3.1

a.) Encapsulation is important since it allows reusability of code. There could be multiple implementation of the same class without affecting the original class. This is done by hiding the implementation of a class by hiding it behind an interface. This allows to be reused in multiple part of the code or other projects as needed.

b.) Throwing exception is accepted when a precondition cannot always be validated since other methods or externals can modify this condition during runtime. If file reader method is called during it the construction of objects but the is deletes by another program then the precondition would be passed during construction, but the exception would be thrown when the program is trying to access it again.

c.) Side effects should be avoided to keep data pristine and clean when this data is not expected to be modified. Side effects are only accepted when said data is expected to be change like the example from the book “a.add(b)” which expects one parameter (“a”) to be modified but not the other (“b”).

3.2

Complex.java

```
package q3_2.math;

public interface Complex {

    /**
     * Print the complex number in string from as the formula.
     * @precondition imaginary != 0 && real != 0.
     * @return Complex number in string form "a + bi".
     * @exception throws exception if
     **/
    String toString();

    /**
     * Accessor for the real number variable.
     * @precondition  real == 0.
     * @return Real number value.
     */
    double r();

    /**
     * Accessor function for the imaginary variable
     * @precondition imaginary != 0 && real != 0.
     * @return Imaginary number value.
     */
    double i();

    /**
     * Add function between two Complex number following formula:
     * (a + bi) + (c + di) = a + b + bi + ci
     * @precondition Complex x != null.
     * @param x Complex number object to be added.
     * @return Complex number object with the result of the operation.
     * @postcondition resultReal != 0 && resultImaginary != 0.
     * @throws NullPointerException if param x == null.
     */
    Complex add (Complex x);

    /**
     * Substraction function between two Complex number following formula:
     * (a + bi) - (c + di) = a - b - bi - ci
     * @precondition  Complex x != null.
     * @param x Complex number object to be Substracted.
     * @return Complex number object with the result of the operation.
     * @postcondition resultReal != 0 && resultImaginary != 0.
     * @throws NullPointerException if param x == null.
     */
    Complex sub (Complex x);

    /**
     * Multiplication function between two Complex number following formula:
     * (a + bi) * (c + di) = (a * c) + (a * di) + (c * bi) + (bi * di)
     * @precondition  Complex x != null.
     * @param x Complex number object to be multiplied.
     * @return Complex number object with the result of the operation.
     * @postcondition resultReal != 0 && resultImaginary != 0.
     * @throws NullPointerException if param x == null.
     */
    Complex mult(Complex x);

    /**
     * Copares the value of 2 complex number objects.
     * @precondition  Complex x != null.
     * @param x Complex number object to be compared.
     * @return Bolean with the result of equality comparison.
     * @throws NullPointerException if param x == null.
     */
    boolean equals (Complex x);

    /**
     * Invariant check if the object still qualifies as a Complex number.
     * or the result of any operation also qualifies as a Complex number.
     * Invarianrt: real != 0 && imiginary != 0
     * @return true if both values are equal to 0.
     */
    boolean isComplexNumber(double real, double imaginary);

}
```

