

UNIT-7

CLASS, OBJECT AND METHOD PART 1

7.1 General form of Classes and Object: -

- Java provides a reserved keyword class to define a class.
- Class is derived datatype; it combines members of different datatypes into one.
- Defines new datatype (primitive ones are not enough).
- For Example: Car, College, Bus etc.
- This new datatype can be used to create objects.
- A class is a template for an object.

Syntax: -

```
<access specifier> class class_name  
{  
    // member variables  
    // class methods  
}
```

In general, class declaration includes the following in the order as it appears:

1. **Modifiers:** A class can be public or has default access.
2. **class keyword:** The class keyword is used to create a class.
3. **Class name:** The name must begin with an initial letter (capitalized by convention).
4. **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.
5. **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.
6. **Body:** The class body surrounded by braces, { }.

```

class Car{
    String company;
    String model;
    double price;
    double mileage;
    .....
}

```

Class: Car



Introduction to Objects in java

What is Object?

An object is an instance of a class.

An object has a state and behaviour.

Example: A dog has

states - colour, name, breed as well as
behaviours – barking, eating.

- The state of an object is stored in fields (variables), while methods (functions) display the object's behaviour.
- An Object is a key to understand Object Oriented Technology.
- An entity that has state and behaviour is known as an object. e.g., Mobile, Car, Door, Laptop etc

Each and every object posses

1. Identity
2. State
3. Behaviour

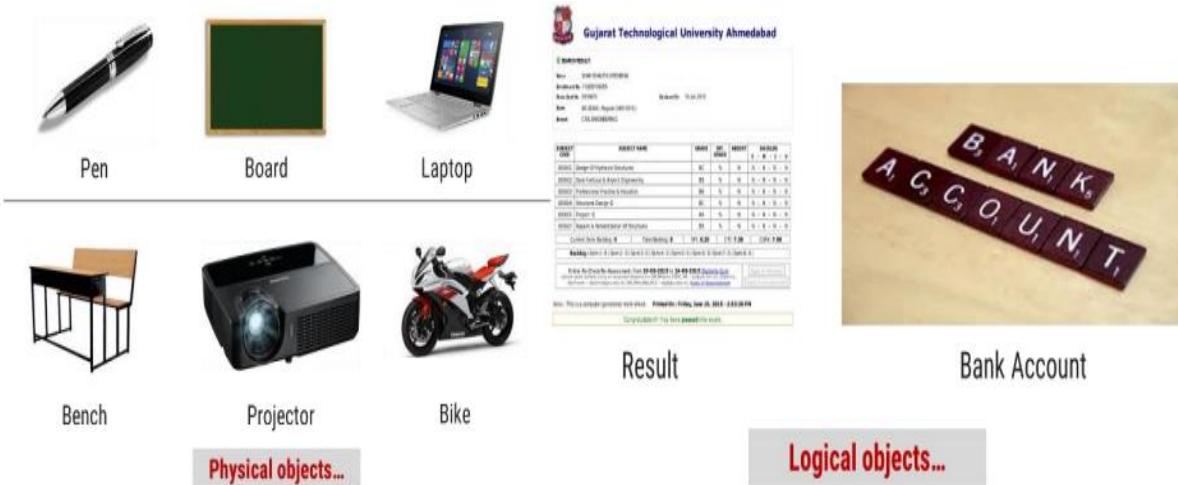
Philosophy of Object Oriented

Our real world is nothing but classification of objects

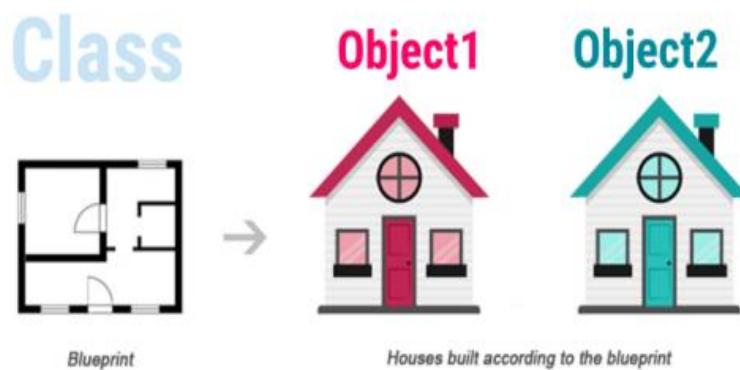
E.g. Human, Vehicle, Library, River, Watch, Fan, etc.

