10 Object Oriented Programming I

Programming languages help a programmer to translate his ideas into a form that the machine can understand. The method of designing and implementing the programs using the features of a language is a programming paradigm. One such method is the Procedural programming which focuses more on procedure than on data. It separates the functions and the data that uses those functions, which is not a very useful thing as it makes our code less maintainable. This leads us to another technique called the object oriented programming.

What is OOPs?

Object means any real world entity like pen, car, chair, etc. Object oriented programming is the programming model that focuses more on the objects than on the procedure for doing it. Thus we can say that oops focuses more on data than on logic or action. In this method the program is made in a way as real world works. In short it is a method to design a program using objects and classes.

Advantages of OOPs:

1. It is more appropriate for real world problems
2. OOP provides better modularity:
   - Data hiding
   - Abstraction
3. OOPs provides better reusability:
   - Inheritance
4. OOP makes our code easy to maintain and modify:
   - Polymorphism

Concepts in OOPs: One of the major shortcomings of the procedural method is that the data and the functions that use those data are separated. This is not preferred as if a data member is changed then all the functions related to it have to be changed again and again. This leads to mistakes and maintaining the code becomes tough.

How to solve?

The best way to solve this problem would be to wrap the data and all the functions related to them in one unit. Each of this unit is an object.

Objects are the basic units of Object-oriented programming. Objects are the real world entities about which we code. Objects have properties and behavior. For instance take an example of a car which has properties like its model and has four tires and behavior of changing speed and applying brakes.

State of the objects is represented by data members and their behavior is represented by methods. Now many objects share common properties and behavior. So, a group of such properties and behavior is made that can generate many instances of itself that have similar characteristics. This group is called a class.

Classes are the blueprints from which objects are created. Like there are many cars which share common properties and behavior. The class provides these properties and behavior to its objects which are the instances of that class.

For example a vehicle class might look like this:

```
public class Vehicle {

    public String brand;
    protected String model;
    private double price;
    int numWheels;
    int yearOfManufacture;
    String color;

    public double getPrice() {
        return price;
    }

    public void printDescription() {
        System.out.println(brand + " " + model + " " + price + " " + numWheels);
    }
}
```

Now each vehicle will be a specific copy of this template.

A class in java contains:

- Data members
- Methods
- Constructor

Now let us talk about each of these in detail:

Data Members: Data members are the properties that are present in a class. The type of these properties can be modified by using special keywords called modifiers. Let us build our own student class and learn about them.

Static and Non Static Properties: Static properties are those that are common to all objects and belong to the class rather each specific object. So each object that we create doesn't have their copy. They are shared by all the objects of the class. We need to write the static keyword before it in order to makeit static.

For e.g.:

```
static int numStudents;
```

Here the number of students in a batch is a property that isn't specific to each student and hence is static.

But the properties like name, roll number etc. can have different values for each student and are object specific and thus are non-static.

An important point to note is that whenever we create a new object only the non-static data member copies are created and the static properties are stored within the class only! This could be considereda very memory efficient practice as static members of a class are made only once.

A general student class might look like this:

```
public class Student {
static int numStudents;
String name;
int rollNo;
}
```

Access Modifiers:

Private: If we make any data member as private it is visible only within the class i.e. it can be accessed by and through the methods of the same class. So we can provide setters and getters function through which they can be accessed outside the class.

For instance in our student class we would like to keep the roll numbers for each student as private as we may not want any other person to modify those roll numbers by making them public. So we will make these private and provide the getter method to access the roll numbers of the students outside the student class.

```
public class Student {
private int rollNo;

public int getRollNo() {
        return this.rollNo;
}
}
```

Default: When we explicitly don't write any modifier it is default. This modifier is package friendly i.e. it can be accessed anywhere within the same package.

Protected: It is accessible only within the same package but can be accessed outside the package using inheritance.

Public: It is accessible everywhere.

An important point to note here is that it is better to make a variable private and then provide getters and setters (in case we wish others to view and change it) than making the variable public. Because by providing setter we can actually add constraints to the function and update value only if they are satisfied (say if we make the marks of a student public someone can even set them to incorrect values like negative numbers. So it would be wise to provide a setter method for marks of the student so that these conditions are checked and correct marks are updated).

Final Keyword: Final keyword can be applied before a variable, method and a class. A final variable is one whose value can't be changed. So we can either initialize a final variable at the time of declaration or in a constructor. The value of a final variable can be assigned only once. A final method is one that can't be overridden. A final class means it can't be inherited (like the String class in java is final). For example if we want the number of students in a class to be equal to 40 and we don't wish to change this number in any circumstances then this data member would be called final.

```
  public class Student {
  final int noOfStudents = 40;
}
```

An important point to note here is that if a static variable is made final then they must be assigned a value with their declaration. This behavior is obvious as static variables are shared among all the objects of a class; creating a new object would change the same static variable which is not allowed if the static variable is final.

Methods: After having discussed about the data members of a class let us move onto the methods contained in the class relating to the data members of the class. We made many members of the class as private or protected. Now to set, modify or get the values of those data members, public getter and setter methods can be made and called on the objects of that class. The methods are called on the object name by using the dot operator.

Now let us look at various modifiers that can be used to modify the types of methods and the differences between them.

Static v/s Non Static Methods: Like data members, methods of a class can also be static which means those methods belong to the class rather than the objects for the class. These methods are directly called by the class name.

As the static methods belong to a class we don't need any instance of a class to access them. An important implication of this point is that the non-static properties thus can't be accessed by the static methods as there is no specific instance of the class associated with them (the non-static properties are specific to each object). So, non-static members and the 'this' keyword can't be used with the static functions. Thus these methods are generally used for the static properties of the class only!

