upGrad Campus

Lecture Notes

Lecture Notes – Classes & Objects

Object-oriented programming, or OOP in short, gives you a particular methodology for implementing large and complex programming projects in a very simple manner. The methodology uses the structure of classes and objects, and four OOP principles, namely abstraction, encapsulation, inheritance and polymorphism, to simplify the large programming projects.

All programming languages are not based on this OOP model of classes and objects, and some are known as procedural programming languages. But most of the modern general-purpose programming languages, such as C++, C#, Java, and Python, are object-oriented programming languages.

Benefits of Object Oriented Programming:

1. The most important benefit of an object-oriented approach is that you tend to build any project in a more organised manner, making the development process very efficient.
2. The second benefit is that an object-oriented approach helps you implement real-world scenarios very naturally.
3. The third benefit is that the modification and updation of each independent module is easier in this case.

All of these benefits are the requirements of a good software development methodology; and through this module, you will see how all these requirements are met well by an object-oriented programming approach.

Classes

What is a class?
A class is a blueprint of an application, where you implement methods (to perform certain actions/functionalities) and declare variables (that describe the properties of the application). For example, Payments could be an entire class in itself in the Ola App.

In Java, every program has at least one class, whose structure looks like this:

A class can have 'variables' that have certain information in them, 'methods' that have functionalities linked with different tasks of that class, and it can also have subclasses. For now, let's focus only on the variables and methods.

Let's look at an example of a university's information management system. Now suppose a particular department needs to manage all the student information. So there can be a class named Student{ }. The pseudocode of this class will be

```
class Student{
   public int rollno;
   public String name;
   public double cgpa;
}
```

Apart from the variables, the classes mainly contain methods; these basically represent the functionalities of different activities related to each class. Now the activities that students may need to perform on the

information management system could be editing the profile, registering for a new course, submitting assignments, or checking their exam results. So in the case of a student class, the methods can be editProfile( ), displayProfile( ), registerCourse( ), submitAssignment( ), checkResult( ), etc. These methods can have their own functionalities. For example, you can have a code for printing a student's name, roll number and cgpa on the screen, implemented inside the displayProfile method, as shown in the sample code. This is more like a pseudocode. You can see that the return-type for this method is set to void as nothing is being returned from this method. Some of these methods, such as editProfile() or submitAssignment(), may have some arguments in them; but right now, you need to go into the actual implementations of these codes. This is a pseudocode representing the structure of a class:

```
public class Student{
   public int rollno;
   public String name;
   public double cgpa;
}
```

You can see the nomenclature pattern of the classes, which is always a single word with the first letter in upper case. So just like you had Student, there can be other classes such as Faculty{ }, Staff { }, etc.

```
public class Student{
      ...
}
public class Faculty{
      ...
}
public class Staff{
      ...
}
```

The **whole idea behind classes is to have independent modules in the program, each class representing one particular entity**. That entity can have various data members, variables, and methods linked to it. For example, Circle & Square can be examples of different classes in a drawing software. The data members for Circle and Square would be variables such as radius and length, respectively.

And as a good practice, keep the **public static void main** method in a separate class that was named **Main** in our example. Also, you can have a different name for the class that contains the main method. However, the name of the Java file has to be the same as the name of that particular class.

Let's say you need to build a program to calculate the area of a circle. The most basic structure for doing this would be to create a Circle class that contains a main( ) method and a findArea( ) method. Now you would start with what you already know about writing a code with a single method only and then convert it into the ideal object-oriented programming structure.

Let's look at this code in detail. You will define your findArea method in the same way as you defined the main method, except for one thing: the return type for this method will be double since the method is returning an area that can be a decimal value. You can declare your function as `public static double findArea(double radius);` and inside this, you can write the formula for calculating the radius, and assign this to a new variable area that is declared as double. Here it is declared in one line, but you can also manage this process in two lines: declaring, assigning, and then returning the area.

Now you can call this function in main by assigning a value, let's say, 2, to the radius variable, and then calling the findArea( ) method and storing it in a variable called area. This process is called method

invoking. And then you can write the command to print this variable. So if you run this program, it will print the result on the console: the area of the circle is 12.56.

