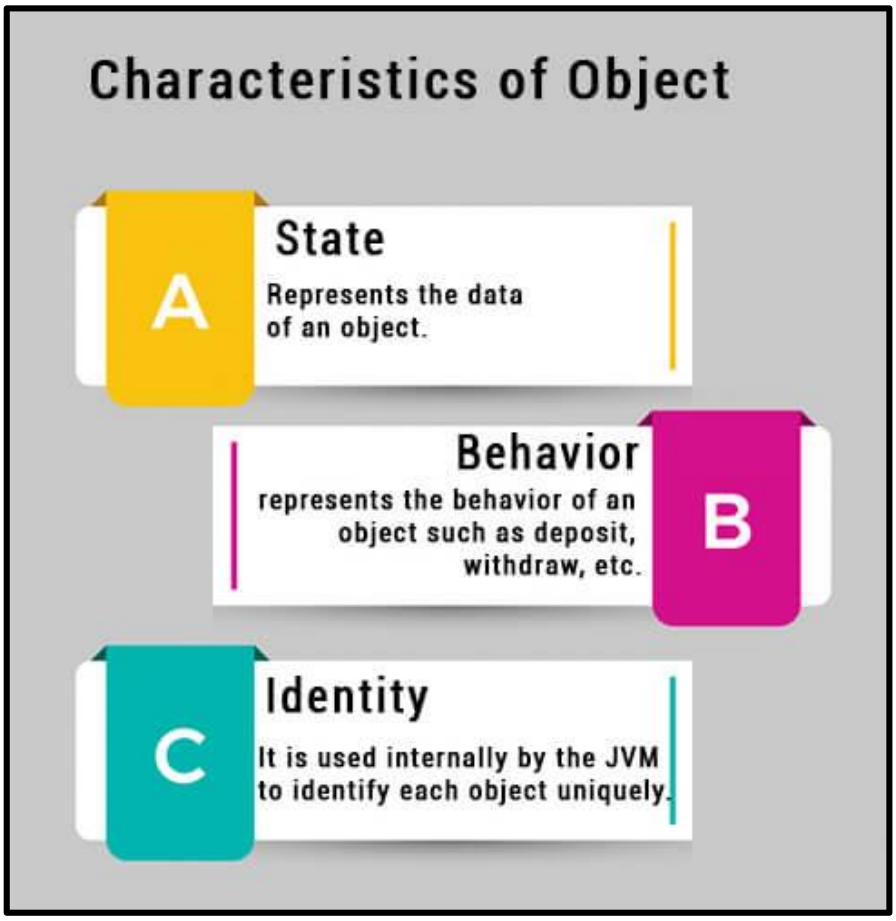
OOP (Object Oriented Programming)

Object Oriented Programming: OOP is a methodology or paradigm to design a program using **Classes** and **Objects**.

Objects & Classes:

Objects: Real world entities that has their own properties and behaviours.

