

TEACHMEBRO.COM

OBJECT ORIENTED PROGRAMMING SYSTEM

COMPLETE OOPS FUNDAMENTAL TUTORIAL
BY ALTAF SHAIKH



Object Oriented Programming (OOPs)

OBJECT ORIENTED PROGRAMMING SYSTEM (OOPs) brings together properties and their associated actions in a single location which makes it easier to understand how a program works.

- It allows users to create the objects that they want and then, create methods to handle those objects.
- The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.
- Object Oriented Programming popularly known as OOP, is used in a modern programming language like Java, Python, C++, etc.



Comparison of OOPS with other programming styles with the help of an Example

Let's understand with an example how OOPs is different than other programming approaches.

Programming languages can be classified into 3 primary types:

1. **Unstructured Programming Languages:** The most primitive of all programming languages having sequentially flow of control. The code is repeated throughout the program. Examples- BASIC, COBOL, and FORTRAN.
2. **Structured Programming Languages:** Has a non-sequentially flow of control. The use of functions allows for re-use of code. Examples- Java, Python, C++
3. **Object Oriented Programming:** Combines Data & Action Together. Examples- Java, Python, C++

Let's understand these 3 types with an example:

Suppose you want to create a Banking Software with functions like

1. Deposit
2. Withdraw
3. Show Balance

Unstructured Programming Languages

The earliest of all programming languages were unstructured programming languages. A very elementary code of banking application in unstructured Programming language will have two variables of one account number and another for account balance

```
int account_number=20;
```

```
int account_balance=100;
```

OOPS Fundamentals

Suppose a deposit of 100 dollars is made.

```
account_balance=account_balance+100
```

Next, you need to display account balance.

```
printf("Account Number=%d,account_number)\nprintf("Account Balance=%d,account_balance)
```

Now the amount of 50 dollars is withdrawn.

```
account_balance=account_balance-50
```

Again, you need to display the account balance.

```
printf("Account Number=%d,account_number)\nprintf("Account Balance=%d,account_balance)
```

The diagram shows a sequence of code blocks for an account simulation. A red box at the top right contains the text "Unstructured Programming" and "same code is repeated". Three green dotted arrows point from this box to three identical blocks of code, each representing a transaction (deposit, display, withdrawal, display). The code blocks are as follows:

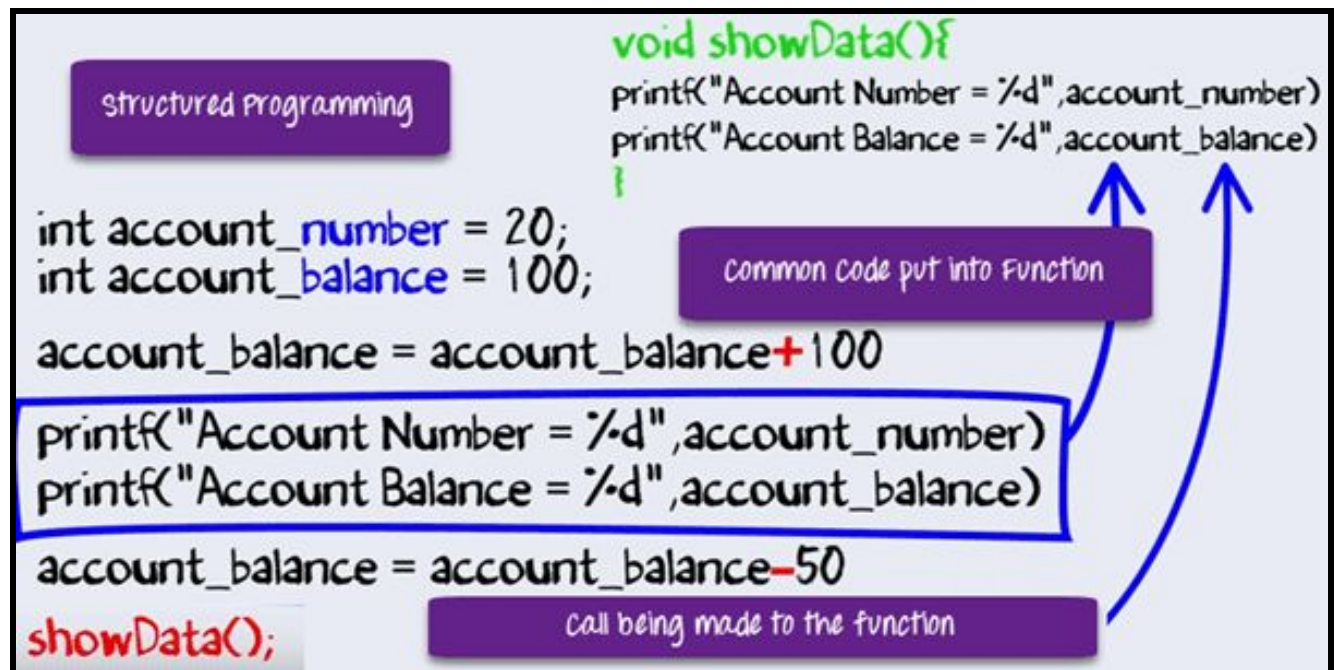
```
int account_number = 20;\nint account_balance = 100;\n\naccount_balance = account_balance+100\nprintf("Account Number = %d",account_number)\nprintf("Account Balance = %d",account_balance)\n\naccount_balance = account_balance-50\nprintf("Account Number = %d",account_number)\nprintf("Account Balance = %d",account_balance)\n\naccount_balance = account_balance-10\nprintf("Account Number = %d",account_number)\nprintf("Account Balance = %d",account_balance)
```


OOPS Fundamentals

For any further deposit or withdrawal operation – you will code repeat the same lines again and again.

