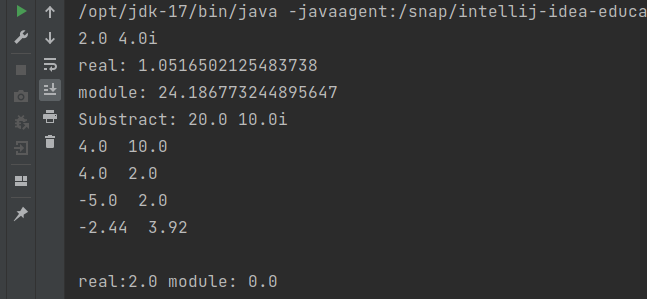
**Лабораторна робота №4**

Використання Gof-патернів у додатках з MVC архітектурою.

**Скриншоти роботи програми**

Робота з програмою:



**Код**

Робочі класи та інтерфейси

ppackage mvc.interfaces;  
  
public interface ComplexValueInterface {  
  
 double getReal();  
  
 void setReal(double real);  
  
 double getImaginary();  
  
 void setImaginary(double imaginary);  
  
}

package mvc;  
  
import mvc.interfaces.ComplexValueInterface;  
  
public class ComplexValue implements ComplexValueInterface {  
  
 protected double real;  
  
 protected double imaginary;  
  
 public ComplexValue() {}  
  
 public ComplexValue(double real, double imaginary) {  
 this.real = real;  
 this.imaginary = imaginary;  
 }  
  
 public ComplexValue add(ComplexValue value) {  
 real += value.real;  
 imaginary += value.imaginary;  
 return this;  
 }  
  
 public ComplexValue subtract(ComplexValue value) {  
 real -= value.real;  
 imaginary -= value.imaginary;  
 return this;  
 }  
  
 public ComplexValue multiply(ComplexValue value) {  
 double temp = real \* value.real - imaginary \* value.imaginary;  
 imaginary = real \* value.imaginary + value.real \* imaginary;  
 real = temp;  
 return this;  
 }  
  
 public ComplexValue divide(ComplexValue value) {  
 if (value.real == 0 && value.imaginary == 0) throw new ArithmeticException();  
 double temp = (real \* value.real + imaginary \* value.imaginary) / (int) (Math.pow(value.real, 2) + Math.pow(value.imaginary, 2));  
 imaginary = (value.real \* imaginary - real \* value.imaginary) / (int) (Math.pow(value.real, 2) + Math.pow(value.imaginary, 2));  
 real = temp;  
 return this;  
 }  
  
 public ComplexValue add\_two(ComplexValue value1, ComplexValue value2) {  
 value1.real += value2.real;  
 value1.imaginary += value2.imaginary;  
 return value1;  
 }  
  
 public ComplexValue subtract\_two(ComplexValue value1, ComplexValue value2) {  
 value1.real -= value2.real;  
 value1.imaginary -= value2.imaginary;  
 return value1;  
 }  
  
 public ComplexValue multiply\_two(ComplexValue value1, ComplexValue value2) {  
 double temp = value1.real \* value2.real - value1.imaginary \* value2.imaginary;  
 imaginary = value1.real \* value2.imaginary + value2.real \* value1.imaginary;  
 value1.real = temp;  
 return value1;  
 }  
  
 public ComplexValue divide\_two(ComplexValue value1, ComplexValue value2) {  
 if (value2.real == 0 && value2.imaginary == 0) throw new ArithmeticException();  
 double temp = (value1.real \* value2.real + value1.imaginary \* value2.imaginary) / (int) (Math.pow(value2.real, 2) + Math.pow(value2.imaginary, 2));  
 imaginary = (value2.real \* value1.imaginary - value1.real \* value2.imaginary) / (int) (Math.pow(value2.real, 2) + Math.pow(value2.imaginary, 2));  
 real = temp;  
 return this;  
 }  
  
 public double getReal() {  
 return real;  
 }  
  
 public void setReal(double real) {  
 this.real = real;  
 }  
  
 public double getImaginary() {  
 return imaginary;  
 }  
  
 public void setImaginary(double imaginary) {  
 this.imaginary = imaginary;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null) {  
 return false;  
 }  
 if (o instanceof ComplexValue) {  
 ComplexValue temp = (ComplexValue) o;  
 return Double.compare(temp.real, real) == 0 && Double.compare(temp.imaginary, imaginary) == 0;  
 } else {  
 return false;  
 }  
 }  
  
 @Override  
 public int hashCode() {  
 int result = Double.hashCode(real);  
 result = 31 \* result + Double.hashCode(imaginary);  
 return result;  
 }  
  
