*Name: Aldrick Gardiner*

*Course: Cop 4331 003*

*Homework: 3*

Question 3.1

Explain why encapsulation is an important principle for OODP.

Encapsulation is an important principle for OODP because it is the hiding of data implementation by restricting access to accessors and mutators. If your implementing a class on a large scale(OOPD), encapsulation helps in a major way by providing a mechanism for restricting the range of the program that is affected by a change to a small subset.

b. When is it OK to throw exceptions as part of the contract ?

Add to your answer an example (no code, just a description).

It is okay to throw an exception as part of the contract when you cannot use a precondition to determine the result. For example checking for a file, its best to use an exception because instead of a precondition because another program could have removed the file immediately after that check and before the constructor call. Thus, existence of the file is not a *verifiable* precondition. In such a situation, throwing an exception is entirely appropriate.

c. Explain why side effects should be avoided

Side effect should be avoided to help with the efficient of the code because code with out side effect can be run numerous times with out any unwanted causes. For example if you have method that adds element to an array but a side effects it deletes elements that is already in the array. That’s a unwanted cause that can causes major problems.

Question 3.2

Filename: Complex.java

public class Complex {

/\*\* The real part \*/

private double r;

/\*\* The imaginary part \*/

private double i;

/\*\* Construct a Complex \*/

Complex(double rr, double ii) {

r = rr;

i = ii;

}

/\*\* Display the current Complex as a String, for use in

\* println() and elsewhere.

\*/

public String toString() {

StringBuffer sb = new StringBuffer().append(r);

if (i>0)

sb.append('+'); // else append(i) appends - sign

return sb.append(i).append('i').toString();

}

/\*\* Return just the Real part \*/

public double getReal() {

return r;

}

/\*\* Return just the Real part \*/

public double getImaginary() {

return i;

}

/\*\* Return the magnitude of a complex number \*/

public double magnitude() {

return Math.sqrt(r\*r + i\*i);

}

/\*\* Add another Complex to this one

\*/

public Complex add(Complex other) {

return add(this, other);

}

/\*\* Add two Complexes

\*/

public static Complex add(Complex c1, Complex c2) {

return new Complex(c1.r+c2.r, c1.i+c2.i);

}

/\*\* Subtract another Complex from this one

\*/

public Complex subtract(Complex other) {

return subtract(this, other);

}

/\*\* Subtract two Complexes

\*/

public static Complex subtract(Complex c1, Complex c2) {

return new Complex(c1.r-c2.r, c1.i-c2.i);

}

/\*\* Multiply this Complex times another one

\*/

public Complex multiply(Complex other) {

return multiply(this, other);

}

/\*\* Multiply two Complexes

\*/

public static Complex multiply(Complex c1, Complex c2) {

return new Complex(c1.r\*c2.r - c1.i\*c2.i, c1.r\*c2.i + c1.i\*c2.r);

}

public static Complex conj(Complex c1, Complex c2){

return new Complex(c1.r, c2.i);

}

/\*\* Divide c1 by c2.

\*/

public static Complex divide(Complex c1, Complex c2) {

return new Complex(

(c1.r\*c2.r+c1.i\*c2.i)/(c2.r\*c2.r+c2.i\*c2.i),

(c1.i\*c2.r-c1.r\*c2.i)/(c2.r\*c2.r+c2.i\*c2.i));

}

/\* Compare this Complex number with another

\*/

public boolean equals(Object o) {

if (!(o instanceof Complex))

throw new IllegalArgumentException(

"Complex.equals argument must be a Complex");

Complex other = (Complex)o;

return r == other.r && i == other.i;

}

public static void main(String[] args) {

Complex c = new Complex(3, 5);

Complex d = new Complex(2, -2);

System.out.println(c);

System.out.println(c + ".getReal() = " + c.getReal());

System.out.println(c + " + " + d + " = " + c.add(d));

System.out.println(c + " + " + d + " = " + Complex.add(c, d));

System.out.println(c + " \* " + d + " = " + c.multiply(d));

System.out.println(Complex.divide(c, d));

System.out.println(Complex.subtract(c, d));

System.out.println(Complex.conj(c, d));

}

}

Filename: ComplexTest.java

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

/\*\*

\*

\* @author Ace

\*/

public class ComplexTest {

public ComplexTest() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() {

}

@After

public void tearDown() {

}

/\*\*

\* Test of toString method, of class Complex.

\*/

@Test

public void testToString() {

System.out.println("toString");

Complex instance = null;

String expResult = "";

String result = instance.toString();

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of getReal method, of class Complex.

\*/

@Test

public void testGetReal() {

System.out.println("getReal");

Complex instance = null;

double expResult = 0.0;

double result = instance.getReal();

assertEquals(expResult, result, 0.0);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of getImaginary method, of class Complex.

