Welcome you all

JAVA PROGRAMMING

DAY : 4

D.Sakthivel
Assistant Professor & Trainer,
KGiSL Micro College
KGiSL Campus, Coimbatore – 641 035.

Day 2

- **Object-Oriented Programming**

- **Object**
- **Class**
- **Inheritance**
- **Polymorphism**
- **Abstraction**
- **Encapsulation**
- **Java Naming Convention**
- Object and Class
- **Method in Java**
  - **Instance Method**
  - **Abstract Method**

Object-Oriented Programming

- Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

- **Simula** is considered the first object-oriented programming language.
- The programming paradigm where everything is represented as an object is known as a truly object-oriented programming language.

- **Smalltalk** is considered the first truly object-oriented programming language.

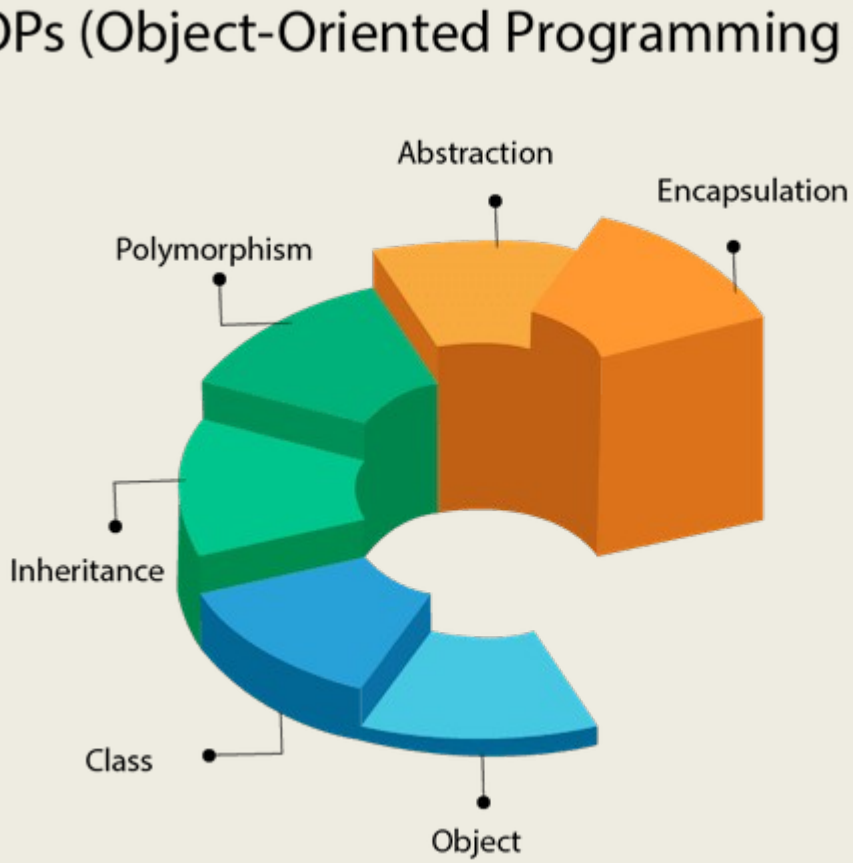
- The popular object-oriented languages are Java, C#, PHP, Python, C++, etc.
- **The main aim of object-oriented programming is to implement real-world entities**,
- for example, object, classes, abstraction, inheritance, polymorphism, etc.

OOPs (Object-Oriented Programming System)

- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc.
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.
- It simplifies software development and maintenance by providing some concepts:

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

OOPs (Object-Oriented Programming System)



Object

Any entity that has state and behavior is known as an object.

For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

- **An Object can be defined as an instance of a class.**
- **An object contains an address and takes up some space in memory.**
- Objects can communicate without knowing the details of each other's data or code.
- The only necessary thing is the type of message accepted and the type of response returned by the objects.