Structured Programming

With the arrival of Structured programming repeated lines on the code were put into structures such as functions or methods. Whenever needed, a simple call to the function is made.



Object-Oriented Programming

In our program, we are dealing with data or performing specific operations on the data.

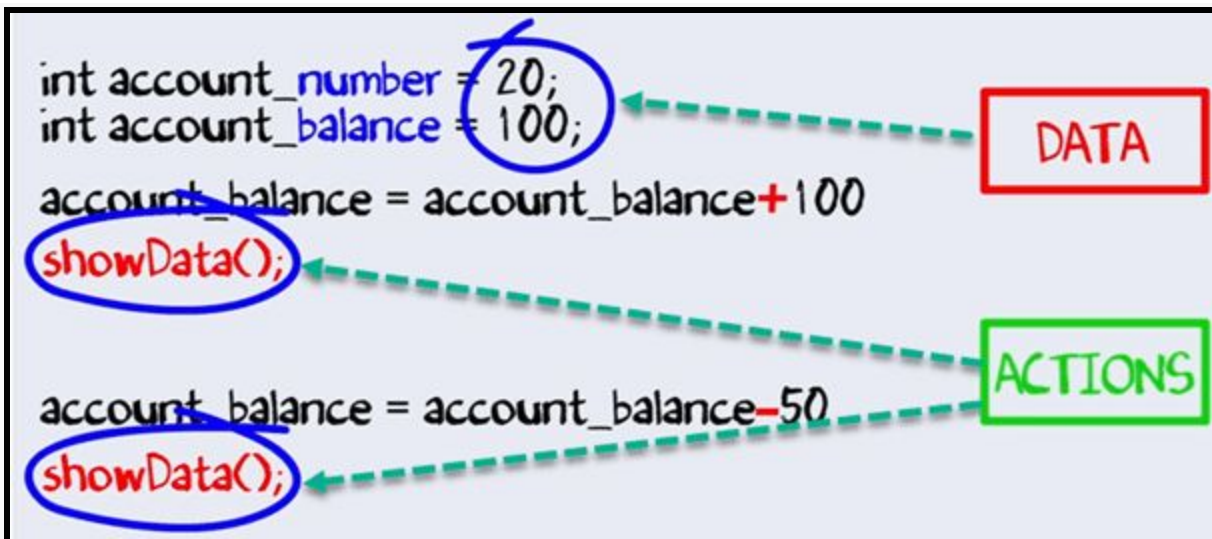
In fact, having data and performing certain operations on that data is a very basic characteristic in any software program.

OOPS Fundamentals

Experts in Software Programming thought of combining the Data and Operations. Therefore, the birth of Object Oriented Programming which is commonly called OOPS.

The same code in OOPS will have the same data and some action performed on that data.

```
Class Account{  
    int account_number;  
    int account_balance;  
  
    public void showdata(){  
        system.out.println("Account Number"+account_number)  
        system.outprintln("Account Balance"+ account_balance)  
    }  
}
```



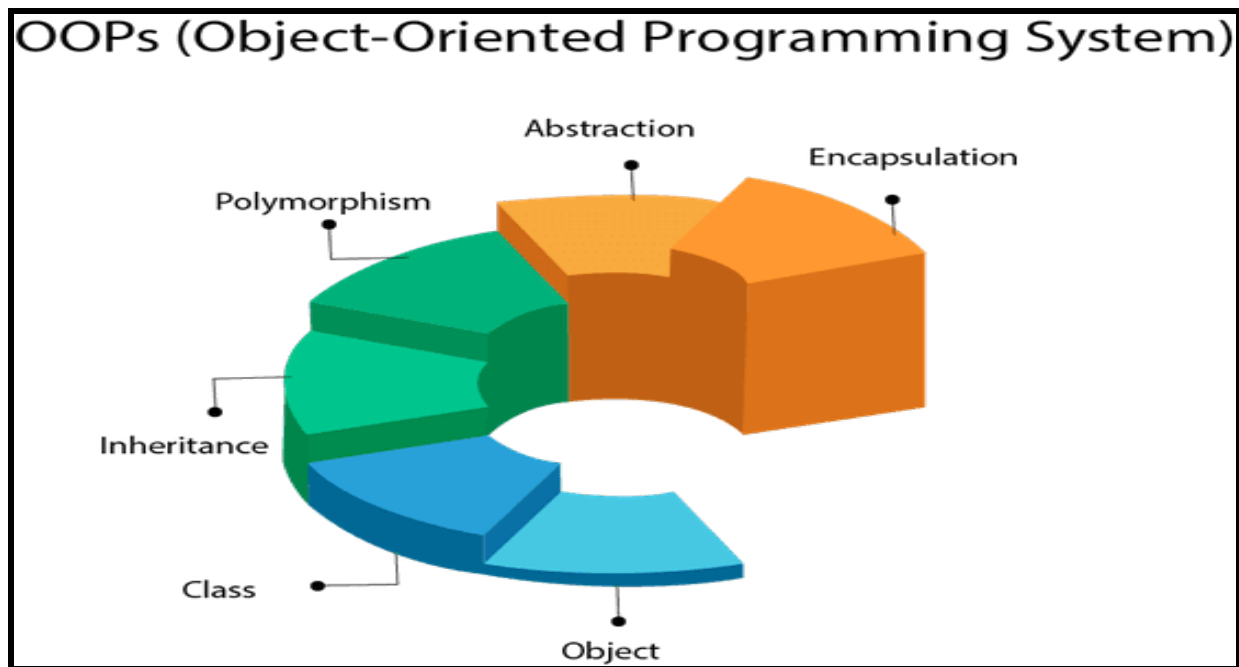
Basic Concepts of OOPs

By combining data and action, we will get many advantages over structural programming viz,