Now you need to modify this code to convert it into a better design as per the object-oriented programming style. As we discussed, you need to have two classes: a main class containing the main method and a Circle class containing the findArea( ) method.

```
public class Main {
  public static void main(String[] args) {
      double area = Circle.findArea(2);
      System.out.println("Area of circle is "+area);
  }
}
public class Circle {
  public static double findArea(double radius) {
      double area = 3.14 * radius * radius;
      return area;
  }
}
```

If you are declaring an independent class, Circle{ }, the best practice is to declare the variable 'double radius' in the class. Declare it in the same way as the method, that is, `public static double radius`. You can modify this method by passing no argument in the findArea( ) method, as your radius is already declared above, and if now, you write the radius inside your method, it will refer to the declared one.

Now you need to modify the main class accordingly; since you declared a variable radius in the Circle class, you can set any value for that radius from the main method by writing Circle.radius = let's say, 2. So you can see that even to access the variable, you need to write the same, 'Circle + dot operator', before writing the name of the variable; it is not in the current class. And now, since your variable radius is assigned a value of 2, you can invoke the findArea( ) method by passing no argument. This will give you the same output as before.

```
public class Main {
  public static void main(String[] args) {
      Circle.radius = 2.0;
      double area = Circle.findArea();
      System.out.println("Area of circle is "+area);
  }
}
public class Circle{
  public static double radius;
  public static double findArea() {
      double area = 3.14 * radius * radius;
      return area;
  }
}
```

So the program can be considered as having a good design, as per an object-oriented style, because it has all the information and functionality related to the circles in the Circle{ } class.

upGrad Campus

Objects

What are objects?

Class is a blueprint and objects are the blocks built upon this blueprint with certain state and behaviour.

What is state and behaviour of an object?

The state of an object refers to the specific values of the variables of the class the object belongs to. The behaviour of an object refers to the action performed by the object when called by a particular method from the class.

Example

Class	Circle		
Object	c1 circle of radius 5f	c2 circle of radius 10f	c3 circle of radius 20f
State of object	radius 5f	radius 10f	radius 20f
Behaviour of object	c1.area( ), which returns 78.525	c1.area( ), which returns 314.1	c1.area( ), which returns 1256.4

Relation between a class and its object

Let's take another example of a house class to understand this better.

Suppose that an architect designs a blueprint for a house. This same blueprint can then be used to build multiple houses. A single blueprint is useful to build several houses that have the same architecture. The concept of classes and objects is similar to this. Here, the **house blueprint acts as a class** and **a particular house represents the object**.

For a Student class, your objects would be different students such as Prateek, Ankit, Ajay, etc. There can be thousands of different objects, with each object having the characteristics of the Student class.

Let's say Ankit is an object of the Student class. This means that Ankit has a roll number, a name and a CGPA. These variables, name, CGPA and roll number, are the instance variables in this case. And the values of these instance variables represent the state of the object, Ankit. So if Ankit displays his profile and registers for a course, these actions represent the behaviour of the object, Ankit. Similarly, other objects of this Student class will also have a roll number, name, CGPA, etc., but the values for these instance variables will be different from Ankit's.

Now that you know what objects are, it's time to show you how you can create objects for a particular class, and how you can use these objects in the main class. So let's go back to the code you built earlier, in which you had two different classes.

You can use the object ‘circle’ in this example. This process is very similar to how earlier, you used the dot operator, with the name of the class, to access variables and invoke methods when you didn’t have objects. Now, you will use the same dot operator, but this time, with a name for the object, which, in this ie, is circle with a small c. So write circle, the dot operator, and the name of the variable from the class you want to set. Now assigning the value 2, here, would mean that the radius of this particular circle is set to 2. So you defined the state of this circle. You can now check the behaviour of this object by invoking the findArea( ) method. To do this, you have a similar command. You need to write the name of the object, ‘circle’, with the dot operator, and the name of the method you want to invoke, which is findArea( ), in this case. You can run this particular program and see the output, which is very similar to your earlier programs. The only difference is that you used an object to run this program.

```
public class Main {
  public static void main(String[] args) {
      Circle circle = new Circle();
      circle.radius = 2.0;
      double area = circle.findArea();
      System.out.println("Area of circle is "+area);
  }
}
public class Circle{
  public double radius;
  public double findArea() {
      double area = 3.14 * radius * radius;
      return area;
  }
}
```

Multiple Objects

For the **Student class**, your **objects would be different students — s1, s2, s3, s4 —** with names such as Prateek, Ankit, and Ajay. There can be thousands of objects, where each object has all the characteristics of the Student class.

Instances: An object of a class is also referred to as an instance. The word “object” or “instance” can be interchangeably used.

For example, s1, s2, s3, etc. are all instances or objects of the Student class.