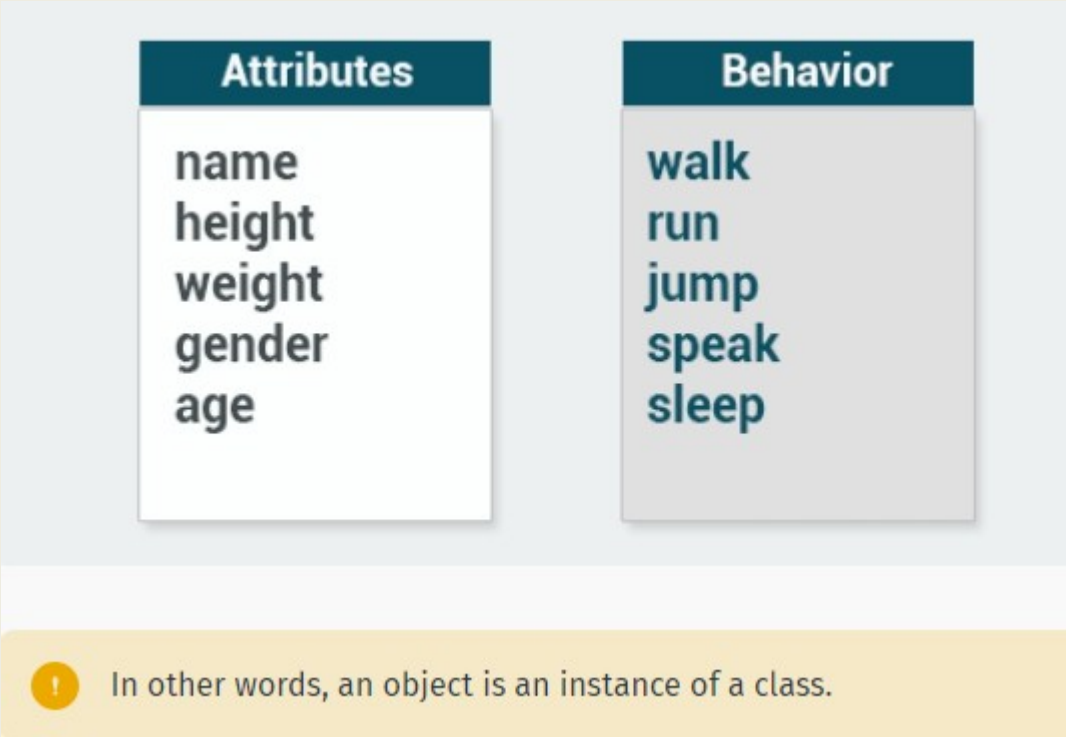
Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

- *Collection of objects* is called class. It is a logical entity.
- **A class can also be defined as a blueprint from which you can create an individual object**. Class doesn't consume any space.
- Classes can be described as blueprints, descriptions, or definitions for an object.
- You can use the same class as a blueprint for creating multiple objects.
- The first step is to define the class, which then becomes a blueprint for object creation.

- Each class has a name, and each is used to define **attributes** and **behavior**.

Class


Attributes
name
height
weight
gender
age
Behavior
walk
run
jump
speak
sleep
In other words, an object is an instance of a class.

Encapsulation

- ***Binding (or wrapping) code and data together into a single unit are known as encapsulation***.
- For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation.
- Java bean is the fully encapsulated class because all the data members are private here.

Capsule

Abstraction

***Hiding internal details and showing functionality* is known as abstraction.**

For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Polymorphism

- **If *one task is performed in different ways*, it is known as polymorphism.**
- For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.
- In Java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Inheritance

- ***When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance.**
- It provides code reusability. It is used to achieve runtime polymorphism.

Java Naming Convention

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
- By using standard Java naming conventions, you make your code easier to read for yourself and other programmers.

Identifiers Type	Naming Rules	Examples

Class	**It should start with the uppercase letter.**
	It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	**It should start with the uppercase letter.**
	It should be an adjective such as Runnable, Remote, ActionListener.
	Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Java Naming Convention

Method	**It should start with lowercase letter.**
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void **draw()**
		{
		//code snippet
		}
		}

Variable	**It should start with a lowercase letter such as id, name.**
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
	Avoid using one-character variables such as x, y, z.
		class Employee
		{
		// variable
		int **id**;
		//code snippet
		}

Java Naming Convention

Package	It should be a lowercase letter such as java, lang.
	If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
		//package
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