- Object
- Class
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

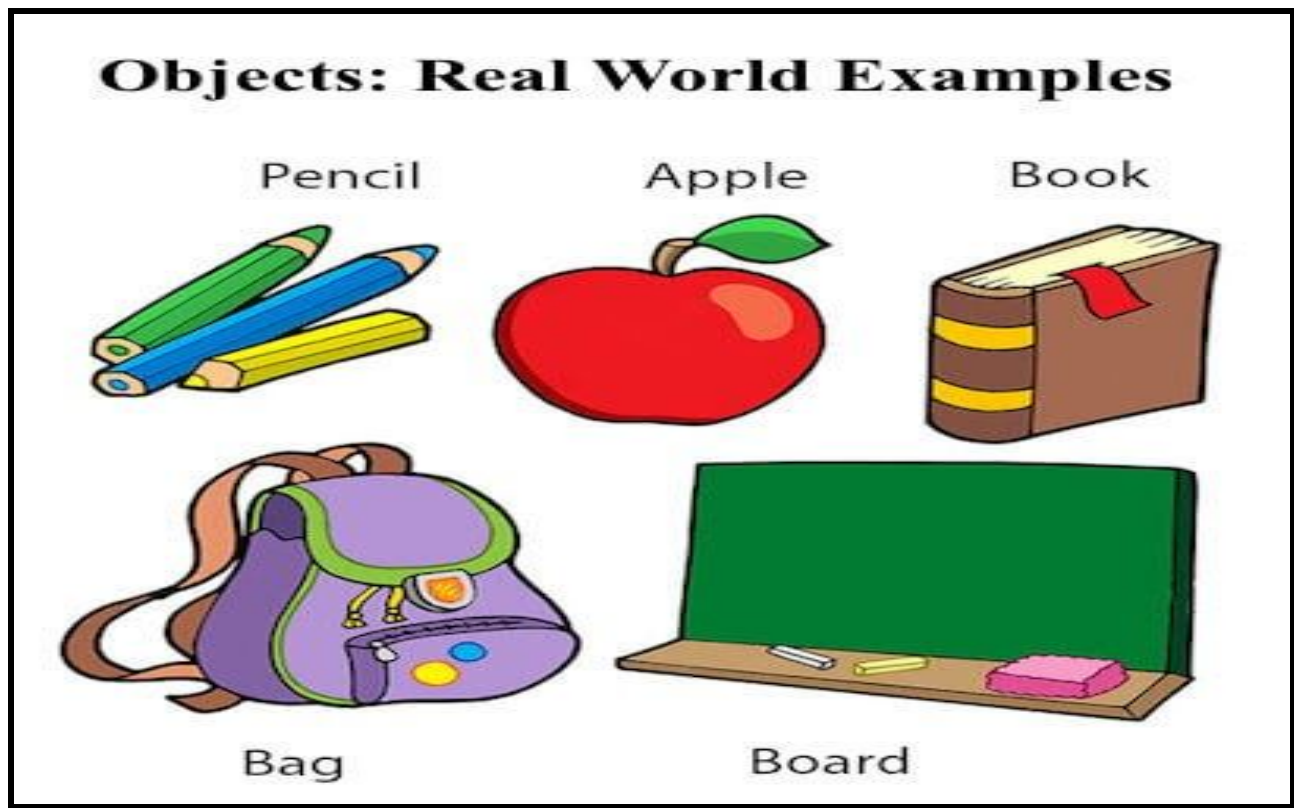
- Coupling
- Cohesion
- Association
- Aggregation
- Composition



Object:

An object can be defined as an instance of a class, and there can be multiple instances of a class in a program. An Object contains both the data or variables (which knows something) and the function or method (which does something). For example - chair, bike, marker, pen, table, car, etc. An object consists of:

- **State:** It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.



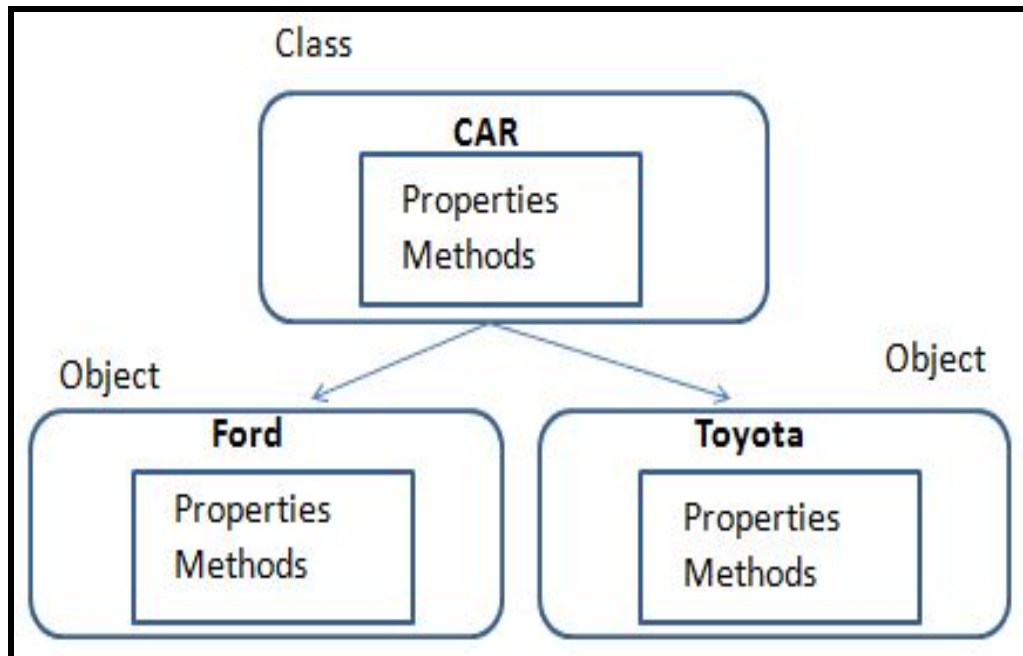
Class:

A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

- **Modifiers:** A class can be public or has default access (In Java, methods and data members of a class/interface can have one of the following four access specifiers viz, private, public, default, protected. However, we cannot declare class/interface with private or protected access specifiers. Otherwise, program fails in compilation. Remember, nested interfaces and classes can have all access specifiers).
- **Class name:** The name should begin with a initial letter (capitalized by convention).
- **Superclass**(if any): The name of the class's parent (superclass), if any, preceded by the keyword *extends*. A class can only extend (subclass) one parent.

OOPS Fundamentals

- **Interfaces**(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword *implements*. A class can implement more than one interface.
- **Body**: The class body surrounded by braces, { }.



Class Creation:

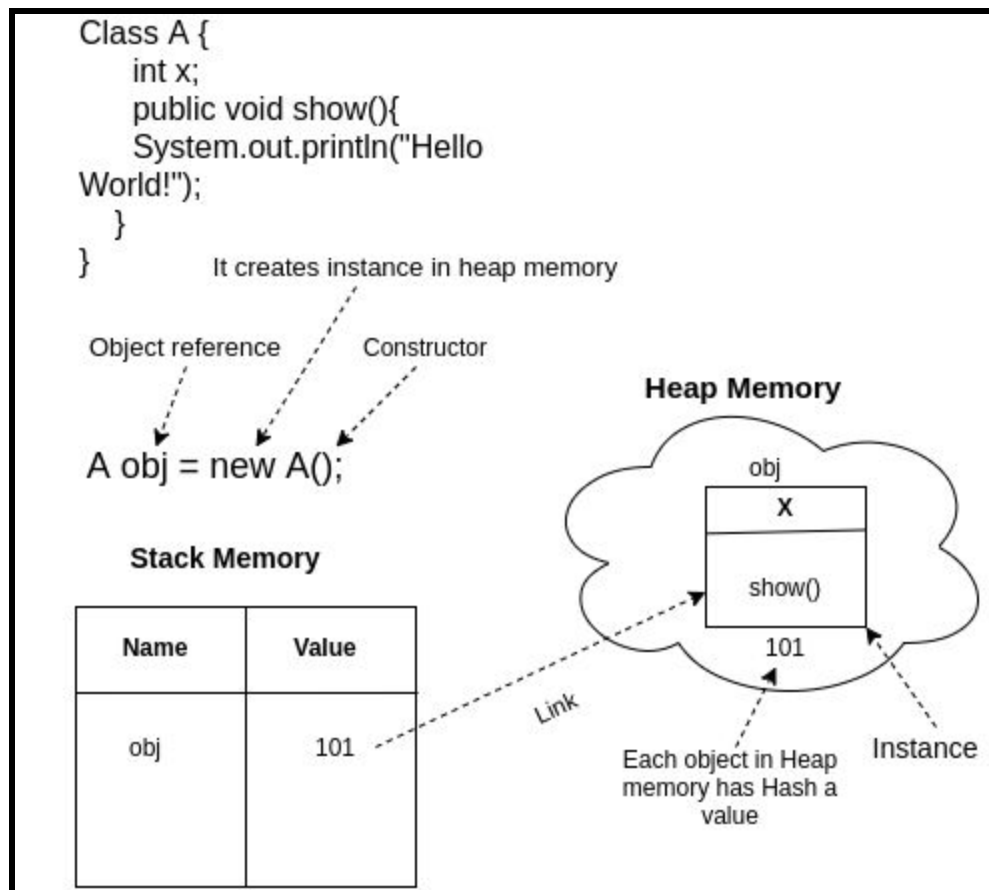
```
# Java program to demonstrate defining
# a class and class variable and method

Class Phone{
    string model;

    public void call(){
    }
}
```

Object creation:

- Object is stored in Heap memory
- New keyword is used to allocate memory inside Heap memory
- Constructor is used to define the size of an object
- Reference variable is stored in stack memory
- Dot(.) operator is used to access the variables and methods of the class



Garbage collection:

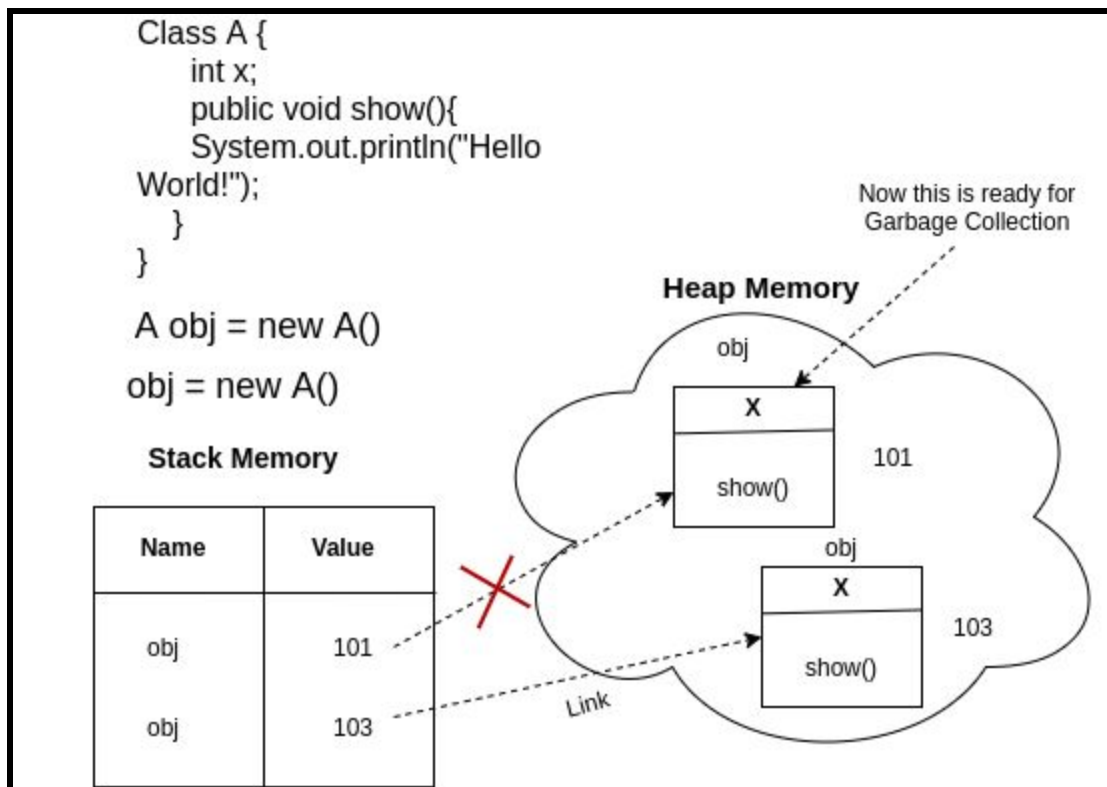
- In Java, the programmer needs not to care for all those objects which are no longer in use. Garbage collector destroys these objects.
- Main objective of Garbage Collector is to free heap memory by destroying unreachable objects.

OOPS Fundamentals

- Unreachable objects : An object is said to be unreachable iff it doesn't contain any reference to it. Also note that objects which are part of island of isolation are also unreachable.

Ways for requesting JVM to run Garbage Collector

- Once we made an object eligible for garbage collection, it may not be destroyed immediately by the garbage collector. Whenever JVM runs the Garbage Collector program, then only the object will be destroyed. But when JVM runs Garbage Collector, we can not expect.
- We can also request JVM to run Garbage Collector. There are two ways to do it :
 - **Using System.gc() method** : System class contains static method gc() for requesting JVM to run Garbage Collector.
 - **Using Runtime.getRuntime().gc() method** : Runtime class allows the application to interface with the JVM in which the application is running. Hence by using its gc() method, we can request JVM to run Garbage Collector.

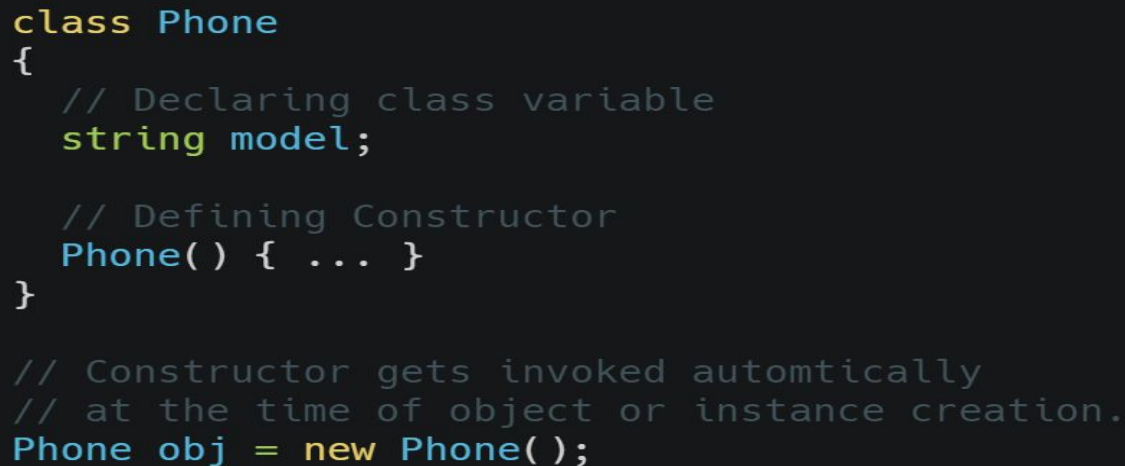


OOPS Fundamentals

By explicitly assigning the 'null' value to the obj. Example: obj=null in the above diagram will also destroy the object and obj gets ready for the Garbage Collection.

Constructor:

Constructors are used for initializing the object's state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of **Object creation**. Every time an object is created using the **new()** keyword, at least one constructor is called to assign initial values to the data members of the same class. It is called **constructor** because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because the java compiler creates a default constructor if your class doesn't have any.



```
class Phone
{
    // Declaring class variable
    string model;

    // Defining Constructor
    Phone() { ... }
}

// Constructor gets invoked automatically
// at the time of object or instance creation.
Phone obj = new Phone();
```

Rules for writing constructor:

OOPS Fundamentals

```
class Phone
{
    // Constructor(s) of a class must have the same name as the class name in which it resides.
    Phone() { ... } ✓

    // A constructor in Java cannot be abstract, final, static, native and synchronized.
    abstract Phone() { .. }
    final Phone() { .. }
    static Phone() { .. }
    native Phone() { .. }
    synchronized Phone() { .. }

    // Access modifiers can be used in constructor declaration to control its access
    // i.e which other class can call the constructor.
    public Phone() { .. }
    private Phone() { .. }
    protected Phone() { .. }
    default Phone() { .. }
}
```