\*/

@Test

public void testGetImaginary() {

System.out.println("getImaginary");

Complex instance = null;

double expResult = 0.0;

double result = instance.getImaginary();

assertEquals(expResult, result, 0.0);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of magnitude method, of class Complex.

\*/

@Test

public void testMagnitude() {

System.out.println("magnitude");

Complex instance = null;

double expResult = 0.0;

double result = instance.magnitude();

assertEquals(expResult, result, 0.0);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of add method, of class Complex.

\*/

@Test

public void testAdd\_Complex() {

System.out.println("add");

Complex other = null;

Complex instance = null;

Complex expResult = null;

Complex result = instance.add(other);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of add method, of class Complex.

\*/

@Test

public void testAdd\_Complex\_Complex() {

System.out.println("add");

Complex c1 = null;

Complex c2 = null;

Complex expResult = null;

Complex result = Complex.add(c1, c2);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of subtract method, of class Complex.

\*/

@Test

public void testSubtract\_Complex() {

System.out.println("subtract");

Complex other = null;

Complex instance = null;

Complex expResult = null;

Complex result = instance.subtract(other);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of subtract method, of class Complex.

\*/

@Test

public void testSubtract\_Complex\_Complex() {

System.out.println("subtract");

Complex c1 = null;

Complex c2 = null;

Complex expResult = null;

Complex result = Complex.subtract(c1, c2);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of multiply method, of class Complex.

\*/

@Test

public void testMultiply\_Complex() {

System.out.println("multiply");

Complex other = null;

Complex instance = null;

Complex expResult = null;

Complex result = instance.multiply(other);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of multiply method, of class Complex.

\*/

@Test

public void testMultiply\_Complex\_Complex() {

System.out.println("multiply");

Complex c1 = null;

Complex c2 = null;

Complex expResult = null;

Complex result = Complex.multiply(c1, c2);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of conj method, of class Complex.

\*/

@Test

public void testConj() {

System.out.println("conj");

Complex c1 = null;

Complex c2 = null;

Complex expResult = null;

Complex result = Complex.conj(c1, c2);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of divide method, of class Complex.

\*/

@Test

public void testDivide() {

System.out.println("divide");

Complex c1 = null;

Complex c2 = null;

Complex expResult = null;

Complex result = Complex.divide(c1, c2);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of equals method, of class Complex.

\*/

@Test

public void testEquals() {

System.out.println("equals");

Object o = null;

Complex instance = null;

boolean expResult = false;

boolean result = instance.equals(o);

assertEquals(expResult, result);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

/\*\*

\* Test of main method, of class Complex.

\*/

@Test

public void testMain() {

System.out.println("main");

String[] args = null;

Complex.main(args);

// TODO review the generated test code and remove the default call to fail.

fail("The test case is a prototype.");

}

}

Question 4.1

Filename: Student.java

import java.util.ArrayList;

import java.util.Comparator;

import java.util.Date;

import java.util.Scanner;

/\*\*

\*

\* @author Ace

\*/

public class Student {

private String name; // first name

private Date enroll;

// construct a new student with given fields

public Student(String name, Date enroll) {

this.name = name;

this.enroll = enroll;

}

public String getName() {

return name;

}

public Date getEnroll() {

return (Date) enroll.clone();

}

public static Comparator<Student> getCompByName()

{

Comparator comp = new Comparator<Student>(){

@Override

public int compare(Student s1, Student s2)

{

s1.name.compareTo(s2.name);

return 0;

}

};

return comp;

}

public static Comparator<Student> getCompByDate()

{

Comparator comp = new Comparator<Student>(){

@Override

public int compare(Student s1, Student s2)

{

s1.enroll.compareTo(s2.enroll);

return 0;

}

};

return comp;

}

// return a string representation of the invoking object

public String toString() {

return name + " " + enroll;

}

// sample client

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int nstudents;

String stName;

int enrollDate;

System.out.print("How many students would you like to enter?");

nstudents = sc.nextInt();

sc.nextLine();

ArrayList studentArray = new ArrayList(nstudents);

for (int i = 0; i < nstudents; i++) {

System.out.print("Enter name: ");

stName = sc.nextLine();

System.out.print("Enter ID: ");

//enroll = sc.;

//studentArray.add(new Student(enrollDate, stName));

}

}

}

Question 4.2

Filename: FillCircle.java

import java.lang.\*;

import java.util.\*;

import java.util.List;

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.geom.\*;

import javax.swing.JButton;

import javax.swing.JPanel;

public class FillCircle extends Frame {

public void paint(Graphics g) {

Graphics2D ga = (Graphics2D)g;

ga.setPaint(Color.red);

ga.fillOval(150,150,100,100);

}

public static void main(String args[])

{

FillCircle frame = new FillCircle();

frame.addWindowListener(

new WindowAdapter()

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

}

);

frame.setSize(400, 400);

frame.setVisible(true);

}

}