Topic: Nested Classes.
OOP concepts involved: Classes, Constructor Methods, Interfaces, Object Instances, Nested Classes, Polymorphism, Abstract Methods, Inheritance.
Programming generic concepts involved: Variables, Access Modifiers, Functions, Data Types.

➢ Theoric introduction

NESTED CLASSES IN JAVA

In Java, it is possible to *define a class within another class*, such classes are known as nested classes. They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and create more readable and maintainable code.

The class written within is called the nested class, and the class that holds the inner class is called the outer class.

SYNTAX OF A NESTED CLASS

```
class Outer_Class{
   // other Outer_Class members
   class Inner_Class{
      // Inner_Class members
   }
}
```

Nested classes are divided into two categories: *static* and *non-static*. Nested classes that are declared static are called static nested classes. Non-static nested classes are called inner classes.

1. INNER CLASSES

Here are a few quick points to remember about non-static nested classes:

- They are also called inner classes

- They can have all types of access modifiers in their declaration (public, private, protected or *package-private*)
- Just like instance variables and methods, inner classes are associated with an instance of the enclosing class
- They have access to all members of the enclosing class, regardless of whether they are static or non-static
- They can only define non-static members

Here's how we can declare an inner class:

```
public class Outer {
       public class Inner {
              // class members
       }
}
```

To instantiate an inner class, we must first instantiate its enclosing class:

```
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
```

1.1 METHOD-LOCAL INNER CLASS

In Java, we can write a class within a method and this will be a local type. Like local variables, the scope of the inner class is restricted within the method. A method-local inner class can be instantiated only within the method where the inner class is defined.

Here are a few points to remember about this type of class:

- They cannot have access modifiers in their declaration
- They have access to both static and non-static members in the enclosing context
- They can only define instance members

The following program shows how to use a method-local inner class.

```
public class NewEnclosing {

    void run() {
        class Local {
```

```
            void run() {
                // method implementation
            }
        }
        Local local = new Local();
        local.run();
    }

    @Test
    public void test() {
        NewEnclosing newEnclosing = new NewEnclosing();
        newEnclosing.run();
    }
}
```

1.2 ANONYMOUS INNER CLASS

Anonymous classes can be used to define an implementation of an interface or an abstract class without having to create a reusable implementation.

Here's a list of points to remember about anonymous classes:

- They cannot have access modifiers in their declaration
- They have access to both static and non-static members in the enclosing context
- They can only define instance members
- They're the only type of nested classes that cannot define constructors or extend/implement other classes or interfaces

To define an anonymous class, let's first define a simple abstract class:

```
abstract class SimpleAbstractClass {
    abstract void run();
}
```

Now, we will define an anonymous class:

```
public class AnonymousInnerTest {

    @Test
    public void whenRunAnonymousClass_thenCorrect() {
```

```
        SimpleAbstractClass simpleAbstractClass = new SimpleAbstractClass() {
            void run() {
                // method implementation
            }
        };
        simpleAbstractClass.run();
    }
}
```

2. STATIC NESTED CLASSES

Here are a few quick points to remember about static nested classes:

- As with static members, these belong to their enclosing class, and not to an instance of the class
- They can have all types of access modifiers in their declaration
- They only have access to static members in the enclosing class
- They can define both static and non-static members

This is an example of how to declare a static nested class:

```
public class Enclosing {

    private static int x = 1;

    public static class StaticNested {
        private void run() {
            // method implementation
        }
    }

    @Test
    public void test() {
        Enclosing.StaticNested nested = new Enclosing.StaticNested();
        nested.run();
    }
}
```