ComplexImpl.java

```
package q3_2.math;


/**
 * Invariable: Both values for both real and imaginary most not equal 0 at any
momment.
 */
public class ComplexImpl implements Complex{

    private final double real;
    private final double imaginary;

    /**
     * Creates new complex number object.
     * @param real Real number in the object.
     * @param imaginary Imaginary number.
     */
    public ComplexImpl (double real, double imaginary){
        this.real = real;
        this.imaginary = imaginary;

        assert this.isComplexNumber(real, imaginary);
    }

    /**
     * Creates Complex number without an imaginary number in the object.
     * Defaults imginaty value to 0.
     * @param real Real number value in the object.
     */
    public ComplexImpl (double real){
        this.imaginary = 0;
        this.real = real;
```

```java
        assert isComplexNumber(real, imaginary);
    }

    /**
     * {@inheritDoc}
     */
    public String toString(){
        if (imaginary >= 0) {
            return real + "+" + imaginary + "i";
        } else {
            return  real + "" + imaginary + "i";
        }
    }

    /**
     * {@inheritDoc}
     */
    public double r() { return  real; }

    /**
     * {@inheritDoc}
     */
    public double i(){ return  imaginary; }

    /**
     * {@inheritDoc}
     */
    public ComplexImpl add (Complex x){
        if (x == null){
            throw new NullPointerException("The parameter provided is null");
        }
        double resultReal;
        double resultImaginary;

        resultReal = real + x.r();
        resultImaginary = imaginary + x.i();

        return new ComplexImpl(resultReal, resultImaginary);
    }

    /**
     * {@inheritDoc}
     */
    public ComplexImpl sub (Complex x){
        if (x == null){
            throw new NullPointerException("The parameter provided is null");
        }

        double resultReal;
        double resultImaginary;

        resultReal = real - x.r();
        resultImaginary = imaginary - x.i();

        return new ComplexImpl(resultReal, resultImaginary);
    }

    /**
     * {@inheritDoc}
     */
    public ComplexImpl mult(Complex x){
        if (x == null){
            throw new NullPointerException("The parameter provided is null");
        }

        double resultReal;
        double resultImaginary;

        resultImaginary = (real * x.i()) + (imaginary * x.r());
        resultReal = imaginary * x.i() * -1;
        resultReal = resultReal + (real * x.r());

        assert this.isComplexNumber(resultReal, resultImaginary);
        return  new ComplexImpl( resultReal, resultImaginary);
    }

    /**
     * {@inheritDoc}
     */
    public boolean equals (Complex x){
        if (x == null){
            throw new NullPointerException("The parameter provided is null");
        }
        return (x.r() == real) && (x.i() == imaginary);
    }


    /**
     * {@inheritDoc}
     */
    public boolean isComplexNumber(double real, double imaginary){
        return real != 0 || imaginary != 0;
    }

}
```

