // дії при натисканні кнопки
     }
});
```

В даному прикладі button - це об'єкт кнопки в JavaFX, до якого ми прикріплюємо обробник подій. setOnAction - це обробник події, яка є специфічною для елемента управління, наприклад, введення Enter в текстовому полі або в даному випадку натискання на кнопку.

new EventHandler<ActionEvent>(){...} - це анонімний клас, який реалізує інтерфейс EventHandler<ActionEvent>. ActionEvent є типом події, на яку ми реагуємо (тобто подія "натискання кнопки").

Анотація @Override позначає те, що метод public void handle(ActionEvent event){...}із інтерфейсу EventHandler перевизначається.

У випадку обробки подій в JavaFX зручно визначати обробники подій за допомогою лямбда виразів, тобто без необхідності створювати окремий клас або імплементувати інтерфейс. Завдяки лямбда-виразам, код стає більш компактним і зрозумілим.

Лямбда-вирази мають наступний синтаксис:

```
(parameters) -> {
    // тіло лямбда-виразу
}
```

Наприклад, можна переписати код обробки події натискання на кнопку (ActionEvent) за допомогою лямбда-виразу наступним чином:

```
button.setOnAction(event -> {
    // дії при натисканні кнопки
});
```

Обробник події натискання на елемент (MouseClicked Event)

```
element.setOnMouseClicked(event -> {
          // дії при натисканні на елементі
});
```

Обробник події наведення курсору на елемент (HoverEvent):

```
element.setOnMouseEntered(event -> {
          // дії при наведенні курсору на елемент
});
```

Обробник події виходу курсору миші з елемента (MouseExited Event):

```
element.setOnMouseExited(event -> {
          // дії при виході курсору миші з елемента
});
```

Обробник події зміни значення текстового поля (TextField):

```
textField.setOnKeyTyped(event -> {
          // дії при зміні значення текстового поля
});
```

Обробник події зміни значення чекбокса (CheckBox):

```
checkBox.setOnAction(event -> {
     // дії при зміні значення чекбокса
});
```

Обробник події зміни вибраного RadioButton в групі:

```
toggleGroup.selectedToggleProperty().addListener(
(observable, oldValue, newValue) -> {
     if (newValue != null) {
       // дії при зміні вибраного RadioButton в групі
       // RadioButton selected = (RadioButton) newValue;
       // System.out.print(selected.getText());
    }
});
```

В цьому прикладі oldValue - це попередній вибраний RadioButton в групі, newValue - нове значення властивості selectedToggleProperty, яке виникло після зміни, тобто новообраний RadioButton.

3. Контрольні питання

1. Що таке семантичні події? Наведіть приклади.
2. Який існує інтерфейс в JavaFX для обробки подій графічного інтерфейсу?
3. Які методи використовуються для встановлення обробників подій?
4. Як можна перевизначити метод handle?

4. Правила техніки безпеки та охорони праці

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

5. Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

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

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

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java в інтегрованому середовищі розробки Eclipse або IntelliJ IDEA.

6. Завдання до лабораторної роботи

Додати обробник подій до створеного у лабораторній роботі №1 вікна JavaFX. Змінити його заголовок на “Лаб. робота 2 | Ваше_Прізвище”.

Варіанти:

1. При натисканні на кнопку показати інформаційне сповіщення з текстом з однорядкового поля.
2. При натисканні на кнопку показати інформаційне сповіщення з текстом з багаторядкового поля.
3. При натисканні на кнопку показати інформаційне сповіщення з обраним варіантом з групи перемикачів.
4. При натисканні на кнопку змінити напис повідомлення на текст з однорядкового поля.
5. При натисканні на кнопку змінити напис повідомлення на текст з багаторядкового поля.
6. При натисканні на кнопку змінити напис повідомлення на текст обраного варіанта з групи перемикачів.
7. При наведенні курсора на кнопку змінити напис повідомлення на текст “Наведено на кнопку”, а при виведення з кнопки - текст “Виведено з кнопки”.

8. При зміні значення однорядкового текстового поля відображати зміни в написі повідомлення.
9. При зміні значення багаторядкового текстового поля відображати зміни в написі повідомлення.
10. При натисканні на зображення змінити напис повідомлення на короткий опис зображеного.

7. Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

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

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

Лабораторна робота №3
Використання потоків і створення анімації в Java

1. Мета роботи

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

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

2. Теоретичні відомості до виконання лабораторної роботи

Багато сучасних мов програмування мають підтримку потоків або сторонні бібліотеки для роботи з потоками для виконання багатозадачних або асинхронних операцій. Мова Java також містить засоби підтримки потоків. Їх можна застосувати в будь-якій програмі при необхідності паралельного виконання декількох завдань.

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.

У Java окремий потік виконання представлений класом java.lang.Thread. Він надає базовий функціонал для керування потоком, такий як створення, запуск і зупинка потоку.

Реалізація використання потоків Thread в програмах мовою Java може виконуватися двома способами:

- розширенням класу Thread;
- реалізацією інтерфейсу Runnable.

При першому способі клас стає потоковим, якщо він створений як розширення класу Thread, наприклад:

```
public class MyClass extends Thread {
     public void run() {
        // Код, який виконується у потоці
    }
}
```

Однак, Java не підтримує множинне наслідування, тому якщо клас вже успадковує інший клас, не можна створити підклас, розширяючи

Thread. Наприклад, оскільки застосунок JavaFX завжди повинен наслідувати клас Application, він не може наслідувати ще й клас Thread.

Для вирішення цієї проблеми для даного класу можна реалізувати інтерфейс Runnable.

Runnable - це інтерфейс у Java, який визначає один метод run(). Цей метод містить код, який виконується у потоці. Коли створюється потік із Runnable, метод run() імплементується у класі, який реалізує інтерфейс Runnable, і цей код буде виконуватися у окремому потоці.

Таким чином можна створити потік, використовуючи клас Thread, і передати об'єкт, що реалізує інтерфейс Runnable, у конструктор потоку. Потім метод run() з інтерфейсу Runnable буде викликаний у створеному потоці. Такий підхід дозволяє розділити логіку виконання коду у потоці від логіки керування потоком.

Створення класу, який реалізує інтерфейс Runnable:

```
public class MyRunnable implements Runnable {
     public void run() {
          // Код, який виконується у потоці
     }
}
```

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 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.

Наприклад, щоб створити об'єкт потоку і передати об'єкт створеного класу MyRunnable у конструктор потоку, можна написати:

```
Thread thread = new Thread(new MyRunnable());
thread.start();
```

Після виклику методу start(), створений потік виконає метод run() з об'єкта MyRunnable у окремому потоці.

Метод public void start() кидає виняток IllegalThreadStateException, 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
```

Отримання строкового представлення потоку, включаючи ідентифікатор, ім'я потоку, пріоритет і ім'я групи:

```
String threadString = thread.toString();
//Приклад результату: Thread[#36,Thread-3,5,main]
```

Отримання імені потоку:

```
String threadName = thread.getName();
```

Встановлення імені потоку:

```
thread.setName("Мій потік");
```

Якщо потрібно, щоб перед продовженням роботи потік чекав визначений час, можна використовувати метод sleep(long millis). Метод sleep() дуже часто використовується в циклах, керуючих анімацією.

Для забезпечення послідовного виконання потоків використовується метод join(). Виклик методу join() для певного потоку призводить до того, що поточний потік (тобто потік, у якому викликається join()) буде блокований до тих пір, поки не завершиться потік, для якого викликається join().

Синтаксис методу join():

public final void join() throws InterruptedException

Виклик join() без аргументів призводить до блокування поточного потоку до завершення потоку, для якого викликається join().

Метод join(long millis) призводить до блокування поточного потоку протягом заданого часу millis (у мілісекундах) або до завершення потоку, для якого викликається join(). Якщо вказаний час вичерпується до завершення потоку, блокування знімається, і потік може продовжити виконання.

Приклад блокування потоків методом join():

```
public class ThreadJoinExample {
     public static void main(String[] args) {
          Thread thread1 = new Thread(new
     MyRunnable("Потік 1"));
          Thread thread2 = new Thread(new
     MyRunnable("Потік 2"));

          thread1.start();
          thread2.start();

          try {
               thread1.join(); // Головний потік чекає на
          завершення потоку 1
               thread2.join(2000); // Головний потік
          чекає на завершення потоку 2 не більше ніж 2
          секунди
          } catch (InterruptedException e) {
               e.printStackTrace();
          }

          System.out.println("Виконання головного потоку
     завершено");
    }
}

class MyRunnable implements Runnable {
    private String name;

    public MyRunnable(String name) {
        this.name = name;
    }
```

```
        public void run() {
             System.out.println("Початок виконання потоку "
        + name);
             try {
                  Thread.sleep(3000);
                  // Імітація довгої операції
             } catch (InterruptedException e) {
                  e.printStackTrace();
             }

             System.out.println("Завершення виконання потоку
         " + name);
        }
    }
```

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

Для перевірки стану потоку використовується метод isAlive(). Він повертає true, якщо потік ще виконується, або false, якщо потік вже завершив виконання або не був ще запущений.

Потоки в Java можуть переривати один одного. Переривання потоків в Java використовується для відправки сигналу потоку про необхідність зупинки або переривання його виконання. Це досягається за допомогою методу interrupt() класу Thread.

Коли потік переривається, йому встановлюється прапорець переривання (interrupt flag).

Потік може перевіряти свій стан переривання за допомогою методу isInterrupted() або interrupted(). Якщо перевірка переривання відбувається за допомогою методу interrupted(), потік буде зупинений або перерваний. Цей метод очищає ознаку переривання для потоку, тобто при повторному виклику методу для цього ж перерваного потоку він поверне значення false.

Якщо потік знаходиться у стані блокування, такому як очікування (wait(), sleep(), join()), переривання може викликати виняток InterruptedException.

Код потоку може вибрати, як обробляти переривання, і реалізувати відповідну логіку.

Звичайно програма 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, програма Java буде виконуватися до тих пір, поки не завершиться останній потік, який не є демоном. Потоки, що не помічені як демони, називаються користувальницькими потоками (user threads).

Щоб потік вважався демоном, треба скористатися методом: setDaemon(boolean on). Якщо параметр on дорівнює true, потік отримує статус демона, якщо false – статус користувальницького потоку. Статус потоку може бути змінений в процесі його виконання.

Метод isDaemon() повертає true, якщо потік є демоном, і false, якщо це користувальницький потік.

Метод enumerate(Thread[] threadArray) 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 activeCount() повідомляє, скільки активних потоків в групі, до якої належить цей потік.

Пріоритети та групи потоків.

Розподіл процесорного часу між потоками в Java виконується за такими правилами: коли потік блокується, тобто припинений, переходить в стан очікування або повинен дочекатися якоїсь події, Java вибирає інший потік з тих, які готові до виконання. Вибирається потік, що має найбільший пріоритет. Якщо таких кілька, вибирається будь-який з них. Однак хоча потік з максимальним пріоритетом має більшу ймовірність виконуватись раніше, немає гарантії, що він завжди буде виконуватись першим. Порядок виконання потоків може варіюватись залежно від контексту виконання та алгоритму планування, який використовується в системі.

Пріоритет потоку можна встановити методом setPriority(int newPriority).

Пріоритет потоку повинен бути числом в діапазоні від Thread.MIN_PRIORITY до Thread.MAX_PRIORITY. Будь-яке значення поза цими межами викликає виключення IllegalArgumentException.

За замовчуванням потоку приписується пріоритет Thread.NORM_PRIORITY.

Значення пріоритету потоку можна визначити за допомогою методу getPriority().

Клас ThreadGroup реалізує стратегію забезпечення безпеки, яка дозволяє впливати один на одного тільки потокам з однієї групи. Наприклад, потік може змінити пріоритет іншого потоку з тієї ж групи або перевести його в стан очікування. Якби не було розбиття потоків на групи, один потік міг би викликати хаос в середовищі Java, перевівши в стан очікування всі інші потоки, або, що ще гірше, завершивши їх.

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

```
ThreadGroup thGroup = new ThreadGroup("Нова група");
```

Можна також створити групу потоків, дочірню по відношенню до існуючої, використовуючи конструктор:

```
public ThreadGroup(ThreadGroup existingGroup, String groupName) throws NullPointerException
```

Строкове представлення групи:

```
String grInfo = thGroup.toString();
```

Отримання імені групи:

```
String grName = thGroup.getName();
```

Встановлення імені групи:

```
thGroup.setName("Ім’я групи");
```

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

Число активних потоків в даній групі:

```
public int activeCount()
```

Число активних груп в даній групі:

```
public int activeGroupCount()
```

Перевірка, чи може потік, що виконується в даний час, модифікувати дану групу:

```
public final void checkAccess()
```

Переривання всіх потоків в даній групі:

```
public final void interrupt()
```

Перевірка і встановлення ознаки потоку-демона для всієї групи:

```
public final boolean isDaemon()
public final void setDaemon(boolean daemon)
```

Обмеження пріоритету потоків з групи:

```
public final synchronized void setMaxPriority(int priority)
```

Визначення максимального пріоритету потоку в групі:

```
public final int getMaxPriority()
```

Визначення батьківської групи потоків:

```
public final ThreadGroup getParent()
```

Перевірка, чи є група g батьківської для даної групи:

```
public final boolean parentOf (ThreadGroup g)
```

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 (object) { //блокування
     // оператори для синхронізації
} // зняття блокування
```

Весь метод може бути позначений як synchronized, щоб гарантувати, що виконання цього методу буде атомарним для всіх потоків, які викликають його на тому ж об'єкті. Коли потік викликає синхронізований метод, він блокується до того часу, поки не отримає доступ до методу.

```
public synchronized void synchronizedMethod() {
     // код методу
}
```

Критична ділянка може включати тільки певну частину методу. Тоді відповідний блок коду може бути відмічений як synchronized. Ключовим блоком є об'єкт, на якому виконується синхронізація.

```
public void simpleMethod() {
     // Несинхронізований код
     synchronized (lockObject) {
          // Синхронізований блок коду
```

```
        }
        // Несинхронізований код
    }
```

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:

Зупинення виконання поточного потоку до отримання повідомлення:

public final void wait() throws InterruptedException, IllegalMonitorStateException

Зупинення виконання поточного потоку до отримання повідомлення або до закінчення заданого інтервалу часу timeout (в мілісекундах):

public final void wait(long timeout) throws InterruptedException, IllegalMonitorStateException, IllegalArgumentException

Метод 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 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.

Подвійна буферизація зображень.

Іноді в процесі роботи із зображеннями може виникнути ефект мерехтіння. Це відбувається через невелику затримки між запуском методу прорисовки і моментом, коли зображення дійсно завантажено. Ефект

мерехтіння можна усунути використовуючи технологію подвійної буферизації зображень.

Основна ідея подвійної буферизації полягає в тому, щоб мати два полотна: одне для малювання нового кадру, інше для відображення поточного кадру.

Подвійна буферизація вже вбудована в JavaFX та автоматично використовується при малюванні на Canvas або інших елементах графічного інтерфейсу.

Canvas в JavaFX - це клас, що представляє площину для малювання графічних об'єктів.

Загальний процес використання Canvas включає створення його екземпляра, отримання GraphicsContext, малювання на ньому графічних об'єктів за допомогою методів GraphicsContext і розміщення Canvas на сцені або у вікні за допомогою інших контейнерів JavaFX.