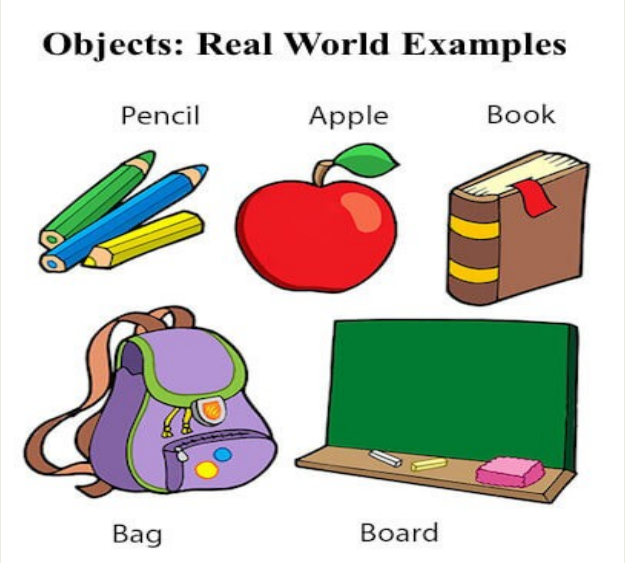
Java Naming Convention

CamelCase in Java naming conventions

- Java follows camel-case syntax for naming the class, interface, method, and variable.
- If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

- Object and Class

What is object?



- **An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.**
- **It can be physical or logical (tangible and intangible).**
- **The example of an intangible object is the banking system.**

An object has three characteristics:

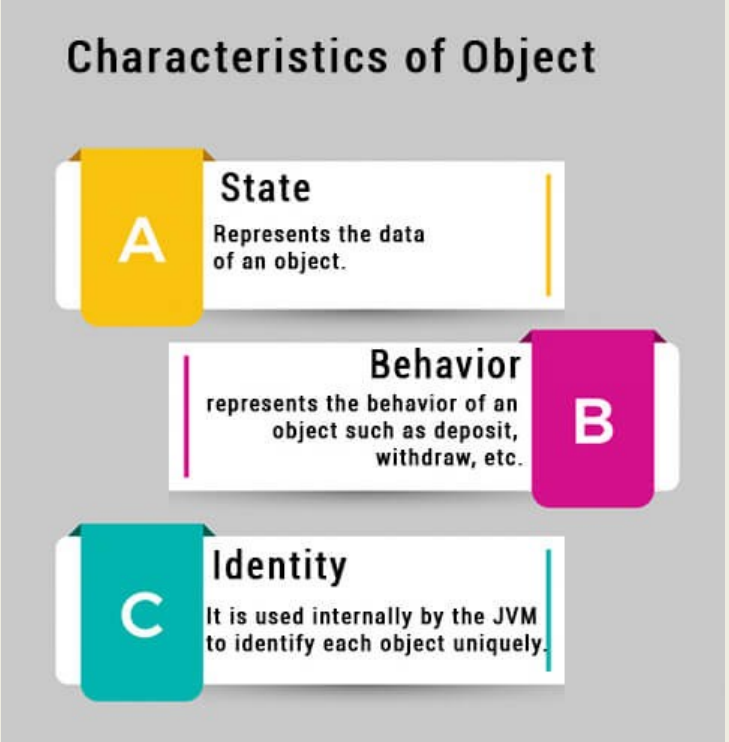
State: represents the data (value) of an object.
Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

What is object?

For Example, Pen is an object. Its name is Reynolds; color is white, known as its state.

It is used to write, so writing is its behavior.

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:

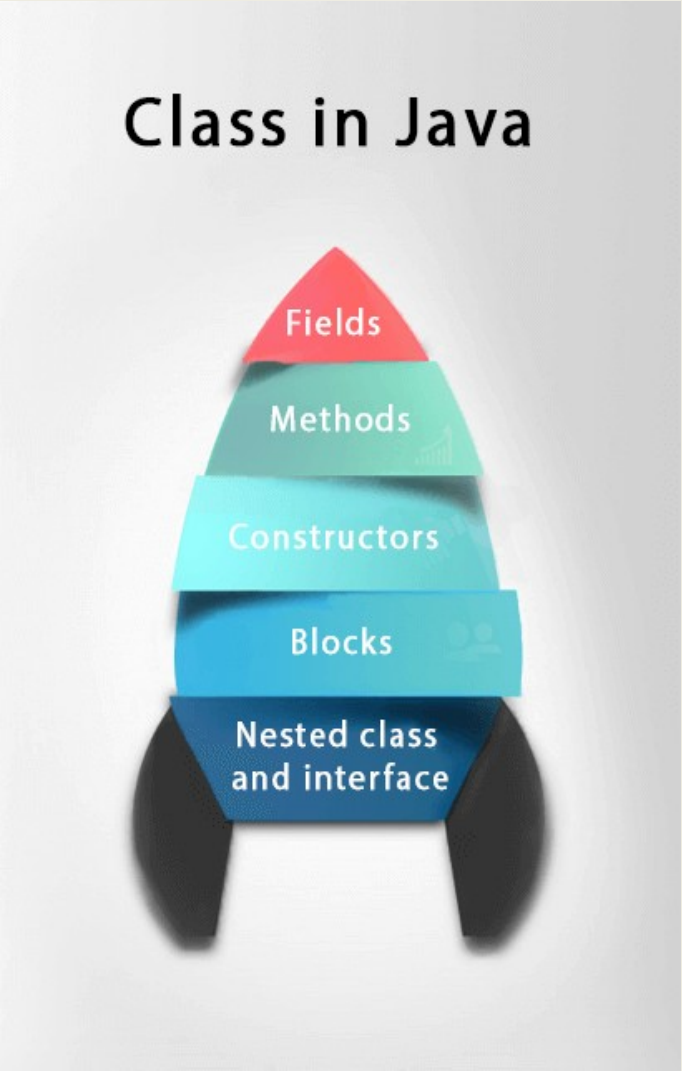
- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

What is class?

- **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**



What is class?

Syntax to declare a class:

```
class <class_name>{
    field;
    method;
}
```

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.

Example Program (Object1 & Object2)

Object and Class Example: main within the class – Object 1 program

Object and Class Example: main outside the class – Object2 program

Example Program :
Object and Class Example: main within t

```java
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Output:

```
0
null
```

KGiSL MicroCollege

Example Program :
Object and Class Example: main outside the class

```
//Java Program to demonstrate having the main method in
//another class
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Output:

```
0
null
```

KGiSL MicroCollege

3 Ways to initialize object

There are 3 ways to initialize object in Java.

- **By reference variable**
- **By method**
- **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.

Initialization through reference

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);
//printing members with a white space
 }
}
```

Output:

```
101 Sonoo
```

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}
```

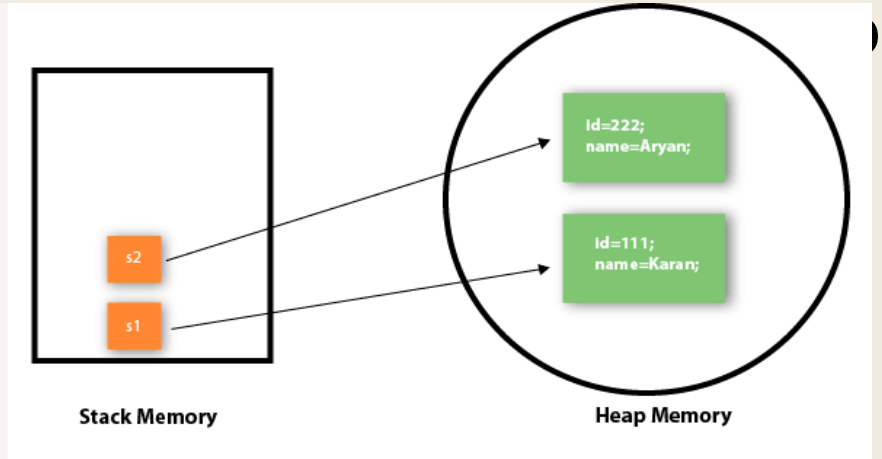
We can also create multiple objects and store information in it through reference variable.

Output:

```
101 Sonoo
102 Amit
```

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 ...rmation() 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.