- Real world is organization of different objects which have their own characteristics, behaviour

- Characteristic of Human: Gender, Age, Height, Weight, Complexion, etc.
- Behaviour of Human: Walk, Eat, Work, React, etc.
- Characteristic of Library: Books, Members, etc.
- Behaviour of Library: New Member, Issue Book, Return Book etc.
- The OO philosophy suggests that the things manipulated by the program should correspond to things in the real world.
- Classification is called a Class in OOP
- Real world entity is called an Object in OOP
- Characteristic is called Property in OOP
- Behaviour is called Method in OOP



Classes and Objects: -



Class is a blueprint of an object
Class describes the object

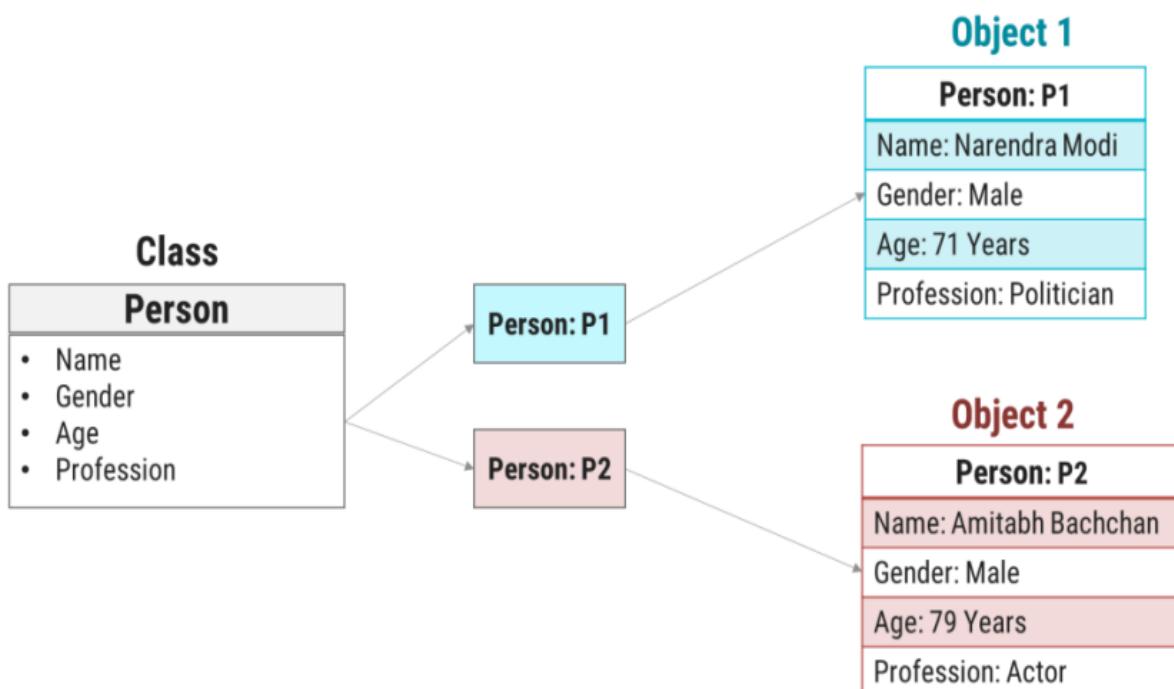
Object1 **Object2**
Objects are instances of classes