Instance variables: Instance variables are variables from the class, which become the characteristics (or states) of the objects/instances. Every object has its own copies of instance variables.

For example, roll number, name, CGPA are all instance variables of the Student class.

State and behaviour of instances: Specific values assigned to these variables for a given instance is represented as the state of that object/instance, calling the methods would be represented as the behaviour of that instance.

So now you can create as many circles as you want, and you can name them c1, c2, c3, and so on. You can also use these circles for any operations you want to perform. This is the real benefit of objects.

```java
public class Main {
  public static void main(String[] args) {
      Circle c1 = new Circle();
      c1.radius = 2.0;

      Circle c2 = new Circle();
      circle2.radius = 4.0;

      // Adding radii of both the circles
       System.out.println(c1.radius + c2.radius);

      // Adding areas of both the circles
       System.out.println(c1.findArea() + c2.findArea());
  }
}

public class Circle{
  public double radius;
  public double findArea() {
      double area = 3.14 * radius * radius;
      return area;
  }
}
```

Modelling Real-life Scenarios

Object-oriented programming allows you to implement real-world scenarios very naturally. So let's see how classes and objects will help you do this. Consider a scenario in the information management system of a university, where a student registers for a course that is taught by a particular faculty. Now for this example, if you consider these three entities as three different classes, then what could the objects for each of these classes be? The instances for the Student class can be Ankit, Prateek, Ajay, etc. Similarly, the instances for the Course class can be C, C++, Java, Python, etc. Finally, the instances for the Faculty class can be Prof. Sujit, Prof. Tricha, Prof. Murli, etc. Now these names would represent the state of the objects for the Faculty class.

Constructors

Constructor is a special type of method used to initialize an object of a class and define values of the instance variables of the object.

You can call a Java constructor while creating a new object. It constructs the values for the instance variables, i.e. provides data for the object. This is why it is known as a constructor.

let's see why customised constructors are required in the first place. Let's take the same example as before, and let's say in this code, instead of defining the radius of the circle as 2, in the next line, the user may forget to assign a value to the instance variable radius, in the case of some objects. Now if you want to make a new circle and assign the radius as 2 in the same line, you can do this by passing the parameter through a constructor, and the parameter will be the radius you want to set, which is 2. Then, you have to create the customised constructor in the Circle class in which, this time, you have to pass an argument. This is because you passed 2 directly into the constructor in the main class. You can create this method by passing an argument, double r, and inside the method, just mention that this r is nothing but the value of the public variable radius defined in this class. So if you look carefully, you'll see that whatever is passed in this method, the argument is stored in the variable radius. In this example, 2 is the argument, so 2 gets stored in the radius for this particular object. So you basically made a customised constructor method so

that you can assign the value of the variables at the same time as you make an object. This will ensure that the values are assigned to the instance variables as soon as a new object is made, so that you don't forget to assign the values to instance variables while creating a new object.

```
public class Main {
  public static void main(String[] args) {
      Circle circle = new Circle(2);
      circle.radius = 2.0;
      double area = circle.findArea();
      System.out.println("Area of circle is "+area);
  }
}
public class Circle{
  public double radius;

   public Circle(double r) {
      radius = r;
   }


  public double findArea() {
      double area = 3.14 * radius * radius;
      return area;
  }
}
```

A constructor is used to initialize the state of an object.

Parameterized constructor: A constructor that has parameters is known as a parameterized constructor. A parameterized constructor is used to assign different values to the instance variables of distinct objects.

Default constructor: A constructor that has no parameter and does nothing apart from creating a new object is known as a default constructor. It is a method that need not be declared if there is no parameterized constructor, or you can declare it and leave it empty in case of a declaration of any other parameterized constructor. Default constructors assign default values to the instance variables of the objects depending on the type, for example, 0, null, etc.

To help better your understanding of how these constructors help you make your codes shorter, and how they take care to ensure that you don't forget to assign values to the instance variables when you make new objects

The 'this' keyword

The **this** keyword is used to **refer to the current class instance variable**. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem.

For example, if there is an instance variable named name declared inside a class, it can be referred to as this.name in all the methods within that class.

The 'static' keyword

The **static keyword** is used for memory management, i.e. it helps in making your program memory-efficient. The static keyword belongs to the class, rather than any particular object.

Static Variable: Variables declared as static are known as static variables. Static variable is used while referring to the common property of all the objects of the class. For example, university name for students in the Student class for an information management system of a university. Static variables are useful if you need to share the same set of information across all objects of a class, such as the University name in our example.