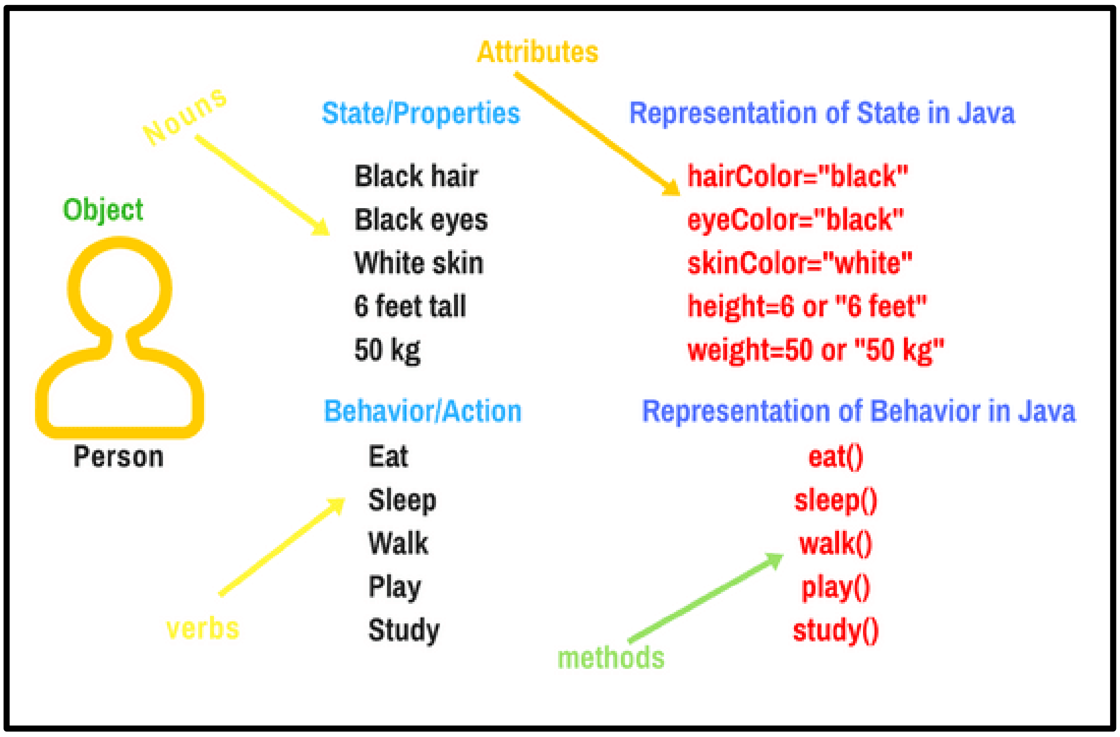


Characteristics of Object

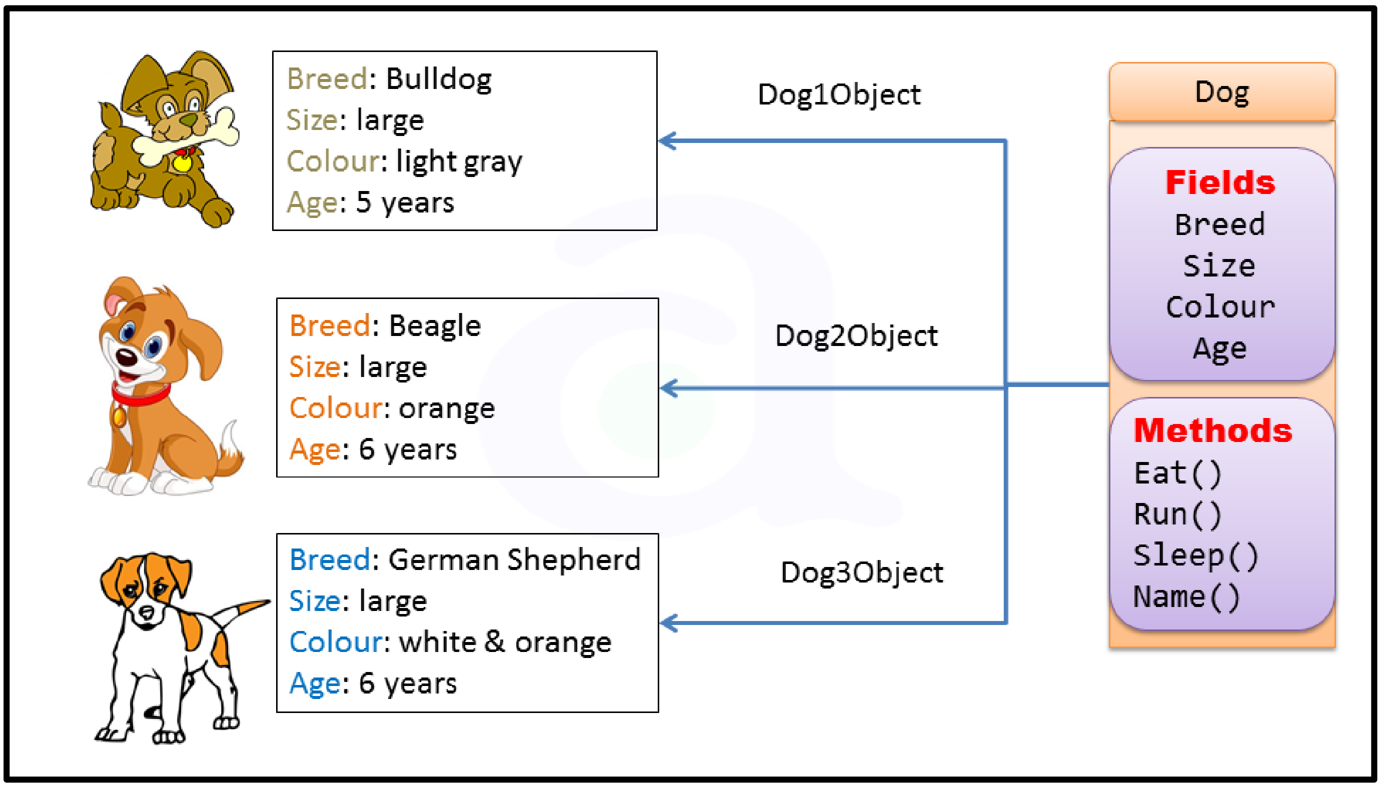
A State
Represents the data of an object.

