Лабораторна робота №14

Паралельне виконання.Шаблон Worker Thread

***Мета робот:***

- реалізація та демонстрація механізму обслуговування черги задач;

- використання шаблонів Worker Thread, Command і Factory Method

для реалізації паралельного виконання обробки колекції об'єктів;

- використання модульного тестування.

**1 ЗАГАЛЬНЕ ЗАВДАННЯ**

Продемонстрировать возможность параллельной обработки элементов

коллекции (поиск минимума, максимума, вычисление среднего значения, отбор

по критерию, статистическая обработка и т.д.).

Управление очередью задач (команд) реализовать с помощью шаблона

Worker Thread.

Обеспечить диалоговый интерфейс с пользователем.

Разработать класс для тестирования функциональности приложения.

Использовать комментарии для автоматической генерации документации

средствами javadoc.

**2 ОПИС ПРОГРАМИ**

**2.1 Ієрархія та структура класів**

**class Info-організовує діалог з користувачем**

**class Calculator виконує індивідуальне завдання**

**class Serializator виконує збереження данних классу Calculator у файл**

**interface Calculator- містить основні методи калькуляторів**

**class Calculator Area –рахує площу**

**class Calculator FactoreArea – створює об єкт типу CalculatorArea**

**class Calculator Perimetr – рахує періметр індивідуального завдання**

**class Calculator FactoryPerimetr- створює об єкт типу CalculatorPerimetr**

**class GuiCalulator-додатковий класс для графічного інтерфейсу**

**class Main –викликає класс контроллер та файл з версткою**

**sample.fxml-містить верстку вікна Windows**

**class Application-організовує роботу по шаблону та точка