CSE 215: Programming Language II Lab

Lab – 6

Lab Officer: Tanzina Tazreen

Encapsulation

Objective:

- To learn about class and objects
- To learn more about Modifiers
- To learn to implement a class using UML
- To learn about Encapsulation

Modifiers

We divide modifiers into two groups:

- **Access Modifiers** - controls the access level
- **Non-Access Modifiers** - do not control access level, but provides other functionality

For **classes**, you can use either public or *default*:

Modifier	Description
`public`	The class is accessible by any other class
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter

For **attributes, methods and constructors**, you can use the one of the following:

Modifier	UML sign	Description
`public`	+	The code is accessible for all classes
`private`	-	The code is only accessible within the declared class
default	none	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter
`protected`	#	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter

Non-Access Modifiers

For **classes**, you can use either final or abstract:

Modifier	Description
`final`	The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter)
`abstract`	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters)

For **attributes and methods**, you can use the one of the following:

Modifier	Description
`final`	Attributes and methods cannot be overridden/modified
`static`	Attributes and methods belongs to the class, rather than an object
`abstract`	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters

Final

```
public class Main {
  final int x = 10;
  final double PI = 3.14;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 50; // will generate an error
    myObj.PI = 25; // will generate an error
    System.out.println(myObj.x);
  }
}
```

Public vs Static

You will often see either static or public attributes and methods.

we created a static method, which can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without
creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by
creating objects");
  }

  // Main method
  public static void main(String[ ] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would output an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method
  }
}
```

Encapsulation

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as private
- provide public **get** and **set** methods to access and update the value of a private variable

```
public class Person {
  private String name; // private = restricted access

  // Getter
  public String getName() {
    return name;
  }
```

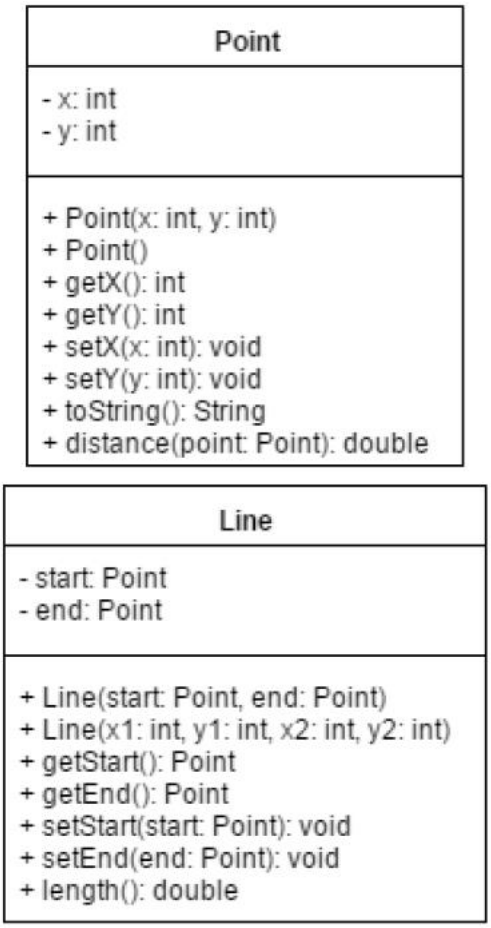
```
  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

To access the private variable in another class:

```
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.setName("John"); // Set the value of the name variable
to "John"
    System.out.println(myObj.getName());
  }
}
```

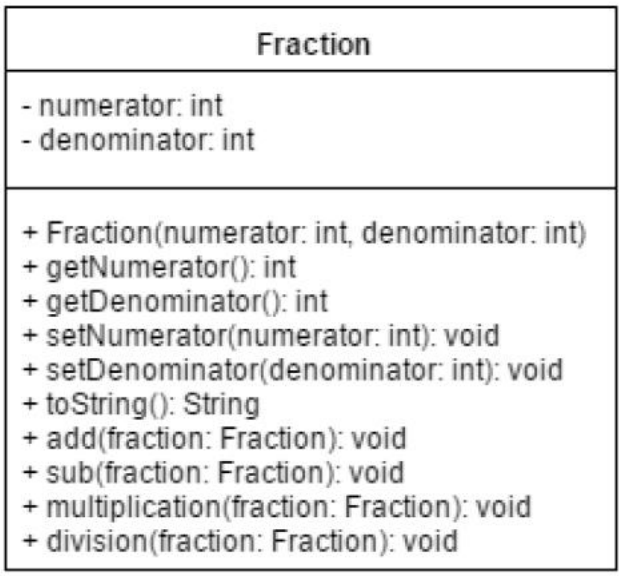
Tasks:

1. Implement the following classes, each on a separate file:



Now create a Line object and invoke the length() method.

2. Implement the following class and test its methods:



`void add(Fraction fraction)`

Adds two Fraction objects and **stores the result** into **calling object**. This is how addition is performed for fractions (assume that this is how it works):

$$\frac{1}{4}+\frac{3}{5}=\frac{1*5+3*4}{4*5}=\frac{17}{20}$$

`String toString()`

The toString() method returns a String representation of the object. In general, you return the valuesof the data fields of the objects in a formatted, readable manner from this method.
This method …

Returns the value of the fraction in $\frac{n}{m}$ format where n is the numerator and m is denominator.

e.g. Your console output should look like: `17 / 20`

Now write a test program, take two Fraction objects. Print both of them. Test add, sub, multiplication and division methods. Print calling object after each method call.