 @Override  
 public String toString() {  
 return real +  
 " " + imaginary +  
 "i";  
 }  
}

package mvc;  
  
public class ExponentialComplexValue extends ComplexValue{  
  
 private double module;  
  
 private double argument;  
  
 public ExponentialComplexValue() {}  
  
 public ExponentialComplexValue(double real, double imaginary) {  
 super(real, imaginary);  
 module = calculateModule(this);  
 argument = calculateArgument(this);  
 }  
  
 public static ExponentialComplexValue transformComplexValue(ComplexValue complexValue){  
 return new ExponentialComplexValue(complexValue.real, complexValue.imaginary);  
 }  
  
 private double calculateModule(ExponentialComplexValue value){  
 return Math.*sqrt*(Math.*pow*(value.real, 2) + Math.*pow*(imaginary, 2));  
 }  
  
 private double calculateArgument(ExponentialComplexValue value){  
 double result = Math.*atan*(value.imaginary/ value.real);  
 return Double.*compare*(value.real, 0) < 0 ? result + Math.*PI* : result;  
 }  
  
 public double getModule() {  
 return module;  
 }  
  
 public void setModule(double module) {  
 this.module = module;  
 }  
  
 public double getArgument() {  
 return argument;  
 }  
  
 public void setArgument(double argument) {  
 this.argument = argument;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null) {  
 return false;  
 }  
 if (o instanceof ExponentialComplexValue) {  
 ExponentialComplexValue temp = (ExponentialComplexValue) o;  
 return Double.compare(temp.real, real) == 0 && Double.compare(temp.imaginary, imaginary) == 0  
 && Double.compare(temp.module, module) == 0 && Double.compare(temp.argument, argument) == 0;  
 } else {  
 return false;  
 }  
 }  
  
 @Override  
 public int hashCode() {  
 int result = super.hashCode();  
 result = 31 \* result + Double.hashCode(module);  
 result = 31 \* result + Double.hashCode(argument);  
 return result;  
 }  
  
 @Override  
 public String toString() {  
 return real +  
 " " + imaginary +  
 "i (" +  
 module + String.format("e%.5fi)", argument);  
 }  
}

MVC:

package mvc;  
  
public class ComplexValueView {  
 public void printCValue(int value){  
 System.*out*.print(value); // Вывод содержимого  
 System.*out*.print(" ");  
 }  
  
 public void printCValue(){  
 System.*out*.println("");  
 }  
}

package mvc;  
  
public class ECValueView {  
 public void printECValue(int value){  
 System.*out*.print(value); // Вывод содержимого  
 System.*out*.print(" ");  
 }  
  
 public void printECValue(){  
 System.*out*.println("");  
 }  
}

Controller

package mvc;  
  
import patterns.Mediator;  
  
public class CValueController{ // контроллер  
 private ComplexValue model;  
 private ComplexValueView view;  
 private static CValueController *instance*;  
  
 public CValueController(ComplexValue model, ComplexValueView view){  
 this.model=model;  
 this.view=view;  
 }  
  
 public ComplexValue add(ComplexValue value1)  
 {  
 return model.add(value1);  
 }  
  
  
  
 public ComplexValue subtract(ComplexValue value1)  
 {  
 return model.subtract(value1);  
 }  
  
 public ComplexValue multiply(ComplexValue value1)  
 {  
 return model.multiply(value1);  
 }  
  
 public ComplexValue divide(ComplexValue value1)  
 {  
 return model.divide(value1);  
 }  
  
 public double getReal()  
 {  
 return model.getReal();  
 }  
  
 public void setReal(double real)  
 {  
 model.setReal(real);  
 }  
  
 public double getImaginary()  
 {  
 return model.getImaginary();  
 }  
  
 public void setImaginary(double imaginary)  
 {  
 model.setImaginary(imaginary);  
 }  
  
  
 public static CValueController getInstance(ComplexValue model, ComplexValueView view) {  
 if (*instance* == null) {  
 *instance* = new CValueController(model, view);  
 }  
 return *instance*;  
 }  
  
 public void checkCValueRealNotEqualToECValueModule(ECValueController value){  
 Mediator.*checkCValueRealNotEqualToECValueModule*(this, value);  
 }  
  
  
}

package mvc;  
  
public class ECValueController {  
  
 final private ExponentialComplexValue model;  
 final private ECValueView view;  
  
 public ECValueController(ExponentialComplexValue model, ECValueView view){  
 this.model=model;  
 this.view=view;  
 }  
  
 public ComplexValue add\_two(ComplexValue value1, ComplexValue value2)  
 {  
 return model.add\_two(value1, value2);  
 }  
  
 public ComplexValue subtract\_two(ComplexValue value1, ComplexValue value2)  
 {  
 return model.subtract\_two(value1, value2);  
 }  
  
 public ComplexValue multiply\_two(ComplexValue value1, ComplexValue value2)  
 {  
 return model.multiply\_two(value1, value2);  
 }  
  
 public ComplexValue divide\_two(ComplexValue value1, ComplexValue value2)  
 {  
 return model.divide\_two(value1, value2);  
 }  
  
 public double getModule()  
 {  
 return model.getModule();  
 }  
  
 public void setModule(double module)  
 {  
 model.setModule(module);  
 }  
  
 public double getArgument()  
 {  
 return model.getArgument();  
 }  
  
 public void setArgument(double argument)  
 {  
 model.setArgument(argument);  
 }  
  