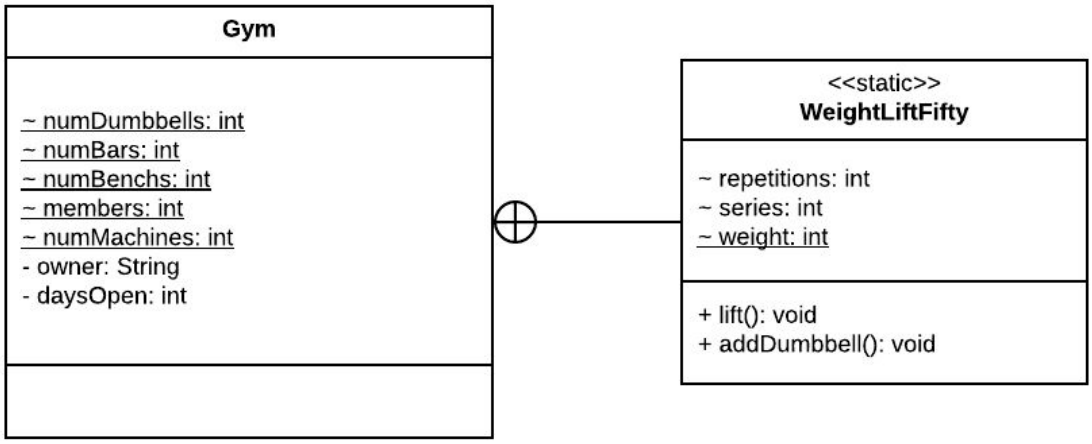
➢ Statement

Create a class called *Gym* that contains static attributes such as **numDumbbells**, **numBars**, etc. This Gym class must have a static nested class called *WeightLiftFifty*, which must have an int attribute called **repetitions**, another int attribute called **series** and a last static int attribute called **weight**, with a value of 50.

Within the WeightLiftFifty class, add a method called *lift*, where the information of the attributes of the outer class and the attributes of the same class are displayed on the screen.

Also, make a method called *addDumbbell* which modifies the value of the static attribute **numDumbbells** belonging to the outer class.

➢ Class design (UML)



➢ **Program Code**

Gym.java

```
public class Gym {
	// Gym static and non-static attributes
	static int numDumbbells = 50;
	static int numBars = 6;
	static int numBenchs = 6;
	static int members = 50;
	static int numMachines = 20;
	private String owner;
	private int daysOpen;

	// Gym constructor
	public Gym(String owner, int daysOpen) {
		this.owner = owner;
		this.daysOpen = daysOpen;
	}

	// Static Nested Class
	public static class WeightLiftFifty {
		// static nested class attributes
		int repetitions;
		int series;
		static int weight = 50; // 50 lbs

		public WeightLiftFifty(int repetitions, int series) {
			this.repetitions = repetitions;
			this.series = series;
		}

		public void lift() {
			// Accessing the outer class (Gym) attributes
			System.out.println("You can make " + series + " series of " +
repetitions + " reps with a weight of " + weight + " lbs whether be\nusing " +
numDumbbells + " dumbbells, or using " + numBars + " \nbars with " + numBenchs + "
benches or using the " + numMachines + " machines available!");
		}

		public void addDumbbell() {
			// modifying the outer class static attribute
			Gym.numDumbbells++;
		}
	}
```

```
	// Getters and Setters for the non-static attributes
	public String getOwner() {
		return owner;
	}

	public void setOwner(String owner) {
		this.owner = owner;
	}

	public int getDaysOpen() {
		return daysOpen;
	}

	public void setDaysOpen(int daysOpen) {
		this.daysOpen = daysOpen;
	}

	public static int getMembers() {
		return members;
	}

	public static void setMembers(int members) {
		Gym.members = members;
	}
}
```

Main.java

```
public class Main {
	public static void main(String[] args) {
		// Creating an instance of the static nested class WeightLiftFifty
		Gym.WeightLiftFifty weightLift = new Gym.WeightLiftFifty(15, 4);
		// calling method to display static values
		weightLift.lift();

		// calling method for modifying the outer class (Gym)
		// static member variable
		weightLift.addDumbbell();

		System.out.println("- Now the Gym has "+ Gym.numDumbbells +"
dumbbells.");
	}
}
```

➢ Program execution

In this program example, we create an object of type *WeightLiftFifty*, which is a nested static class. When creating this object, we passed by parameter two values, which are non-static values belonging to the *WeightLiftFifty* class.

Then we called the only method of that class: **lift()**, in which we accessed the static attributes of the outer class, as well as the own *WeightLiftFifty* static and non-static attributes.

In the end, we made the call of the method **addDumbbell()**, which modified the value of the static attribute of the outer class *Gym*. After this, we checked the new value of the *numDumbbells* attribute and we could notice that it was changed.

```
You can make 4 series of 15 reps with a weight of 50 lbs whether be
using 50 dumbbells, or using 6 bars with 6 benches or using the 20
machines available!
- Now the Gym has 51 dumbbells.
```

➢ Conclusions

As we covered in the previous section, a nested class is a class which is within an outer class. There are two types of nested classes: static nested classes and non-static nested classes (also called inner classes).

In general, nested classes are used for grouping classes that are only used in one place, and for the ability to have a better encapsulation, as well as having a more readable code.

The only nested class that doesn't permit to have a constructor are the anonymous inner classes, which normally are used to implement an interface without having an actual implementation of a class that uses that interface.

A static nested class doesn't have permission to access the outer class non-static members. It only has access to static members in the enclosing class.