TestComplex.java

```java
import org.junit.Assert;
import org.junit.Test;
import q3_2.math.Complex;
import q3_2.math.ComplexImpl;

public class TestComplex {


    @Test
    public void testAdd(){
        System.out.println("Running test add...");
        double a = 1, b = 2, c = -3, d = 4;
        double e = a + c, f = b + d;
        ComplexImpl x = new ComplexImpl(a,b);
        ComplexImpl y = new ComplexImpl(c,d);
        ComplexImpl w = x.add(y);
        ComplexImpl z = new ComplexImpl(e,f);
        // set up Complex objects
        // test condition using the Complex equals() method:
```

```java
        Assert.assertTrue(z.equals(w));
    }

    @Test
    public void testAddWithNoImaginaryNumber(){
        System.out.println("Running test add without an imaginary number...");
        double a = 1, b = 2;
        double e = a + b;
        ComplexImpl x = new ComplexImpl(a);
        ComplexImpl y = new ComplexImpl(b);
        ComplexImpl w = x.add(y);
        ComplexImpl z = new ComplexImpl(e);

        Assert.assertTrue(z.equals(w));
    }

    @Test
    public void testSub(){
        System.out.println("Running test sub...");
        double a = 1, b = 2, c = -3, d = 4;
        double e = a + c, f = b + d;
        ComplexImpl x = new ComplexImpl(a,b);
        ComplexImpl y = new ComplexImpl(c,d);
        ComplexImpl w = x.add(y);
        ComplexImpl z = new ComplexImpl(e,f);
        // set up Complex objects
        // test condition using the Complex equals() method:
        Assert.assertTrue(z.equals(w));
    }

    @Test
    public void testSubWithNoImaginaryNumber(){
        System.out.println("Running test sub without imaginary number...");
        double a = 1, b = 2;
        double e = a - b;
        ComplexImpl x = new ComplexImpl(a);
        ComplexImpl y = new ComplexImpl(b);
        ComplexImpl w = x.sub(y);
        ComplexImpl z = new ComplexImpl(e);

        Assert.assertTrue(z.equals(w));
    }

    @Test
    public void testMult() {
        System.out.println("Running test mult...");
        double a = 1, b = 2, c = -3, d = 4;
        double e = (a * c) + (b * d * -1);
        double f = (a * d) + (b * c);

        ComplexImpl x = new ComplexImpl(a, b);
        ComplexImpl y = new ComplexImpl(c,d);
        ComplexImpl w = x.mult(y);
        ComplexImpl z = new ComplexImpl(e,f);

        Assert.assertTrue(z.equals(w));
    }

    @Test
    public void testMultNoImaginaryNumber(){
        System.out.println("Running test mult without imaginary number...");
        double a = 1, c = -3;
        double e = (a * c) ;

        ComplexImpl x = new ComplexImpl(a);
        ComplexImpl y = new ComplexImpl(c);
        ComplexImpl w = x.mult(y);
        ComplexImpl z = new ComplexImpl(e);

        Assert.assertTrue(z.equals(w));
    }

    @Test
    public void testToString(){
        System.out.println("Running test toString...");
        double a = 1, b = 2;
        String expectedStr = a + "+" + b + "i";

        ComplexImpl x = new ComplexImpl(a,b);
        String testStr = x.toString();

        Assert.assertEquals(expectedStr, testStr);
    }

    @Test
    public void testToStringNoImaginaryNumber(){
        System.out.println("Running test toString without imaginary number...");
        double a = 1;
        String expectedStr = a + "+" + 0.0 + "i";

        ComplexImpl x = new ComplexImpl(a);
        String testStr = x.toString();

        Assert.assertEquals(expectedStr, testStr);
    }

    @Test
    public void testToStringWithNegativeImaginaryNumber(){
        System.out.println("Running test toString with negative imaginary number...");
        double a = -1, b = -1;
        String expectedStr = a + "" + b + "i";

        ComplexImpl x = new ComplexImpl(a,b);
        String testStr = x.toString();

        Assert.assertEquals(expectedStr, testStr);
    }

    @Test
    public void  testIsComplexNumber(){
        System.out.println("Running test isComplexNumber...");
        ComplexImpl complex = new ComplexImpl(1,2);
        Assert.assertTrue(complex.isComplexNumber(complex.r(), complex.i()));
    }

    @Test(expected = NullPointerException.class)
    public void testAddNullPointerException(){
        System.out.println("Running test testAddNullPointerException...");
        double a = 1, b = 2;
        ComplexImpl x = new ComplexImpl(a,b);
        ComplexImpl y = null;
```

```java
        x.add(y);
    }

    @Test(expected = NullPointerException.class)
    public void testSubNullPointerException(){
        System.out.println("Running test testSubNullPointerException...");
        double a = 1, b = 2;
        ComplexImpl x = new ComplexImpl(a,b);
        ComplexImpl y = null;

        x.sub(y);
    }

    @Test(expected = NullPointerException.class)
    public void testMultNullPointerException(){
        System.out.println("Running test testMultNullPointerException...");
        double a = 1, b = 2;
        ComplexImpl x = new ComplexImpl(a,b);
        ComplexImpl y = null;

        x.mult(y);
    }

    @Test(expected = NullPointerException.class)
    public void testEqualsNullPointerException(){
        System.out.println("Running test testEqualsNullPointerException...");
        double a = 1, b = 2;
        ComplexImpl x = new ComplexImpl(a,b);
        ComplexImpl y = null;

        x.equals(y);
    }



}
```

