МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«Харківський Політехнічний Інститут»  
  
Кафедра «Стратегічного Управління»

ЗВІТ

з лабораторної роботи № 4

«Використання рефлексії та метапрограмування. Робота з потоками виконання»

Перевірила: старший викладач  
Вільхівська О. В.  
Виконав: ст. гр. КН-27

Харків, 2018

Лабораторна робота № 4  
Тема: Використання рефлексії та метапрограмування. Робота з потоками виконання  
Завдання

### **Індивідуальне завдання**

Створити програму графічного інтерфейсу користувача, яка призначена для побудови графіку довільних функцій. Користувач повинен увести дійсні значення a і b, функції f(x) і g(x) у вигляді рядків, які відповідають синтаксису Java. У програмі здійснюється обчислення функції h(x) відповідно до індивідуального завдання:

|  |  |
| --- | --- |
| Номери варіанту | Функція *h(x)* |
| 2 | *f(x + a) + g(x – b)* |

Після введення необхідних функцій, діапазону відображення графіку і натиснення відповідної кнопки здійснюється побудова графіку. Слід також передбачити функцію очищення рядків уведення й графіку.  
  
Для програмної реалізації обробки введених виразів слід застосувати динамічну компіляцію коду. Для створення застосунку графічного інтерфейсу користувача слід використати засоби JavaFX. Рекомендований підхід - використання компоненту LineChart.

### **Перегляд всіх полів класу**

Створити консольний застосунок, в якому користувач вводить ім'я класу і отримує інформацію про всі поля цього класу (включаючи закриті і захищені).

### **Створення застосунку графічного інтерфейсу користувача для отримання простих множників чисел**

За допомогою засобів JavaFX розробити застосунок графічного інтерфейсу користувача, в якому користувач вводить діапазон чисел (від і до), а у вікні відображаються числа і їх прості множники. Реалізувати можливість призупинення, відновлення потоку, а також повного припинення і повторного обчислення з новими даними.

### **Робота з BlockingQueue**

Створити консольну програму, в якій один потік виконання додає цілі числа до черги BlockingQueue,а інший обчислює їх середнє арифметичне.

### **Виклик функції для обраного класу (додаткове завдання)**

Створити класи з однойменними методами. Вибрати клас за ім'ям і викликати його метод.

### **Інтерпретація математичних виразів (додаткове завдання)**

Створити консольний застосунок, який дозволяє вводити математичні вирази, обчислювати і виводити результат. Вираз може складатися з констант, математичних операцій і дужок. Для реалізації використовувати засоби пакету javax.script.  
  
Примітка. Синтаксис математичних виразів JavaScript аналогічний Java. Результат можна виводити за допомогою функції print() без створення додаткових змінних.

### **Обчислення π в окремому потоці виконання (додаткове завдання)**

Реалізувати програму обчислення π з точністю до заданого ε як суму послідовності:  
  
Обчислення здійснювати в окремому потоці виконання. Під час виконання обчислення надавати користувачеві можливість уводити запит про кількість обчислених елементів суми.



### **Робота з потоками даних (додаткове завдання)**

Створити консольну програму, в якій виводяться всі додатні цілі числа, сума цифр яких дорівнює заданому значенню. Використати потоки даних.

Хід роботи

### **Завдання 1. Індивідуальне завдання**

*Код програми*

*Вміст GUIFX.java*

|  |
| --- |
| package task\_1;  import javafx.application.Application; import javafx.fxml.FXMLLoader; import javafx.scene.Scene; import javafx.scene.chart.LineChart; import javafx.scene.chart.NumberAxis; import javafx.scene.chart.XYChart; import javafx.scene.layout.BorderPane; import javafx.stage.Stage;  import java.io.IOException; import java.util.function.DoubleUnaryOperator;  public class GUIFX extends Application {  public void start(Stage primaryStage) {  FXMLLoader loader = new FXMLLoader();  try {  BorderPane root = (BorderPane)FXMLLoader.load(getClass().getResource("GUI.fxml"));  Scene scene = new Scene(root);  primaryStage.setScene(scene);  primaryStage.setTitle("Графики");  primaryStage.show();  } catch (IOException e) {  e.printStackTrace();  }  }  public static void main(String[] args) {  launch(args);  } } |