```
Canvas canvas = new Canvas(WIDTH, HEIGHT);
GraphicsContext gc = canvas.getGraphicsContext2D();
StackPane root = new StackPane(canvas);
Scene scene = new Scene(root);
primaryStage.setScene(scene);
primaryStage.show();
```

Цей код створює вікно з одним Canvas в контейнері типу StackPane, на якому можна малювати графічні об'єкти.

Наприклад, можна створити анімацію з масиву зображень (Image[] frames) в окремому потоці.

```
class ImageAnimation implements Runnable{
     @Override
     public void run() {
          while (true) {
               gc.clearRect(0, 0, WIDTH, HEIGHT);
               gc.drawImage(frames[frameIndex], 0, 0,
WIDTH, HEIGHT);
               try {
                    Thread.sleep(500);
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
          frameIndex = (frameIndex + 1) % frames.length;
          }
     }
}
```

Та створити і запустити новий потік в start():

```
Thread thread = new Thread(new ImageAnimation());
thread.start();
```

В даному випадку в кожній ітерації нескінченного циклу виконується очищення заданої прямокутної області на Canvas за допомогою методу gc.clearRect. Після виклику clearRect() область стає чистою (transparent) і готовою для нового візуального відображення. Далі виконується малювання наступного зображення з масиву зображень.

Кожний раз при виклику метода drawImage() внутрішній буфер автоматично оновлюється. Потім вміст буфера автоматично копіюється на екран у відповідний момент. Thread.sleep() використовується для налаштування необхідної частоти кадрів.

Оновлення інтерфейсу у багатопоточній програмі

У JavaFX графічний інтерфейс повинен бути оновлюваний лише з основного потоку заходів. Якщо ви хочете оновлювати графічний інтерфейс з іншого потоку, вам потрібно використовувати Platform.runLater() для виконання цієї операції з основного потоку JavaFX. Наприклад,

```
public class Example extends Application {
     @Override
     public void start(Stage primaryStage) {
          StackPane root = new StackPane();
          Label label = new Label("Напис");
          root.getChildren().add(label);
          Thread     thread     =     new     Thread(new
     MyRunnable(label));
          thread.start();
          Scene scene = new Scene(root, 300, 200);
          primaryStage.setScene(scene);
          primaryStage.show();
     }

     private static class MyRunnable implements Runnable{

          private final Label label;

          public MyRunnable(Label label) {
               this.label = label;
          }
```

```
            @Override
            public void run() {
                 Platform.runLater(() ->
            label.setText("Новий напис"));
            }

        }

        public static void main(String[] args) {
            launch(args);
        }
}
```

При запуску додатку на екрані з'являється вікно з надписом "Напис". Після цього створюється новий потік, який виконує клас MyRunnable. В конструктор MyRunnable передається посилання на мітку label.

В методі run() класу MyRunnable використовується метод Platform.runLater(), який забезпечує безпечний доступ до графічного інтерфейсу з основного потоку JavaFX. Використовуючи лямбда-вираз, змінюється текст мітки на "Новий напис". Це відбувається з іншого потоку, але безпечно згідно з правилами JavaFX.

3. Контрольні питання

1. Якими способами можна створити потоки в мові програмування Java?
2. Яким чином можна забезпечити послідовне виконання потоків в програмі?
3. Якими методами можна керувати виконанням потоків у Java?
4. Яким чином Java розподіляє процесорний час між потоками під час виконання багатопотокових програм?
5. Як можна встановити пріоритет потоку?
6. В якому потоці можна оновлювати графічний інтерфейс JavaFX? Який метод можна використовувати, щоб оновити інтерфейс з іншого потоку?

4. Правила техніки безпеки та охорони праці

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

5. Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

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

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

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java в інтегрованому середовищі розробки Eclipse або IntelliJ IDEA.

6. Завдання до лабораторної роботи

Створити вікно JavaFX з заголовком “Лаб. робота 3 | Ваше_Прізвище”. Створити графічний застосунок, який імітує гонки машин. На екрані повинні бути кілька учасників гонок, що рухаються по дорозі. Можна представити машини різнокольоровими прямокутниками, а дорогу - сірим фоном. Кожен учасник має бути окремим потоком. Швидкість руху можна встановлювати випадковим чином за допомогою функції Math.random(). Після завершення гонки (коли всі учасники будуть на фініші) за допомогою окремого потоку, який очікує завершення всіх потоків учасників, виведіть повідомлення: “Гонку завершено!”.

7. Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

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

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

Лабораторна робота №4
Програмування потоків вводу/виводу

1. Мета роботи

Метою роботи є отримання практичних навичок роботи використання потоків вводу-виводу.

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

2. Теоретичні відомості до виконання лабораторної роботи

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

Клас File

Клас File дозволяє отримати від системи всі дані, починаючи з імені файлу і закінчуючи часом останньої його модифікації. Клас File можна використовувати для створення нових каталогів, а також для видалення і перейменування файлів. Для створення об'єкта File можна 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 табл. 2.1.

Таблиця 2.1 - Основні методи для роботи з об’єктами класу File

boolean createNewFile()	Створює новий файл. Повертає true, якщо файл створений успішно, і false, якщо файл

	вже існує або сталася помилка.
boolean mkdir()	Створює новий каталог. Повертає true, якщо каталог створений успішно, і false, якщо каталог вже існує або сталася помилка.
boolean mkdirs()	Створює новий каталог разом з усіма проміжними каталогами, які не існують. Повертає true, якщо каталог(и) створені успішно, і false, якщо сталася помилка
boolean renameTo(File dest)	Перейменовує файл або каталог на задане ім'я, вказане параметром dest. Повертає true, якщо перейменування відбулося успішно, і false в іншому випадку.
boolean delete()	Видаляє файл або каталог. Повертає true, якщо видалення відбулося успішно, і false, якщо сталася помилка.
boolean exists()	Перевіряє, чи існує файл або каталог.
boolean isFile()	Перевіряє, чи є даний об'єкт файлом.
boolean isDirectory()	Перевіряє, чи є даний об'єкт каталогом.
long length()	Повертає розмір файлу в байтах.
String getName()	Повертає ім'я файлу або каталогу.
String getParent()	Повертає ім'я батьківського каталогу.
String getPath()	Повертає відносний шлях до файлу або каталогу.
String getAbsolutePath()	Повертає абсолютний шлях до файлу або каталогу.
boolean canRead()	Перевіряє, чи можна читати файл.
boolean canWrite()	Перевіряє, чи можна записувати в файл.
boolean setReadable(boolean readable)	Встановлює або знімає право на читання файлу.
boolean	Встановлює або знімає право на запис у файл.

setWritable(boolean writable)	
boolean setExecutable(boolean executable)	Встановлює або знімає право на виконання файлу.
long lastModified()	Повертає час останньої зміни файлу в мілісекундах, починаючи з 1 січня 1970 року.
boolean setLastModified(long time)	Задає час останньої зміни файлу в мілісекундах.
boolean isHidden()	Перевіряє, чи є файл прихованим.
boolean setHidden(boolean hidden)	Встановлює або знімає прихованість файлу (доступно на деяких платформах).
String[] list()	Повертає масив імен файлів та підкаталогів у поточному каталозі.
String[] list (FilenameFilter filter)	Повертає масив імен всіх файлів, що задовольняють фільтру імен по інтерфейсу FilenameFilter.
File[] listFiles()	Повертає масив об'єктів класу File для файлів та підкаталогів у поточному каталозі.
File[] listFiles (FilenameFilter filter)	Повертає масив об'єктів класу File, що задовольняють фільтру імен по інтерфейсу FilenameFilter.

Метод list (FilenameFilter filter) та listFiles (FilenameFilter filter) приймають як параметр об'єкт класу, який реалізує інтерфейс типу FilenameFilter.

Цей інтерфейс визначає єдиний метод accept(), який має наступний вигляд:

boolean accept(File dir, String name)

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

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

boolean accept(File path)

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

Приклад використання цих методів для для отримання списку імен файлів та об'єктів File відповідно, які задовольняють критерію фільтрації (розширення ".java"):

```
import java.io.File;
import java.io.FilenameFilter;

public class JavaFileFilter implements FilenameFilter {
     @Override
     public boolean accept(File dir, String name) {
          return name.toLowerCase().endsWith(".java");
     }

     public static void main(String[] args) {
          File directory = new File("C:/example");
          JavaFileFilter filter = new JavaFileFilter();

          String[] filteredFilesNames =
     directory.list(filter);

          if (filteredFilesNames != null) {
               System.out.println("Список файлів з
          розширенням .java:");
               for (String fileName :
          filteredFilesNames){
                    System.out.println(fileName);
               }
          }

          File[] filteredFiles =
     directory.listFiles(filter);

          if (filteredFiles != null) {
```

```
                    System.out.println("\nСписок об'єктів File
               з розширенням .java:");
                    for (File file : filteredFiles) {
                         System.out.println(file.getName());
                    }
               }
          }
     }
```

Змінна separatorChar класу File повертає символ поділення імен каталогів і файлів ("\" в Windows і "/" – в Unix), а змінна pathSeparatorChar містить символ-роздільник шляхів у списку (";" в Windows і ":" – в Unix).

Вибір файлів в JavaFX

Для створення діалогового вікна для вибору каталогів і файлів в JavaFX використовується клас FileChooser з пакету javafx.stage. FileChooser дозволяє вибрати один або кілька файлів або каталогів з файлової системи.

Створення об’єкту FileChooser:

```
FileChooser fileChooser = new FileChooser();
```

Встановлення заголовку вікна:

```
fileChooser.setTitle("Виберіть файл або каталог");
```

Створення фільтрів для типів файлів, якщо потрібно:

```
FileChooser.ExtensionFilter txtFilter = new
FileChooser.ExtensionFilter("Текстові файли (*.txt)","*.txt");
FileChooser.ExtensionFilter imageFilter = new
FileChooser.ExtensionFilter("Зображення (*.jpg, *.png)",
"*.jpg", "*.png");
```

Додавання фільтрів:

```
fileChooser.getExtensionFilters().add(txtFilter);
fileChooser.getExtensionFilters().add(imageFilter );
```

або

```
fileChooser.getExtensionFilters().addAll(txtFilter,imageF
ilter);
```

Відображення діалогового вікна вибору файлу або каталогу:

```
File selectedFile = fileChooser.showOpenDialog(stage);
```

Відображення діалогового вікна вибору місця збереження файлу:

```
File selectedDir = fileChooser.showSaveDialog(stage);
```

Перевірка, чи користувач щось вибрав:

```
if (selectedFile != null) {
     System.out.println("Вибраний файл або каталог: " +
selectedFile.getAbsolutePath());
```