Object and Class Example: Initialization through method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Output:

```
111 Karan
222 Aryan
```

KGiSL MicroCollege

Object and Class Example: Initialization through a constructor

```
class Employee{
   int id;
   String name;
   float salary;
   void insert(int i, String n, float s) {
      id=i;
      name=n;
      salary=s;
   }
   void display(){System.out.println(id+" "+name+" "+salary);}
}
```

```
public class TestEmployee {
public static void main(String[] args) {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}
}
```

Output:

```
101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0
```

Object and Class Example: Rectangle

```
class Rectangle{
 int length;
 int width;
 void insert(int l, int w){
  length=l;
  width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
```

```
class TestRectangle1{
 public static void main(String args[]){
  Rectangle r1=new Rectangle();
  Rectangle r2=new Rectangle();
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
 }
}
```

Output:

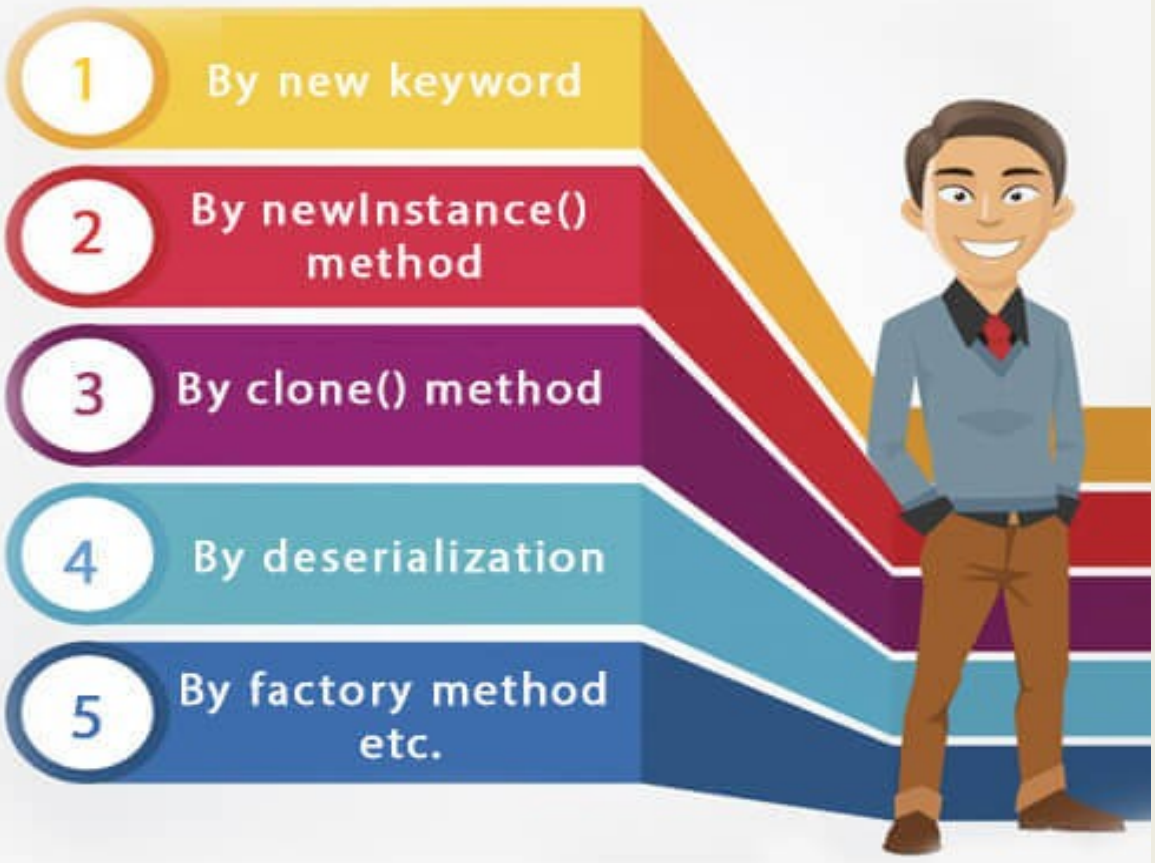
```
55
45
```

What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc.