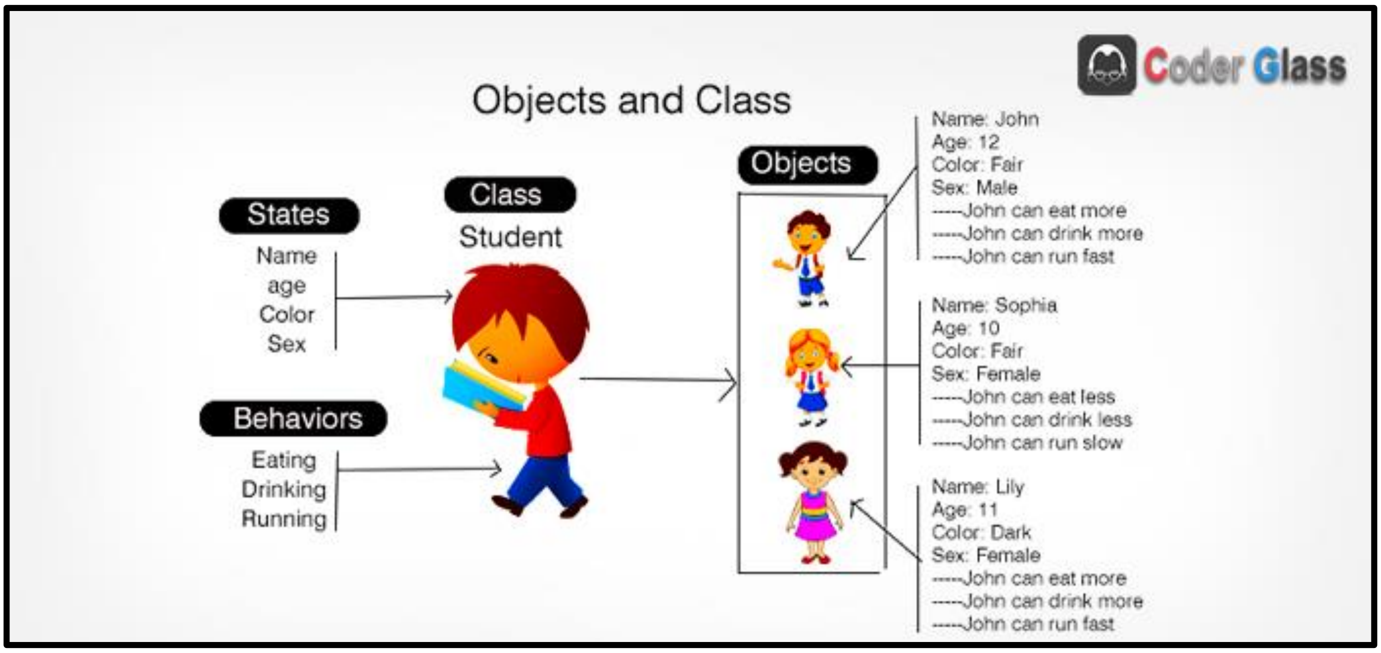
B Behavior
represents the behavior of an object such as deposit, withdraw, etc.