```
} else {
     System.out.println("Вибір скасовано.");
}
```

Важливо, що клас File працює лише з метаданими файлів та каталогів і не надає можливостей для фактичного читання або запису даних у файл. Для операцій зчитування і запису даних з файлів використовуються інші класи.

Організація вводу-виводу в Java

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

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

В Java є два основні типи потоків: байтові і символьні.

Байтові потоки (InputStream, OutputStream) працюють з байтами (byte) і є базовими для вводу-виводу даних у вигляді послідовності байтів. Наприклад, FileInputStream та FileOutputStream - це потоки, які дозволяють зчитувати та записувати дані у вигляді байтів. Ці потоки зручні для роботи з бінарними даними, такими як зображення, аудіо- або відеофайли.

Символьні потоки (Reader, Writer) працюють з символами (char) і дозволяють зчитувати та записувати тексти у вигляді послідовності символів. Вони зручні для роботи з текстовими даними, такими як текстові файли, рядки тощо. Наприклад, FileReader та FileWriter - це потоки, які дозволяють читати та записувати тексти.

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

Практично кожен метод класів в пакеті java.io здатний генерувати ту чи іншу форму виняткової ситуації IOException. Тому для поліпшення

стилю програмування слід завжди поміщати виклики операцій вводу/виводу до блоку try/catch.

Байтові потоки вводу/виводу

Класи InputStream та OutputStream

Абстрактний клас InputStream забезпечує базові функції всіх потоків вводу байтів.

Основні методи класу InputStream наведені в табл. 2.2.

Таблиця 2.2 - Основні методи класу InputStream

int read()	Читає наступний байт з потоку і повертає його у вигляді цілого числа (0-255) або -1, якщо потік досяг кінця даних.
int read(byte[] b)	Читає блок байтів з потоку та зберігає їх у масив b. Повертає кількість зчитаних байтів або -1, якщо потік досяг кінця даних.
int read(byte[] b, int off, int len)	Читає максимум len байтів з потоку та зберігає їх у масив b, починаючи з індексу off масиву. Повертає кількість зчитаних байтів або -1, якщо потік досяг кінця даних.
long skip(long n)	Пропускає n байтів у потоці.
int available()	Повертає кількість байтів, які можна зчитати без блокування.
void close()	Закриває потік та звільняє ресурси.
void mark(int readlimit)	Встановлює позначку в потоці на поточному місці.
void reset()	Повертає позицію потоку до останньої позначки, яка була встановлена методом mark().
boolean markSupported()	Перевіряє, чи підтримує потік можливість встановлення позначки (mark() і reset()).

Абстрактний клас OutputStream забезпечує базові функції всіх потоків виводу байтів.

Основні методи класу OutputStream наведені в табл. 2.3.

Таблиця 2.3 - Основні методи класу OutputStream

void write(int b)	Записує один байт у потік. Аргумент b повинен бути цілим числом (0-255), яке представляє байт для запису.
void write(byte[] b)	Записує весь масив байтів b у потік.
void write(byte[] b, int off, int len)	Записує len байтів із масиву b, починаючи з індексу off масиву, у потік.
void flush()	Виконує зміну буферизованих даних у вихідному потоці, переконуючись, що всі дані були записані.
void close()	Закриває потік та звільняє ресурси.

Важливо пам'ятати, що класи InputStream та OutputStream є абстрактними, тобто неможливо створити їх екземпляри напряму. Замість цього, необхідно використовувати їхні підкласи, які надають конкретну реалізацію для запису даних у відповідні джерела виводу. Наприклад, це класи FileInputStream і FileOutputStream, ByteArrayInputStream і ByteArrayOutputStream, BufferedInputStream і BufferedOutputStream.

Класи FileInputStream і FileOutputStream

Клас FileInputStream представляє потік читання байтів з файлу. Для використання FileInputStream спочатку потрібно створити об'єкт цього класу за допомогою одного з наступних конструкторів:

FileInputStream (String filepath)

FileInputSream(File file)

Клас FileInputStream реалізує методи класу InputStream: available(), close(), skip() і всі форми методу read(). Крім того, в цьому класі доданий метод getFD(), який повертає об’єкт FileDescriptor для відкритого файлу і метод finalize() для очищення зв'язку з файлом і виклику методу close().

Клас FileOutputStream представляє потік запису байтів в файл. Зазвичай використовуються такі конструктори цього класу:

FileOutputStream(String filePath)

FileOutputStream(File fileObj)

FileOutputStream(String filePath, boolean append)

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

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

Класи ByteArrayInputStream і ByteArrayOutputStream

Клас ByteArrayInputStream представляє потік вводу байтів, який зчитує дані з масиву байтів. Цей клас особливо корисний, коли потрібно зчитати дані з пам'яті або із заздалегідь відомого масиву байтів, а не з файлу чи мережі. Цей клас є зручним для тестування або роботи з даними у форматі байтів без необхідності створення файлу або сполучення з мережею.

Клас ByteArrayInputStream має два конструктора, кожен з яких використовує байтовий масив в якості джерела даних:

ByteArrayInputStream(byte array[])

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

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

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

Клас ByteArrayOutputStream представляє потік виводу байтів, який записує дані во внутрішній масив байтів (byte[]). Цей клас є зручним, коли

потрібно тимчасово зберігати дані у пам'яті перед подальшою обробкою або відправкою.

Клас ByteArrayOutputStream має два конструктора:

ByteArrayOutputStream()

ByteArrayOutputStream(int len)

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

Додатково до методів, успадкованих від базового класу OutputStream, клас ByteArrayOutputStream ще підтримує такі методи (табл. 2.4).

Таблиця 2.4 - Додаткові методи класу ByteArrayOutputStream

byte[] toByteArray()	Повертає масив байтів, який містить дані, що були записані у потік.
int size()	Повертає поточний розмір внутрішнього масиву байтів.
void reset()	Знищує всі дані у потоці і знову встановлює внутрішній масив байтів до початкового стану.
String toString()	Перетворює вміст буфера в рядок відповідно до правил кодування символів за замовчуванням.
void writeTo (OutputStream out)	Записує весь вміст потоку в інший потік.

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

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 byte[] buf класу BufferedInputStream містить внутрішній буфер вхідного потоку, а змінна int count містить лічильник кількості введених байт (ця величина змінюється від нуля до величини масиву size).

Буферизація полягає в тому, що BufferedInputStream зчитує дані з потоку вводу в буфер певного розміру, а потім дозволяє вашій програмі читати дані з буфера. Завдяки цьому операції зчитування можуть бути більш ефективними, оскільки не завжди потрібно читати дані безпосередньо з джерела вводу, а можна використовувати дані, що знаходяться в буфері.

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

BufferedOutputStream(OutputStream out)

BufferedOutputStream(OutputStream out, int size)

Перша форма створює об'єкт BufferedOutputStream з буфером за замовчуванням (512 байт), а друга - з буфером заданого розміру.

Змінна класу byte[] buf повертає вміст внутрішнього буфера, а int count – кількість виведених у буфер байт. Клас BufferedOutputStream реалізує методи write() запису одного байта і частини масиву байт, а також методи close() і flush() класу OutputStream.

Клас SequenceInputStream

Клас SequenceInputStream дозволяє об'єднувати кілька потоків вводу (InputStream) у послідовність, що дозволяє читати дані з них як з одного потоку.

Основна форма конструктора цього класу використовує в якості параметрів два об'єкти класу InputStream або його підкласів:

SequenceInputStream(InputStream firstStream, InputStream secondStream)

Клас також підтримує інші конструктори, які дозволяють об'єднувати більше ніж два потоки вводу.

Клас виконує запити читання першого об'єкта типу InputStream (параметр firstStream), поки він не закінчиться, і потім перемикається на другий (параметр secondStream). У свою чергу, використовуючи як одного з об'єктів об'єкт класу SequenceInputStream, можна організувати введення більш ніж двох об'єктів вхідного потоку.

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

Приклад створення двох потоків та читання з них даних після об’єднання за допомогою SequenceInputStream:

```
try {
     InputStream input1 = new FileInputStream("file1.txt");
     InputStream input2 = new FileInputStream("file2.txt");

     SequenceInputStream sequenceInput = new
SequenceInputStream(input1, input2);

     int data;
     while ((data = sequenceInput.read()) != -1) {
          System.out.print((char) data);
     }

     sequenceInput.close();
} catch (IOException e) {
     e.printStackTrace();
}
```

Символьні потоки вводу/виводу

Класи Reader і Writer

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

Абстрактний клас Reader надає основний функціонал для всіх підкласів, які реалізують різні способи читання символів з різних джерел.

Основні методи класу Reader наведені в табл. 2.5.

Таблиця 2.5 - Основні методи класу Reader

int read()	Читає наступний символ з потоку і повертає його у вигляді цілого числа (0-65535) або -1, якщо потік досяг кінця даних.
int read(char[] cbuf)	Читає безліч символів та зберігає їх у масив cbuf. Повертає кількість зчитаних символів або -1, якщо потік досяг кінця.
int read(char[] cbuf, int off, int len)	Читає максимум len символів з потоку та зберігає їх у масив cbuf, починаючи з індексу off масиву. Повертає кількість зчитаних символів або -1, якщо потік досяг кінця.
long skip(long n)	Пропускає n символів у потоці.
boolean ready()	Перевіряє, чи можна зчитати символ без блокування.
void close()	Закриває потік та звільняє ресурси.
void mark(int readAheadLimit)	Встановлює позначку в потоці на поточному місці.
void reset()	Повертає позицію потоку до останньої позначки, яка була встановлена методом mark().
boolean markSupported()	Перевіряє, чи підтримує потік можливість встановлення позначки (mark() і reset()).

Абстрактний клас Writer надає основний функціонал для всіх підкласів, які реалізують різні способи запису символів в різні джерела.

Основні методи класу Writer наведені в табл. 2.6.

Таблиця 2.6 - Основні методи класу Writer

void write(int c)	Записує один символ у потік. Аргумент c повинен бути цілим числом, що представляє код символу.
void write(char[] cbuf)	Записує масив символів cbuf у потік.

void write(char[] cbuf, int off, int len)	Записує len символів з масиву cbuf, починаючи з індексу off масиву, у потік.
void write(String str)	Записує рядок str у потік.
void write(String str, int off, int len)	void write(String str, int off, int len)
void flush()	Завершує запис даних у потік і забезпечує, щоб всі символи були записані.
void close()	Закриває потік і звільняє ресурси.

Класи Reader та Writer є абстрактними, тобто неможливо створити їх екземпляри напряму. Замість цього, необхідно використовувати їхні підкласи, які надають конкретну реалізацію відповідних методів.

Класи InputStreamReader і OutputStreamWriter

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

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

InputStreamReader (InputStream in)

Цей конструктор створює з байтового потоку in символьний потік відповідно до кодуванням за замовчуванням (кодування за замовчуванням залежить від реалізації віртуальної машини Java і встановлених для неї локальних значень).

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

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

OutputStreamWriter (OutputStream out)

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

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

Класи FileReader і FileWriter

Клас FileReader призначений для читання символів з текстового файлу. Для створення об'єкта класу FileReader можна використовувати один з наступних конструкторів:

FileReader (String fileName)

FileReader (File file)

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

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

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

FileWriter (String fileName)

FileWriter (File file)

FileWriter (String fileName, boolean append)

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

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

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

Приклад запису вмісту текстового файлу в інший файл за допомогою FileReader та FileWriter:

```
try {
     // Відкриваємо файл для читання
     FileReader fileReader = new FileReader("input.txt");
     // Відкриваємо файл для запису
     FileWriter fileWriter = new FileWriter("output.txt");

     // Читаємо та записуємо дані
     int character;
     while ((character = fileReader.read()) != -1) {
          // Записуємо символ в інший файл
          fileWriter.write(character);
```

```
        }

        // Закриваємо потоки
        fileReader.close();
        fileWriter.close();

        System.out.println("Файл успішно скопійовано.");
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Класи CharArrayReader і CharArrayWriter

Клас CharArrayReader дозволяє читати символи з масиву символів (char[]) як з потоку. Цей клас дозволяє зчитувати символи з масиву, що може бути корисним у випадку, коли потрібно прочитати текстові дані зі збережених у пам'яті даних. Для класу CharArrayReader визначені наступні конструктори:

CharArrayReader (char[] buf)

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

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

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

Клас CharArrayWriter дозволяє записувати символи у масив символів (char[]) так само, як у потік символів. Цей клас має два конструктора:

CharArrayWriter ()

CharArrayWriter (int initialSize)

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

Додатково до реалізації методів write() для запису одного символу, частини масиву символів і частини рядка, flush() і close() класу Writer, в класі CharArrayWriter реалізовані власні методи, що наведені в табл. 2.7.

Таблиця 2.7 - Додаткові методи класу CharArrayWriter

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

Клас PrintWriter

Клас PrintWriter дозволяє зручно виводити різні типи даних у текстовий потік.

Для класу PrintWriter визначені наступні конструктори:

PrintWriter (OutputStream out)

PrintWriter (OutputStream out, boolean autoFlush)

PrintWriter (Writer out)

PrintWriter (Writer out, boolean autoFlush)

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

Методи цього класу не кидають виключення вводу-виводу. Замість них використовуються методи даного класу setError() і checkError(), які відповідно встановлюють стан помилки в true і перевіряють стан помилки.

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

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

Консольний ввод

Для читання введених даних з клавіатури або з інших джерел вводу, які спрямовуються на стандартний вхід програми зазвичай використовується System.in. System.in є статичним об'єктом класу InputStream, тому він має доступ до методів, які дозволяють читати дані з потоку.

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

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

```
try {
     InputStreamReader inputStreamReader = new
InputStreamReader(System.in);
     BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);

     System.out.print("Введіть рядок: ");
     String inputLine = bufferedReader.readLine();
     System.out.println("Ви ввели: " + inputLine);

     bufferedReader.close();
} catch (IOException e) {
     e.printStackTrace();
}
```

У цьому прикладі використовується InputStreamReader для зчитування потоку байтів з System.in та перетворення їх у символи. Потім використовується BufferedReader для читання рядків з InputStreamReader. Користувач може ввести рядок у консоль, і цей рядок буде зчитаний та виведений на екран.

Клас StreamTokenizer

При читанні символьних потоків дуже часто потрібно розбивати дані на окремі частини. Для розбору потоків символів на токени використовується клас StreamTokenizer. Токени - це окремі частини тексту,

наприклад, слова, числа, розділові знаки тощо. StreamTokenizer дозволяє легко аналізувати текстові дані, зчитуючи їх по-токеново.

Тип токену може бути одним зі сталих: TT_EOF (кінець потоку), TT_WORD (слово), TT_NUMBER (число) або TT_EOL (кінець рядка).

Властивість int nval містить значення поточного слова, якщо воно число. Властивість String sval містить значення поточного слова, якщо воно – рядок. Властивість int ttype містить тип тільки що прочитаного слова.

Клас StreamTokenizer включає безліч різних методів, основні з яких наведені в таблиці 2.8.

Таблиця 2.8 - Основні методи класу StreamTokenizer

void resetSyntax()	Скидає таблицю символів для розпізнавання токенів. Всі символи вважаються роздільниками.
void wordChars(int low, int hi)	Встановлює, які символи можуть входити до складу слів
void whitespaceChars (int low, int hi)	Задає символи, які сприймаються як пробіли
void ordinaryChars(int low, int hi)	Встановлює, що символи з кодами в діапазоні від low до hi є звичайними.
void ordinaryChar(int ch)	Додає символ ch до таблиці символів для розпізнавання як звичайного символу.
int nextToken()	Зчитує наступний токен з потоку і повертає його тип
void quoteChar(int char)	Визначає символ, що використовується для виділення текстових рядків.
int lineno()	Повертає номер поточного рядка.
void commentChar (int char)	Визначає символ однострочного коментаря.
void eolIsSignificant (boolean flag)	Дозволяє вказати, чи слід вважати ознаку кінця рядка роздільником слів.
void lowerCaseMode (boolean flag)	Дозволяє перевести екземпляр об'єкта в режим примусового переведення всіх виділених слів в нижній регістр.

void parseNumbers()	Дозволяє визначити, чи слід робити спроби розпізнавання чисел і перетворення їх у відповідну числову форму.
void slashSlashComments (boolean flag)	Задає режим обробки однорядкових коментарів, що виконані в стилі C ++.
void slashStarComments (boolean flag)	Задає режим обробки багаторядкових коментарів, що виконані у стилі C.
String toString()	Повертає представлення поточної лексеми у вигляді об'єкта класу String.

Таким чином, клас StreamTokenizer дозволяє аналізувати вхідний потік символів і розпізнавати окремі токени за допомогою методу nextToken(). Можна задавати правила для розпізнавання слів, чисел та роздільників за допомогою методів wordChars(), ordinaryChars(), ordinaryChar() тощо. Після отримання токену, його тип можна перевірити за допомогою змінних sval та nval для рядкових та числових значень відповідно.

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

3. Контрольні питання

1. Для яких цілей використовується клас File? Які операції над файлами визначені в класі File?
2. Що таке потік даних? Які види потоків визначені в Java?
3. Які методи читання і запису даних визначені в класах InputStream і OutputStream?
4. Для яких цілей використовуються класи ByteArrayInputStream і ByteArrayOutputStream?
5. Як в Java виконується об'єднання двох потоків вводу в один потік?
6. Як в Java реалізований перехід між байтовими і символьними потоками?
7. Для яких цілей використовується клас StreamTokenizer?

4. Правила техніки безпеки та охорони праці

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

5. Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

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

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

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java в інтегрованому середовищі розробки Eclipse або IntelliJ IDEA.

6. Завдання до лабораторної роботи

Створити вікно JavaFX з заголовком “Лаб. робота 4 | Ваше_Прізвище”.

Варіанти:

1. Створіть програму копіювання файлу з одного каталогу в інший. Компоненти графічного вікна: напис "Копіювання файлу" зверху в центрі вікна, кнопка "Копіювати" у нижній частині вікна. У центрі вікна два стовпця. В першому стовпці: напис "Оберіть файл для копіювання:", кнопка "Обрати файл…" та напис "Файл не обрано". При натисканні кнопки "Обрати файл…" відкривається вікно FileChooser, після вибору файла напис "Файл не обрано" змінюється на шлях до обраного файлу. В другому стовпці: напис "Оберіть каталог для копії:", кнопка "Обрати каталог…" та напис "Каталог не обрано". При натисканні кнопки "Обрати каталог…" відкривається вікно FileChooser, після вибору каталогу напис "Каталог не обрано" змінюється на шлях до обраного каталогу. Копіювання файлу виконується при натисканні кнопки "Копіювати", якщо обрані і файл, і каталог.