Different ways to create an object in Java



Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach. For example:

```
new Calculation();//anonymous object
```

Calling method through a reference:

```
Calculation c=new Calculation();
c.fact(5);
```

Calling method through an anonymous object

```
new Calculation().fact(5);
```

Let's see the full example of an anonymous object in Java.

```
class Calculation{
 void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
 System.out.println("factorial is "+fact);
}
public static void main(String args[]){
 new Calculation().fact(5);//calling method with anonymous object
}
}
```

Output:

```
Factorial is 120
```

KGiSL MicroCollege

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

```
int a=10, b=20;
```

Initialization of reference variables:

```
Rectangle r1=new Rectangle(), r2=new Rectangle();//
creating two objects
```

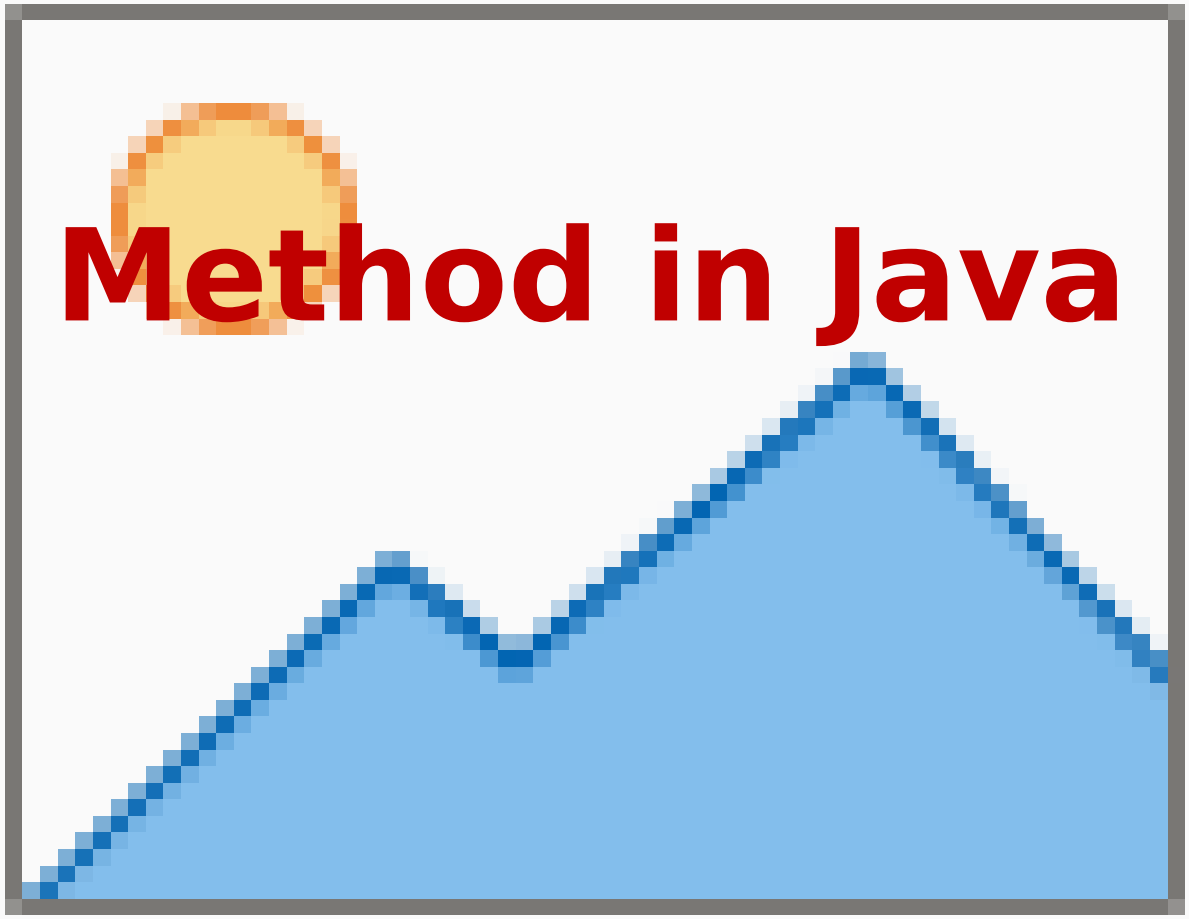
```java
//Java Program to illustrate the use of Rectangle class which
//has length and width data members
class Rectangle{
 int length;
 int width;
 void insert(int l,int w){
  length=l;
  width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle2{
 public static void main(String args[]){
  Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
  r1.insert(11,5);
  r2.insert(3,15);
  r1.calculateArea();
  r2.calculateArea();
 }
}
```

Output:

```
55
45
```

KGiSL MicroCollege

Click icon to add picture



Method in Java

- In general, a **method** is a way to perform some task.
- Similarly, the **method in Java** is a collection of instructions that performs a specific task.
- It provides the reusability of code. We can also easily modify code using **methods**.

