TITLE: OBJECT ORIENTED PROGRAMMING II

NAME: FRANCIS MUIRURI

REG NO : SCM211-1370/2021

**ANONYMOUS, ABSTRACT AND INNER CLASSES**

**Anonymous Classes**

Anonymous classes (classes with no name) may appear in expressions that expect a class item following a new operator.

Syntax of a new expression with an anonymous class:

new SuperType(constructor parameters)

{ instance methods and instance variables of inner class }

The SuperType may be:

1. A class:

Then the anonymous class extends SuperType, and the parameters are for a call superclass constructor.

1. An interface:

Then the anonymous class implements SuperType (and extends Object), and there can be no constructor parameters.

Since an anonymous class has no name, it cannot have an explicit constructor—only the default constructor provided by Java is available.

Example

Printable [] pt = new Printable[3];

pt[0] = fi;

pt[1] = si;

pt[2] = new Printable() { public void printNum(int n) { System.out.println(">>>" + n + "<<<");

}

};

for (int k=0; k>>45<<<

In anonymous classes, objects are created whenever they are required. That is, objects are created to perform some specific tasks.

Here, an object of the anonymous class is created dynamically when we need to override the display() method.

Anonymous classes also help us to make our code concise.

Use anonymous class when...

1. only need to create one object of the class

2. class is short

3. don't need a constructor or no static fields

**Abstract classes**

• The reserved word abstract can be used as a modifier for an instance method or a class.

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

• An abstract method has no body. It must be overridden in a subclass of its class to be made into a “concrete”, callable method.

• Any class with an abstract method must be declared as an abstract class, although any class can be made abstract using the modifier.

• An abstract class cannot be instantiated. It must be extended by a “concrete” subclass to create

|  |  |
| --- | --- |
| **Interface** | **Abstract Class** |
| Keyword used: interface | Keyword used: abstract |
| Sub classes can implement an interface | Sub classes have to extend abstract class |
| Multiple interfaces can be implemented | One abstract class can be extended |
| Supports Multiple Inheritance | Cannot support Multiple Inheritance |

objects. Those objects may be assigned to a variable of the type of the abstract superclass (upcasting).

**Difference between an interface and an abstract class**

**Rules for java Abstract class**

1. An abstract class must be declared with an abstract keyword.
2. It cannot be instantiated
3. It can have abstract and non-abstract methods.
4. It can have  [constructors and static methods also](https://www.javatpoint.com/java-constructor)
5. It can have final methods which will force the subclass not to change the body of the method.
6. ***Via interface***
7. **An interface** in Java is a boundary between the method and the class implementing it. An interface in Java holds the method signature in it, but never the implementation of the method. In Java, we use the interface to achieve abstraction.
8. package simplilearn;
9. public interface Area {
10. public void Square();
11. public void Circle();
12. public void Rectangle();
13. public void Triangle();
14. }
15. //Class
16. package simplilearn;
17. import java.util.Scanner;
18. public class shapeArea implements Area {
19. public void Circle() {
20. Scanner kb = new Scanner(System.in);
21. System.out.println("Enter the radius of the circle");
22. double r = kb.nextInt();
23. double areaOfCircle = 3.142 \* r \* r;
24. System.out.println("Area of the circle is" + areaOfCircle);
25. }
26. @Override
27. public void Square() {
28. // TODO Auto-generated method stub
29. Scanner kb2 = new Scanner(System.in);
30. System.out.println("Value of the side the square");
31. double s = kb2.nextInt();
32. double areaOfSquare = s \* s;
33. System.out.println("Area of the square is" + areaOfSquare);
34. }
35. @Override
36. public void Rectangle() {
37. // TODO Auto-generated method stub
38. Scanner kb3 = new Scanner(System.in);
39. System.out.println("Enter the length of the Rectangle");
40. double l = kb3.nextInt();
41. System.out.println("Enter the breadth of the Rectangle");
42. double b = kb3.nextInt();
43. double areaOfRectangle = l \* b;
44. System.out.println("Area of the Rectangle is" + areaOfRectangle);
45. }
46. @Override
47. public void Triangle() {
48. // TODO Auto-generated method stub
49. Scanner kb4 = new Scanner(System.in);
50. System.out.println("Enter the base of the Triangle");
51. double base = kb4.nextInt();
52. System.out.println("Enter the height of the Triangle");
53. double h = kb4.nextInt();
54. double areaOfTriangle = 0.5 \* base \* h;
55. System.out.println("Area of the Triangle is" + areaOfTriangle);
56. }
57. public static void main(String[] args) {
58. shapeArea geometry = new shapeArea();
59. geometry.Circle();
60. geometry.Square();
61. geometry.Rectangle();
62. geometry.Triangle();
63. }
64. }

**Inner Classes**

All of the classes defined by us so far have been “top-level” classes, meaning that they lie at the top level of packages and files.

As a consequence, names of classes and their members must be visible to all classes in a package or to none at all.

Inner classes allow us to restrict visibility by making a class “local” to another class or to a method.

An inner class may be defined inside:

1. Another class.

2. A method (a block).

3. An expression (using an anonymous class).

Each object created from an inner class is “associated” with an object created from the class in which it is defined.

Inner classes may not have static members.

The name of an inner class must be different from the names of all of its enclosing classes.

**TYPES OF INNER CLASSES**

***Static Inner Classes***

These are the simplest type of inner classes. Static inner classes are those that are declared inside a class and marked static. It should be noted that these classes can only be accessed using an instance of the outer class. You can take advantage of static nested classes for grouping related classes together.

#### *Non-static Inner Classes*

A non-static inner class as the name suggests, is associated with an instance of an outer class. All the members (variables and methods) of the outer class are accessible from these classes.

#### *Local Inner Classes*

Local inner classes are defined within a method. They have access to all the members (variables and methods) of the enclosing class but they cannot be instantiated from outside the method in which they are defined. An inner class defined locally may only be instantiated inside its method where it has been defined. A method local inner class is accessible only within the method in which it was defined, and cannot be referenced by any other code outside the method in which it is defined. A method local inner class can access local variables from the enclosing scope (including final variables).

#### *Anonymous Inner Classes*

Inner classes that don’t have names are also known as anonymous inner classes. Both the declaration and instantiation of an anonymous inner class occur simultaneously. Anonymous inner classes cannot have explicit constructors, just like all local inner classes. Anonymous inner classes are useful when you have to use a local inner class only once.

public class MyOuterClass {

public class MyInnerClass {

public void display() {

System.out.println("Inside the inner class");

}

}

public static void main(String[] args) {

MyOuterClass objOuterClass = new MyOuterClass();

MyOuterClass.MyInnerClass objInnerClass = objOuterClass.new MyInnerClass();

objInnerClass.display();

}

}