2. Створіть програму переміщення файлу з одного каталогу в інший. Компоненти графічного вікна: напис "Переміщення файлу" зверху в центрі вікна, кнопка "Перемістити" у нижній частині вікна. У центрі вікна два

стовпця. В першому стовпці: напис "Оберіть файл для переміщення:", кнопка "Обрати файл…" та напис "Файл не обрано". При натисканні кнопки "Обрати файл…" відкривається вікно FileChooser, після вибору файла напис "Файл не обрано" змінюється на шлях до обраного файлу. В другому стовпці: напис "Оберіть каталог для переміщення:", кнопка "Обрати каталог…" та напис "Каталог не обрано". При натисканні кнопки "Обрати каталог…" відкривається вікно FileChooser, після вибору каталогу напис "Каталог не обрано" змінюється на шлях до обраного каталогу. Переміщення файлу виконується при натисканні кнопки "Перемістити", якщо обрані і файл, і каталог.

3. Створіть програму перейменування файлу. Компоненти графічного вікна: напис "Перейменування файлу" зверху в центрі вікна, кнопка "Перейменувати" у нижній частині вікна. У центрі вікна два стовпця. В першому стовпці: напис "Оберіть файл для перейменування:", кнопка "Обрати файл…" та напис "Файл не обрано". При натисканні кнопки "Обрати файл…" відкривається вікно FileChooser, після вибору файла напис "Файл не обрано" змінюється на шлях до обраного файлу. В другому стовпці: напис "Введіть нове ім’я файлу:" та текстове поле. Перейменування файлу виконується при натисканні кнопки "Перейменувати", якщо обраний файл і введене нове ім’я.

4. Створіть програму для видалення файлу. Компоненти графічного вікна: напис "Видалення файлу" зверху в центрі вікна, кнопка "Видалити" у нижній частині вікна. У центрі вікна напис "Оберіть файл для видалення:", кнопка "Обрати файл…" та напис "Файл не обрано". При натисканні кнопки "Обрати файл…" відкривається вікно FileChooser, після вибору файла напис "Файл не обрано" змінюється на шлях до обраного файлу. Видалення файлу виконується при натисканні кнопки після виведення діалогового вікна для підтвердження або скасування видалення файлу.

5. Створіть програму для перегляду даних в текстовому файлі з використанням класу StreamTokenizer (шлях до файлу прописаний у програмі). Кожен рядок файлу містить ім'я користувача, пароль і довільні відомості про користувача (облікову інформацію користувача) через роздільник ";". Компоненти графічного вікна: напис "Отримання даних користувача" зверху в центрі вікна, кнопка "Знайти користувача" у нижній частині вікна. У центрі вікна два стовпця. В першому стовпці: напис "Введіть ім'я:" і текстове поле для введення імені користувача. В другому

стовпці: 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 "Знайти користувача" файл проглядається і, якщо користувач знайдений, його пароль і облікова інформація виводяться у відповідних полях, інакше виводиться повідомлення "Користувач не знайдений".

6. Створіть програму виведення файлів каталогу з заданим розширенням. Компоненти графічного вікна: напис "Виведення файлів заданих типів" зверху в центрі вікна, кнопка "Вивести список файлів" у нижній частині вікна. У центрі вікна два стовпця. В першому стовпці: напис "Оберіть каталог:", кнопка "Обрати каталог…", напис "Каталог не обрано", напис "Введіть тип файлу:" та текстове поле для введення типу файлу (наприклад, ".doc"). При натисканні кнопки "Обрати каталог…" відкривається вікно FileChooser, після вибору каталогу напис "Каталог не обрано" змінюється на шлях до обраного каталогу. В другому стовпці: напис "Список файлів обраного типу" та текстове поле для виведення списку файлів. Виведення файлів каталогу з заданим розширенням виконується при натисканні кнопки "Вивести список файлів".

7. Створіть програму для отримання характеристик файлу з використанням методів класу File. Компоненти графічного вікна: напис "Отримання характеристик файлу" зверху в центрі вікна, кнопка "Отримати характеристики" у нижній частині вікна. У центрі вікна: напис "Оберіть файл для отримання характеристик:", кнопка "Обрати файл…", напис "Файл не обрано" та текстове поле для виводу інформації. При натисканні кнопки "Обрати файл…" відкривається вікно FileChooser, після вибору файла напис "Файл не обрано" змінюється на шлях до обраного файлу. При натисканні кнопки "Отримання характеристик файлу" виводиться ім'я обраного файлу, його тип, розмір і дата останньої модифікації.

8. Створіть програму для перегляду даних в текстовому файлі з використанням класу StreamTokenizer (шлях до файлу прописаний у програмі). Кожен рядок файлу містить ім'я товару, ціну і довільні відомості про товар через роздільник ";". Компоненти графічного вікна: напис "Відомості про товар" зверху в центрі вікна. У вікні: спадне меню 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.

9. Створіть програму виведення файлів каталогу менше заданого розміру в обраному каталозі. Компоненти графічного вікна: напис "Виведення файлів заданого розміру" зверху в центрі вікна, кнопка "Вивести список файлів" у нижній частині вікна. У центрі вікна два стовпці. У першому стовпці: напис "Оберіть каталог:", кнопка "Обрати каталог…", напис "Каталог не обрано", напис "Введіть максимальний розмір файлу (КБ):" та текстове поле для введення числа. При натисканні кнопки "Обрати каталог…" відкривається вікно FileChooser, після вибору каталогу напис "Каталог не обрано" змінюється на шлях до обраного каталогу. В другому стовпці: напис "Список файлів обраного типу" та текстове поле для виведення списку файлів. Виведення файлів каталогу менше заданого розміру виконується при натисканні кнопки "Вивести список файлів".

10. Створіть програму злиття вмісту обраних файлів в каталозі. Компоненти графічного вікна: напис "Злиття файлів" зверху в центрі вікна, кнопка "Злити" у нижній частині вікна. У центрі вікна два стовпці. У першому стовпці: напис "Оберіть файл 1:", кнопка "Обрати файл…", напис "Файл не обрано", напис "Оберіть файл 2:", кнопка "Обрати файл…" та напис "Файл не обрано". При натисканні кнопок "Обрати файл…" відкривається вікно FileChooser, після вибору файла відповідний напис "Файл не обрано" змінюється на шлях до обраного файлу. У другому стовпці: напис "Оберіть каталог для нового файлу:", кнопка "Обрати каталог…", напис "Каталог не обрано", напис "Ім’я об'єднаного файлу" та текстове поле для введення імені створюваного файлу. При натисканні кнопки "Обрати каталог…" відкривається вікно FileChooser, після вибору каталогу напис "Каталог не обрано" змінюється на шлях до обраного каталогу. Файли зливаються при натисканні кнопки "Злити".

7. Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

- титульний лист, де вказано номер і назва лабораторної роботи, відомості про виконавця,
- номер варіанта роботи та текст завдання,

- відповіді на контрольні запитання до лабораторної роботи,
- листинг програмного коду,
- результати виконання програми,
- висновки.

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

Лабораторна робота №5
Програмування колекцій в Java

1. Мета роботи

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

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

2. Теоретичні відомості до виконання лабораторної роботи

Колекції

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 у зручний і безпечний спосіб. 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.

У Java колекції реалізовані через інтерфейси і конкретні класи пакету java.util, які надають різні способи зберігання даних.

Стандартним API для роботи з колекціями в мові програмування Java є Java Collections Framework (JCF), який включає набір інтерфейсів та класів для зберігання і обробки даних у вигляді списків, множин, карт, черг та інших структур.

Всі Java Collections Framework містять наступні три компоненти:

Інтерфейси: Набір інтерфейсів, таких як List, Set, Map, Queue, що визначають основні операції і поведінку колекцій.

Класи: Конкретні реалізації інтерфейсів, наприклад, ArrayList, LinkedList, HashSet, HashMap, TreeSet і т.д.

Алгоритми: Сортування, пошук і інші алгоритми, які працюють з колекціями даних.

Інтерфейси колекцій

Ключові інтерфейси колекцій 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.

Інтерфейс Collection

Інтерфейс Collection представляє базовий контракт для всіх структур даних, що зберігають і управляють групами об'єктів. Він визначає набір загальних методів, які дозволяють працювати з колекціями безпосередньо та динамічно, незалежно від конкретної реалізації колекції. 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 основні операції (табл. 3.1).

Таблиця 3.1 - Методи та операції інтерфейсу Collection

int size()	Визначення кількості елементів в колекції.
boolean isEmpty()	Перевірка, чи містить колекція елементи.
boolean contains (Object o)	Перевірка, чи перебуває даний елемент в колекції.
boolean equals(Object o)	Порівняння заданого об'єкта з даною колекцією на рівність.
boolean add (Object o)	Додавання елемента до колекції.
boolean remove (Object o)	Видалення елемента з колекції.
void clear()	Очищення колекції.
Iterator iterator ()	Забезпечення ітераційних операцій над колекцією
Object[] toArray(), Object[] toArray(Object a[])	Забезпечення моста між колекціями і старими інтерфейсами прикладних програм, які припускали передачу їм параметрів об'єкта у вигляді масиву.
boolean containsAll(Collection c)	Перевірка, чи містяться всі елементи колекції с в даній колекції.

boolean addAll(Collection c)	Додавання всіх елементів колекції c до даної колекції
boolean removeAll (Collection c)	Видалення з даної колекції всіх елементів, що містяться в колекції c.
boolean retainAll (Collection c)	Залишення в даній колекції тільки тих елементів, які містяться в колекції c.

Метод iterator() повертає об'єкт інтерфейсу Iterator. Інтерфейс Iterator оголошує наступні методи для колекцій (табл. 3.2).

Таблиця 3.2 - Методи інтерфейсу Iterator

boolean hasNext()	Перевірка, чи є наступний елемент.
Object next()	Повернення наступного елементу.
void remove ()	Видалення даного елементу.

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

Інтерфейс Set є підінтерфейсом Collection і представляє неупорядковану колекцію унікальних елементів. Він не допускає дублювання елементів, тобто кожен елемент у ньому може бути представлений лише один раз.

Інтерфейс Set містить ті ж методи, що і батьківський інтерфейс. Проте методи в цьому інтерфейсі мають більш конкретний математичний сенс. Наприклад:

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

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

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

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

Інтерфейс List також є підінтерфейсом Collection. Він представляє впорядкований список елементів (послідовність), де дозволяється дублювання елементів, тобто один і той самий елемент може бути присутнім у списку більше одного разу. Впорядкований список означає, що порядок елементів у списку відповідає їх взаємному розташуванню.

Додатково до операцій, успадкованих від Collection, інтерфейс List оголошує наступні операції (табл. 3.3)

Таблиця 3.3 - Додаткові методи інтерфейсу List

Object get(int index)	Отримання значення елемента із заданим індексом.
Object set(int index, Object element)	Установка значення елемента із заданим індексом.
int indexOf(Object o)	Отримання індексу першого входження заданого елемента у списку.
int lastIndexOf(Object o)	Отримання індексу останнього входження заданого елемента у списку.
void add(int index, Object element)	Додавання елемента по заданому індексу.
Object remove(int index)	Видалення елемента по заданому індексу.
void clear()	Очищення списку.
List subList (int fromIndex, int toIndex)	Операції над частиною списку.
ListIterator listIterator(), ListIterator listIterator (int index)	Ітераційні операції у списку.

Метод listIterator() повертає об'єкт інтерфейсу ListIterator. Інтерфейс ListIterator розширює інтерфейс Iterator і містить наступні методи перегляду списку (табл. 3.4).

Таблиця 3.4 - Методи інтерфейсу ListIterator

boolean hasNext()	Перевіряє, чи є наступний елемент.
boolean hasPrevious()	Перевіряє, чи є попередній елемент.
Object next()	Повертає наступний елемент.
Object previous()	Повертає попередній елемент.
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 set (Object obj)	Призначає obj на поточний елемент (останній елемент, повернутий викликом методу next() або previous())
void remove ()	Видаляє поточний елемент із списку.

Інтерфейс SortedSet

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 (табл. 3.5).

Таблиця 3.5 - Додаткові методи інтерфейсу SortedSet

SortedSet subSet(Object fromElement, Object toElement)	Операції над частиною набору.
SortedSet headSet(Object toElement)	Операції над головною частиною набору.
SortedSet tailSet(Object fromElement)	Операції над хвостовою частиною набору
Object first()	Повернення першого елемента відсортованого набору.

Object last()	Повернення останнього елемента відсортованого набору.
Comparator comparator()	Доступ до інтерфейсу Comparator.

Звичайно об'єкти в програмах Java упорядковано за допомогою методу compareTo (Object obj), оголошеного в інтерфейсі Comparable. 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 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 compare (Object o1, Object o2) і 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 Map

Інтерфейс Map представляє структуру даних "ключ-значення". У Map кожен елемент представлений у вигляді пари, де ключ є унікальним і відображається на відповідне значення. Інтерфейс Map не може містити дублікатів ключів, і для кожного ключа може бути асоційовано тільки одне значення. Таку колекцію називають відображенням (map), словником (dictionary) або асоціативним масивом (associative array).

В інтерфейсі Map визначені наступні основні операції (табл. 3.6)

Таблиця 3.6 - Методи та операції інтерфейсу Map

int size()	Визначення кількості ключів-значень в мапі.

boolean isEmpty()	Перевірка, чи містить відображення елементи.
Object put(Object key, Object value)	Додавання нового запису з ключем та значенням до відображення. Якщо вже існує запис з таким ключем, він замінюється новим значенням, а старе значення повертається.
Object get(Object key)	Отримання значення елемента із заданим ключем.
boolean equals(Object o)	Порівняння відображення з заданим об'єктом на рівність.
Set keySet()	Повертає множину ключів, що містяться в мапі.
Object remove (Object key)	Видалення елемента із заданим ключем.
void clear()	Очистка відображення.
boolean containsKey (Object key)	Перевірка наявності елемента із заданим ключем.
boolean containsValue (Object value)	Перевірка наявності елемента із заданим значенням.
void putAll(Map t)	Додавання до даного відображення всіх пар із заданого відображення.
Collection values()	Представлення всіх значень даного відображення у вигляді колекції
Set entrySet()	Повертає множину усіх записів (пар ключ-значення) у відображенні у вигляді об'єктів типу Map.Entry

Інтерфейс Map.Entry описує методи роботи з парами «ключ-значення», отриманими методом entrySet (табл. 3.7).