*Вміст GUIController.java*

|  |
| --- |
| package task\_1;  import java.io.IOException;  import java.io.PrintWriter;  import java.net.URL;  import java.util.ResourceBundle;  import java.util.function.DoubleBinaryOperator;  import javax.script.Bindings;  import javax.script.ScriptContext;  import javax.script.ScriptEngine;  import javax.script.ScriptEngineManager;  import javax.script.ScriptException;  import javax.tools.JavaCompiler;  import javax.tools.ToolProvider;  import javafx.application.Platform;  import javafx.event.ActionEvent;  import javafx.fxml.FXML;  import javafx.fxml.Initializable;  import javafx.scene.chart.LineChart;  import javafx.scene.chart.XYChart;  import javafx.scene.control.Label;  import javafx.scene.control.TableColumn;  import javafx.scene.control.TableView;  import javafx.scene.control.TextField;  import javafx.scene.control.TableColumn.CellEditEvent;  import javafx.scene.control.cell.PropertyValueFactory;  import javafx.scene.control.cell.TextFieldTableCell;  public class GUIController implements Initializable{  @FXML TextField textFieldF;  @FXML TextField textFieldG;  @FXML TextField textFieldA;  @FXML TextField textFieldB;  @FXML LineChart<Number,Number> chart;    private double a=0;  private double x=0;  private double b=0;  private static DoubleBinaryOperator f=(a,x)->x; // Default  private DoubleBinaryOperator g=(a,x)->-x; // Default  // private int number=1;  private ScriptEngineManager factory = new ScriptEngineManager();  private ScriptEngine engine = factory.getEngineByName("JavaScript");  private Bindings bindings = engine.getBindings(ScriptContext.ENGINE\_SCOPE);  @FXML public void doAChanged(ActionEvent event){  a = Double.valueOf(textFieldA.getText());  engine.put("a", new Double(a));  updateView();  }  @FXML public void doBChanged(ActionEvent event){  b = Double.valueOf(textFieldB.getText());  engine.put("b", new Double(b));  updateView();  }  @FXML public void doGChanged(ActionEvent event) throws ScriptException {  g = (b,x)->{  try {  return Double.valueOf(engine.eval("y = "+textFieldG.getText()).toString());  } catch (ScriptException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return b;  };  updateView();  }  @FXML public void doFChanged(ActionEvent event) throws ScriptException {  f = (a,x)->{  try {  return Double.valueOf(engine.eval("y = "+textFieldF.getText()).toString());  } catch (ScriptException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  return a;  };  updateView();  // JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();  // genSource(textFieldF.getText(),"F",++number);  // try {  // if(compiler.run(null, null, null, "src/task\_1/FunctionF"+number+".java")==0) {  // Class<?> cls = Class.forName("task\_1.FunctionF" + (number-1));  // cls.getMethod("change").invoke(null);  // updateView();  // }  // }  // catch(Exception e) {  // e.printStackTrace();  // }  }  @FXML public void doClear(ActionEvent event) {  textFieldF.clear();  textFieldG.clear();  textFieldA.clear();  textFieldB.clear();  chart.getData().clear();  }  @FXML public void doExit(ActionEvent event) {  Platform.exit();  }  public static void setF(DoubleBinaryOperator func) {  f = func;  System.out.println(func);  }  public LineChart<Number, Number> updateView() {  chart.setCreateSymbols(false);  chart.getData().clear();  double xTo = 10, xFrom = -10;  double h = (xTo - xFrom) / 100;  // Додаємо ім'я і точки першої функції:  XYChart.Series<Number, Number> fSeries = new XYChart.Series<>();  fSeries.setName("f(x)");  for (double x = xFrom; x <= xTo; x += h) {  engine.put("x", new Double(x));  fSeries.getData().add(new XYChart.Data<>(x, f.applyAsDouble(a,x)));  }  // Додаємо ім'я і точки другої функції:  XYChart.Series<Number, Number> gSeries = new XYChart.Series<>();  gSeries.setName("g(x)");  for (double x = xFrom; x <= xTo; x += h) {  engine.put("x", new Double(x));  gSeries.getData().add(new XYChart.Data<>(x, g.applyAsDouble(a,x)));  }  XYChart.Series<Number, Number> hSeries = new XYChart.Series<>();  hSeries.setName("h(x)");  for (double x = xFrom; x <= xTo; x += h) {  engine.put("x", new Double(x));  hSeries.getData().add(new XYChart.Data<>(x, f.applyAsDouble(a,x)+g.applyAsDouble(b,x)));  }  // chart.getData().addAll(fSeries, gSeries);  chart.getData().addAll(hSeries);  return chart;  }  // void genSource(String expression, String name, int number) {  // try (PrintWriter out = new PrintWriter("src/task\_1/Function" + name + number + ".java")) {  // out.println("package task\_1;");  // out.println("public class Function" + name +number+ "{");  // out.println(" public static void change() {");  // out.println(" GUIController.setF((a,x)->"+textFieldF.getText()+");");  // out.println(" }");  // out.println("}");  // }  // catch (IOException e) {  // e.printStackTrace();  // }  // }  public static void main(String[] args) {  GUIFX.main(new String[] {});  }  @Override  public void initialize(URL location, ResourceBundle resources) {  engine.put("a", new Double(a));  engine.put("b", new Double(b));  updateView();  }  } |