Types of constructor:

- No-argument constructor:

```
// No-argument constructor or Default constructor //
```

// Case 1: If we don't define a constructor in a class,
// then compiler creates default constructor(with no arguments) for the class.

```
class Phone{ String model; }

Phone obj = new Phone(); //Compiler creates default constructor by itself.
```

// Case 2: If we write a constructor with arguments or no-arguments,
// then the compiler does not create a default constructor.

```
class Phone{
    String model;
    // Defining no-arg constructor.
    Phone()
    {
        this.model = "Samsung"; // Assign certain value to class variable
        System.out.println("Constructor Called"); // Or print any stmt
        // Or leave body blank
    }
}

Phone obj = new Phone(); //Compiler calls defined constructor.
```

- **Parameterized Constructor:**

```
// Parameterized Constructor //
```

```
class Phone{
    String model;
    // Defining parameterized constructor.
    Phone(String model)
    {
        this.model = model; // Assign value to class variable
        System.out.println("Constructor Called"); // Or print any stmt
        // Or leave body blank
    }
}

Phone obj = new Phone("Samsung"); //Required arg are passed while obj creation itself..
```

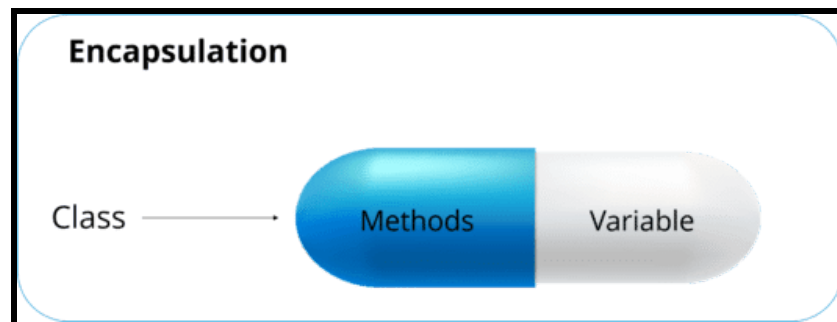
#Note:

- 1) If any assignment is not made to any class variable in both the constructor, default values are provided to class variables of the object like 0, null, etc depending on the data type.

- 2) There are no “return value” statements in constructor, but constructor returns current class instance. We can write ‘return’ inside a constructor. Constructor(s) do not return any type while method(s) have the return type or void if does not return any value.
- 3) Constructor Overloading is possible.

Encapsulation:

Encapsulation in Java is a process of wrapping code and data together into a single unit. The best way to understand encapsulation is to look at the example of a medical capsule, where the drug is always safe inside the capsule. Similarly, through encapsulation the methods and variables of a class are well hidden and safe.



We can achieve encapsulation in Java by:

- Declaring the variables of a class as private.
- Providing public setter and getter methods to modify and view the variables values.

Often encapsulation is misunderstood with Abstraction.

- Encapsulation is more about **"How"** to achieve a functionality.
- Abstraction is more about **"What"** a class can do.

A simple example to understand this difference is a mobile phone. Where the complex logic in the circuit board is encapsulated in a touch screen, and the interface is provided to abstract it out.

Simple Java program to demonstrate encapsulation:

```
class Phone
{
    // Declaring class variable as private
    private String model;

    // Defining getter method
    public String getModel() { return this.model; }

    // Defining setter method
    public void setModel(String model) { this.model = model; }
}

Phone obj = new Phone();

// Not possible as class variable is private
obj.model = "Samsung";
System.out.println("Model Name " + obj.model);

// Possible as getter and setter methods are public
obj.setModel("Samsung");
System.out.println("Model Name " + obj.getModel());
```



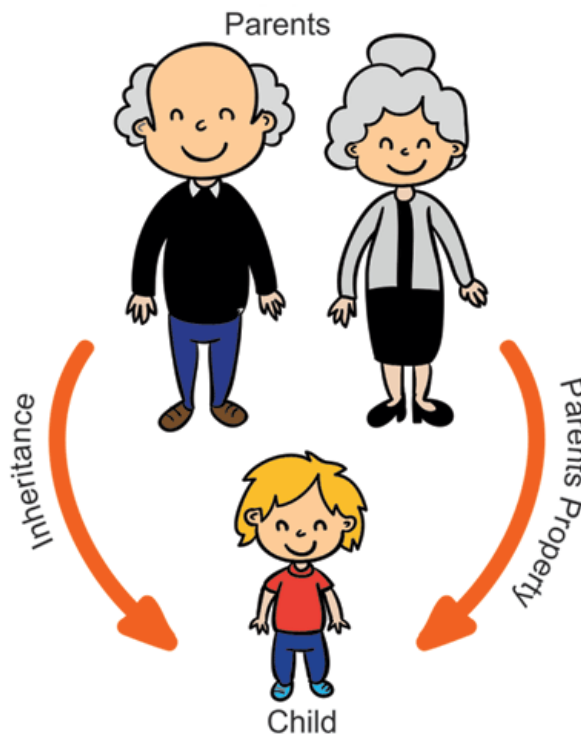
Advantages of Encapsulation:

- The user will have no idea about the inner implementation of the class.
- We can make the variables of the class as read-only or write-only depending on our requirement.
- Encapsulation also improves re-usability and is easy to change with new requirements.
- Encapsulated code is easy to test for unit testing.

Inheritance:

In OOP, computer programs are designed in such a way where everything is an object that interacts with one another. Inheritance is one such concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes.