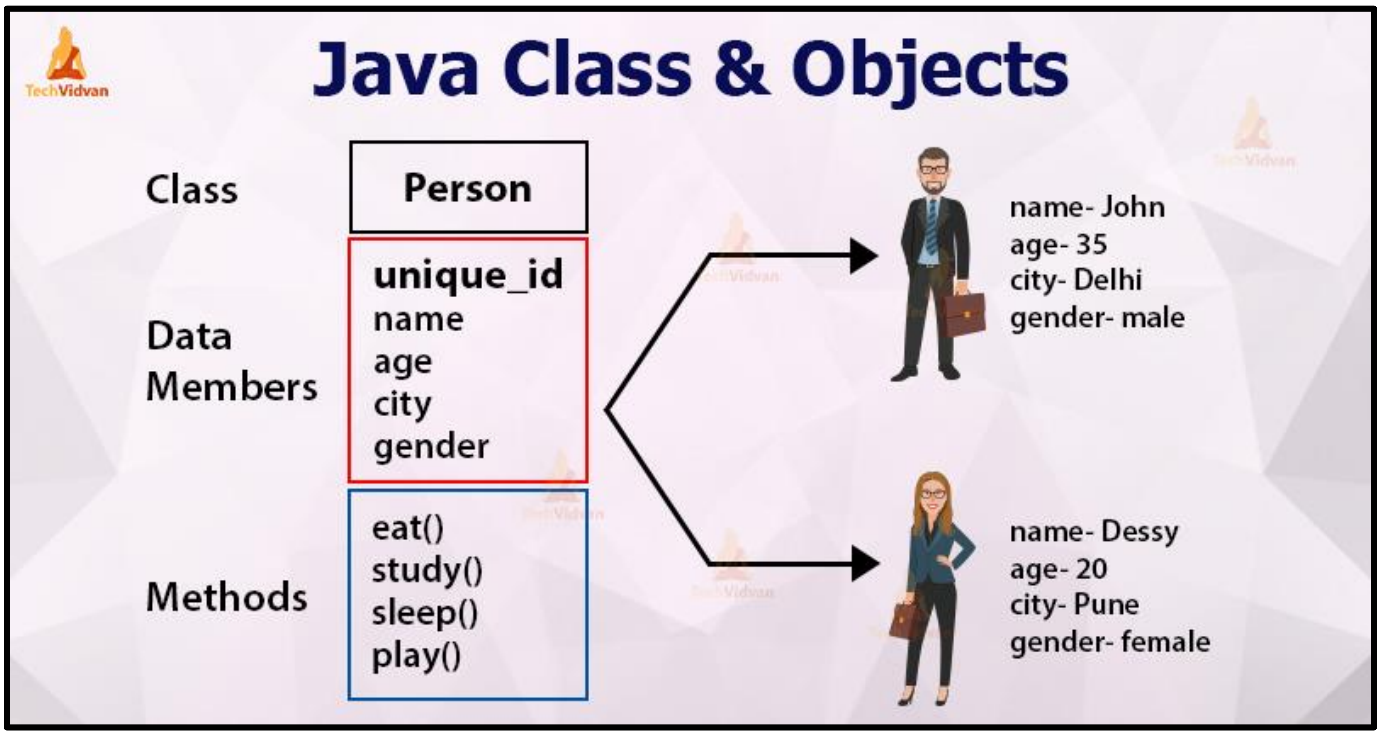
C Identity
It is used internally by the JVM to identify each object uniquely.



Class: Blueprint from which an objects properties and behaviours are decided.





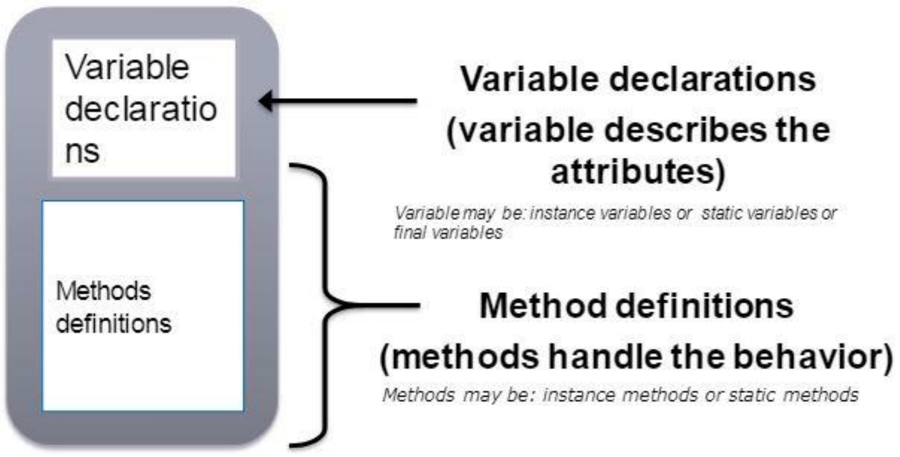


Difference between Object Oriented Programming & Procedure Oriented Programming

OOP	POP
• Object oriented.	• Structure oriented.
• Program is divided into objects.	• Program is divided into functions.
• Bottom-up approach.	• Top-down approach.
• Inheritance property is used.	• Inheritance is not allowed.
• It uses access specifier.	• It doesn’t use access specifier.
• Encapsulation is used to hide the data.	• No data hiding.
• More secure	• Less secure
• Objects can move & communicate with	• Data can move freely from function
• Each other through member functions	• To function in the system.
• Supports Overloading	• Do not support Overloading
• **Example: C++, Java.**	• **Example: C, Pascal.**