The static variable is allocated memory only once, at the time of loading a class.

Static Method: Methods declared as static are known as static methods. A static method belongs to the class, rather than object of a class. It can be called without creating the object of the class, using ClassName.staticMethod( ) notation.

Static methods cannot access non-static variables, or call non-static methods. But static methods can access any static variable and change its value.

The 'final' keyword

The **final keyword** is used to restrict the user. If any method is declared as final, you cannot override that particular method.

Final variables: If any variable is declared as final, you cannot change the value of that variable. It can be assigned any value only once, and then it remains constant throughout the program. Final variables are useful if you have global constants in your programs that should never be changed.

Final methods: If you declare any method as final, you cannot override that.

Abstraction

What is Abstraction?
Abstraction is a principle based on **hiding the details of implementations of classes**, and access only certain features/functionalities given to the users/other parts of the program. This is done by writing your program using the framework of classes and objects.

Advantages of abstraction:

1. Abstraction allows you to use the functionality (behaviour) of the objects from other parts of the code without showing the internal implementation of that functionality (methods).

2. Rights to change the implementation of a class can be given only to a certain set of users. Others cannot go and arbitrarily change anything within the class, apart from using the behaviors defined by that class by creating and using its objects.

While programming projects also, abstraction is sometimes very necessary in cases where you don't have to show the implementation to certain users, but you just need to give them access to use the functionality. You can create multiple objects for a class, assign values to them, invoke methods, perform operations, and do a lot more from the main class. But you can't go and edit the class and the methods inside. That implementation is limited to specific people only.

Public & Private Access Modifiers

Access modifiers are used to restrict the accessibility of methods or variables of a class. There are four types of access modifiers—public, private, protected, and default—in Java. Let's learn more about them.

Public Access Modifier

When you declare a variable or a method as public in a class, it signifies that it can be accessed throughout the class. You can access these variables and methods from other classes as well. When a method of a class is declared as public, it can be accessed by creating an object of its class in other classes. If it is a static method, then you can directly access it using className.method() without creating an instance.

This creates potential issues because other classes can now access these public variables and methods and write code that depends on them. This will reduce your flexibility to change the implementation of these public variables and methods at a later point. This is because it will affect other classes that depends on these public variables and methods.

Private Access Modifier

When you declare a variable as private in a class, it signifies that it can be accessed throughout the class but not outside of it. You can't access it from another class. When a method is declared private, it cannot be accessed by creating an object outside the class. It is restricted only to the class that contains it.

Getter Methods

Getter methods are used to access private variables from outside the class in which they are declared. The standard way in which a private variable can be declared along with its getter method is as follows:

```
class Demo {
  private int var = 5;      //assigning a value to the private int variable

  public int getVar() {
      return var;
  }
}
```

After the previous code, you can access the value assigned to this variable from outside the class. You can do this by calling the getVar( ) method from outside the class in the following manner.

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

        Demo d = new Demo();

        d.getVar();        //calling getVar method to access its value
  }
```

Setter Methods

Setter methods are used to set a new value to private variables, or modify their values from outside the class in which they are declared. The standard way in which a private variable can be declared along with its setter method is as follows:

```
class Demo {
  private int var;      //just declaring a private int variable

  public void setVar(int var) {
      this.var = var;
  }
}
```

After the previous code, you can assign a new value to this variable from outside the class by calling the setVar( ) method from outside the class in the following manner.

```
public class ClassesAndObjects {
  public static void main(String[] args) {
        Demo d = new Demo();

        d.setVar(5);        //calling setVar method to modify its value
  }
}
```

Difference between Constructors & Setter Methods

1. **Constructors are automatically called as soon as the object** is being created, whereas **setter methods can be called after the object is created**, for example, when you want to set/modify the value of any instance variable.
2. Name of a constructor is always the same as the name of the class.
3. No return type (void, int, String, etc) is mentioned in case of constructors.
4. Different constraints for the value of variable can be added to the setter methods. You can also call the setter method from the constructor, if you want to place constraints on the values of the instance variables of the class while creating a new object. This can be done in the following way:

```
class Demo {
  private int var;         //just declaring a private int variable

public Demo(int var) {       //declaring parameterised setter method
      setVar(var);
  }

  public void setVar(int var) {     //declaring setter method
      this.var = var;
      if (this.var < 0){                //adding constraint
         this.var = 0;
      }
  }
}
```

Encapsulation

What is encapsulation?
Encapsulation is a principle based on **hiding the state of objects and restricting their access** to various parts of your program. Encapsulation is achieved with the use of private access modifiers and the getter and setter methods.

How do you create a fully encapsulated class?

1. Make all the instance variables of a class private.
2. Only use getter and setter methods to read or write values of the instance variables.

Advantages of encapsulation:

1. You can make your class read-only or write-only by declaring only getter or setter methods. This prevents other code or malicious code from accessing instance variables in your class that they should not read or modify.
2. You can add variable logic/constraints in the setter methods, so you have full control of the data that can be assigned to the instance variables.

Both of the above points contribute to the **objective of code-safety**, which we discussed at the start of this session.

Classes whose variables are declared as private and can only be accessed through methods are known as encapsulated classes. Encapsulation is also referred to as the process of wrapping all data and code acting on the data i.e. the methods together, into a single unit, which is your class in this case. In your encapsulated class, the information or data of that class is private or hidden from the user. Let's look at some of the benefits of encapsulation. The first one is,you have some control on what values can be stored in the variables. No one can come in and save any junk values in these variables. The second benefit is you can make these variables either read-only or write-only depending on your program. What this means is if you want to make the instance variable write-only, then you can create a setter method for that without creating any getter method. And in case of the read-only variable, you can just create a getter method without creating a setter method.

Module Summary

So far, you learnt about two important principles of object-oriented programming. The first one is abstraction - i.e. implementing the structure of your classes and objects in your program appropriately, to hide implementation of the classes from the user and by solely giving access only to the instance variables and the ability to invoke the methods for different objects. The second design principle is encapsulation - this will keep all the instance variables private and gives its access to the user through getter and setter functions only. This is done to hide the information or variables of that class and give access to that information, only through certain methods. This also restricts the storage of junk values in the variables. You have seen the examples of encapsulation through the Circle class.

1. **Classes & Objects:** 'Classes and Objects' is the framework of Object Oriented Programming which revolve around the real-life entities.

Class: A class is a blueprint from which objects are created. It represents a set of attributes and methods that represents a group of entities, and common to all the objects of one type.

Object: An object represents a real life entity, which belongs to a class with same properties as that of a class. The object can perform all functionalities declared in a class, by invoking the methods declared there.

2. **Constructors:** A constructor is used to initialize the state of an object.

.

Default constructor: A constructor that doesn't need to be declared and is without any parameter is known as a default constructor. A default constructor can assign user defined ors the default values to the instance variables of the object depending on the type. For example, 0, null, etc..

Parameterized constructor: A constructor that takes in some parameter for the initialisation of instances variables while creating an object is known as a parameterized constructor. A parameterized constructor is used to assign different values to the instance variables when creating of distinct objects.

3. **The this, static & final keywords:**

.

The this keyword: The this keyword can be used to refer to the current class instance variables. If there is ambiguity between the instance variables and parameters, the this keyword resolves the problem.

The static keyword: The static keyword is used for memory management and helps in making your program memory-efficient. Anything which is declared static belongs to the entire class, rather than only the instance of the class, i.e. a static member is the common property of all the objects of the class.

The final keyword: The final keyword is used to restrict the user from updating the value of instance variables. If any variable is declared as final, you cannot change the value of that variable. If any method is declared as final, you cannot override that particular method.

4. **Abstraction:** Abstraction is a principle based on **hiding the details of implementations of classes**, and access only certain features/functionalities given to the users/other parts of the program. This is done by writing your program using the framework of classes and objects.

5. **Private access modifier:** A type which is declared while declaring any member of a class (instance variables, methods, etc.). Private access modifiers restrict the access of members within their own classes only.

6. **Declaring instance variables of a class as private:** All instance variables of a class should be declared as private in order to restrict the access to the variable's data from outside the class, which makes the program safer.

7. **Getter methods to read the value of private variables:** A public getter method is declared in the class to give read-only access of the private variable from outside the class.

8. **Setter methods to write the value of private variables:** A public setter method is declared in the class to gives write-only access to the private variable from outside the class. You can add constraints/logic of values that can be assigned to these variables.

9. **Declaring private methods and constructors as private:** Methods declared as private cannot be called from outside that class. Constructors declared as private don't allow the creation of objects of that particular class outside that class..

10. **Making a fully encapsulated class:** We can boost the safety of a program by declaring all the data members/instance variables of that class as private. These classes can be made read-only, or write-only by declaring only getter or setter methods.