Inheritance represents the IS-A relationship which is also known as a **parent-child relationship**.



Base Class: The class whose features are inherited is known as base class(or a super class or a parent class).

Let below snippet, be considered as base class of our code:


```
// Base Class //  
  
class Phone  
{  
    // Declaring class variable  
    String model;  
    // Defining Constructor  
    Phone(String model) { this.model = model; }  
    // Defining Methods  
    public void call() { .. }  
    public String info()  
    {  
        return("Model Name: " + this.model + "\n");  
    }  
}
```

Child Class: The class that inherits the other class is known as child class(or a derived class, extended class, or sub class). The subclass can add its own fields and methods in addition to the superclass fields and methods. The keyword used for inheritance is *extends*.

Let below snippet, be considered as child class of our code:

```
// Derived Class //

class Mobile extends Phone
{
    // Declaring more class variable
    String provider;

    // Defining Constructor
    Mobile(String model, String provider)
    {
        // invoking base-class(Phone) constructor
        super(model);
        this.provider = provider;
    }

    // Defining Methods
    // Adding more methods required
    public void sms() { .. }

    // Overriding to print more info
    @Override
    public String info()
    {
        return(super.info()+ "Provider: "+this.provider);
    }
}
```

super keyword : Reference variable which is used to refer to an immediate parent class object. Whenever you create an instance of a subclass, an instance of the parent class is created implicitly which is referred to by a super reference variable.

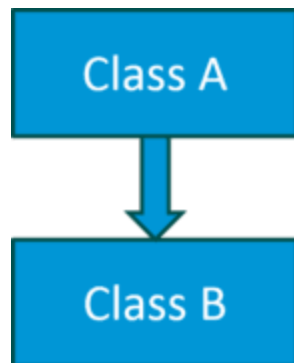
Driver class which lets us see the output of our code:

```
// Driver class //
```

```
public class Test
{
    public static void main(String args[])
    {
        Mobile mb = new Mobile("Samsung","Idea");
        System.out.println(mb.info());
        // Output: //
        // Model Name: Samsung
        // Provider: Idea
    }
}
```

Types of inheritance:

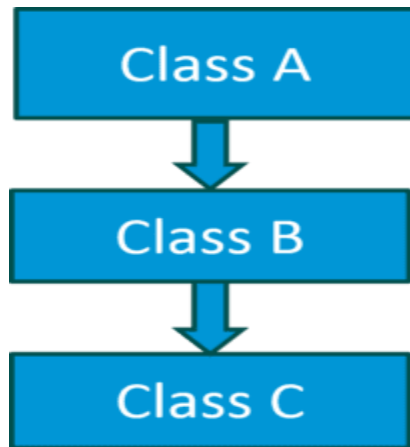
1. **Single Inheritance:** In single inheritance, subclasses inherit the features of one superclass.



Syntax:

```
Class A { ... }
Class B extends A { ... }
```

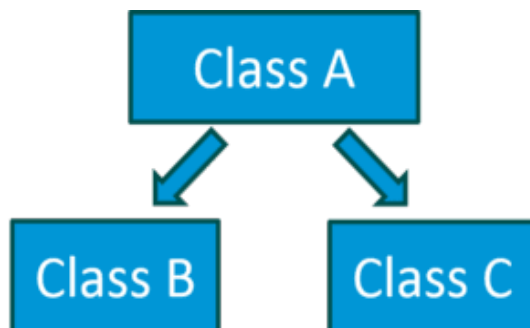
2. **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.



Syntax:

```
class A { ... }  
class B extends A { ... }  
class C extends B { ... }
```

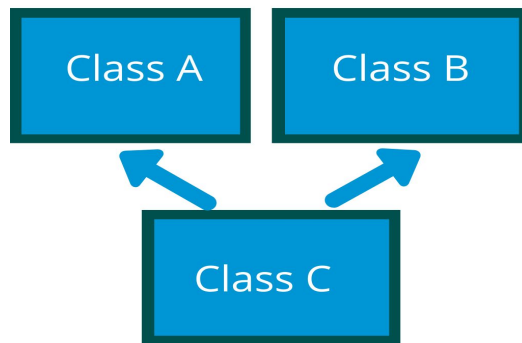
3. **Hierarchical Inheritance** : In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.



Syntax:

```
class A { ... }  
class B extends A { ... }  
class C extends A { ... }
```

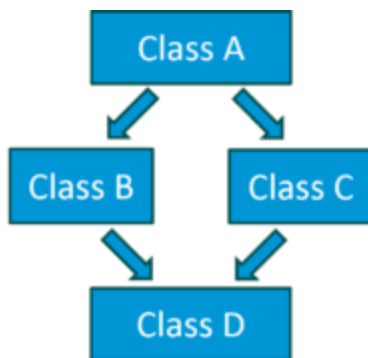
4. **Multiple Inheritance (Not supported in Java, but is achievable through Interfaces)** : In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes.



Syntax:

```
interface A { ... }
interface B { ... }
interface C extends A,B { ... }
```

5. **Hybrid Inheritance (Not supported in Java, but is achievable through Interfaces)** : Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance.



Syntax:

```
public class A { ... }
public interface B { ... }
public interface C { ... }
public class D extends A implements B,C { ... }
```

Polymorphism:

Polymorphism refers to the ability of a variable, object or function to take on multiple forms. Polymorphism means taking many forms, where '**poly**' means many and '**morph**' means forms. For example, in English, the verb **run** has a different meaning if you use it with a laptop, a foot race, and business. Here, we understand the meaning of *run* based on the other words used along with it. The same is also applied to Polymorphism. In other words, polymorphism allows you to define one interface or method and have multiple implementations.