The non-static methods on the other hand are called on an instance of a class or an object and can thus access both static and non-static properties present in the object.

The access modifiers work the same with the methods as they do with the data members. The public methods can be accessed anywhere whereas the private methods are available only within the same class. Thus private methods can be used to work with the data members that we don't wish to expose to the clients.

Now let us add some methods to our student class.

```
public class Student {
final static int noOfStudents = 40;
String name;
private int rollNo;
public String getName() {
        return this.name;
}
public int getRollNo() {
        return this.rollNo;
}
public int getNumStudents() {
        return Student.noOfStudents;
}
}
```

Constructor: As we have our student class ready, we can now create its objects. Each student will be a specific copy of this template. The syntax to create an object is:

```
public static void main(String[] args) {
        Student s = new Student();
        // s.name - will give access to this student's name
}
```

Notice that there is a **new** keyword and a method is called with the same name as that of the class. The new keyword creates a java object and occupies the memory for it on the heap. The method called constructor, is a special method used to initialize a new object and its name is same as that of the class name.

Even though in our student class we haven't created an explicit constructor there is a default constructor implicitly there. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class. It initializes member data variable to default values (numeric values are initialized as *0*, Boolean values are initialized as *false* and references are initialized as *null*).

We can also create our own constructors. One important point to note here is that as soon as we create our constructor the default constructor goes off. We can also make multiple constructors each varying in the number of arguments being passed (i.e. constructor overloading). The constructor that will be called will be decided on runtime depending on the type and number of arguments specified while creating the object.

Below is our own custom constructor for our student class.

```
public class Student {
String name;
int rollNo;
public Student(String name) {
	this.name = name;
}
public Student(String name, introllNo) {
	this.name = name;
	this.rollNo = rollNo;
}
}
```

The 'this' keyword: Here **this** is a keyword that refers to current object. So, **this.name** refers to the data member (i.e. name) of this object and not the argument variable name.

In general, there can be many uses of 'this' keyword.

- The 'this' keyword can be used to refer current class instance variable.
- The this() can be used to invoke current class constructor.
- The 'this' keyword can be used to invoke current class method (implicitly).
- The 'this' can be passed as an argument in the method call.
- The 'this' can be passed as argument in the constructor call.
- The 'this' keyword can also be used to return the current class instance.

There is also a special type of constructor called the **Copy Constructor**. Java doesn't have a default copy constructor but we can create one of our own. Given below is an example of a copy constructor.

```
public class Student {
String name;
int rollNo;
public Student(String name, int rollNo) {
	this.name = name;
	this.rollNo = rollNo;
}
```

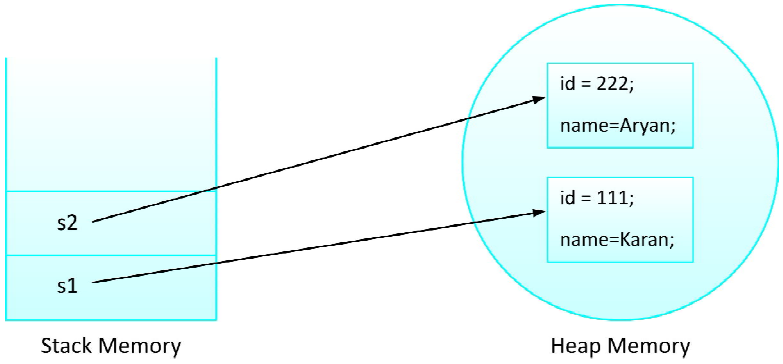
```
    public Student(Student s) {
            this.rollNo = s.rollNo;
            this.name = s.name;
    }
}
```

After leaning so much about the objects and constructors let us summarize the initialization of an object in simple steps.

As any new object is created, following steps take place:

1. Memory is allocated on heap and its reference is stored in stack.
2. The non staticdata members are initialized to their default values.
3. The 'this' keyword is set to the current object.
4. The instance initialize is run and the fields are initialized to their respected values.
5. The constructor code is executed.

The final memory map of the instance of our student class is given below:



Encapsulation-The First Pillar of OOPs: We learnt that one of the major disadvantages of the procedural paradigm was that the functions and the data were separated which made the maintainability of the code poor! To overcome this we made classes which contained both the data members and their methods. This wrapping up of data and its functions into a single unit (called class) is known as **Encapsulation**.

A class classifies its members into three types: private, protected and public. Thus **Data Hiding** is implemented through private and protected members. These private and protected members can be accessed or set using public functions.

Also, the outside world needs to know only the essential details through public members. The rest of the implementation details remain hidden from the outside world which is nothing but **Abstraction**.

Like for instance if we wish to know the aggregate marks of a student, we need not know how the aggregate is calculated. We are just interested in the marks and for that we only need to know that we need to call a public function called 'totalMarks' on any student object.

We have an additional benefit that comes bounded with encapsulation which is **Modularity**. Modularity is nothing but partitioning our code into small individual components called modules. It makes our code less complex and easy to understand. Like for example, our code represents a school. A school has many individual components like students, teachers, other helping staff, etc. now each of them are complete units in themselves yet they are part of the school. This is called modularity.

After having the overview of benefits of encapsulation let us dive into the details of data hiding and abstraction.

Let us suppose that we make all the data members of our student class as public. Now accidently if any of our client changes the roll numbers of the student objects, all are data would be ruined. Thus public data is not safe. So, to make our data safe, we need to hide our data by making it either private or protected! This technique is called information hiding. The private data can be accessed only via a proper channel that is through their access methods which are made public.

To use a class, we only need to know the public API of the class. We only need to know the signature of the public methods that is their input and output forms to access a class. Thus, only the essential details are shown to the end user and all the complex implementation details are kept hidden. This is Abstraction. After all sometimes ignorance is bliss!