}

Patterns:

package patterns;  
import mvc.\*;  
  
public class Mediator{  
 public static void checkCValueRealNotEqualToECValueModule(CValueController val1, ECValueController val2){  
 System.out.println();  
 System.out.println("real:" + val1.getReal() + " " + "module: " + val2.getModule() );  
 }  
}

package mvc;  
  
public class ECVControllerFacade {  
  
 final private ExponentialComplexValue model;  
 final private ECValueView view;  
  
 public ECVControllerFacade(ExponentialComplexValue model, ECValueView view) {  
 this.model = model;  
 this.view = view;  
 }  
  
 public void active(ComplexValue val1, ComplexValue val2) {  
 ECValueController controller = new ECValueController(this.model, this.view);  
  
 System.*out*.println( controller.add\_two(val1, val2).getReal() + " " + controller.add\_two(val1, val2).getImaginary());  
  
 System.*out*.println(controller.subtract\_two(val1, val2).getReal() + " " + controller.subtract\_two(val1, val2).getImaginary());  
  
 System.*out*.println(controller.multiply\_two(val1, val2).getReal() + " " + controller.multiply\_two(val1, val2).getImaginary());  
  
 System.*out*.println(controller.divide\_two(val1, val2).getReal() + " " + controller.divide\_two(val1, val2).getImaginary());  
 }  
  
}

package mvc;  
  
public class ECVProxyController {  
  
 private ECValueController ecValueController;  
  
 public ECVProxyController(ExponentialComplexValue model, ECValueView view) {  
 this.ecValueController = new ECValueController(model, view);  
 }  
  
 public ComplexValue add\_two(ComplexValue value1, ComplexValue value2) {  
 return ecValueController.add\_two(value1, value2);  
 }  
  
 public ComplexValue subtract\_two(ComplexValue value1, ComplexValue value2) {  
 return ecValueController.subtract\_two(value1, value2);  
 }  
  
 public ComplexValue multiply\_two(ComplexValue value1, ComplexValue value2) {  
 return ecValueController.multiply\_two(value1, value2);  
 }  
  
 public ComplexValue divide\_two(ComplexValue value1, ComplexValue value2) {  
 return ecValueController.divide\_two(value1, value2);  
 }  
  
 public double getModule() {  
 return ecValueController.getModule();  
 }  
  
 public void setModule(double module) {  
 ecValueController.setModule(module);  
 }  
  
 public double getArgument() {  
 return ecValueController.getArgument();  
 }  
  
 public void setArgument(double argument) {  
 ecValueController.setArgument(argument);  
 }  
}

Main

package mvc;  
  
import java.lang.reflect.InvocationTargetException;  
  
public class Main {  
 public static void main(String[] args) throws InvocationTargetException, IllegalAccessException {  
 // Singlton  
 ComplexValue model1 = new ComplexValue(1,2);  
 ComplexValueView view1 = new ComplexValueView();  
 // create StackXController using Singlton  
 CValueController controller1 = CValueController.*getInstance*(model1, view1);  
  
 System.*out*.println(controller1.add(model1));  
  
  
 // Proxy  
 ExponentialComplexValue model2 = new ExponentialComplexValue(12, 21);  
 ECValueView view2 = new ECValueView();  
 //Create Controller Proxy  
 ECVProxyController controller2 = new ECVProxyController(model2, view2);  
 System.*out*.println("real: " + controller2.getArgument());  
 System.*out*.println("module: " + controller2.getModule());  
 System.*out*.println("Substract: " + controller2.subtract\_two(new ComplexValue(22,11), new ComplexValue(2,1)));  
  
 // Facade  
 ExponentialComplexValue model3 = new ExponentialComplexValue();  
 ECValueView view3 = new ECValueView();  
 // create facade  
 ECVControllerFacade controllerFacade = new ECVControllerFacade(model2, view2);  
 controllerFacade.active(new ComplexValue(1,2), new ComplexValue(3, 4)); // использование фасада  
  
  
  
 // Mediator  
 ExponentialComplexValue model4 = new ExponentialComplexValue();  
 ECValueView view4 = new ECValueView();  
 //Создание контролера  
 ECValueController controller4 = new ECValueController(model4, view4);  
 controller1.checkCValueRealNotEqualToECValueModule(controller4);  
  
 }  
}