Таблиця 3.7 - Методи інтерфейсу Map.Entry

Object getKey()	Отримання ключа.
Object getValue()	Отримання значення.
Object setValue (Object value)	Зміна значення.

boolean equals (Object o)	Порівняння заданого об'єкта з даною парою «ключ-значення» на рівність.

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

Інтерфейс SortedMap

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

Додатково до операцій інтерфейсу Map Інтерфейс SortedMap реалізує наступні операції (табл. 3.8).

Таблиця 3.8 - Додаткові методи інтерфейсу SortedSet

SortedMap subMap (Object fromKey, Object toKey)	Виділення частини відображення.
SortedSet headMap(Object toElement)	Операції над головною частиною набору.
SortedMap tailMap (Object fromKey)	Операції над хвостовою частиною відображення.
Object firstKey()	Повернення першого елемента відображення.
Object lastKey()	Повернення останнього елемента відображення.
Comparator comparator()	Доступ до інтерфейсу Comparator.

В цілому, інтерфейс SortedMap є аналогом інтерфейсу SortedSet.

Реалізації колекцій і алгоритми

Реалізації колекцій - це клас, які реалізують (імплементують) інтерфейси з фреймворка Java Collections Framework.

Абстрактні класи в JCF є особливим типом класів, які забезпечують більшу гнучкість і спрощують реалізацію ієрархій колекцій.

Основна характеристика абстрактних класів - це те, що вони можуть містити як конкретні (реалізовані) методи, так і абстрактні методи. Абстрактний метод є методом, який не має тіла в абстрактному класі, тобто він не має реалізації в цьому класі, але обов'язково повинен бути перевизначений (реалізований) у підкласах. Це дозволяє абстрактним класам встановлювати загальний шаблон для підкласів, але водночас залишати деякі методи відкритими для унікальної реалізації у підкласах.

Абстрактні класи використовуються для створення базових реалізацій деяких інтерфейсів, або навіть цілі ієрархії колекцій.

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

Клас AbstractList є суперкласом для класів AbstractSequentialList, ArrayList, Vector і містить реалізації методів add(), методів addAll(), clear(), equals(), get(), indexOf(), iterator(), lastIndexOf(), listIterator(), remove(), set(), sublist() інтерфейсу List. Клас AbstractList містить також метод removeRange (int fromIndex, int toIndex) який дозволяє видалити елементи списку в заданому діапазоні.

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

Клас AbstractMap є суперкласом для класів HashMap, TreeMap і WeakHashMap і містить реалізації методів clear(), containsKey(), containsValue(), entrySet(), equals(), get(), isEmpty(), keySet(), put(), putAll(), remove(), size() і values() інтерфейсу Map. Крім того, клас AbstractMap містить метод clone(), який дозволяє отримати порожню копію відображення (ключі і значення не копіюються), а також метод toString(), який дозволяє перетворити відображення в строкове представлення.

Існують три основні типи реалізацій колекцій: загальноцільові, пакувальники та альтернативні.

Загальноцільові реалізації (General-Purpose Implementations)

Ці реалізації призначені для загальних сценаріїв використання і підтримують широкий спектр операцій. Вони забезпечують ефективність та зручний інтерфейс для багатьох різних випадків використання колекцій. Загальноцільові реалізації та їх зв'язок з інтерфейсами і структурами даних представлені в таблиці 3.9.

Таблиця 3.9 - Загальноцільові реалізації

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

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

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

Серед загальноцільових реалізацій можна виділити найбільш часто використовувані: ArrayList, LinkedList, HashSet та HashMap.

Клас ArrayList

ArrayList забезпечує реалізацію інтерфейсу Set та реалізує динамічний масив, який автоматично збільшується, коли заповнюються всі доступні місця. Використовується для швидкого доступу до елементів по індексу.

Об'єкти ArrayList мають дві характеристики – ємність і розмір масиву. Якщо розмір списку перевищить ємність, ємність автоматично збільшується на деяку кількість елементів.

Клас ArrayList має наступні конструктори:

ArrayList() – створює порожній список;

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

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

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

Крім того, клас містить наступні власні методи (табл. 3.10).

Таблиця 3.10 - Власні методи класу ArrayList

Object clone()	Отримання порожній копії об'єкта ArrayList (без елементів).
void ensureCapacity (int minCapacity)	Збільшує ємність екземпляру ArrayList, якщо це необхідно, для того, щоб він міг містити число елементів, задане в minCapacity.
void removeRange (int fromIndex, int toIndex)	Видаляє елементи в заданому діапазоні індексів.
void trimToSize()	Зменшує ємність об'єкта ArrayList до його попереднього значення.

Клас LinkedList

Клас LinkedList, на додаток до реалізації методів інтерфейсу List забезпечує методи для отримання, видалення і вставки елементів в список, що дозволяє використовувати зв'язані списки як стеки, черги або черги з двома кінцями. LinkedList реалізує зв'язаний список, що забезпечує швидкі операції вставки та видалення, але повільний доступ до елементів по індексу.

Клас LinkedList має два конструктора:

LinkedList () – створює порожній зв'язаний список;

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

На додаток до двох методів add(), двох методів addAll(), методів clear(), contains(), get(), indexOf(), isEmpty(), lastIndexOf(), listIterator(), двох методів remove(), методів set(), size() і двох методів toArray() інтерфейсу List, клас LinkedList забезпечує наступні методи (табл. 3.11).

Таблиця 3.11 - Власні методи класу LinkedList

void addFirst(Object o)	Додавання елемента в початок зв'язаного списку.
void addLast(Object o)	Додавання елемента в кінець зв'язаного списку.
Object getFirst()	Отримання першого елемента зв'язаного списку.
Object getLast()	Отримання останнього елемента зв'язаного списку.
Object removeFirst()	Видалення першого елемента зв'язаного списку.
Object removeLast()	Видалення останнього елемента зв'язаного списку.
Object clone()	Отримання порожньої копії об'єкта LinkedList (без елементів).

Клас ArrayList є кращим за LinkedList у випадку, якщо необхідна висока продуктивність. Однак, якщо часто доводиться додавати елементи в початок списку або видаляти елементи з його середини, кращим є клас LinkedList.

Клас HashSet

Клас HashSet реалізує інтерфейс Set і забезпечує збереження унікальних елементів у хеш-таблиці (зазвичай є реалізацією HashMap). Не гарантує порядку елементів.

Клас HashSet має наступні конструктори:

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

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

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

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

У класі HashSet реалізовані наступні методи інтерфейсу Set: add(), clear(), contains(), isEmpty(), iterator(), remove() і size(). Метод clone() дозволяє отримати порожню копію об'єкта HashSet (без елементів).

Клас HashMap

Клас HashMap є реалізацією інтерфейсу Map з використанням хеш-таблиць і за своїми можливостями відповідає класу HashTable. Надає швидкий доступ до значень за ключами.

Клас HashMap має наступні конструктори:

HashMap() – створює нове, пусте відображення з ємністю за замовчуванням 16 і фактором завантаження, рівним 0.75;

HashMap(Map m) – створює нове відображення, що містить елементи заданого відображення;

HashMap(int initialCapacity) – створює нове, пусте відображення із заданою ємністю і фактором завантаження, рівним 0.75;

HashMap(int initialCapacity, float loadFactor) – створює нове, пусте відображення із заданою ємністю і заданим фактором завантаження.

Клас HashMap реалізує наступні методи інтерфейсу Map: clear(), containsKey(), containsValue(), entrySet(), get(), isEmpty(), keySet(), put(), putAll(), remove(), size() і values().

Пакувальники (Wrapper Classes)

Ці реалізації "упаковують" існуючі колекції, надаючи їм додатковий функціонал або модифікуючи їх поведінку. Вони є декораторами навколо основних реалізацій і дозволяють розширити їх функціональність. Серед пакувальників можна виділити:

Collections: Клас, що надає статичні методи для роботи з колекціями, такі як сортування, пошук, зміна порядку елементів тощо.

Synchronized: Класи, які надають синхронізацію операцій для роботи зі збалансованими деревами, списками і мапами.

Checked: Класи, які додають перевірку типів для колекцій, що гарантує, що вони містять тільки елементи з заданим типом.

Альтернативні реалізації (Convenience)

Ці реалізації надають додатковий комфорт і зручність при використанні колекцій в певних ситуаціях. Вони можуть бути підходящими для певних типів завдань або просто спрощувати роботу з колекціями. Серед альтернативних реалізацій можна виділити:

LinkedHashSet: Подібно до HashSet, але зберігає порядок додавання елементів.

TreeSet: Забезпечує впорядковане збереження елементів у дереві. Використовує порівняння для сортування.

LinkedHashMap: Подібно до HashMap, але зберігає порядок додавання пар "ключ-значення".

Ці типи реалізацій допомагають підтримувати гнучкість, продуктивність і простоту в роботі з колекціями, а вибір конкретного типу залежить від потреб та вимог в програмі.

Ієрархія інтерфейсів та класів Java Collections Framework наведена на рис. 2.1.

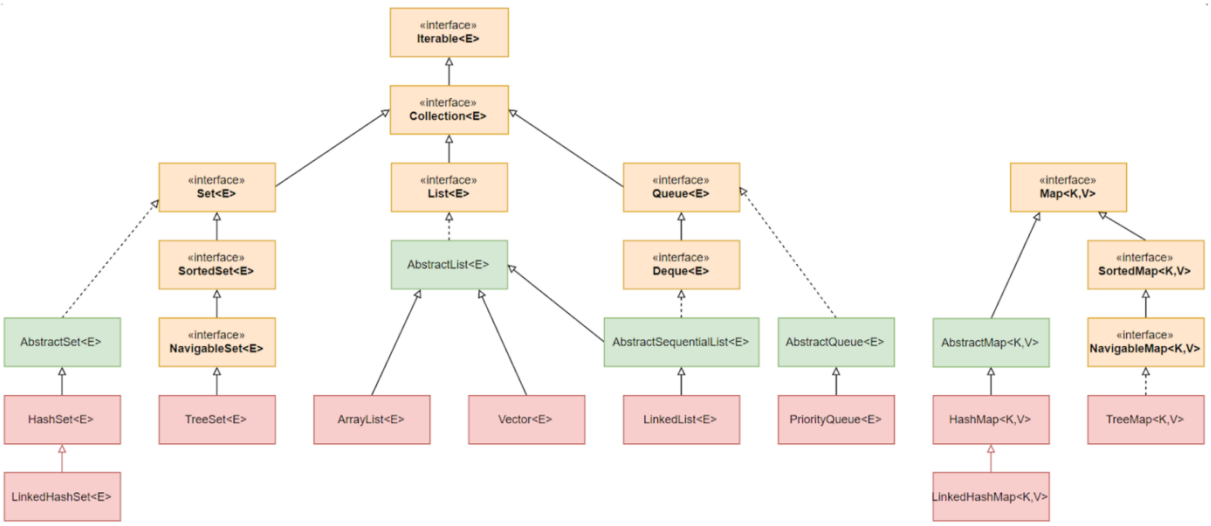

Рисунок 2.1 - Ієрархія інтерфейсів та класів Java Collections Framework

3. Контрольні питання

1. Які інтерфейси колекцій визначені в мові Java?
2. Які операції над колекціями визначені за допомогою методів інтерфейсу Collection?
3. Які додаткові методи для колекцій визначені в інтерфейсі List?
4. Як визначається інтерфейс Map, і які операції визначені в цьому інтерфейсі?
5. Які типи реалізацій колекцій існують у мові Java? Дайте коротку характеристику кожного типу.
6. Які абстрактні класи реалізації колекцій визначені в Java?

4. Правила техніки безпеки та охорони праці

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

5. Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

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

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

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java в інтегрованому середовищі розробки Eclipse або IntelliJ IDEA.

6. Завдання до лабораторної роботи

Створити вікно JavaFX з заголовком “Лаб. робота 5 | Ваше_Прізвище”.

Варіанти:

1. Створіть додаток JavaFX 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 одразу 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 JavaFX 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, рік видання і ціну книги. 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 кнопка "Видалити".

3. Створіть додаток JavaFX 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.

4. Створіть додаток JavaFX 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 "*".

5. Створіть додаток JavaFX 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, рік видання і ціну книги. Після введення записи одразу 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.

Приклад створення HashMap, додавання та сортування елементів:

```
    TextField tfPhoneNumber, tfLastName, tfFirstName,
tfMiddleName, tfAddress;
    HashMap<String, Contact> phoneBook = new HashMap<>();
    …
    String phoneNumber = tfPhoneNumber.getText().trim();
    String lastName = tfLastName.getText().trim();
    String firstName = tfFirstName.getText().trim();
    String middleName = tfMiddleName.getText().trim();
```

```
     String address = tfAddress.getText().trim();
     Contact contact = new Contact(lastName, firstName,
middleName, address);
     phoneBook.put(phoneNumber, contact);
     …
     // сортування ключів
     sortedKeys = new ArrayList<>(phoneBook.keySet());
     Collections.sort(sortedKeys);
     …
     // отримання елементу за індексом після сортування
     String phoneNumber = sortedKeys.get(index);
     Contact contact = phoneBook.get(phoneNumber);

     // клас Contact
     private static class Contact {
        private String lastName;
        private String firstName;
        private String middleName;
        private String address;

        public Contact(String lastName, String firstName,
String middleName, String address) {
            this.lastName = lastName;
            this.firstName = firstName;
            this.middleName = middleName;
            this.address = address;
        }
     }
```

7. Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

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

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

Лабораторна робота №6
Управління мережевими з’єднаннями.
Використання сокетів у розподілених додатках

1. Мета роботи

Метою роботи є придбання навичок створення мережевих додатків на мові Java з використанням бібліотеки java.net.

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

2. Теоретичні відомості до виконання лабораторної роботи

При роботі з мережевими з'єднаннями в Java-програмах будуть використовуватися основні поняття, що їх характеризують: IP-адресація, доменна служба імен DNS, URL-адреси, протоколи TCP і UDP, прикладні протоколи, сокети. У зв'язку з цим рекомендується повторити згадані теми за конспектом лекцій з дисципліни «Комп'ютерні мережі».

Класи та інтерфейси, що дозволяють управляти мережевими з'єднаннями в Java-програмах, визначені в пакеті java.net. Деякі з завдань, які можуть бути вирішені засобами цього пакету:

- робота з URL-адресами;
- завантаження вмісту з Інтернету;
- створення клієнт-серверних додатків і відправка та отримання даних по мережі;
- робота з дейтаграмами;
- взаємодія з веб-сервісами, виконання HTTP-запитів та обробка відповідей сервера;
- створення мережевого з'єднання між двома процесами на одній локальній машині;
- реалізація мережевих протоколів, таких як FTP, SMTP, POP3 тощо;
- налаштування безпеки мережевих з'єднань, використовуючи SSL або TLS.