входу у программу**

**interface Queue-робить чергу для запитів до додатку**

**class Menu-містить запити у своїй коллекції та виконує їх**

**interface Command-містить різні типи команд**

**class Command Queue –Організовує чергу команд на основі двох интерфейсів**

**2.2 Опис програми**

На рисунку 2.1 наведена структура розробленої програми

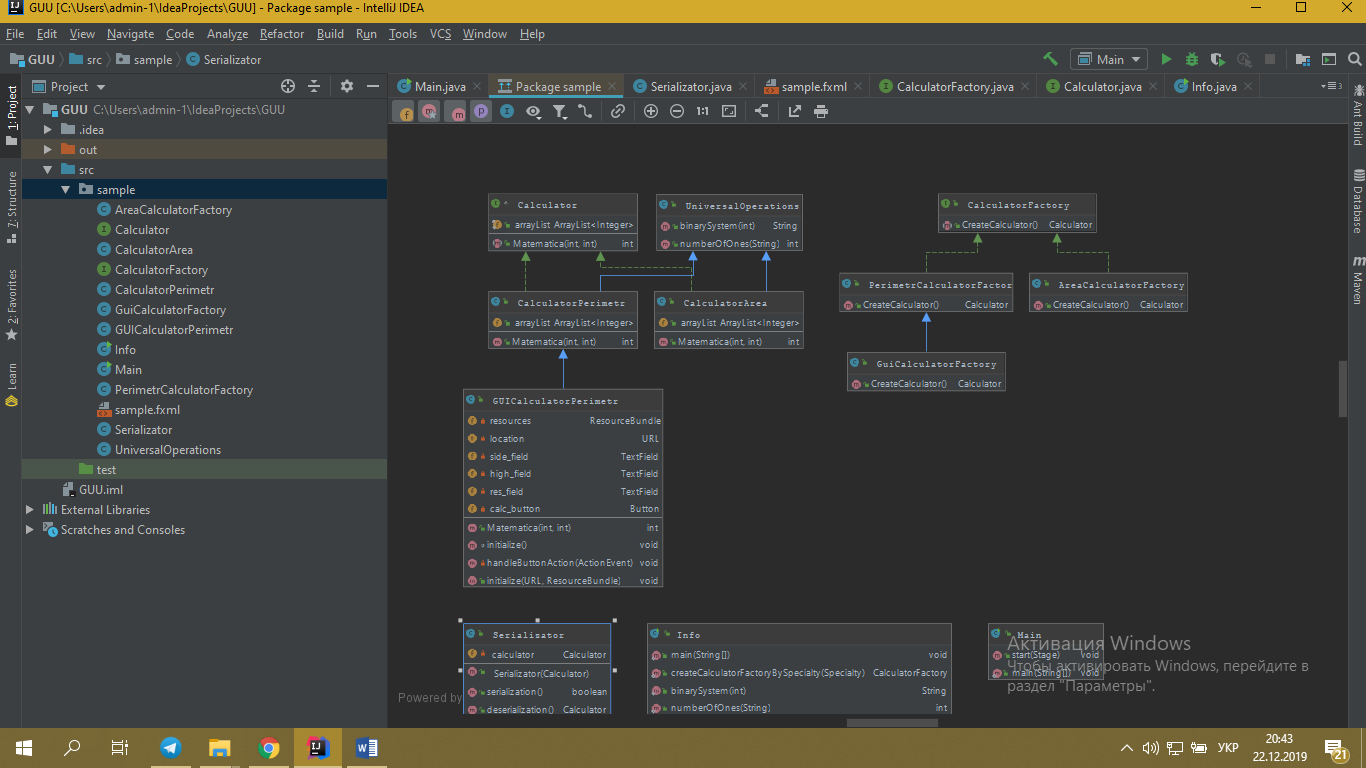


Рис 2.1 – структура програми

**2.3 Важливі фрагменти програми**

2.3.1 Файл*Application.java*

public class Application {  
  
 private static Application *application* =new Application();  
  
 private static Calculator *calculator* = new PerimetrCalculatorFactory().CreateCalculator();  
 private Application()  
 {  
  
 }  
  
 public static synchronized Application getInstance()  
 {  
  
 return *application*;  
 }  
 public void run()  
 {  
  
Menu menu =new Menu();  
menu.add(new IsEmptyCommand(*calculator*));  
menu.add(new InsertCommand(*calculator*,12));  
menu.add(new InsertCommand(*calculator*,1));  
menu.add(new InsertCommand(*calculator*,3));  
menu.add(new InsertCommand(*calculator*,37));  
  
menu.add(new ExecuteCommand(*calculator*));  
menu.execute();  
 }  
  
}

2.3.1 Файл*Menu.java*

import java.util.ArrayList;  
  
public class Menu implements Command {  
  
ArrayList<Command> menu =new ArrayList<Command>();  
  
  
public Command add(Command command)  
{  
 menu.add(command);  
 return command;  
}  
  
 public void execute() {  
 for (Command c:menu  
 ) {  
  
 c.execute();  
 }  
 }  
}

2.3.1 Файл*QueueCommand.java*

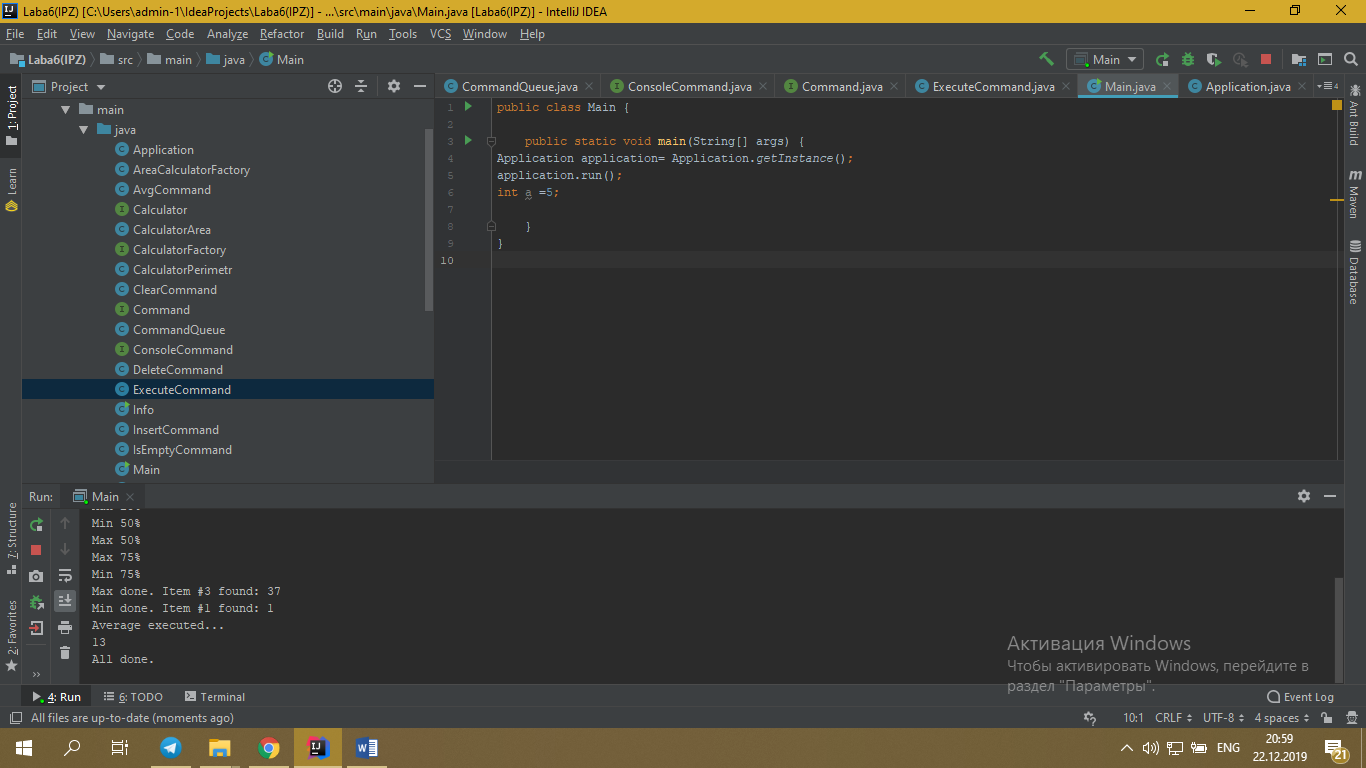
import java.util.Vector;  
  