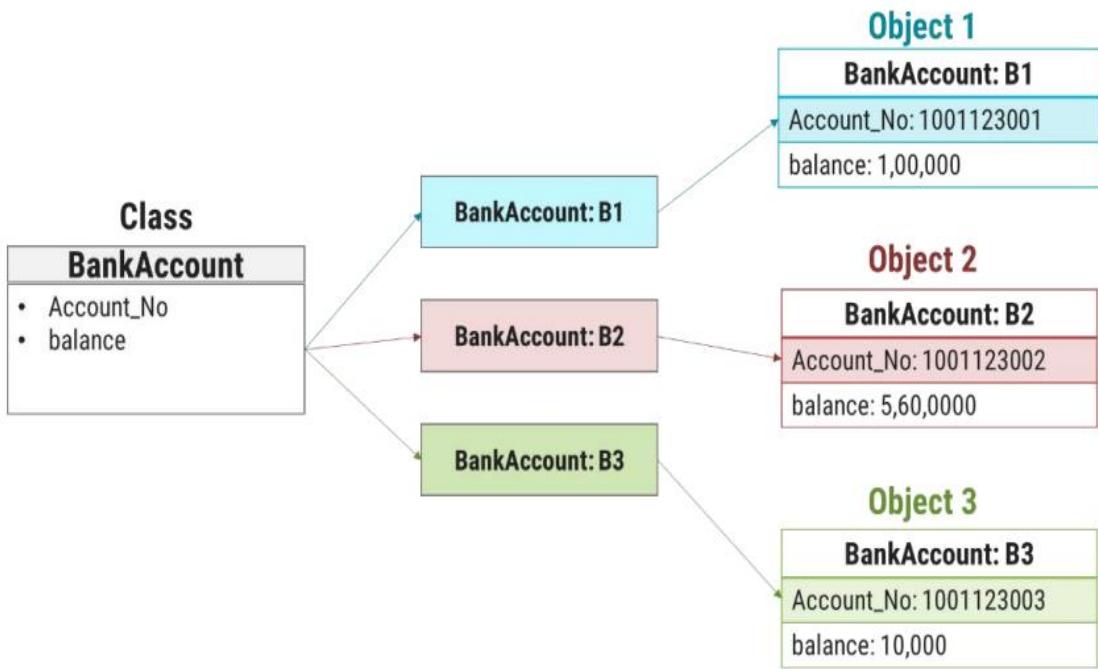
Class can be defined in multiple ways

- A class is the building block.
- A class is a blueprint for an object.
- A class is a user-defined data type.
- A class is a collection of objects of the similar kind.
- A class is a user-defined data type which combines data and methods.
- A class describes both the data and behaviors of objects.
- Class contains data members (also known as field or property or data) and member functions (also known as method or action or behavior)
- Classes are similar to structures in C.
- Class name can be given as per the Identifier Naming Conventions.

Object Definition:

- An Object is an instance of a Class.
- An Object is a variable of a specific Class
- An Object is a data structure that encapsulates data and functions in a single construct.
- Object is a basic run-time entity
- Objects are analogous to the real-world entities.





Creating Object & Accessing members: -

- new keyword creates new object

Syntax:

- ClassName objName = new ClassName();

Example:

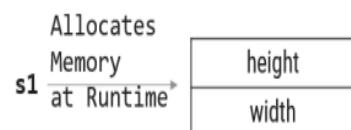
- SmartPhone iPhone = new SmartPhone();
- Object variables and methods can be accessed using the dot (.) operator
- Example: iPhone.storage = 8000;
- Declaring an Object
- When we create a class, we are creating a new data type.
- Object of that data type will have all the attributes and abilities that are designed in the class.
- The new operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it.
- This reference is, more or less, the address in memory of the object allocated by new.
- This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.

```

1. class Square{
2.     double height;
3.     double width;
4. }
5. class MyProg{
6.     public static void main(String[] args) {
7.         Square s1; //declare reference to object
8.         s1= new Square(); //allocate a Square object
9.     }
10.

```

An object reference is similar to a memory pointer.



`new` operator dynamically allocates memory for an object

- Here, `s1` is a variable of the class type.
- The class name followed by parentheses specifies the constructor for the class.
- It is important to understand that `new` allocates memory for an object during run time.

Assigning Object Reference

| | |
|---|--|
| <p>ObjectDemo.java</p> <pre>Rectangle r1=new Rectangle(); Rectangle r2=r1;</pre> <p>A diagram showing two variables, <code>r1</code> and <code>r2</code>, represented as gray boxes. Both <code>r1</code> and <code>r2</code> have arrows pointing to a single shared object, which is represented as a gray rectangle divided into two horizontal sections: "Width" on top and "Height" on the bottom.</p> | <p>ObjectDemo.java</p> <pre>Rectangle r1=new Rectangle(); Rectangle r2=r1; ... R1=null</pre> <p>A diagram showing two variables, <code>r1</code> and <code>r2</code>, represented as gray boxes. The variable <code>r1</code> has an arrow pointing to a gray box labeled "null". The variable <code>r2</code> has an arrow pointing to the same shared object as in the previous diagram, which is represented as a gray rectangle divided into two horizontal sections: "Width" on top and "Height" on the bottom.</p> |
| <p>Here, <code>r1</code> and <code>r2</code> will both refer to the same object. The assignment of <code>r1</code> to <code>r2</code> did not allocate any memory or copy any part of the original object. It simply makes <code>r2</code> refer to the same object as does <code>r1</code></p> | <p>Here, <code>r1</code> has been set to null, but <code>r2</code> still points to the original object.</p> |