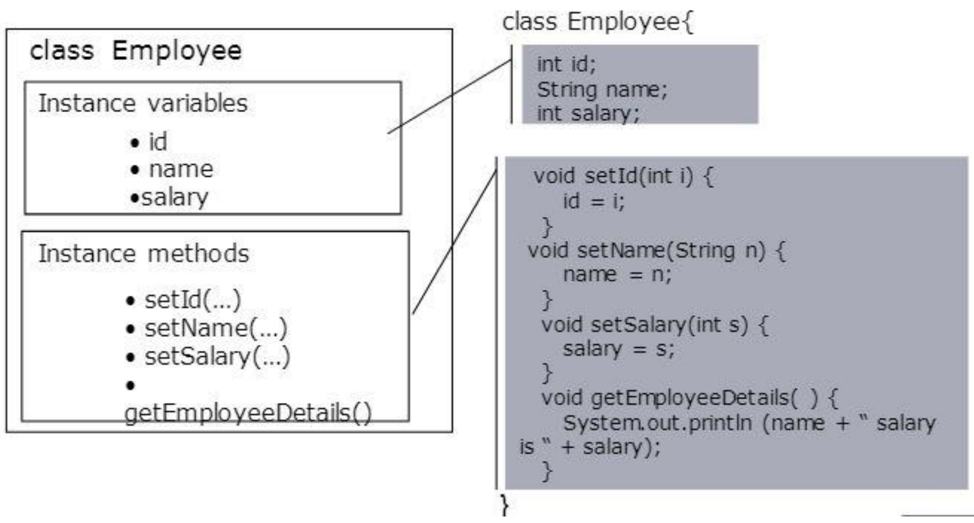
Classes

A class contains variable declarations and method definitions


Variable declaratio ns
Methods definitions
Variable declarations (variable describes the attributes)
Variable may be: instance variables or static variables or final variables
Method definitions (methods handle the behavior)
Methods may be: instance methods or static methods

