Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Passing

Java - What is OOP?

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

Benefits of OOP

i) Modularity for easier troubleshooting

Something has gone wrong, and you have no idea where to look. Is the problem in the Widget file, or is it the WhaleFlumper? Will you have to trudge through that "sewage.c" file? Hope you commented your code!

When working with object-oriented programming languages, you know exactly where to look. "Oh, the car object broke down? The problem must be in the Car class!" You don't have to muck through anything else.

That's the beauty of encapsulation. Objects are self-contained, and each bit of functionality does its own thing while leaving the other bits alone. Also, this modality allows an IT team to work on multiple objects simultaneously while minimizing the chance that one person might duplicate someone else's functionality.

ii) Reuse of code through inheritance

Suppose that in addition to your Car object, one colleague needs a RaceCar object, and another needs a Limousine object. Everyone builds their objects separately but discover commonalities between them. In fact, each object is really just a different kind of Car. This is where the inheritance technique saves time: Create one generic class (Car), and then define the subclasses (RaceCar and Limousine) that are to inherit the generic class's traits.

Of course, Limousine and RaceCar still have their unique attributes and functions. If the RaceCar object needs a method to "fireAfterBurners" and the Limousine object requires a Chauffeur, each class could implement separate functions just for itself. However, because both classes inherit key aspects from the Car class, for example the "drive" or "fillUpGas" methods, your inheriting classes can simply reuse existing code instead of writing these functions all over again.

What if you want to make a change to all Car objects, regardless of type? This is another advantage of the OO approach. Simply make a change to your Car class, and all car objects will simply inherit the new code.

iii. Flexibility through polymorphism

Riffing on this example, you now need just a few drivers, or functions, like "driveCar," driveRaceCar" and "DriveLimousine." RaceCarDrivers share some traits with LimousineDrivers, but other things, like RaceHelmets and BeverageSponsorships, are unique.

This is where object-oriented programming's sweet polymorphism comes into play. Because a single function can shape-shift to adapt to whichever class it's in, you could create one function in the parent Car class called "drive" — not "driveCar" or "driveRaceCar," but just "drive." This one function would work with the RaceCarDriver, LimousineDriver, etc. In fact, you could even have "raceCar.drive(myRaceCarDriver)" or "limo.drive(myChauffeur)."

iv. Effective problem solving

A language like C has an amazing legacy in programming history, but writing software in a top-down language is like playing Jenga while wearing mittens. The more complex it gets, the greater the chance it will collapse. Meanwhile, writing a functional-style program in a language like Haskell or ML can be a chore.

Object-oriented programming is often the most natural and pragmatic approach, once you get the hang of it. OOP languages allows you to break down your software into bite-sized problems that you then can solve — one object at a time.

This isn't to say that OOP is the One True Way. However, the advantages of object-oriented programming are many. When you need to solve complex programming challenges and want to add code tools to your skill set, OOP is your friend — and has much greater longevity and utility than Pac-Man or parachute pants.

Java - What are Classes and Objects?

- **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
- **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

class

Fruit

objects

Apple

Banana

Mango

Another example:

class

Car

objects

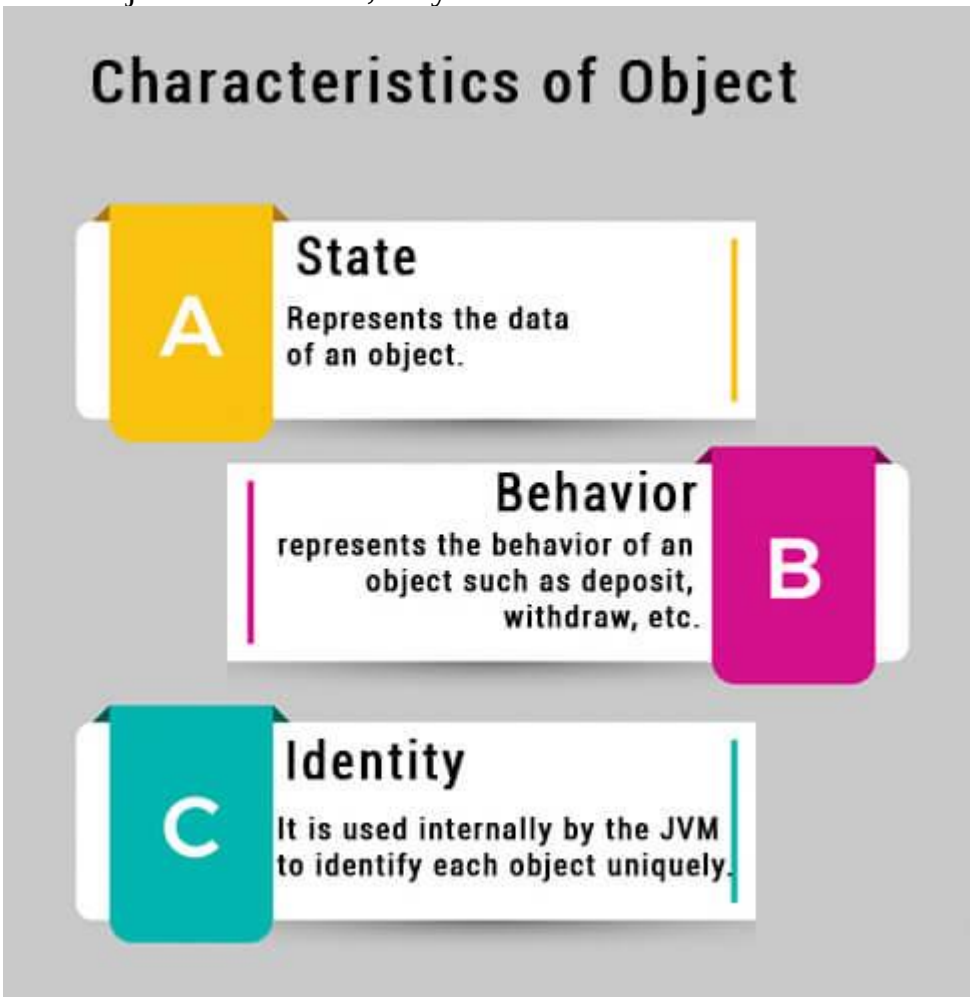
Volvo

Audi

Toyota

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.



An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

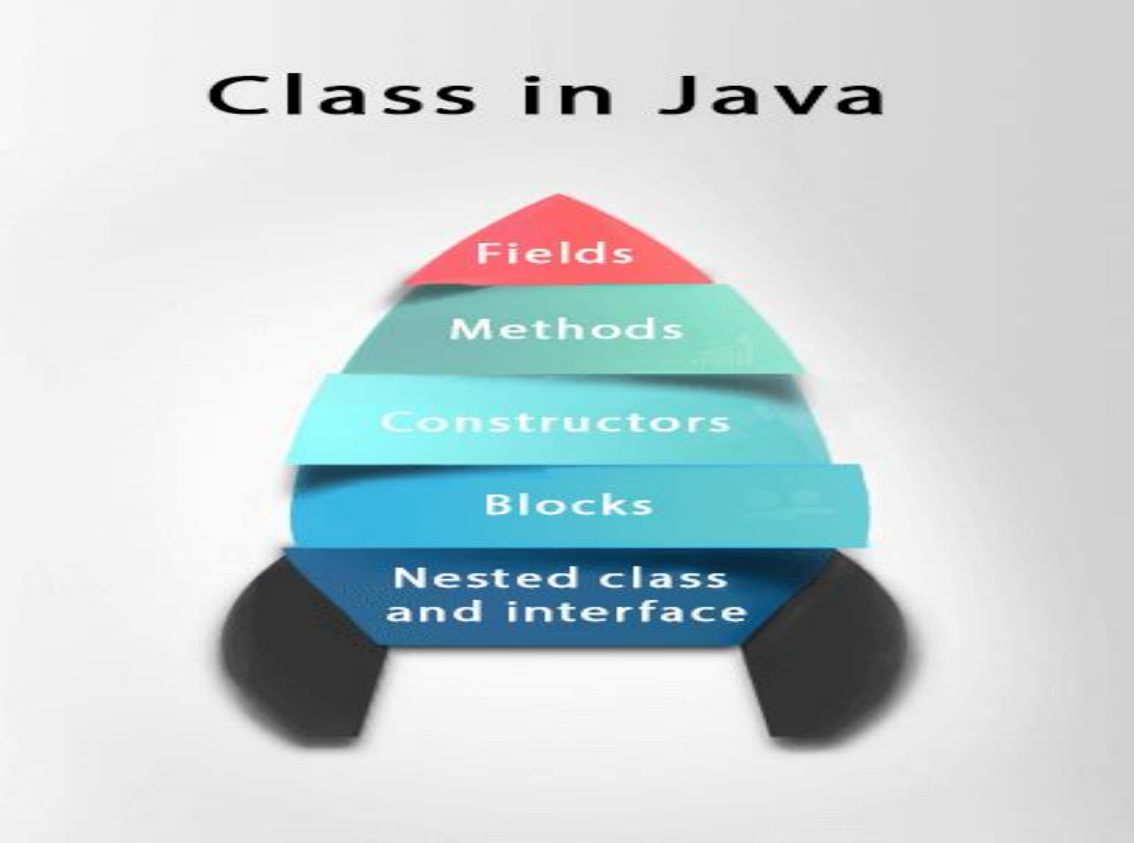
- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**



Syntax to declare a class:

```
1. class <class_name>{
2.     field;
3.     method;
4. }
```

What is Object in Java?

OBJECT is an instance of a class. An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. For example color name, table, bag, barking. When you send a message to an object, you are asking the object to invoke or execute one of its methods as defined in the class.

From a programming point of view, an object can include a data structure, a variable, or a function. It has a memory location allocated. The object is designed as class hierarchies.

Syntax

```
ClassName ReferenceVariable = new ClassName();
```

What is the Difference Between Object & Class?

A **class** is a **blueprint or prototype** that defines the variables and the methods (functions) common to all objects of a certain kind.