WAP using class Person to display name and age

```
1. class MyProgram {  
2.     public static void main(String[] args) {  
3.         Person p1= new Person();  
4.         Person p2= new Person();  
5.         p1.name="modi";  
6.         p1.age=71;  
7.         p2.name="bachchan";  
8.         p2.age=80;  
9.         System.out.println("p1.name="+p1.name);  
10.        System.out.println("p2.name="+p2.name);  
11.        System.out.println("p1.age="+p1.age);  
12.        System.out.println("p2.age="+p2.age);  
13.    } //main()  
14. } //class myProgram
```

```
15.     class Person  
16.     {  
17.         String name;  
18.         int age;  
19.     } //class person  
20.
```

Output:

```
p1.name=modi  
p2.name=bachchan  
p1.age=71  
p2.age=80
```

WAP using class Person to display name and age with method

```
1. class MyProgram {  
2.     public static void main(String[] args){  
3.         Person p1=new Person();  
4.         Person p2=new Person();  
5.         p1.name="modi";  
6.         p1.age=71;  
7.         p2.name="bachchan";  
8.         p2.age=80;  
9.         p1.displayName();  
10.            p2.displayName();  
11.            p1.displayAge();  
12.            p2.displayAge();  
13.        } //main()  
14.    } //class myProgram  
  
15. class Person{  
16.     String name;  
17.     int age;  
18.     public void displayName(){  
19.         System.out.println("name="+name);  
20.     }  
21.     public void displayAge(){  
22.         System.out.println("age="+age);  
23.     }  
24. } //class person
```

Output:

```
name=modi  
name=bachchan  
age=71  
age=80
```

Class vs. Object

| Class | Object |
|---|---|
| Class is a Blueprint or template. | Object is the instance of a class. |
| Class creates a logical framework that defines the relationship between its members. | When you declare an object of a class, you are creating an instance of that class. |
| Class is a logical construct. | An object has physical reality. (i.e., an object occupies space in memory.) |
| Class is a group or collection of similar object. e.g. Car | An Object is defined as real-world entity e.g.: Audi, Volkswagen, Tesla, Ferrari etc. |
| Class is declared only once | An Object can be created as many times as required |
| Class doesn't allocate memory when it is created. | An Object allocates the memory upon creation |
| Class can exist without its objects. | An Object can't exist without a class. |
| Class: Profession Class: Mobile Class: Subject Class: Student Class: Color | Object: Doctor, Teacher, Lawyer, Politician... Object: iPhone, Samsung, 1plus... Object: Maths, English, Science, Computer... Object: John, Aarav, Smita... Object: Blue, Green, Red, Yellow, Violet, Black... |

Characteristics of OOP

OOps in java is to improve code readability and reusability by defining a Java program efficiently. The main principles of object-oriented programming are abstraction, encapsulation, inheritance, and polymorphism. These concepts aim to implement real-world entities in programs.

What is Abstraction?

Abstraction is a process which displays only the information needed and hides the unnecessary information. We can say that the main purpose of abstraction is data hiding. Abstraction means selecting data from a large number of data to show the information needed, which helps in reducing programming complexity and efforts.

Suppose we want to create a student application and ask to collect information about the student.

We collect the following information.

Name

Class

Address

Dob

Fathers name

Mothers' names and so on.

We may not require every information that we have collected to fill out the application. So, we select the data that is required to fill out the application. Hence, we have fetched, removed, and selected the data, the student information from large data. This process is known as abstraction in the oops concept.

What is Inheritance?

Inheritance is a method in which one object acquires/inherits another object's properties, and inheritance also supports hierarchical classification. The idea behind this is that we can create new classes built on existing classes, i.e., when you inherit from an existing class, we can reuse methods and fields of the parent class. Inheritance represents the parent-child relationship.

What is Polymorphism?

Polymorphism refers to many forms, or it is a process that performs a single action in different ways. It occurs when we have many classes related to each other by inheritance. Polymorphism is of two different types, i.e., compile-time polymorphism and runtime polymorphism. One of the examples of Compile time polymorphism is that when we overload a static method in java. Run time polymorphism also called a dynamic method dispatch is a method in which a call to an overridden method is resolved at run time rather than compile time. In this method, the overridden method is always called through the reference variable. By using method overloading and method overriding, we can perform polymorphism. Generally, the concept of polymorphism is often expressed as one interface, and multiple methods. This reduces complexity by allowing the same interface to be used as a general class of action.

