



## CSE 215: Programming Language II Lab

Sec – 8, Faculty - MUO

Lab Officer: Tanzina Tazreen

### Lab – 7

## 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
<b>public</b>	The class is accessible by any other class
<i>default</i>	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 <a href="#">Packages chapter</a>

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

Modifier	UML sign	Description
<code>public</code>	+	The code is accessible for all classes
<code>private</code>	-	The code is only accessible within the declared class
<code>default</code>	<b>none</b>	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 <a href="#">Packages chapter</a>
<code>protected</code>	#	The code is accessible in the same package and <b>subclasses</b> . You will learn more about subclasses and superclasses in the <a href="#">Inheritance chapter</a>

## Non-Access Modifiers

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

Modifier	Description
<code>final</code>	The class cannot be inherited by other classes (You will learn more about inheritance in the <a href="#">Inheritance chapter</a> )
<code>abstract</code>	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 <a href="#">Inheritance</a> and <a href="#">Abstraction</a> chapters)

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

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

Point
- x: int - y: int
+ Point(x: int, y: int) + Point() + getX(): int + getY(): int + setX(x: int): void + setY(y: int): void + toString(): String + distance(point: Point): double

Line
- start: Point - end: Point
+ Line(start: Point, end: Point) + Line(x1: int, y1: int, x2: int, y2: int) + getStart(): Point + getEnd(): Point + setStart(start: Point): void + setEnd(end: Point): void + length(): double

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

2. Implement the following class and test its methods:

Fraction
- numerator: int - denominator: int
+ Fraction(numerator: int, denominator: int) + getNumerator(): int + getDenominator(): int + setNumerator(numerator: int): void + setDenominator(denominator: int): void + toString(): String + add(fraction: Fraction): void + sub(fraction: Fraction): void + multiplication(fraction: Fraction): void + division(fraction: Fraction): void

## **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 values of 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.