*Вміст GUI.fxml*

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?import javafx.scene.image.\*?>  <?import javafx.scene.chart.\*?>  <?import javafx.scene.control.\*?>  <?import java.lang.\*?>  <?import javafx.scene.layout.\*?>  <BorderPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="task\_1.GUIController">  <center>  <LineChart fx:id="chart" title="h(x) = f(x+a)+g(x-b)" BorderPane.alignment="CENTER">  <xAxis>  <NumberAxis side="BOTTOM" />  </xAxis>  <yAxis>  <NumberAxis side="LEFT" />  </yAxis>  </LineChart>  </center>  <bottom>  <AnchorPane prefHeight="143.0" prefWidth="600.0" BorderPane.alignment="CENTER">  <children>  <Label layoutX="50.0" layoutY="21.0" text="f(x)" />  <Label layoutX="48.0" layoutY="72.0" text="g(x)" />  <TextField fx:id="textFieldF" layoutX="48.0" layoutY="47.0" onAction="#doFChanged" />  <TextField fx:id="textFieldG" layoutX="48.0" layoutY="98.0" onAction="#doGChanged" />  <TextField fx:id="textFieldA" layoutX="247.0" layoutY="47.0" onAction="#doAChanged" />  <TextField fx:id="textFieldB" layoutX="247.0" layoutY="98.0" onAction="#doBChanged" />  <Label layoutX="247.0" layoutY="21.0" text="a" />  <Label layoutX="247.0" layoutY="81.0" text="b" />  </children>  </AnchorPane>  </bottom>  <top>  <MenuBar BorderPane.alignment="CENTER">  <menus>  <Menu mnemonicParsing="false" text="Файл">  <items>  <MenuItem mnemonicParsing="false" text="Очистить" onAction="#doClear" />  <MenuItem mnemonicParsing="false" text="Выход" onAction="#doExit" />  </items>  </Menu>  </menus>  </MenuBar>  </top>  </BorderPane> |

*Приклад виконання програми*

|  |
| --- |
|  |

### **Завдання 2. Перегляд всіх полів класу**

*Код програми*

*Вміст task\_2.java*

|  |
| --- |
| package task\_2;  import java.lang.reflect.Field;  import java.util.Scanner;  public class task\_2 {  public static void main (String[] args) throws Exception {  Scanner scanner = new Scanner(System.in);  String className = scanner.nextLine();    for (Field field : Class.forName(className).getDeclaredFields()) {  System.out.println(field.getName());  }  }  } |