What is Encapsulation?

Encapsulation is one of the concepts in OOPs concepts; it is the process that binds together the data and code into a single unit and keeps both from being safe from outside interference and misuse. In this process, the data is hidden from other classes and can be accessed only through the current class's methods. Hence, it is also known as data hiding. Encapsulation acts as a protective wrapper that prevents the code and data from being accessed by outsiders.

7.2 Scope and lifetime of variables: -

The scope of a variable refers to the areas or the sections of the program in which the variable can be accessed, and the lifetime of a variable indicates how long the variable stays alive in the memory.

A joint statement defining the scope and lifetime of a variable is “how and where the variable is defined.”

Types of Variables and its Scope

There are three types of variables.

1. Instance Variables
2. Class Variables
3. Local Variables

1. Instance Variables

- A variable which is declared inside a class, but is declared outside any methods and blocks is known as instance variable.
- **Scope:** Throughout the class except in the static methods.
- **Lifetime:** Until the object of the class stays in the memory.

2. Class Variables

- A variable which is declared inside a class, outside all the blocks and is declared as static is known as class variable.
- **Scope:** Throughout the class.
- **Lifetime:** Until the end of the program.

3. Local Variables

- All variables which are not instance or class variables are known as local variables.
- **Scope:** Within the block it is declared.
- **Lifetime:** Until control leaves the block in which it is declared.

Example: -

```
public class scope_and_lifetime {  
    int num1, num2; //Instance Variables  
    static int result; //Class Variable  
    int add(int a, int b){ //Local Variables  
        num1 = a;  
        num2 = b;  
        return a+b;  
    }  
    public static void main(String args[]){
```

```

        scope_and_lifetime ob = new scope_and_lifetime();
        result = ob.add(10, 20);
        System.out.println("Sum = " + result);
    }
}

```

Example: -

```

public class Demo
{
//instance variable
String name = "Andrew";
//class and static variable
static double height= 5.9;
public static void main(String args[])
{
//local variable
int marks = 72;
}
}

```

Examples: -

VariableScopeExample1.java

```

public class VariableScopeExample1
{
public static void main(String args[])
{
int x=10;
{
//y has limited scope to this block only
int y=20;
System.out.println("Sum of x+y = " + (x+y));
}
//here y is unknown
y=100;
//x is still known
x=50;
}

```

```
}
```

Output:

```
/VariableScopeExample1.java:12: error: cannot find symbol
y=100;
^
    symbol:  variable y
    location: class VariableScopeExample1
1 error
```

We see that `y=100` is unknown. If you want to compile and run the above program remove or comment the statement `y=100`. After removing the statement, the above program runs successfully and shows the following output.

Sum of `x+y = 30`

There is another variable named an instance variable. These are declared inside a class but outside any method, constructor, or block. When an instance variable is declared using the keyword static is known as a static variable. Their scope is class level but visible to the method, constructor, or block that is defined inside the class.

Example: -

```
public class BlockScopeExample1
{
    public static void main(String args[])
    {
        for (int x = 0; x < 10; x++)
        {
            System.out.println(x);
        }
        System.out.println(x);
    }
}
```

Output:

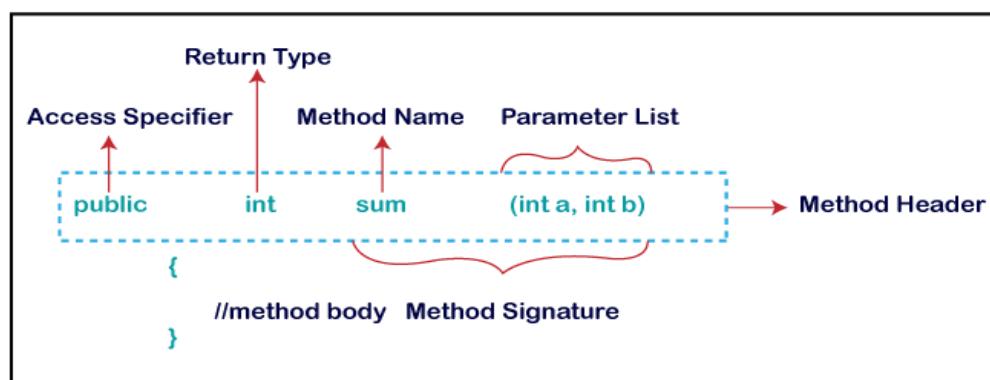
```
/BlockScopeExample.java:9: error: cannot find symbol
System.out.println(x);
^
    symbol:  variable x
    location: class BlockScopeExample
1 error
```