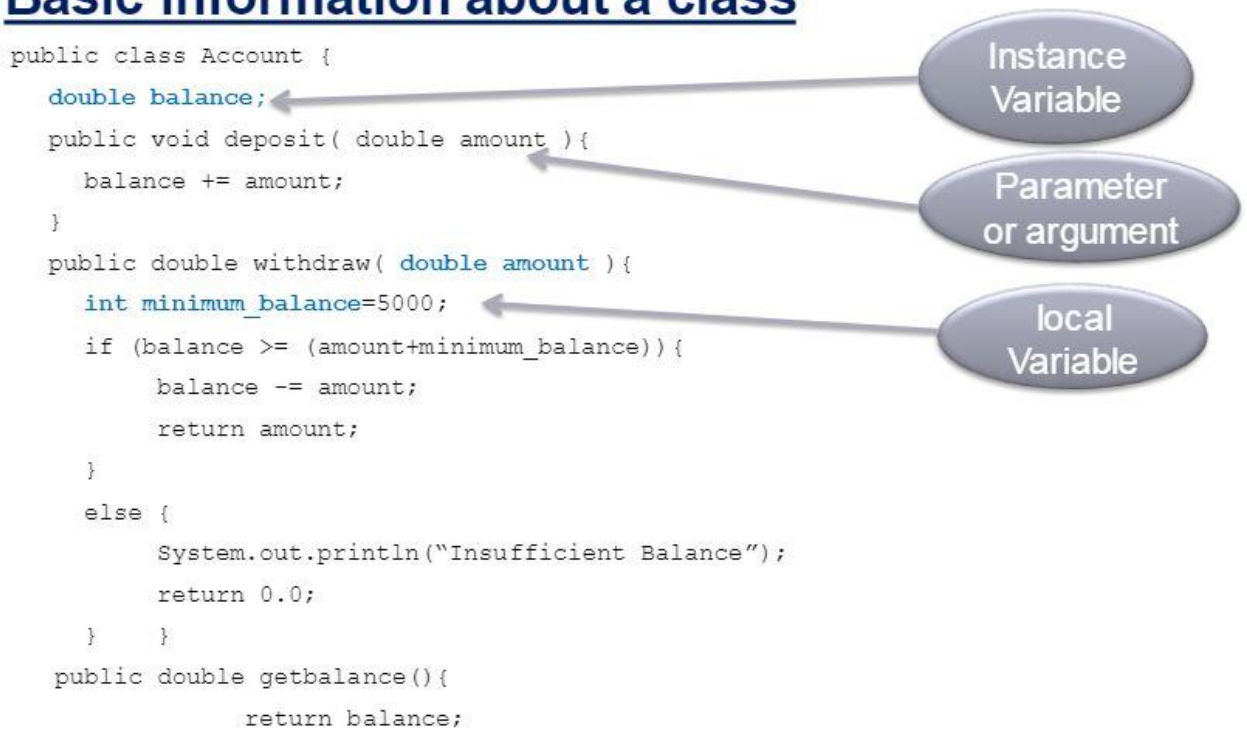
Defining a Class in java

Define an Employee class with instance variables and instance methods



```
class Employee{
  int id;
  String name;
  int salary;

  void setId(int i) {
     id = i;
   }
  void setName(String n) {
     name = n;
   }
   void setSalary(int s) {
     salary = s;
   }
   void getEmployeeDetails( ) {
     System.out.println (name + " salary
is " + salary);
   }
}
```

Basic information about a class



```
public class Account {
   double balance;
   public void deposit( double amount ){
     balance += amount;
   }
   public double withdraw( double amount ){
     int minimum_balance=5000;
     if (balance >= (amount+minimum_balance)){
          balance -= amount;
          return amount;
     }
     else {
          System.out.println("Insufficient Balance");
          return 0.0;
     }    }
   public double getbalance(){
               return balance;
```

Member variables

- The previous slide contains definition of a class called Accounts.
- A class contains members which can either be variables(fields) or methods(behaviors).
- A variable declared within a class(outside any method) is known as an **instance variable**.
- A variable declared within a method is known as **local variable**.
- Variables with method declarations are known as **parameters or arguments**.
- A class variable can also be declared as static where as a local variable cannot be static.

Objects and References

- Once a class is defined, you can declare a variable (object reference) of type class

```
Student stud1;
Employee emp1;
```

- The new operator is used to create an object of that reference type

```
Employee emp = new Employee();
          ↑                  ↖
   Object reference          object
```

- Object references are used to store objects.
- Reference can be created for any type of classes (like concrete classes, abstract classes) and interfaces.

Objects and References (Contd.).

- The new operator,

  Dynamically allocates memory for an object

  Creates the object on the heap

  Returns a reference to it

  The reference is then stored in the variable

Employee class - Example

```
class Employee{
int id;
String name;
int salary;
void setId(int no){
id = no;
}
void setName(String n){
name = n;
}
void setSalary(int s){
salary = s;
}
void getEmployeeDetails(){
System.out.println(name + " salary is "+ salary);
}
}
public class EmployeeDemo {
public static void main(String[] args) {
Employee emp1 = new Employee();
emp1.setId(101);
emp1.setName("John");
emp1.setSalary(12000);
emp1.getEmployeeDetails();
}
}
```

Output:

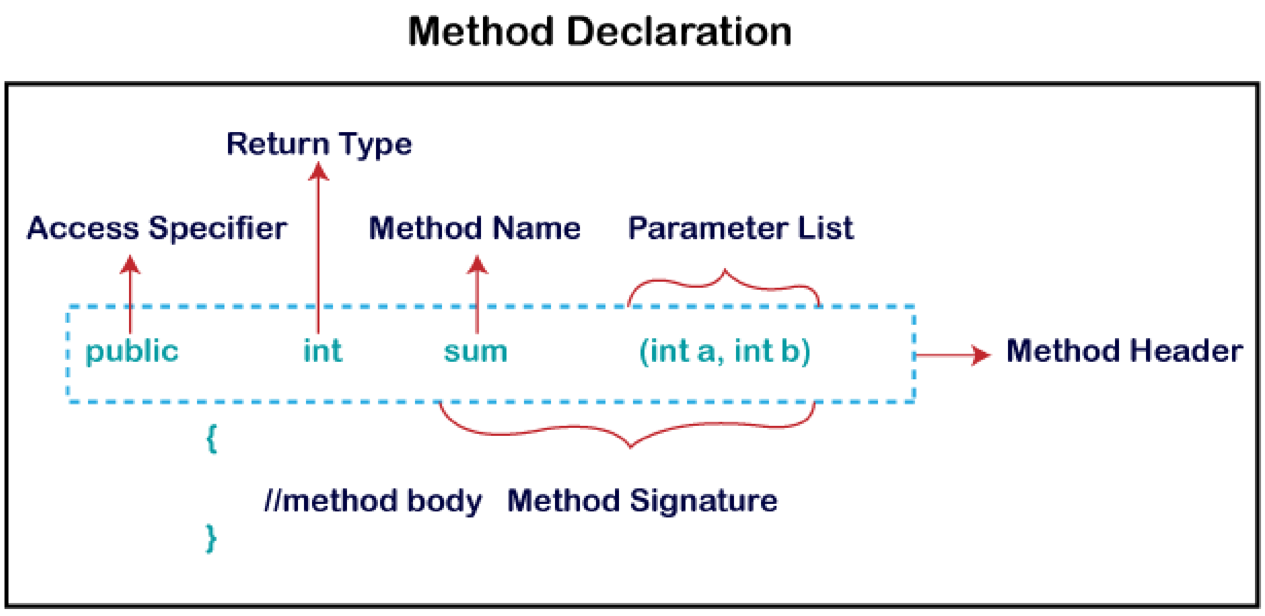
```
John salary is 12000
```

Methods:

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as method header, as we have shown in the following figure.