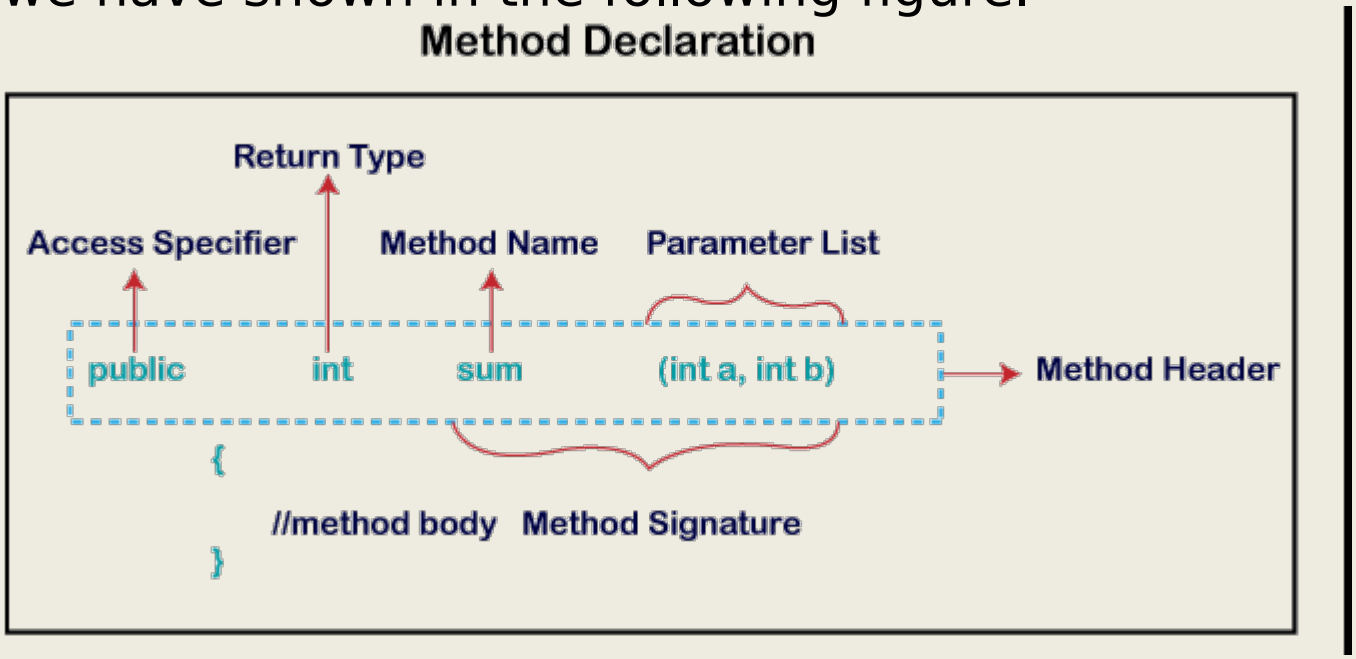
What is a method in Java?

- 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. We write a method once and use it many times.
- We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.
- The method is executed only when we call or invoke it.
- The most important method in Java is the **main()** method.

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.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.
Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:
Public: The method is accessible by all classes when we use public specifier in our application.
Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.
Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter.

If the method name has more than two words, the first name must be a verb followed by adjective or noun.

In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

- **Single-word method name:** sum(), area()
- **Multi-word method name:** areaOfCircle(), stringComparision()

Types of Method

There are two types of methods in Java:

- **Predefined Method**
- **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**.
- We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc.