*Приклад виконання програми*

|  |
| --- |
| java.lang.String value hash serialVersionUID serialPersistentFields CASE\_INSENSITIVE\_ORDER |

### **Завдання 3. Створення застосунку графічного інтерфейсу користувача для отримання простих множників чисел**

*Код програми*

*Вміст task\_3FX.java*

|  |
| --- |
| package task\_3;  import javafx.application.Application;  import javafx.fxml.FXMLLoader;  import javafx.scene.Scene;  import javafx.scene.layout.BorderPane;  import javafx.stage.Stage;  import java.io.FileInputStream;  import java.io.IOException;  public class task\_3FX extends Application {  public static void main(String[] args) {  launch(args);  }  @Override  public void start(Stage primaryStage) {  FXMLLoader loader = new FXMLLoader();  try {  BorderPane root = (BorderPane)FXMLLoader.load(getClass().getResource("task\_3.fxml"));  Scene scene = new Scene(root);  primaryStage.setScene(scene);  primaryStage.setTitle("Множители числа");  primaryStage.show();  } catch (IOException e) {  e.printStackTrace();  }  }  } |

*Вміст task\_3Controller.java*

|  |
| --- |
| package task\_3;  import javafx.event.ActionEvent;  import javafx.fxml.FXML;  import javafx.scene.control.\*;  public class task\_3Controller {  @FXML private TextField textFieldNumber;  @FXML private Button buttonStart;  @FXML private Button buttonSuspend;  @FXML private Button buttonResume;  @FXML private Button buttonStop;  @FXML private ProgressBar progressBar;  @FXML private TextArea textAreaResults;  private PrimeNumbers primeNumbers = new PrimeNumbers(this::addToTextArea, this::setProgress, this::finish);  @FXML private void startClick(ActionEvent actionEvent) {  try {  primeNumbers.setNumber(Integer.parseInt(textFieldNumber.getText()));  textAreaResults.setText("");  progressBar.setProgress(0);  buttonStart.setDisable(true);  buttonSuspend.setDisable(false);  buttonResume.setDisable(true);  buttonStop.setDisable(false);  primeNumbers.start();  } catch (NumberFormatException e) {  Alert alert = new Alert(Alert.AlertType.ERROR);  alert.setTitle("Помилка");  alert.setHeaderText("Хибний діапазон!");  alert.showAndWait();  }  }  @FXML private void suspendClick(ActionEvent actionEvent) {  primeNumbers.suspend();  buttonStart.setDisable(true);  buttonSuspend.setDisable(true);  buttonResume.setDisable(false);  buttonStop.setDisable(false);  }  @FXML private void resumeClick(ActionEvent actionEvent) {  primeNumbers.resume();  buttonStart.setDisable(true);  buttonSuspend.setDisable(false);  buttonResume.setDisable(true);  buttonStop.setDisable(false);  }  @FXML private void stopClick(ActionEvent actionEvent) {  primeNumbers.stop();  }  private void addToTextArea() {  textAreaResults.setText(textAreaResults.getText() + primeNumbers.getLastFound() + " ");  }  private void setProgress() {  progressBar.setProgress(primeNumbers.getPercentage());  }  private void finish() {  buttonStart.setDisable(false);  buttonSuspend.setDisable(true);  buttonResume.setDisable(true);  buttonStop.setDisable(true);  }  } |