Naming a Method (Camel case convenstion)

Single-word method name: sum(), area()

Multi-word method name: areaOfCircle(), stringComparision()

Types of Method

There are two types of methods in Java:

1. Predefined Method
2. User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method

Examples: length(), equals(), compareTo(), sqrt(),main(), print(), and max()

User-defined Method

The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

Examples: isEven(), isOdd(), calculateSum()

How to Call or Invoke a User-defined Method

Once we have defined a method, it should be called. The calling of a method in a program is simple. When we call or invoke a user-defined method, the program control transfer to the called method.

Example:

```
import java.util.Scanner;
public class EvenOdd
{
        public static void main (String args[])
        {
                //creating Scanner class object
                Scanner scan=new Scanner(System.in);
                System.out.print("Enter the number: ");
                //reading value from user
                int num=scan.nextInt();
                //method calling
                findEvenOdd(num);
        }
        //user defined method
        public static void findEvenOdd(int num)
        {
                //method body
                if(num%2==0)
                System.out.println(num+" is even");
                else
                System.out.println(num+" is odd");
        }
}
```

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name.

Example:

```
public class Display
{
        public static void main(String[] args)
        {
                show();
        }
        static void show()
        {
        System.out.println("It is an example of static method.");
        }
}
```

Instance Method

The method of the class is known as an instance method. It is a non-static method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class.

Example:

```
public class InstanceMethodExample
{
        int s;
        public static void main(String [] args)
        {
                //Creating an object of the class
                InstanceMethodExample obj = new InstanceMethodExample();
                //invoking instance method
                System.out.println("The sum is: "+obj.add(12, 13));
        }
        //user-defined method because we have not used static keyword
        public int add(int a, int b)
        {
                s = a+b;
                //returning the sum
                return s;
        }
}
```

There are two types of instance method:

1. Accessor Method
2. Mutator Method

Accessor Method: The method(s) that reads the instance variable(s) is known as the accessor method. We can easily identify it because the method is prefixed with the word **get**.

It is also known as **getters**. It returns the value of the private field. It is used to get the value of the private field.

Example

```
public int getId()

{

        return Id;

}
```

Mutator Method: The method(s) read the instance variable(s) and also modify the values. We can easily identify it because the method is prefixed with the word set.

It is also known as setters or modifiers. It does not return anything. It accepts a parameter of the same data type that depends on the field. It is used to set the value of the private field.

Example

```
public void setRoll(int roll)
{
        this.roll = roll;
}
```

Abstract Method

The method that does not has method body is known as abstract method. In other words, without an implementation is known as abstract method. It always declares in the abstract class.

It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword abstract.

Syntax

```
abstract void method_name();
```

Factory method

It is a method that returns an object to the class to which it belongs. All static methods are factory methods.

Example:

```
NumberFormat obj = NumberFormat.getNumberInstance();
```

Constructors

- While designing a class, the class designer can define within the class, a special method called 'constructor'
- Constructor is automatically invoked whenever an object of the class is created
- Rules to define a constructor
  - A constructor has the same name as the class name
  - A constructor should not have a return type
  - A constructor can be defined with any access specifier (like private, public)
  - A class can contain more than one constructor, So it can be overloaded

Constructor - Example

```
class Sample{
private int id;
Sample(){
id = 101;
System.out.println("Default constructor, with ID: "+id);
}
Sample(int no){
id = no;
System.out.println("One argument constructor,with ID: "+ id);
}
}
public class ConstDemo {
public static void main(String[] args) {
Sample s1 = new Sample();
Sample s2 = new Sample(102);
}
}
```

Output:

```
Default constructor, with ID: 101
One argument constructor,with ID: 102
```

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: 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 java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

