**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();
    }
}
```

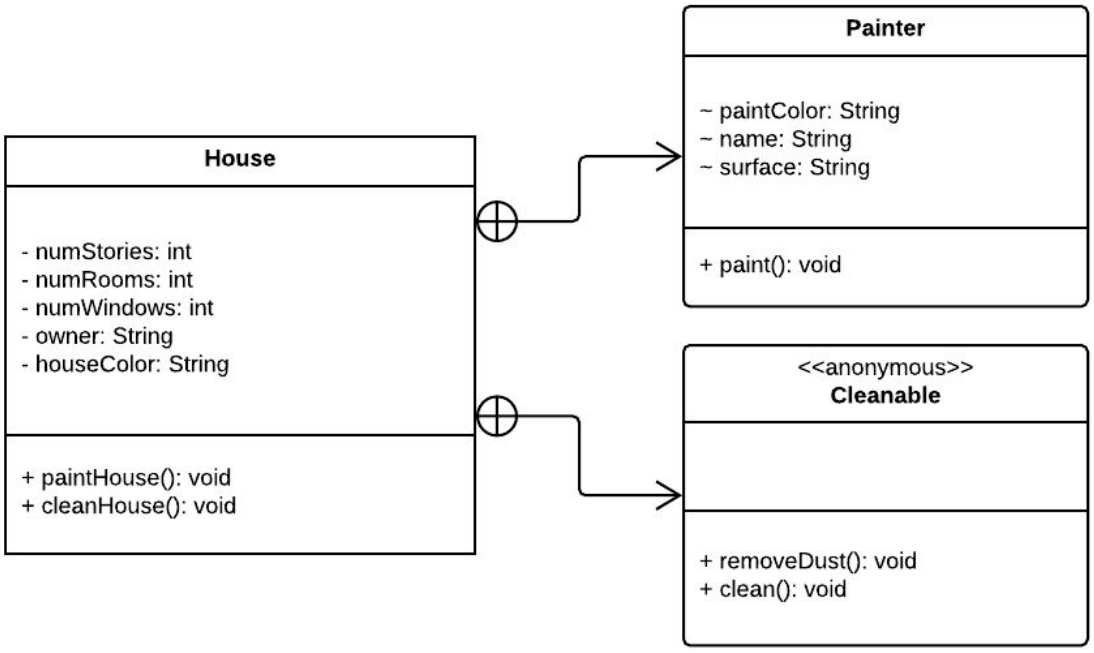
## ➢ Statement

Create a class named *House* that contains the following attributes: *numStories* (int), *numRooms* (int), *numWindows* (int), *owner* (String) and *houseColor* (String). After this create a method named *paintHouse* which creates a Method Local Inner class named *Painter*. This *Painter* class has these attributes: *paintColor* (String), *name* (String) and *surface* (String). It also needs to have a method named *paint* were it displays a message composed of the House (outer class) attributes as well as the own attributes., and it needs to change the *houseColor* attribute depending on the *paintColor* value.

After the *Painter* class is defined within the *paintHouse* method, you need to create an instance of that class and then call the *paint* method.

The *House* class needs to have another method named *cleanHouse* were an anonymous inner class is create in base of an interface created previously named *Cleanable*. This anonymous inner class needs to provide an implementation of the two abstract methods (removeDust and clean) belonging to the *Cleanable* interface.

## ➢ Class design (UML)



➢ **Program Code**

Cleanable.java

```
public interface Cleanable {
	void clean();
	void removeDust();
}
```

House.java

```
public class House {
	// House attributes
	private int numStories;
	private int numRooms;
	private int numWindows;
	private String owner;
	private String houseColor;

	// constructor
	public House(int numStories, int numRooms, int numWindows, String owner,
String houseColor) {
		super();
		this.numStories = numStories;
		this.numRooms = numRooms;
		this.numWindows = numWindows;
		this.owner = owner;
		this.houseColor = houseColor;
	}

	public void paintHouse() {
		// Method Local Inner class
		class Painter {
			// Painter attributes
			String paintColor;
			String name;
			String surface;

			public Painter(String paintColor, String name, String surface){
				this.paintColor = paintColor;
				this.name = name;
				this.surface = surface;
			}

			// Painter method were it has access to the outer
			// class attributes
```

```
                    public void paint() {
                          System.out.println("- "+owner + " hired " + name + " and
he is painting a " + surface + " with " + paintColor + " painting.");
                          houseColor = paintColor; // Changing outer class
                                                   // (House) attribute
                    }
              }

              // Creating an instance of the method local Painter class
              Painter painter = new Painter("Red", "Jason Ralboi", "Wall");
              painter.paint();
       }

       public void cleanHouse() {
              // Anonymous inner class
              Cleanable cleanable = new Cleanable() {

                    // We can't have any constructor method within
                    // anonymous inner classes.

                    // Adding its own implementation of the abstract methods
                    @Override
                    public void removeDust() {
                          System.out.println("\n- "+owner + " is removing the dust
coming from the " + numRooms + " existing rooms\n of the " + numStories + "-story "
+ houseColor + " house.");
                    }

                    @Override
                    public void clean() {
                          System.out.println("- "+owner+" is cleaning the dirt of
the "+numWindows+" windows.");
                    }
              };

              // calling the Cleanable methods from the previous anonymous instance
              cleanable.removeDust();
              cleanable.clean();
       }

       // Getters and Setters
       public int getNumStories() {
              return numStories;
       }

       public void setNumStories(int numStories) {
              this.numStories = numStories;
```

```
	}

	public int getNumRooms() {
		return numRooms;
	}

	public void setNumRooms(int numRooms) {
		this.numRooms = numRooms;
	}

	public int getNumWindows() {
		return numWindows;
	}

	public void setNumWindows(int numWindows) {
		this.numWindows = numWindows;
	}

	public String getOwner() {
		return owner;
	}

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

	public String getHouseColor() {
		return houseColor;
	}

	public void setHouseColor(String houseColor) {
		this.houseColor = houseColor;
	}
}
```

Main.java

```
public class Main {
	public static void main(String[] args) {

		// Creating an instance of the House class
		House house = new House(2, 4, 6, "Lobic Rous", "Green");

		System.out.println("The house color is: "+house.getHouseColor());
		house.paintHouse();
```

```
            System.out.println("Now the house color is: "+house.getHouseColor());
            house.cleanHouse();
        }
}
```

### ➢ Program execution

In this brief example, we create a *House* type object, passing the necessary attributes as parameters. Then we call its *paintHouse* and *cleanHouse* methods that internally create instances of either an anonymous inner class or a method local inner class.

```
The house color is: Green
- Lobic Rous hired Jason Ralboi and he is painting a Wall with Red painting.
Now the house color is: Red

- Lobic Rous is removing the dust coming from the 4 existing rooms
 of the 2-story Red house.
- Lobic Rous is cleaning the dirt of the 6 windows.
```

### ➢ 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.