*Вміст task\_3.fxml*

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?import javafx.geometry.\*?>  <?import javafx.scene.control.\*?>  <?import java.lang.\*?>  <?import javafx.scene.layout.\*?>  <?import javafx.geometry.Insets?>  <?import javafx.scene.control.Button?>  <?import javafx.scene.control.Label?>  <?import javafx.scene.control.TextArea?>  <?import javafx.scene.control.TextField?>  <?import javafx.scene.layout.BorderPane?>  <?import javafx.scene.layout.HBox?>  <BorderPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="task\_3.task\_3Controller">  <top>  <HBox prefHeight="3.0" prefWidth="600.0" spacing="10.0" BorderPane.alignment="CENTER">  <padding>  <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />  </padding>  <children>  <Label text="Число" />  <TextField fx:id="textFieldNumber" text="1000" />  <Button fx:id="buttonStart" minWidth="80.0" onAction="#startClick" text="Старт" />  <Button fx:id="buttonSuspend" disable="true" minWidth="80.0" onAction="#suspendClick" text="Пауза" />  <Button fx:id="buttonResume" disable="true" minWidth="80.0" onAction="#resumeClick" text="Продолжить" />  <Button fx:id="buttonStop" disable="true" minWidth="80.0" onAction="#stopClick" text="Завершить" />  </children>  </HBox>  </top>  <center>  <TextArea fx:id="textAreaResults" editable="false" wrapText="true" BorderPane.alignment="CENTER" />  </center>  <bottom>  <ProgressBar fx:id="progressBar" maxWidth="Infinity" progress="0" />  </bottom>  </BorderPane> |

*Вміст PrimeNumbers.java*

|  |
| --- |
| package task\_3;  import javafx.application.Platform;  public class PrimeNumbers implements Runnable {  private Thread primesThread; // нитка обчислення простих чисел  private int number; // кінець діапазону обчислення простих чисел  private int lastFound; // останнє знайдене просте число  private Runnable displayFunc; // функція, яка викликається для виведення знайденого числа  private Runnable percentageFunc; // функція, яка оновлює кількість відсотків виконаного процесу  private Runnable finishFunc; // функція, яка викликається після закінчення  private double percentage;  private boolean suspended;  private boolean stopped;  public PrimeNumbers(Runnable addFunc, Runnable percentageFunc, Runnable finishFunc) {  this.displayFunc = addFunc;  this.percentageFunc = percentageFunc;  this.finishFunc = finishFunc;  }  public int getTo() {  return number;  }  public void setNumber(int number) {  this.number = number;  }  public synchronized int getLastFound() {  return lastFound;  }  private synchronized void setLastFound(int lastFound) {  this.lastFound = lastFound;  }  public synchronized double getPercentage() {  return percentage;  }  private synchronized void setPercentage(double percentage) {  this.percentage = percentage;  }  public synchronized boolean isSuspended() {  return suspended;  }  private synchronized void setSuspended(boolean suspended) {  this.suspended = suspended;  }  public synchronized boolean isStopped() {  return stopped;  }  private synchronized void setStopped(boolean stopped) {  this.stopped = stopped;  }  @Override  public void run() {  boolean prime;  for (int n = 2; n <= number/2; n++)  {  try {  setPercentage(n \* 1.0 / number\*2);  // Оновлюємо кількість відсотків:  if (percentageFunc != null) {  Platform.runLater(percentageFunc);  }  prime = true;  for (int i = 2; i \* i <= n; i++) {  if (n % i == 0) {  prime = false;  break;  }  }  Thread.sleep(20);  if ((prime)&&(number % n==0)) {  setLastFound(n);  // Відображаємо знайдене просте число:  if (displayFunc != null) {  displayFunc.run();  }  }  }  catch (InterruptedException e) {  // залежно від стану об'єкта чекаємо на продовження або завершуємо пошук:  while (isSuspended()) {  try {  Thread.sleep(100);  }  catch (InterruptedException e1) {  // Перервали у стані очікування:  if (isStopped()) {  break;  }  }  }  if (isStopped()) {  break;  }  }  }  prime = true;  for (int i=2; i\*i<=number; i++) {  if (number % i == 0) {  System.out.println(i);  prime = false;  break;  }  }  if (prime) {  setLastFound(number);  if (displayFunc != null) {  displayFunc.run();  }  }  if (finishFunc != null) {  Platform.runLater(finishFunc);  }  }  public void start() {  primesThread = new Thread(this);  setSuspended(false);  setStopped(false);  primesThread.start();  }  public void suspend() {  setSuspended(true);  primesThread.interrupt();  }  public void resume() {  setSuspended(false);  primesThread.interrupt();  }  public void stop() {  setStopped(true);  primesThread.interrupt();  }  } |