Розглянемо деякі з цих завдань більш докладно.

Клас InetAddress

Клас InetAddress в Java представляє собою об'єкт, який містить інформацію про IP-адресу та ім'я хоста. Цей клас дозволяє виконувати

операції над IP-адресами та отримувати інформацію про хост, пов'язаний з заданою IP-адресою.

Клас InetAddress не має видимих конструкторів. Для створення об'єкта потрібно використовувати один з доступних фабричних методів.

Фабричні методи – угода, за допомогою якого статичні методи в класі повертають екземпляр даного класу. Вони надають зручний спосіб створення об'єктів без прямого виклику конструктора, що дозволяє уникнути прямої залежності коду від конкретного класу. Це зроблено, щоб уникнути перевантаження конструктора різними списками параметрів, коли наявність унікальних імен методів призводить до більш ясних результатів.

Для створення об'єктів InetAddress можна використовувати три статичні методи (табл. 4.1).

Таблиця 4.1 - Фабричні методи для створення об'єктів InetAddress

InetAddress getLocalHost()	Повертає об'єкт класу InetAddress для локального комп'ютера.
InetAddress getByName (String hostName)	Повертає об'єкт класу InetAddress для комп'ютера, DNS-ім'я якого передається в параметрі hostName.
InetAddress[] getAllByName (String hostName)	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.

Основні нестатичні методи класу InetAddress наведені в табл. 4.2.

Таблиця 4.2 - Основні методи класу InetAddress

byte[] getAddress()	Повертає чотирьохелементний масив байтів, який представляє IP-адресу об'єкта.
String	Повертає IP-адресу в форматі String у вигляді

getHostAddress()	текстового представлення.
String getHostName()	Повертає ім'я хоста для даної IP-адреси. Якщо ім'я невідоме, повертає текстове представлення IP-адреси.
boolean equals(Object other)	Перевіряє, чи рівні два об'єкти InetAddress, порівнюючи IP-адреси.
int hashCode()	Повертає хеш-код об'єкта.
boolean isMulticastAddress()	Повертає true, якщо IP-адреса об'єкта цього класу групова (multicast, починається з "224.").
boolean isReachable(int timeout)	Перевіряє, чи можна досягти хоста за допомогою пінгування. timeout - максимальний час очікування в мілісекундах для відповіді від хоста.
String toString()	Повертає рядок, який містить доменну і IP-адресу хоста (наприклад, "sterwave.com/192.147.170.6").

Розглянемо різні випадки застосування розглянутих раніше методів класу InetAddress.

Зверніть увагу на розходження при використанні методу getByName() з параметром null і методу getLocalHost(). Якщо локальна мережа побудована на протоколі TCP/IP, то кожному комп'ютеру присвоюється IP-адреса. Тому метод getLocalHost() повертає дану IP-адресу цієї локальної мережі. З іншого боку метод getByName() з параметром null в будь-якому випадку повертає адресу 127.0.0.1.

```
InetAddress localaddr = InetAddress.getLocalHost();
System.out.println("Local address (getLocalHost): " +
localaddr.getHostAddress() + " | "+ localaddr.getHostName());

localaddr = InetAddress.getByName(null);
System.out.println("Local address (getByName): " +
localaddr.getHostAddress() + " | "+ localaddr.getHostName());

InetAddress addr = InetAddress.getByName("odeku.edu.ua");
System.out.println("ODEKU address: " + addr.getHostAddress()+
" | " + addr.getHostName());

InetAddress[] multaddr=InetAddress.getAllByName("java2s.com");
```

```
System.out.println("java2s addresses: ");
for(int i=0; i<multaddr.length; i++)
System.out.println((i+1)+ ": " + multaddr[i].getHostAddress()
+ " | " + multaddr[i].getHostName());
```

Результат роботи:

```
Local address (getLocalHost): 172.22.192.1 | HOME-PC
Local address (getByName): 127.0.0.1 | localhost
ODEKU address: 195.138.69.231 | odeku.edu.ua
java2s addresses:
1: 52.217.10.139 | java2s.com
2: 54.231.132.181 | java2s.com
3: 52.217.235.245 | java2s.com
4: 52.216.144.138 | java2s.com
5: 52.216.100.130 | java2s.com
6: 52.217.224.253 | java2s.com
7: 52.216.154.219 | java2s.com
8: 52.216.213.13 | java2s.com
```

Класи URL і URLConnection

Клас URL в Java представляє собою об'єкт, який використовується для представлення та роботи з URL-адресами.

URL забезпечує зручну форму ідентифікації ресурсів мережі Internet і може складатися з 4-х основних складових: протоколу, доменного імені або IP-адреси комп'ютера, номера порту і шляху до файлу.

В якості протоколу можуть використовуватися http, https, ftp, file. Ім'я протоколу в URL завершується двокрапкою, потім після подвійного слеш (//) вказується DNS-ім'я комп'ютера або IP-адреса, наприклад https://www.example.com/ 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

Конструктори класу URL:

URL (String spec);

URL (String protocol, String host, String file);

URL (String protocol, String host, int port, String file);

де spec – рядок, який містить повну URL-адресу,

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.

Основні методи класу URL наведені в табл. 4.3.

Таблиця 4.3 - Основні методи класу URL

String getProtocol()	Повертає протокол даної URL-адреси.
String getHost()	Повертає ім'я хоста (домен) даної URL-адреси.
int getPort()	Повертає номер порту даної URL-адреси.
String getFile()	Повертає шлях до ресурсу на сервері.
openStream()	Відкриває потік вхідних даних InputStream і дозволяє здійснити завантаження об'єкта класу URL.

Приклад завантаження на екран вмісту Web-сторінки, розташованої за адресою https://www.google.com/.

```
try{
     URL myURL = new URL("https://www.google.com/");
     // Створюємо потік вводу
     InputStream in = myURL.openStream();
     // Читаємо з потоку і виводимо на екран
     int ch;
     while((ch=in.read())!=-1)
          System.out.print( (char)ch);
     in.close();
}
catch(IOException e){
     System.out.println(e.getMessage());
     System.exit(0);
}
```

Результат роботи:

```
<!doctype html><html itemscope=""
itemtype="http://schema.org/WebPage" lang="uk"><head><meta
content="text/html; charset=UTF-8" http-equiv="Content-
```

```
Type"><meta
content="/images/branding/googleg/1x/googleg_standard_color_12
8dp.png" itemprop="image"><title>Google</title><script
nonce="0e9BC9L8oeDY0WdoS6Hsrg">(function(){var _g={
    …
```

Сокети і сокетне з’єднання

Сокет (Socket) є одним із основних механізмів для забезпечення зв'язку між процесами або комп'ютерами в комп'ютерних мережах. Він представляє собою інтерфейс для взаємодії між програмами, які працюють на різних комп'ютерах через мережу, і дозволяє передавати дані в обидва напрямки.

Сокети можуть бути використані для з'єднання клієнтів та серверів або для зв'язку між двома незалежними програмами. Вони дозволяють надсилати та отримувати дані з використанням різних мережевих протоколів, таких як TCP (Transmission Control Protocol) або UDP (User Datagram Protocol).

Сокет визначається номером порту та IP-адресою. При цьому IP-адреса використовується для ідентифікації комп'ютера, номер порту – для ідентифікації процесу, що працює на комп'ютері. Коли один додаток знає сокет іншого, створюється сокетне з'єднання. Клієнт намагається з'єднатися з сервером, ініціалізувавши сокетне з'єднання. Сервер чекає, поки клієнт зв'яжеться з ним. Перше повідомлення, що посилається клієнтом на сервер, містить сокет клієнта. Сервер в свою чергу створює сокет, який буде використовуватися для зв'язку з клієнтом, і посилає його клієнту з першим повідомленням. Після цього встановлюється комунікаційне з'єднання.

У Java для програмування TCP/IP сокетів використовуються два основних класи пакету java.net: Socket та ServerSocket.

Клас Socket представляє клієнтський сокет, який встановлює з'єднання з сервером. Він надає можливість надсилати та отримувати дані з сервера. Клієнтський сокет створюється з IP-адресою або існуючим об'єктом класу InetAddress та портом сервера, до якого потрібно підключитися.

```
try {
     Socket socket = new Socket("192.168.0.100", 8080);
} catch (IOException e) {
     System.out.println("Помилка: " + e);
```

```
}
```

Основні методи класу Socket наведені в табл. 4.4.

Таблиця 4.4 - Методи класу Socket

InetAddress getInetAddress()	Повертає об'єкт InetAddress, пов'язаний з об'єктом Socket.
int getPort()	Повертає віддалений порт, з яким з'єднаний даний об'єкт Socket.
int getLocalPort()	Повертає локальний порт, з яким з'єднаний даний об'єкт Socket.
InetAddress getLocalAddress()	Повертає адресу комп'ютера-клієнта.

Клас ServerSocket представляє серверний сокет, який слухає певний порт та приймає з'єднання від клієнтів. Коли серверний сокет отримує запит на з'єднання від клієнта, він створює новий екземпляр класу Socket, який представляє з'єднання з цим клієнтом.

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 clientSocket = null;
     ServerSocket serverSocket = new ServerSocket(8080);
     clientSocket = serverSocket.accept();
} catch (IOException e) {
     System.out.println("Помилка: " + e);
}
```

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.

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.

Організація серверного сокета

Приклад програмування сервера для прослуховування заданого порту і передачі через нього даних клієнту. У даному прикладі сервер посилає клієнтові рядок "Добрий день!", після чого розриває зв'язок.

```
import java.io.*;
import java.net.*;

public class MyServer{
     public static void main(String[] args) {
          Socket s = null;
          try {
               // відправка рядка клієнту
               ServerSocket server = new ServerSocket(2000);
               System.out.println("Сервер запущений...");
               s = server.accept();
               PrintStream ps = new
               PrintStream(s.getOutputStream());
               ps.println("Добрий день!");
               ps.flush();
               s.close();
          } catch (IOException e) {
               System.out.println("Помилка: " + e);
          }
     }
}
```

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.

Організація клієнтського сокета

Приклад програмування клієнта, який підключається до створеного раніше серверу, зчитує дані і виводить їх на екран.

```
import java.io.*;
import java.net.*;

public class MyClient {
     public static void main(String[] args) {
          System.out.println("Клієнт запущений...");
          try {
               //Отримання рядка клієнтом
               InetAddress local = InetAddress.getLocalHost();
               Socket socket = new Socket(local, 2000);
               BufferedReader dis = new BufferedReader (new
               InputStreamReader(socket.getInputStream()));
               String msg = dis.readLine();
               System.out.println(msg);
               socket.close();
               dis.close();
          } catch (IOException e) {
               System.out.println("Помилка: " + e);
          }
     }
}
```

Аналогічно клієнт може послати дані серверу через потік виводу за допомогою методу getOutputStream(), а сервер може отримати дані за допомогою методу getInputStream().

В результаті у класі MyServer створюється серверний сокет на порті 2000 і починає прослуховувати з'єднання. Коли клієнт підключається, сервер приймає з'єднання, створює вихідний потік і надсилає рядок "Добрий день!" клієнту. Після відправки рядка сервер закриває з'єднання. У класі MyClient створюється клієнтський сокет та пробує підключитися до сервера на IP-адресі та порті 2000 (в даному випадку, локальна адреса, де запущений сервер). Після успішного підключення клієнт отримує вхідний потік, з якого зчитує рядок, який відправив сервер, і виводить його на екран. Потім клієнт закриває з'єднання та вхідний потік.

Результат запуску залежатиме від послідовності запуску MyServer і MyClient. Якщо спочатку запустити MyServer, а потім MyClient, MyServer виведе повідомлення "Сервер запущений..." і буде очікувати підключення клієнта. А коли MyClient буде запущений, з’явиться повідомлення "Клієнт

запущений..." і MyClient успішно підключиться до сервера, отримає рядок "Добрий день!" та виведе його на екран. Після цього з'єднання будуть закриті (рис. 3.1).

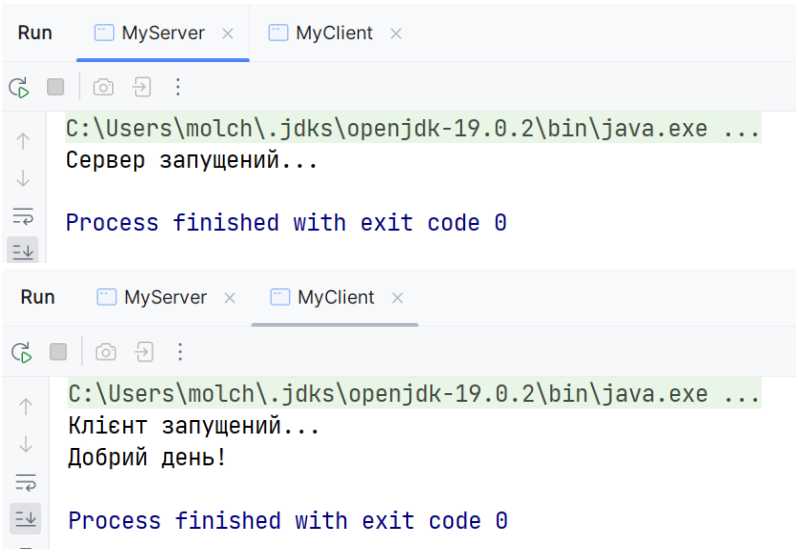

Рисунок 3.1 - Результат роботи класів MyServer та MyClient

Якщо запустити спочатку MyClient, а потім MyServer, то клієнт виведе помилку: java.net.ConnectException: Connection refused: connect. Це означає, що з'єднання було відхилено, тому що сервер ще не запущений і не прослуховує визначений порт.

Нижче наводиться приклад, що демонструє діалог сервера з клієнтом в режимі "питання/відповідь", при якому обидві сторони приймають і вводять повідомлення з терміналу. Серверний додаток TCPServer:

```
import java.io.*;
import java.net.*;

public class TCPServer {
   public static void main(String[] args) {
       try {
           ServerSocket serverSocket = new ServerSocket(9876);
           System.out.println("Сервер запущений");

           Socket clientSocket = serverSocket.accept();

           BufferedReader clientReader = new
BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

```
            PrintWriter clientWriter = new
PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader consoleReader = new
BufferedReader(new InputStreamReader(System.in));

            while (true) {
                String receivedMessage =
clientReader.readLine();
                if (receivedMessage == null) {
                    break;
                }
                System.out.println("Клієнт: " +
receivedMessage);
                System.out.print("Повідомлення для клієнта: ");
                String responseMessage =
consoleReader.readLine();
                clientWriter.println(responseMessage);
            }
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Клієнтський додаток TCPClient приймає від користувача повідомлення у вигляді рядка і передає його на сервер. Програма сервера відображає його на екрані, приймає і передає набрану на клавіатурі відповідь за зворотною адресою клієнта.