Polymorphism in Java is of two types:

- **Runtime polymorphism:**

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

- Method Overriding

If subclass (child class) has the same method as declared in the parent class.

```
// Method Overriding //

class Phone
{
    // Method to be overridden by child class
    void info() { System.out.println("Parent: Phone"); }
}

class Mobile extends Phone
{
    // The name and parameter of the method are the same
    // There is IS-A relationship between the classes
    // Method overriding is possible
    void info() { System.out.println("Child: Mobile"); }
}

class Telephone extends Phone
{
    // Method overriding is possible
    // Everything is satisfied
    void info() { System.out.println("Child: Telephone"); }
}
```

#Note: Static method cannot be overridden. It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area. That is why, java's main method cannot be overridden as it is also a static method.

- **Compile time polymorphism**

In Java, compile time polymorphism refers to a process in which a call to an overloaded method is resolved at compile time rather than at run time. It is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

- Method overloading:

```
// Method overloading //  
  
class DivideTask {  
    // Method with 2 parameter  
    static int Divide(int a, int b)  
    {  
        return a / b;  
    }  
  
    // Method with same name & no. of arg but different datatype  
    static double Divide(double a, double b)  
    {  
        return a / b;  
    }  
  
    // Method with the same name & datatype but 3 parameter  
    static int Divide(int a, int b, int c)  
    {  
        return a / b / c;  
    }  
}
```

Abstraction:

Abstraction refers to the quality of dealing with ideas rather than events. It basically deals with **hiding** the details and showing the essential things to the user. Whenever we get a call, we get an option to either pick it up or just reject it. But in reality, there is a lot of code that runs in the background. So you don't know the internal processing of how a call is generated, that's the beauty of

OOPS Fundamentals

abstraction. Therefore, abstraction helps to reduce complexity. You can achieve abstraction in two ways:



a) **Abstract Class:**

Abstract class in Java contains the '**abstract**' keyword. Now what does the abstract keyword mean? If a class is declared abstract, it cannot be instantiated, which means you cannot create an object of an abstract class. Also, an abstract class can contain abstract as well as concrete methods.

Note: You can achieve 0-100% abstraction using abstract class.

To use an abstract class, you have to inherit it from another class where you have to provide implementations for the abstract methods there itself, else it will also become an abstract class.

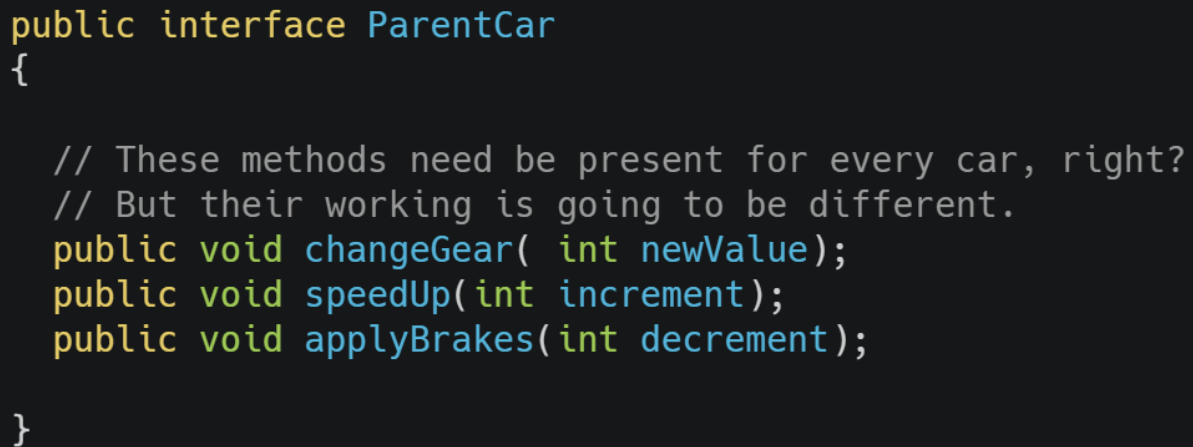
b) **Interface**

OOPS Fundamentals

Interface in Java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. Along with abstraction, interface also helps to achieve multiple inheritance in Java.

Note: You can achieve 100% abstraction using interfaces.

So an interface basically is a group of related methods with empty bodies.



```
public interface ParentCar
{
    // These methods need be present for every car, right?
    // But their working is going to be different.
    public void changeGear( int newValue);
    public void speedUp(int increment);
    public void applyBrakes(int decrement);
}
```




```
public class Audi implements ParentCar
{
    int speed=0;
    int gear=1;

    // Providing functionalities to the different methods declared in interface class.
    // Implementing an interface allows a class to become more formal
    // about the behavior it promises to provide.
    public void changeGear(int value) { gear=value; }
    public void speedUp( int increment) { speed=speed+increment; }
    public void applyBrakes(int decrement) { speed=speed-decrement; }
    void printStates(){ System.out.println("speed:"+speed+"gear:"+gear); }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Audi A6= new Audi();
        A6.speedUp(50);
        A6.printStates();
        A6.changeGear(4);
        A6.SpeedUp(100);
        A6.printStates();
    }
}
```

Advantages of OOPS:

- OOP offers easy to understand and a clear modular structure for programs.
- Objects created for Object-Oriented Programs can be reused in other programs. Thus it saves significant development costs.
- Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.
- It also enhances program modularity because every object exists independently.

Join TeachMeBro Community



Follow our [Linkedin](#) Page



Join Our [Telegram](#) Group



Visit our [Website](#) for more tutorials



Subscribe us on [Youtube](#)



Follow us on [Instagram](#)

Thank You