*Приклад виконання програми*

|  |
| --- |
|  |

### **Завдання 4. Робота з BlockingQueue**

*Код програми*

*Вмст task\_4.java*

|  |
| --- |
| package task\_4;  import java.util.concurrent.ArrayBlockingQueue;  import java.util.concurrent.BlockingQueue;  public class task\_4 {  public static void main(String[] args) throws InterruptedException {  // Створюємо чергу з 10 елементів:  BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(10);    // Створюємо і запускаємо два потоки - для запису і читання:  Thread producerThread = new Thread(new Producer(queue, 10));  Thread consumerThread = new Thread(new Consumer(queue, 10));  producerThread.start();  consumerThread.start();    // Чекаємо 10 секунд і перериваємо перший потік:  Thread.sleep(1000);  producerThread.interrupt();  }  } |

*Вмст Producer.java*

|  |
| --- |
| package task\_4;  import java.util.concurrent.BlockingQueue;  public class Producer implements Runnable {  private BlockingQueue<Integer> queue;  int countToAdd;  public Producer(BlockingQueue<Integer> queue, int countToAdd) {  this.queue = queue;  this.countToAdd = countToAdd;  }  public void run() {  // Намагаємося додавати числа:  try {  for (int i = 1; i <= countToAdd; i++) {  System.out.printf("Added: %d%n", i);  queue.put(i);  Thread.sleep(100);  }  }  catch (InterruptedException e) {  System.out.println("Producer interrupted");  }  }  } |

*Вмст Consumer.java*

|  |
| --- |
| package task\_4;  import java.util.concurrent.BlockingQueue;  public class Consumer implements Runnable {  private BlockingQueue<Integer> queue;  int countToTake;  double avg=0;  public Consumer(BlockingQueue<Integer> queue, int countToTake) {  this.queue = queue;  this.countToTake = countToTake;  }  public void run() {  // Вилучаємо числа:  try {  for (int i = 1; i <= countToTake; i++) {  int number = queue.take();  avg = (number+avg\*(i-1))/i;  System.out.printf("Taken by customer: %d, Average: %f%n", number, avg);  }  }  catch (InterruptedException e) {  System.out.println("Consumer interrupted");  }  }  } |

*Приклад виконання програми*

|  |
| --- |
| Added: 1 Taken by customer: 1, Average: 1.000000 Added: 2 Taken by customer: 2, Average: 1.500000 Added: 3 Taken by customer: 3, Average: 2.000000 Added: 4 Taken by customer: 4, Average: 2.500000 Added: 5 Taken by customer: 5, Average: 3.000000 Added: 6 Taken by customer: 6, Average: 3.500000 Added: 7 Taken by customer: 7, Average: 4.000000 Added: 8 Taken by customer: 8, Average: 4.500000 Added: 9 Taken by customer: 9, Average: 5.000000 Added: 10 Taken by customer: 10, Average: 5.500000 Producer interrupted |

### **Завдання 5. Виклик функції для обраного класу (додаткове завдання)**

*Код програми*

*Вміст task\_5.java*

|  |
| --- |
| package task\_5;  public class task\_5 {  public static void main (String[] args) {  Class<?> c1 = class1.class;  Class<?> c2 = class2.class;   try {  c1.getMethod("print", null).invoke(null);  c2.getMethod("print", String.class).invoke(c2.newInstance(), "Some text from a nonStatic method of Class2");  }  catch (Exception e) {  e.printStackTrace();  }  } } |

*Вміст class1.java*

|  |
| --- |
| package task\_5;  public class class1 {  public static void print() {  System.out.println("This is a static print method of Class1");  } } |

*Вміст class2.java*

|  |
| --- |
| package task\_5;  public class class2 {  public void print(String str) {  System.out.println(str);  } } |

*Приклад виконання програми*

|  |
| --- |
| This is a static print method of Class1 Some text from a nonStatic method of Class2 |