Predefined Method

- Each and every predefined method is defined inside a class.
- Such as **print()** method is defined in the **java.io.PrintStream** class.
- It prints the statement that we write inside the method.
- For example, **print("Java")**, it prints Java on the console.

```
public class Demo
{
public static void main(String[] args)
{
// using the max() method of Math class
System.out.print("The maximum number is: " + Math.m
ax(9,7));
}
}
```

Output:

```
The maximum number is: 9
```

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.

How to Create a User-defined Method

```
//user defined method
public static void findEvenOdd(i
nt num)
{
//method body
if(num%2==0)
System.out.println(num+" is even
");
else
System.out.println(num+" is odd"
);
}
```

How to Call or Invoke a User-defined Method

```
import java.util.Scanner;
public class EvenOdd
{
public static void main (String a
rgs[])
{
//creating Scanner class object
Scanner scan=new Scanner(Syst
em.in);
System.out.print("Enter the numb
er: ");
//reading value from the user
int num=scan.nextInt();
//method calling
findEvenOdd(num);
}
```

Let's combine both snippets of codes in a single program and execute it.

```
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");
}
}
```

```
Enter the number: 12
12 is even
```

KGiSL MicroCollege

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.
- We can also create a static method by using the keyword **static** before the method name.
- 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. The best example of a static method is the **main()** method.

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

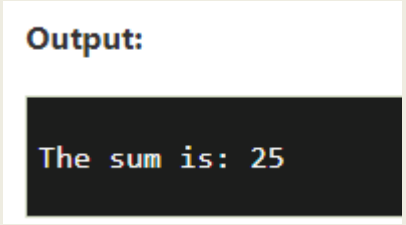
Output:

```
It is an example of a 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.
- Let's see an example of an instance method.

```
public class InstanceMethodExample
{
public static void main(String [] args)
{
//Creating an object of the class
InstanceMethodExample obj = new InstanceMethodExampl
e();
//invoking instance method
System.out.println("The sum is: "+obj.add(12, 13));
}
int s;
//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:

- **Accessor Method**
- **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.

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

```
public class Student
{
private int roll;
private String name;
public int getRoll()    //accessor method
{
return roll;
}
public void setRoll(int roll) //mutator method
{
this.roll = roll;
}
public String getName()
{
return name;
}
public void setName(String name)
{
this.name = name;
}
public void display()
{
System.out.println("Roll no.: "+roll);
System.out.println("Student name: "+name);
}
}
```

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_nam
e();
```

Abstract Method

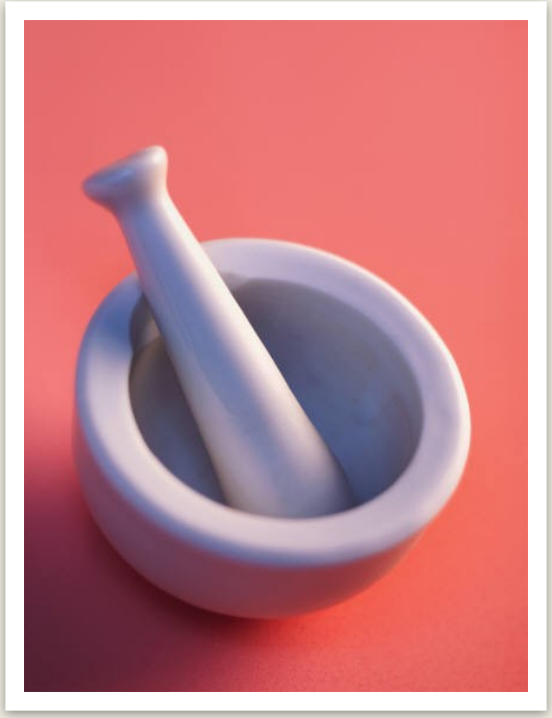
```
abstract class Demo //abstract class
{
//abstract method declaration
abstract void display();
}
public class MyClass extends Demo
{
//method impelmentation
void display()
{
System.out.println("Abstract method?");
}
```

```
public static void main(String args[])
{
//creating object of abstract class
Demo obj = new MyClass();
//invoking abstract method
obj.display();
}
}
```

Output:

```
Abstract method...
```

KGiSL MicroCollege



Thank You