4.1

Student.java

```
package q4_1;

import java.util.*;

/**
 * Invariable: values must not equal null.
 */
public class Student {

    private final String name;
    private final Date enrollmentDate;

    /**
     * Contructor for Student object
     * @precondition name and  date != null
     * @param name Student name in format last, first
     * @param whenEnrolled date of enrollment.
     * @throws NullPointerException if any of the parameters != 0.
     */
    public Student(String name, Date whenEnrolled) {
        if (name == null || whenEnrolled == null){
            throw new NullPointerException("Values provided are null");
        }
        this.name = name;
        this.enrollmentDate = whenEnrolled;
    }

    /**
     * Accessor for name value.
     * @precondition name !=  null.
     * @return value of name.
     */
    public String getName (){ return name; }

    /**
     * Accessor for date value
     * @precondition Date != 0
     * @return Value of date.
     */
    public Date getEnrollmentDate() { return (Date) enrollmentDate.clone(); }

    /**
     * Coparator for value name between two Strudent objects.
      * @return 0 if both values are equal. Greater than 0 if student1 name value is
lexicographically
     * less than the student2 name vlaue. Less than 0 if the student1 name value is
lexicographically greater
     * than the value of the student2 name.
     * @throws NullPointerException if the value of name is null for any of the two
student objects
     */
    public static Comparator<Student> getCompByName() {
        return new Comparator<Student>(){
            public int compare(Student student1, Student student2){
                if (student1.getName() == null  || student2.getName() == null){
                    throw new NullPointerException("The value name if onw of the 
objects is set to null");
                }
                return student1.getName().compareTo(student2.getName());
            }
        };
    }

    /**
     * Coparator for value Date between two Strudent objects.
     * @return 0 if both values are equal. Greater than 0 if student1 Date value 
before
     * student2 Dtae. Less than 0 if the student1 Date after student2 Date.
     * @throws NullPointerException if the value of Date is null for any of the two 
student objects
     */
    public static Comparator<Student> getCompByDate() {
        return  new Comparator<Student>() {
            public int compare(Student student1, Student student2) {
                if (student1.getEnrollmentDate() == null  || 
student2.getEnrollmentDate() == null){
                    throw new NullPointerException("The value whenEnrroll if onw of 
the objects is set to null");
                }
                return 
student1.getEnrollmentDate().compareTo(student2.getEnrollmentDate());

            }
        };
    }


    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
        students.add(new Student("Doe, John", new Date(118,10,10)));
        students.add(new Student("Doe, Jane", new Date(118,10,7)));
        students.add(new Student("Smith, Sonia", new Date(115, 8, 20)));
        students.add(new Student("Messi, Leo", new Date(119, 8, 11)));
        students.add(new Student("Iniesta, Andres", new Date(120, 1, 5)));

        System.out.println("Sorting by Name:");
        Collections.sort(students, Student.getCompByDate());
        students.forEach((student) -> {
            System.out.println(student.getName() + ", " + 
student.getEnrollmentDate().toString());
        });

        System.out.println("Sorting by Date:");

        Collections.sort(students, Student.getCompByDate());
        students.forEach((student) -> {
            System.out.println(student.getName() + ", " + 
student.getEnrollmentDate().toString());
        });

    }
}
```

4.2

ColorFrame.java

```java
package q4_2;


import q4_1.Student;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ColorFrame  {
    private final int size = 5;
    Color color;
    JLabel jLabel;
    JFrame jFrame;
    ColorIcon icon;

    public ColorFrame(){
        jLabel = new JLabel();
        icon = new ColorIcon();
        jFrame = new JFrame();
        jFrame.setLayout(new FlowLayout());
        jLabel = new JLabel(icon);

    }

    public JButton createButton(String name){
        JButton button = new JButton(name);
        button.addActionListener(createColorChangeActionListener(name, this));
        return button;
    }

    public void setColor(String colorName){
        switch (colorName.toLowerCase()){
            case "red":
                color = Color.RED;
                break;
            case "blue":
                color = Color.BLUE;
                break;
            case "green":
                color = Color.GREEN;
                break;
            default:
                color = Color.white;
        }
        icon.setIconColor(color);
    }

    public static ActionListener createColorChangeActionListener(final String name,
ColorFrame frame)  {
        return new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println(name);
                frame.setColor(name);
                frame.jLabel.repaint();
            }
        };
    };


    public static void main(String[] args) throws Exception {
        ColorFrame colorFrame = new ColorFrame();

        String[] colors = new String[] {"RED","GREEN","BLUE"};
        JButton[] buttons = new JButton[3];

        for (int x = 0 ; x < colors.length; x++){
            buttons[x] = colorFrame.createButton(colors[x]);
            colorFrame.color = (Color) Color.class.getField(colors[x]).get(null);

            colorFrame.jFrame.add(buttons[x]);
        }
        colorFrame.jFrame.add(colorFrame.jLabel);
        colorFrame.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        colorFrame.jFrame.pack();
        colorFrame.jFrame.setVisible(true);

    }
}
```

ColorIcon.java

```
package q4_2;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.Ellipse2D;

public class ColorIcon implements Icon {
    private  int size = 50;
    Graphics2D g2;
    Color color = Color.WHITE;

    public void setIconColor(Color color){
        this.color = color;
        g2.setColor(this.color);
    }

    @Override
    public void paintIcon(Component c, Graphics g, int x, int y) {
        g2 = (Graphics2D) g;
        Ellipse2D.Double icon = new Ellipse2D.Double(x, y, size, size);
        setIconColor(color);
        g2.fill(icon);

    }

    @Override
    public int getIconWidth() {
        return size;
    }

    @Override
    public int getIconHeight() {
        return size;
    }
}
```