public class CommandQueue implements Queue {  
private Vector<Command> tasks;  
  
private boolean waiting;  
  
private boolean shutdown;  
  
public void shutDown()  
  
{  
 shutdown=true;  
}  
  
public CommandQueue()  
{  
 tasks =new Vector<Command>();  
 waiting =false;  
 new Thread(new Worker()).start();  
}  
 public void put(Command cmd) {  
tasks.add(cmd);  
if (waiting)  
{  
 synchronized (this){  
 notifyAll();  
 }  
}  
 }  
  
 public Command take() {  
 if (tasks.isEmpty())  
 {  
 synchronized (this)  
 {  
 waiting=true;  
 try  
 {  
 try {  
 wait();  
 } catch (InterruptedException e) {  
 waiting=false;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 return (Command) tasks.remove(0);  
 }  
  
 private class Worker implements Runnable  
 {  
  
 public void run() {  
while(!shutdown)  
{  
 Command r =take();  
 r.execute();  
}  
 }  
 }  
  
}

2.3.1 Файл*ExecuteCommand.java*

public interface Command {  
  
 public void execute();  
}

import java.util.concurrent.TimeUnit;  
  
  
public class ExecuteCommand implements Command{  
  
 */\*\* Объект, реализующий интерфейс  
 \* \* обслуживает коллекцию объектов \*/* private Calculator calculator;  
 */\*\* Возвращает поле {****@linkplain*** *\* \* @return значение {****@linkplain*** *\*/* public Calculator getView() {  
 return calculator;  
 }  
 */\*\* Устанавливает поле {****@linkplaiw****}  
  
 \** ***@return*** *новое значение {****@linkpview****}  
 \*/* public Calculator setCalculator(Calculator calculator) {  
 return this.calculator = calculator;  
 }  
*/\*\* Инициализирует поле  
 \** ***@param*** *calculator объект, реализующий  
 \*/*public ExecuteCommand(Calculator calculator) {  
 this.calculator = calculator;  
}  
  
 public char getKey() {  
 return 'e';  
 }  
 @Override  
 public String toString() {  
 return "'e'xecute";  
 }  
  
 public void execute() {  
  
 CommandQueue queue1 = new CommandQueue();  
 CommandQueue queue2 = new CommandQueue();  
  
*/\*\*  
  
 ExecutorService exec1 = Executors.newSingleThreadExecutor();  
 ExecutorService exec2 = Executors.newSingleThreadExecutor();  
 /\*\*/* MaxCommand maxCommand = new MaxCommand((CalculatorPerimetr) calculator);  
 AvgCommand avgCommand = new AvgCommand((CalculatorPerimetr)calculator);  
 MinCommand minCommand = new MinCommand((CalculatorPerimetr)calculator);  
 System.*out*.println("Execute all threads...");  
  
*/\*\*  
  
 exec1.execute(minMaxCommand);  
 exec2.execute(maxCommand);  
 exec2.execute(avgCommand);  
 /\*\*/* queue1.put(minCommand);  
 queue2.put(maxCommand);  
 queue2.put(avgCommand);  
  
 /\*\*/  
 try {  
 while (  
 maxCommand.running() ||  
 minCommand.running()) {  
 TimeUnit.*MILLISECONDS*.sleep(100);  
 }  
*/\*\*  
  
 exec1.shutdown();  
 exec2.shutdown();  
 /\*\*/* queue1.shutDown();  
 queue2.shutDown();  
  
 /\*\*/  
  
 TimeUnit.*SECONDS*.sleep(1);  
 } catch (InterruptedException e) {  
 System.*err*.println(e);  
 }  
 System.*out*.println("All done.");  
 }  
}

**3 РЕЗУЛЬТАТИ**



**ВИСНОВКИ**

Навчились працювати тестами та шаблонами .