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:

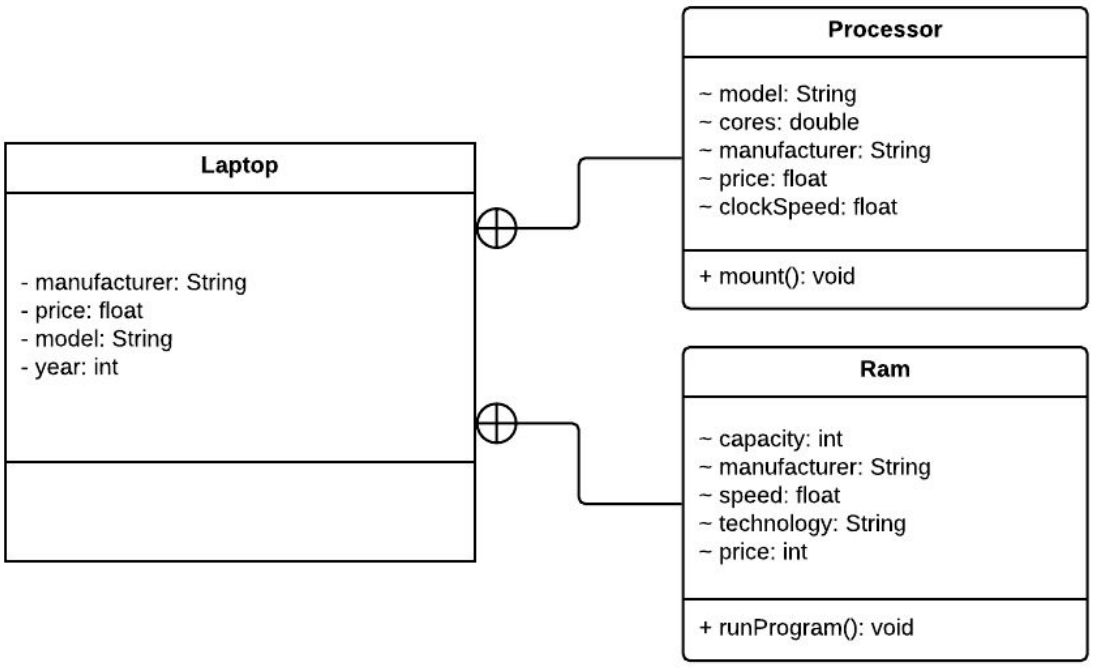
```
public class Enclosing {

    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 named *Laptop* that contains the following attributes: *manufacturer* (String), *price* (float), *model* (String) and *year* (int). After this place a class named *Processor* inside this Outer class (Laptop). *Processor* class contains the following attributes: *model* (String), *cores* (double), *manufacturer* (String), *price* (float) and *clockSpeed* (float). It also needs to have a method named *mount***.**

After, add another class named *Ram* within the *Laptop* class. This *Ram* inner class needs to have the following attributes: capacity (int), manufacturer (String), speed (float), technology (String) and price (int). It also needs to have a method named *runProgram*.

➢ Class design (UML)



➢ Program Code

Laptop.java

```
public class Laptop {
	// Laptop attributes
	private String manufacturer;
	private float price;
	private String model;
	private int year;

	// Laptop constructor methods
	public Laptop(String manufacturer, float price, String model, int year) {
		this.manufacturer = manufacturer;
		this.price = price;
		this.model = model;
		this.year = year;
	}

	// Member Inner class
	class Processor {
		// Processor own attributes
		String model;
		double cores;
		String manufacturer;
		float price;
		float clockSpeed;

		public Processor(double cores, String manufacturer, float price, float
clockSpeed, String model) {
			this.cores = cores;
			this.manufacturer = manufacturer;
			this.price = price;
			this.clockSpeed = clockSpeed;
			this.model = model;
		}

		// Processor method. Accessing the outer class attributes.
		public void mount() {
			System.out.println("- The " + manufacturer + " " + model + "
with a clock speed of " + clockSpeed + " GHz and " + cores + " cores\n with a price
of $" + price + "has been mounted on a " + Laptop.this.year + " " +
Laptop.this.manufacturer + " " + Laptop.this.model);
		}

	}
```

```
	// Member Inner class
	class Ram {
		// Ram own Atributes
		int capacity;
		String manufacturer;
		float speed;
		String technology;
		int price;

		public Ram(int capacity, String manufacturer, float speed, String
technology, int price) {
			this.capacity = capacity;
			this.manufacturer = manufacturer;
			this.speed = speed;
			this.technology = technology;
			this.price = price;
		}

		// Ram method. Accessing the outer class attributes.
		public void runProgram() {
			System.out.println("- Running programs on a " +
Laptop.this.manufacturer + " " + Laptop.this.model + " laptop with" + capacity + "
GBs of " + technology + " RAM\n at " + speed+ " MHz");
		}
	}

	// Getters and Setters
	public String getManufacturer() {
		return manufacturer;
	}

	public void setManufacturer(String manufacturer) {
		this.manufacturer = manufacturer;
	}

	public float getPrice() {
		return price;
	}

	public void setPrice(float price) {
		this.price = price;
	}

	public String getModel() {
		return model;
	}
```

```
	public void setModel(String model) {
		this.model = model;
	}

	public int getYear() {
		return year;
	}

	public void setYear(int year) {
		this.year = year;
	}

}
```

Main.java

```
public class Main {
	public static void main(String[] args) {
		// Creating an instance of a Laptop class
		Laptop laptop = new Laptop("Dell", 800, "Inspiron 7559", 2016);

		// Creating an instance of Processor (inner class), dependent
		// on the instance of the outer/enclosing class "Laptop"
		Laptop.Processor proccesor = laptop.new Processor(4, "Intel", 200,
2.8f, "Core I7 6700HQ");

		// Creating an instance of Ram (inner class), dependent on
		// the instance of the outer/enclosing class "Laptop"
		Laptop.Ram ram = laptop.new Ram(8, "Kingston", 1600, "DDR4", 100);

		// Running inner classes methods
		proccesor.mount();

		ram.runProgram();
	}
}
```

➢ Program execution

In this program example, we need to create an object of type *Laptop* for us to be able to make the other two instances of the inner classes of *Laptop* (*Processor* and *Ram*). This is because *Ram* and *Processor* classes are non-static nested classes, so we need an instance of the outer class so we can instance these inner classes.

These two inner classes have methods where we access the outer class attributes at the same time we access the inner class attributes.

```
- The Intel Core I7 6700HQ with a clock speed of 2.8 GHz and 4.0 cores
 with a price of $200.0has been mounted on a 2016 Dell Inspiron 7559
- Running programs on a Dell Inspiron 7559 laptop with8 GBs of DDR4 RAM
 at 1600.0 MHz
```

➢ 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.