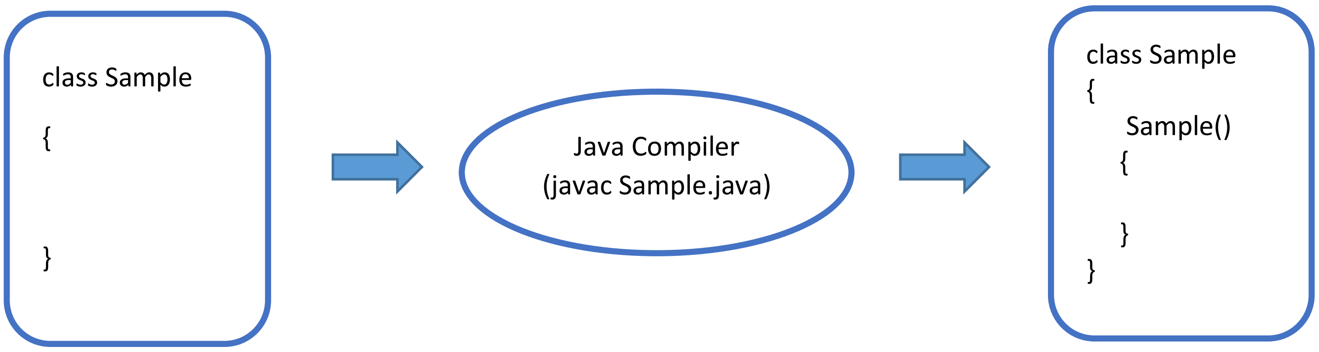
Syntax of default constructor:

```
<class_name>()

{

}
```

Example:

```
public class Sample
{
        Sample()  // default constructor
        {
                System.out.println("welcome to sample class constructor");
        }
        Public static void main(String ar[])
        {
                Sample s= new Sample();   // default constructor will be executed
        }

}
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.



```
class Sample

{


}
```

```
class Sample
{
   Sample()
   {

   }
}
```

What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example:

```
class Employee
{
	int id;
	String name;
	//method to display the value of id and name
	void display()
	{
		System.out.println(id+" "+name);
	}

	public static void main(String args[])
	{
		//creating objects
		Employee e1=new Employee();
		Employee e2=new Employee();
		//displaying values of the object
		e1.display();
		e2.display();
	}
}
```

Explanation: In the above class, you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Syntax:

```
<class-name>(param1, param2)

{

}
```

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example:

```
class Employee
{
        int id;
        String name;
        Employee(int i, String n)
        {
                id=i;
                name=n;
        }
        //method to display the value of id and name
        void display()
        {
                System.out.println(id+" "+name);
        }

        public static void main(String args[])
        {
                //creating objects
                Employee e1=new Employee(10,”ravi”);
                Employee e2=new Employee(20,”ramu”);
                //displaying values of the object
                e1.display();
                e2.display();
        }
}
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

They are differentiated by the compiler by the number of parameters in the list and their types.

Example:

```
class Employee
{
        int id;
        String name;
        int age;

        Employee(int i, String n)
        {
                id=i;
                name=n;
        }
        Employee(int i, String n, int a)
        {
                id=i;
                name=n;
                age=a;
        }

        void display()
        {
                System.out.println(id+" "+name+" "+age);
        }

        public static void main(String args[])
        {
                //creating objects
                Employee e1=new Employee(10,”ravi”);
                Employee e2=new Employee(20,”ramu”,25);
                //displaying values of the object
                e1.display();
                e2.display();
        }
}
```

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

Example:

```
class Employee
{
        int id;
        String name;
        Employee(int i, String n)
        {
                id=i;
                name=n;
        }
        Employee(Employee e1)
        {
                id=e1.id;
                name=e1.name
        }
        //method to display the value of id and name
        void display()
        {
                System.out.println(id+" "+name);
        }

        public static void main(String args[])
        {
                //creating objects
                Employee e1=new Employee(10,”ravi”);
                Employee e2=new Employee(e1);
                //displaying values of the object
                e1.display();
                e2.display();
        }
}
```

Example2: Copying values without constructor

```
class Employee
{
        int id;
        String name;
        Employee(int i, String n)
        {
                id=i;
                name=n;
        }
        //method to display the value of id and name
        void display()
        {
                System.out.println(id+" "+name);
        }

        public static void main(String args[])
        {
                //creating objects
                Employee e1=new Employee(10,”ravi”);
                Employee e2=new Employee();
                e2.id=e1.id;
                e2.name=e1.name;
                //displaying values of the object
                e1.display();
                e2.display();
        }
}
```

Does constructor return any value?

Yes, it is the current class instance (You cannot use return type yet it returns a value).

Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Is there Constructor class in Java?

Yes.

What is the purpose of Constructor class?

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.