```
import java.io.*;
import java.net.*;

public class TCPClient {
   public static void main(String[] args) {
       try {
            Socket socket = new Socket("localhost", 9876);

            BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter writer = new
PrintWriter(socket.getOutputStream(), true);

            InputStreamReader inputStreamReader = new
InputStreamReader(System.in);
            BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);
```

```
            while (true) {
                System.out.print("Повідомлення для сервера: ");
                String message = bufferedReader.readLine();
                writer.println(message);
                String receivedMessage = reader.readLine();
                System.out.println("Сервер: " +
receivedMessage);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Результати роботи додатку зображені на рис. 3.2.

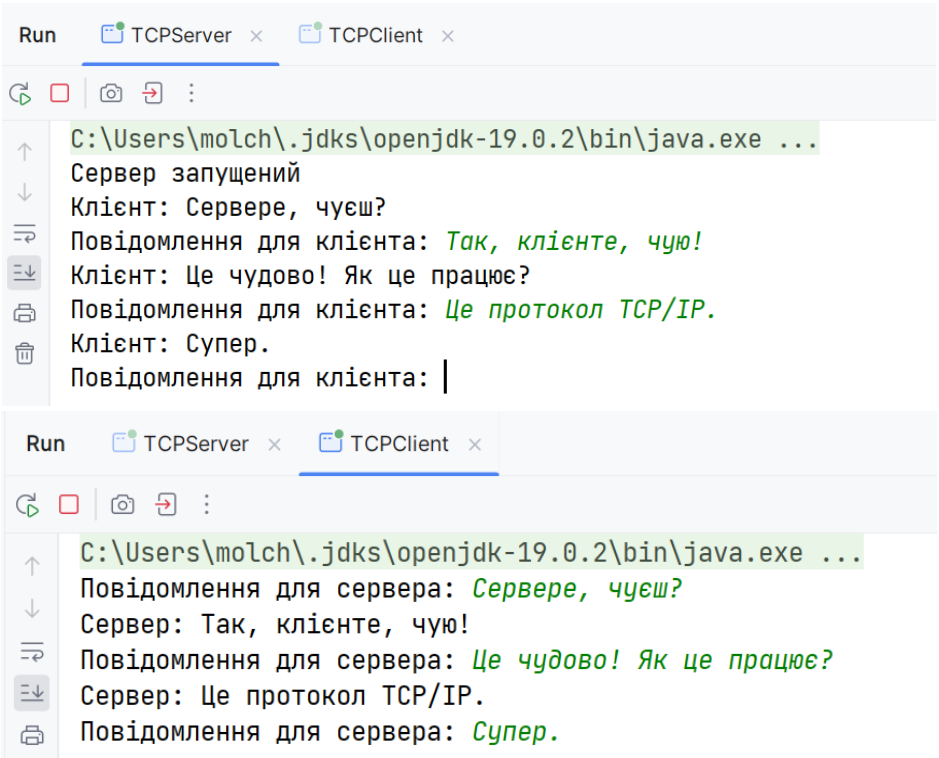

Рисунок 3.2 - Результат роботи класів TCPServer та TCPClient

Багатопоточність сервера та клієнта

Сервер повинен підтримувати багатопоточність, інакше він буде не в змозі обробляти кілька з'єднань одночасно. Сервер містить цикл, що очікує нового клієнтського з'єднання. Кожен раз, коли клієнт просить з'єднання, сервер створює новий потік. У наступному прикладі створюється клас

NetServerThread, що розширює клас Thread. Сервер передає кожному новому підключеному клієнту його порядковий номер.

```
import java.net.*;
import java.io.*;

public class NetServerThread extends Thread {
     Socket socket;
     int num; //номер підключення
     PrintStream ps;
     String msg;

     public NetServerThread(Socket s, int num) {
          socket = s;
          this.num=num;
          try {
               ps = new PrintStream(s.getOutputStream());
          } catch (IOException e) {
               System.out.println("Помилка: " + e);
          }
     }

     public static void main(String[] args) {
          try {
               ServerSocket server = new ServerSocket(2000);
               System.out.println("Сервер запущений...");
               int i=0; //лічильник підключень
               while (true) {
                    Socket s;
                    s = server.accept();
                    i++;
                    NetServerThread       nst       =       new
               NetServerThread(s,i);
                    nst.start();
               }
          } catch (IOException e) {
               System.out.println("Помилка: " + e);
          }
     }

     public void run() {
          msg = "Повідомлення: " + num;
          sendMessage(msg);
     }

     public void sendMessage(String msg) {
          ps.println(msg);
          System.out.println(msg + " <передача>");
          ps.flush();
     }
```

```
}
```

Для клієнтських додатків також необхідна підтримка багатопоточності. Наприклад, один потік очікує виконання операції вводу/виводу, а інші потоки виконують свої функції. Нижче наведено приклад програми, що реалізує отримання повідомлення клієнтом в потоці.

```
import java.net.*;
import java.io.*;

public class NetClientThread extends Thread {
   BufferedReader br = null;
   Socket s = null;

   public NetClientThread() {
       try {
           s = new Socket("127.0.0.1", 2000);
           InputStreamReader isr =
                   new InputStreamReader (s.getInputStream());
           br = new BufferedReader(isr);
       } catch (IOException e) {
           System.out.println("Помилка: " + e);
       }}

   public static void main(String[] args) {
       NetClientThread nct = new NetClientThread();
       nct.start();
   }

   public void run() {
       while (true) {
           try {
               String msg = br.readLine();
               if (msg == null) break;
               else System.out.println(msg);
           } catch (IOException e) {
               System.out.println("Помилка: " + e);
           }
       }
   }
}
```

В результаті в класі NetServerThread створюється серверний сокет на порті 2000 і починає прослуховувати з'єднання. Кожен раз, коли клієнт підключається, сервер створює для нього новий об'єкт NetServerThread в окремому потоці і відправляє повідомлення клієнту з номером підключення. У класі NetClientThread створюється клієнтський сокет і спробує підключитися до сервера на IP-адресі 127.0.0.1 (локальний хост) та

порті 2000. Після успішного підключення клієнт отримує вхідний потік для прийому повідомлень від сервера та виводить отримані повідомлення на екран. Програма NetClientThread працює в безкінечному циклі, постійно чекаючи на нові повідомлення від сервера (рис. 3.3).

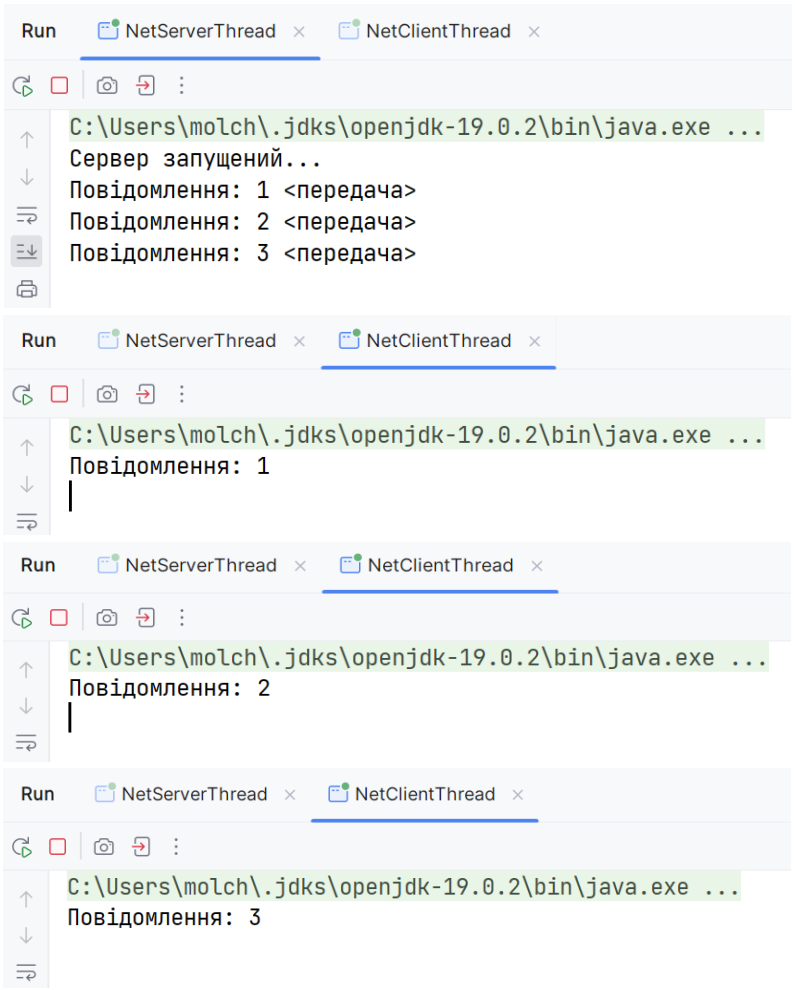

Рисунок 3.3 - Результат роботи класів NetServerThread та NetClientThread

Програмування дейтаграм

Дейтаграми – невеликі за обсягом пакети даних, які передаються між комп'ютерами по мережі на основі протоколу UDP (User Datagram Protocol). Для роботи з дейтаграммами в пакеті java.net існують два класи: DatagramSocket і DatagramPacket. Об'єкт DatagramSocket створює сам сокет, а дейтаграма являє собою об'єкт класу DatagramPacket.

У класі DatagramSocket є наступні конструктори:

DatagramSocket();

DatagramSocket(int port);
DatagramSocket(int port, InetAddress addr);
де port – номер порту, addr – адреса підключення.
Методи класу DatagramSocket наведені в табл. 4.5.

Таблиця 4.5 - Методи класу DatagramSocket

InetAddress getInetAddress()	Повертає адресу, до якої здійснюється підключення.
InetAddress getLocalAddress()	Повертає локальну адресу комп'ютера, з якої виконується підключення.
int getPort()	Повертає порт, до якого здійснюється підключення.
int getLocalPort()	Повертає локальний порт, через який здійснюється підключення.
void receive(DatagramPacket)	Чекає отримання дейтаграми і копіює дані в спеціальний DatagramPacket.
void send(DatagramPacket)	Посилає DatagramPacket.
void close()	Закриває сокет.

Конструктор класу DatagramPacket для отримання пакета:

```
DatagramPacket (byte[] buf, int length);
```

Конструктор класу DatagramPacket для відправки пакета:

```
DatagramPacket (byte[] buf, int length, InetAddress address, int port);
```

де buf – масив байт, що представляють пакет дейтаграми,
length – розмір даного пакету,
address – Internet-адреса відправки,
port – порт комп'ютера, на який відправляється дейтаграмма.
Методи класу DatagramPacket наведені в табл. 4.6.

Таблиця 4.6 - Методи класу DatagramPacket

byte[] getData()	Повертає масив байт, що представляють собою вміст дейтаграми.
void setData(byte[])	Записує в пакет дані, отримуючи в якості параметрів байтові масиви.
InetAddress getAddress()	Повертає адресу підключення.

void setAddress(InetAddress)	Встановлює адресу підключення.
int getPort()	Повертає порт підключення.
void setPort(int)	Встановлює порт підключення.
int getLength()	Повертає розмір дейтаграми.
void setLength(int)	Встановлює розмір дейтаграми.

У процесі роботи з об'єктами дейтаграм можуть виникати такі виняткові ситуації:

SocketException – помилка протоколу при зверненні до сокета;

SecurityException – помилка доступу до системи безпеки;

UnknownHostException – невірно вказано адресу InetAddress.

Нижче наводиться приклад, що демонструє діалог сервера з клієнтом в режимі "питання/відповідь", при якому обидві сторони приймають і вводять повідомлення з терміналу. Серверний додаток UDPServer створює свій сокет і пакет для прийому повідомлень від клієнта.

```
import java.io.*;
import java.net.*;

public class UDPServer {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket(9876);

            while (true) {
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
                socket.receive(receivePacket);

                String receivedMessage = new
String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Клієнт: " +
receivedMessage);

                InetAddress clientAddress =
receivePacket.getAddress();
                int clientPort = receivePacket.getPort();

                // Статична відповідь
                // String responseMessage = "Я отримав твоє
повідомлення: "+receivedMessage;
```

```
                // Відповідь з клавіатури
                InputStreamReader inputStreamReader = new
InputStreamReader(System.in);
                BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);
                System.out.print("Введіть повідомлення для
клієнта: ");
                String responseMessage =
bufferedReader.readLine();

                byte[] sendData = responseMessage.getBytes();
                DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length, clientAddress,
clientPort);
                socket.send(sendPacket);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Клієнтський додаток UDPClient приймає від користувача повідомлення у вигляді рядка і передає його на сервер. Програма сервера відображає його на екрані, приймає і передає набрану на клавіатурі відповідь за зворотною адресою клієнта.

```
import java.io.*;
import java.net.*;

public class UDPClient {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket();
            InputStreamReader inputStreamReader = new
InputStreamReader(System.in);
            BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);

            while (true) {
                System.out.print("Введіть повідомлення для
сервера ('exit' для виходу): ");

                String message = bufferedReader.readLine();
                if ("exit".equalsIgnoreCase(message)) {
                    break;
                }
```

```
                byte[] sendData = message.getBytes();
                InetAddress serverAddress =
InetAddress.getByName("localhost");
                int serverPort = 9876;
                DatagramPacket sendPacket = new
DatagramPacket(sendData, sendData.length, serverAddress,
serverPort);
                socket.send(sendPacket);

                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
                socket.receive(receivePacket);

                String receivedMessage = new
String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Сервер: " +
receivedMessage);
            }
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Результат роботи додатку (у варіанті відповідей з клавіатури) зображений на рис. 3.4.