### **Завдання 6. Інтерпретація математичних виразів (додаткове завдання)**

*Код програми*

*Вміст task\_6.java*

|  |
| --- |
| package task\_6\_2;  import java.io.\*; import java.lang.reflect.Method; import java.util.Scanner; import javax.tools.\*;  public class task\_6 {  final String sourceFile = "src/task\_6\_2/NumberFun.java";   void genSource(String expression) {  try (PrintWriter out = new PrintWriter(sourceFile)) {  out.println("package task\_6\_2;");  out.println("public class NumberFun {");  out.println(" public static double transform(double a) {");  // Додаємо порожній рядок, щоб перетворити на рядок результат будь-якого типу:  out.println(" return " + expression + ";");  out.println(" }");  out.println("}");  }  catch (IOException e) {  e.printStackTrace();  }   }   boolean compile() {  JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();  return compiler.run(null, null, null, sourceFile) == 0;  }   @SuppressWarnings("resource")  public static void main(String[] args) {  task\_6 sp = new task\_6();  Scanner scan = new Scanner(System.in);  System.out.println("Введите выражение, выполняемое над переменной типа double a:");  String expression = scan.nextLine().replaceAll("\"", "\\\"");  sp.genSource(expression);  try {  if (sp.compile()) {  System.out.println("Введите значение a:");  double a = scan.nextDouble();  Class<?> cls = Class.forName("task\_6\_2.NumberFun");  Method m = cls.getMethod("transform", double.class);  System.out.println(m.invoke(null, a));  }  else {  System.out.println("Ошибка ввода!");  }  }  catch (Exception e) {  e.printStackTrace();  }  } } |

*Вміст NumberFun.java*

|  |
| --- |
| package task\_6\_2; public class NumberFun {  public static double transform(double a) {  return (a+1)/2;  } } |

*Приклад виконання програми*

|  |
| --- |
| Введите выражение, выполняемое над переменной типа double a: a-10 Введите значение a: 22 12.0 |

### **Завдання 7. Обчислення π в окремому потоці виконання (додаткове завдання)**

*Код програми*

*Вміст piThread.java*

|  |
| --- |
| package task\_7;  public class piThread implements Runnable {  final double eps;  double pi=0;  private int completness;  piThread(double eps){  this.eps = eps;  }  synchronized public int getCompletness() {  return completness;  }  synchronized public void setCompletness(int completness) {  this.completness = completness;  }  synchronized public double getPi() {  return pi;  }  synchronized public void setPi(double pi) {  this.pi = pi;  }  @Override  public void run() {  double addend = 1;  for (int i=1,j=1; Math.abs(addend)>eps; i++,j\*=-1) {  addend=j\*4.0/(i\*2-1);  pi+=addend;  setCompletness(i);  try {  Thread.sleep(1);  } catch (InterruptedException e) {  // TODO Auto-generated catch block  e.printStackTrace();  }  }  Thread.currentThread().interrupt();  }  public static void main(String[] args) {  // TODO Auto-generated method stub  }  } |

*Вміст task\_7.java*

|  |
| --- |
| package task\_7;  import java.util.Scanner;  public class task\_7 {   public static void main(String[] args) throws Exception{  piThread th = new piThread(0.0001);  Thread pi = new Thread(th);  pi.start();  while(pi.isAlive()) {  Scanner scanner = new Scanner(System.in);  if (scanner.hasNextLine()) {  System.out.println(th.getCompletness()+" "+th.getPi());  }  }  pi.interrupt();  }  } |

*Приклад виконання програми*

|  |
| --- |
| 256 3.137686418490669  750 3.1402593208490526  1171 3.142446624399089  1537 3.1422432716081237  1871 3.1421271270951805 |

Висновки

Навідміну від процесів потоки розділяють спільну пам’ять. Під час виконання багатьох потоків завдяки квантуванню часу створюється ілюзія одночасного виконання. Багатопотоковість є популярною моделлю програмування, бо дозволяє багатьом потокам існувати в контексті одного процесу.

Рефлексія дозволяє отримувати дані про класи, методи під час виконання програми, а також обробляти типи, що з’явилися під час виконання програми.