An **object** is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life.

No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects.**
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once.**
6)	Object **allocates memory when it is created.**	Class **doesn't allocated memory when it is created.**
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.

Let's see some real life example of class and object in java to understand the difference well:

Class: Human **Object:** Man, Woman

Class: Fruit **Object:** Apple, Banana, Mango, Guava wtc.

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

---

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object and Class Example: main within the class

In this example, we have created a Car class which has two data members id and name. We are creating the object of the Car class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

```
//Java Program to illustrate how to define a class and fields
//Defining a Car class.
class Car {
 //defining fields
 int id;//field or data member or instance variable
 String name,model;  float price;
 //creating main method inside the Car class
 public static void main(String args[]){
  //Creating an object or instance
  Car c1=new Car();//creating an object of Student
  c1.id=500;
  c1.name="BMW";
  c1.model="ASTRA";
  c1.price=240000;

  //Printing values of the object
  System.out.println(c1.id);//accessing member through reference variable
  System.out.println(c1.name);
  System.out.println(c1.model);
  System.out.println(c1.price);

  Car C2=new Car();//creating an object of Student

  C2.id=9500;
  C2.name="Tesla";
```

```
C2.model="Cavier";
C2.price=50000;

//Printing values of the object
System.out.println(C2.id);//accessing member through reference variable
System.out.println(C2.name);
System.out.println(C2.model);
System.out.println(C2.price);

 }
}
```

Object and Class Example: main outside the class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

File: TestStudent1.java

```
1. //Java Program to demonstrate having the main method in
2. //another class
3. //Creating Student class.
4. class Student{
5.  int id;
6.  String name;
7. }
```

```
8.  //Creating another class TestStudent1 which contains the main method
9.  class TestStudent1{
10. public static void main(String args[]){
11.  Student s1=new Student();
12.  System.out.println(s1.id);
13.  System.out.println(s1.name);
14. }
15.}
```

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

File: TestStudent2.java

```
1.  class Student{
2.   int id;
3.   String name;
4.  }
5.  class TestStudent2{
6.   public static void main(String args[]){
7.    Student s1=new Student();
8.    s1.id=101;
9.    s1.name="Sonoo";
10.  System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12.}
```

We can also create multiple objects and store information in it through reference variable.

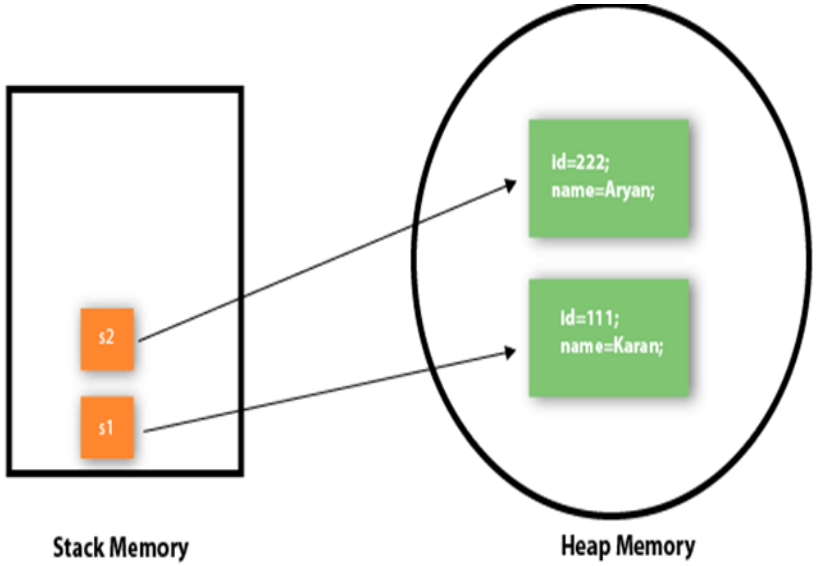
File: TestStudent.java

```
1. class Student{
2.  int id;
3.  String name;
4. }
5. class TestStudent{
6.  public static void main(String args[]){
7.   //Creating objects
8.   Student s1=new Student();
9.   Student s2=new Student();
10.  //Initializing objects
11.  s1.id=101;
12.  s1.name="Sonoo";
13.  s2.id=102;
14.  s2.name="Amit";
15.  //Printing data
16.  System.out.println(s1.id+" "+s1.name);
17.  System.out.println(s2.id+" "+s2.name);
18. }
19.}
```

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation()

method.



As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

File: TestStudent4.java

```
1.  class Student{
2.   int rollno;
3.   String name;
4.   void insertRecord(int r, String n){
5.    rollno=r;
6.    name=n;
7.   }
8.   void displayInformation(){System.out.println(rollno+" "+name);}
9.  }
10.class TestStudent4{
11. public static void main(String args[]){
12.  Student s1=new Student();
13.  Student s2=new Student();
14.  s1.insertRecord(111,"SAYMA");
15.  s2.insertRecord(222,"AZAN");
16.  s1.displayInformation();
17.  s2.displayInformation();
18. }
19.}
```