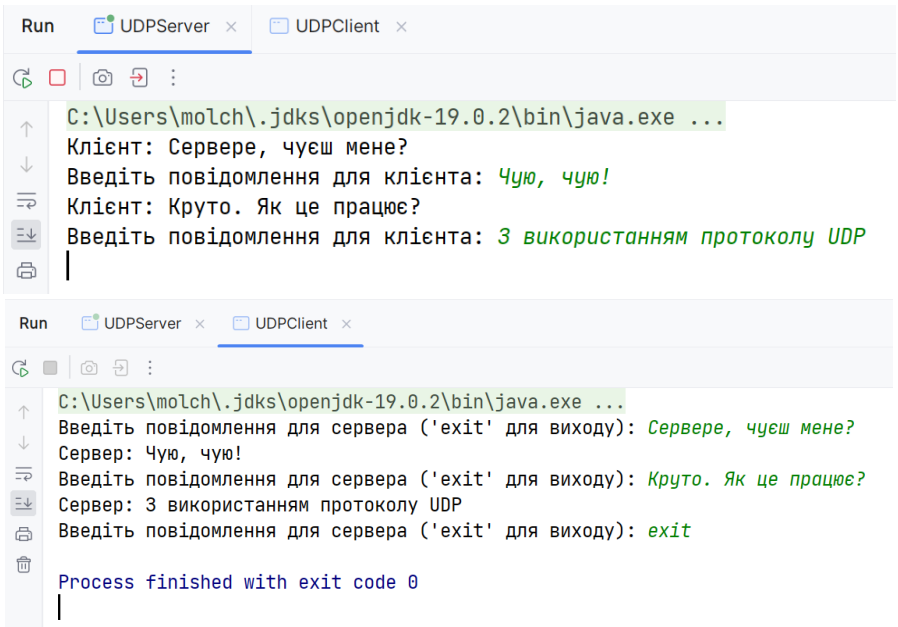

Рисунок 3.4 - Результат роботи класів UDPServer та UDPClient

Приклад чату з графічним інтерфейсом, в якому два і більше клієнтів спілкуються через сервер з використанням TCP/IP сокетів. Сервер чекає підключення клієнтів, а потім передає дані від одного клієнта іншому.

Код сервера:

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;

public class ServerWindow extends Application {

   TextArea taChat;
   private ArrayList<PrintWriter> clientOutputStreams;

   public static void main(String[] args) {
       launch(args);
   }

   @Override
   public void start(Stage primaryStage) {
       primaryStage.setTitle("Сервер");

       taChat = new TextArea();
       taChat.setEditable(false);
       Scene scene = new Scene(taChat, 400, 200);
       primaryStage.setScene(scene);
       primaryStage.show();

       clientOutputStreams = new ArrayList<>();
       new Thread(new ServerStart()).start();
       taChat.appendText("Сервер запущений.\n");
   }

   public class ServerStart implements Runnable {
       @Override
       public void run() {
           try {
```

```
                ServerSocket serverSock = new
ServerSocket(2222);
                while (true) {
                     Socket clientSock = serverSock.accept();
                     PrintWriter writer = new
                PrintWriter(clientSock.getOutputStream());
                  clientOutputStreams.add(writer);
                     Thread listener = new Thread(new
                ClientHandler(clientSock, writer));
                     listener.start();
                     taChat.appendText("З'єднання встановлено.
                \n");
                }
            } catch (IOException e) {
                taChat.appendText("Помилка при з'єднанні!\n");
            }
        }
    }

    public class ClientHandler implements Runnable {
        BufferedReader reader;
        Socket sock;
        PrintWriter client;

        public ClientHandler(Socket clientSocket, PrintWriter
user) {
            client = user;
            try {
               sock = clientSocket;
  reader = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
            } catch (IOException e) {
               taChat.appendText("Помилка!\n");
            }
        }

        @Override
        public void run() {
            String message;
            try {
                while ((message = reader.readLine()) != null) {
                    taChat.appendText("Отримано: " + message +
"\n");
                     Iterator<PrintWriter> it =
clientOutputStreams.iterator();
                    while (it.hasNext()) {
                        try {
                          PrintWriter writer = it.next();
                          writer.println(message);
                          taChat.appendText("Відправлено: " +
                     message + "\n");
```

```
                        writer.flush();
                    } catch (Exception e) {
                        taChat.appendText("Помилка при
                    розсилці повідомлення всім користувачам.
                    \n");
                    }
                }
            }
        } catch (IOException ex) {
            taChat.appendText("З'єднання втрачено. \n");
            clientOutputStreams.remove(client);
        }
    }
  }
}
```

Цей клас створює простий сервер. У вікні сервера - тільки TextArea для повідомлень. При створенні вікна автоматично створюється ServerSocket і починає відслідковувати підключення клієнтів. Внутрішній клас ServerStart імплементує інтерфейс Runnable і виконує функціонал старту сервера. Цей клас відслідковує підключення клієнтів, створюючи для кожного клієнта окремий потік ClientHandler для обробки комунікації. Внутрішній клас ClientHandler відповідає за обробку комунікації з окремим клієнтом. Він читає повідомлення з клієнта, додає їх до TextArea, а потім розсилає ці повідомлення всім підключеним клієнтам.

Код клієнта:

```
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Random;

public class ClientWindow extends Application {

   private String username;
   private boolean isConnected = false;
   private BufferedReader reader;
   private PrintWriter writer;
```

```
    private TextArea taChat;
    private TextField tfUsername, tfMessage;
    private Button btnConnect;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Клієнт");
        Label lbUsername = new Label("Ім'я користувача");
        tfUsername = new TextField();
        btnConnect = new Button("З'єдатись з сервером");
        taChat = new TextArea();
        tfMessage = new TextField();
        Button btnSend = new Button("Відправити");
        VBox vbox = new VBox(10);
        vbox.setPadding(new Insets(10));
        vbox.getChildren().addAll(lbUsername, tfUsername,
      btnConnect,taChat,tfMessage, btnSend);
        Scene scene = new Scene(vbox, 300, 400);
        primaryStage.setScene(scene);
        primaryStage.show();
        btnConnect.setOnAction(event -> connectToServer());
        btnSend.setOnAction(event -> sendMessage());
    }

    private void connectToServer() {
        if (!isConnected) {
            String enteredUsername =
tfUsername.getText().trim();
            if (!enteredUsername.isEmpty()) {
                username = enteredUsername;
            } else {
                username = "Клієнт "+ new Random().nextInt(999)
+ 1;
            }
            tfUsername.setText(username);
            tfUsername.setEditable(false);
            btnConnect.setDisable(true);

            try {
                Socket sock = new Socket("localhost", 2222);
                reader = new BufferedReader(new
InputStreamReader(sock.getInputStream()));
                writer = new
PrintWriter(sock.getOutputStream());
                writer.println(username + ": доєднався(-лася)
до чату.\n");
                writer.flush();
```

```
                isConnected = true;
                taChat.appendText("Підключено до сервера.\n");
            } catch (Exception e) {
                taChat.appendText("Помилка   при   підключенні!
Спробуйте знову.\n");
                tfUsername.setEditable(true);
                btnConnect.setDisable(false);
            }

            new Thread(() -> {
                try {
                    String stream;
                    while((stream = reader.readLine())!=null){
                        taChat.appendText(stream + "\n");
                    }
                } catch (Exception e) {
                    taChat.appendText("Помилка!\n");
                }
            }).start();
        } else {
            taChat.appendText("Ви вже підключені до сервера.
\n");
        }
    }

    private void sendMessage() {
        if (!tfMessage.getText().isEmpty()) {
            try {
                writer.println(username + ": " +
tfMessage.getText());
                writer.flush();
            } catch (Exception ex) {
                taChat.appendText("Помилка! Повідомлення не
відправлено!\n");
            }
            tfMessage.clear();
            tfMessage.requestFocus();
        }
    }
}
```

Цей клас створює клієнта чату. Метод connectToServer виконує підключення до сервера за допомогою сокета. Метод sendMessage відправляє повідомлення на сервер, використовуючи потік writer. Внутрішній анонімний клас new Thread(() -> {...}).start() створює новий потік для постійного слухання вхідного потоку reader та виводу повідомлень у вікно. Результат зображений на рис. 3.5.

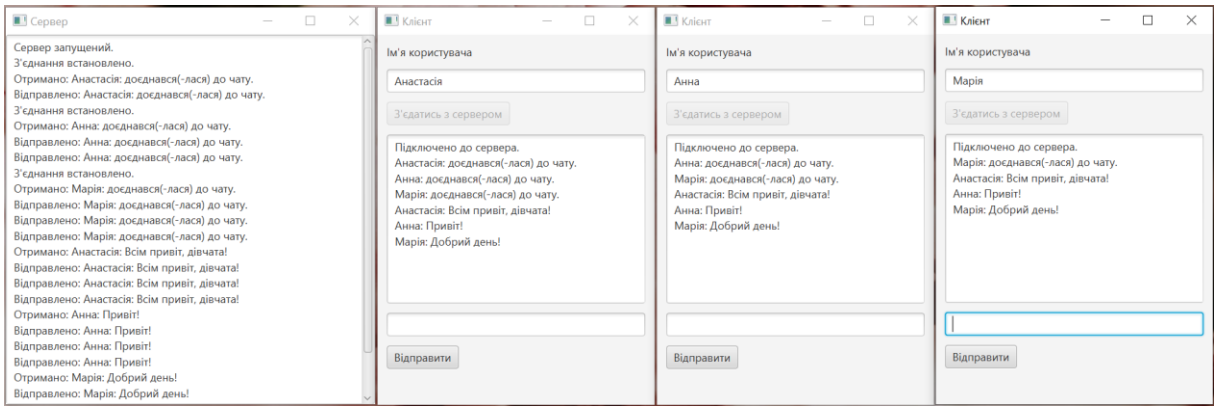

Рисунок 3.5 - Результат роботи класів SeverWindow та ClientWindow

Примітка. Щоб в IntelliJIDEA запустити одразу декілька екземплярів класу ClientWindow, необхідно відкрити налаштування Run/Debug Configurations, створити нову конфігурацію запуску та встановити галочку для опції Allow multiple instances (рис. 3.6).

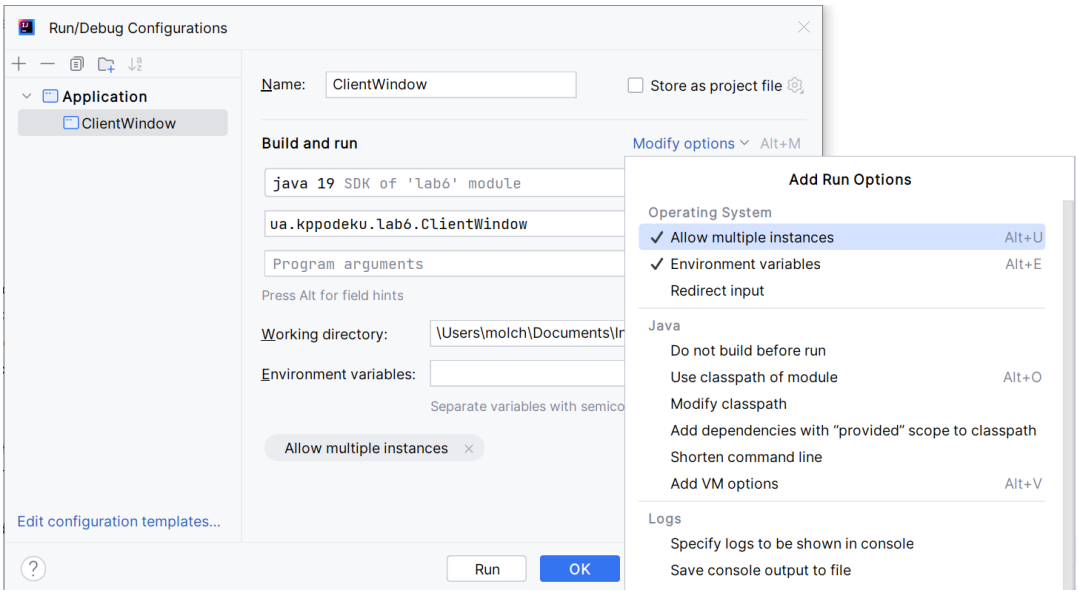

Рисунок 3.6 - Налаштування конфігурації запуску класів в IntelliJIDEA

3. Контрольні питання

1. Для чого призначений і яке значення повертає метод getAllByName() класу java.net.InetAddress?
2. 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?
5. Які дії необхідно зробити для обміну даними по UDP протоколу?

4. Правила техніки безпеки та охорони праці

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

5. Порядок проведення лабораторної роботи

Порядок проведення лабораторної роботи передбачає:

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

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

– отримання студентом варіанту індивідуального завдання;

– створення програмного коду мовою програмування Java в інтегрованому середовищі розробки Eclipse або IntelliJ IDEA.

6. Завдання до лабораторної роботи

Створити програму, яка дозволяє здійснювати взаємодію клієнта і сервера.

Варіанти:

1. Розробити консольну програму вирішення квадратного рівняння на сервері. Клієнт передає серверу значення коефіцієнтів через заданий роздільник. Сервер розраховує корені та повертає клієнту. Використовувати TCP-сервіс.
2. Розробити консольну програму вирішення квадратного рівняння на сервері. Клієнт передає серверу значення коефіцієнтів через заданий роздільник. Сервер розраховує корені та повертає клієнту. Використовувати UDP-сервіс.
3. Розробити консольну програму перевірки існування файлу. Клієнт передає серверу шлях до файла. Сервер повертає основну інформацію про цей файл, якщо він існує.

4. Розробити консольну програму відображення вмісту текстового файлу. Клієнт передає серверу шлях до файла. Сервер читає файл та повертає його вміст клієнту.
5. Розробити консольну програму обчислення факторіала числа на сервері. Клієнт передає серверу ціле число, для якого потрібно знайти факторіал. Сервер обчислює факторіал цього числа та повертає його клієнту.

7. Порядок оформлення звіту та його подання і захист

Підготовлений до захисту звіт до лабораторної роботи повинен містити:

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

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

Список літератури

Основна

1. Кузніченко С.Д., Терещенко Т.М. Крос-платформне програмування. Конспект лекцій. – Одеса: ОДЕКУ, 2016. – 104 с.

2. Галкін О.В., Катеринич Л.О., Шкільняк О.С. Програмування на Java 8: Навчальний посібник для студентів факультету комп'ютерних наук та кібернетики. – К.: ЛОГОС, 2017. – 186 с. URL: http://surl.li/kuyte

Додаткова література

3. Офіційний сайт Oracle. The Java Tutorials. URL: http://docs.oracle.com/javase/tutorial/index.html

4. Cay S. Horstmann. Core Java. Volume 1– Fundamentals. 12th Edition. 2022. URL: http://surl.li/kuycf

5. Cay S. Horsary Cornell. Core Java. Volume II–Advanced Features. 9th Edition. 2013. URL: http://surl.li/kuyhh

6. Joshua Bloch. Effective Java. 2th Edition. 2008. URL: http://surl.li/kuykf

7. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software. URL: http://surl.li/kuyob

8. Steve McConnell. Code complete. A practical handbook of software construction. 2th Edition. 2004. URL: http://surl.li/kuyqt

9. Brian Goetz. Java concurrency in practice. 2006. URL: http://surl.li/kuyxn

10. Эккель Б. Философия Java. Библиотека программиста. / Б. Эккель. – 4-е изд. URL: http://surl.li/kuzbo

11. Блинов И. Н. Java. Промышленное программирование: практ. пособ. / И. Н. Блинов, В. С. Романчик. URL:http://surl.li/kuzch