7.3 Introducing Method, Method definition, calling and signature of method: -

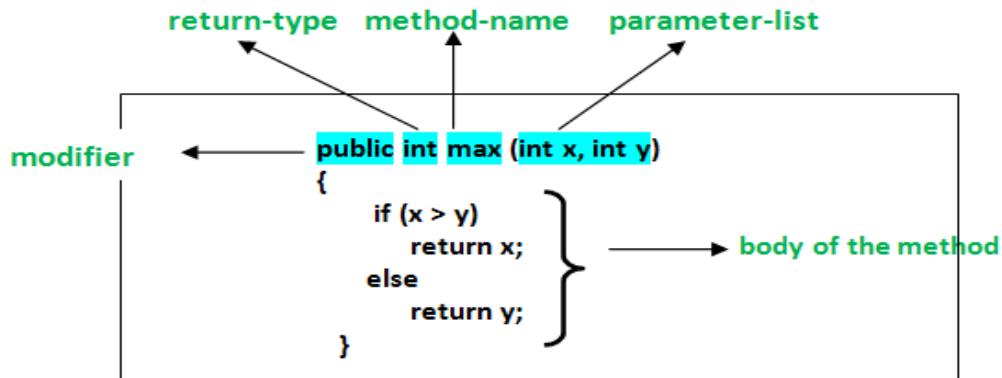
- Method in Java or Java Method is a collection of statements that perform some specific task and return the result to the caller.
- It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy 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.
- **Advantages of Method**
- Reduced Code Redundancy: Rewriting the same logic or code again and again in a program can be avoided.
- Reusability of Code: Same function can be call from multiple times without rewriting code.
- Reduction in size of program: Instead of writing many lines, just function need to be called.
- Saves Development Time: Instead of changing code multiple times, code in a function need to be changed.
- More Traceability of Code: Large program can be easily understood or traced when it is dividing into functions.
- Easy to Test & Debug: Testing and debugging of code for errors can be done easily in individual function.

Method Definition

- 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.



It consists of the method name and a parameter list (number of parameters, type of the parameters, and order of the parameters). The return type and exceptions are not considered as part of it.

Method Signature of the above function:

`max(int x, int y)` Number of parameters is 2, Type of parameter is int.

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.

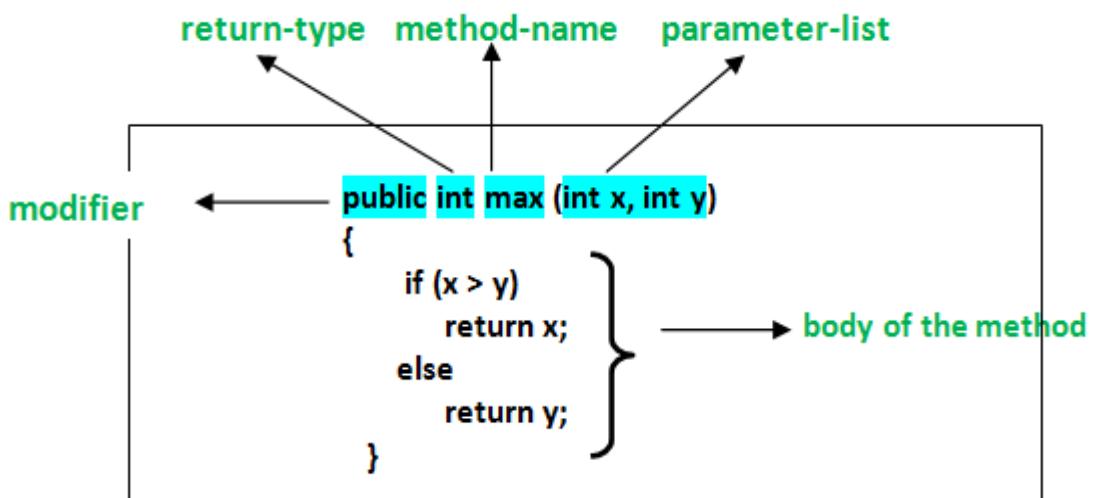
Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

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, leave 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()

It is also possible that a method has the same name as another method name in the same class, it is known as method overloading.

Types of Method

There are two types of methods in Java:

1. Predefined Method
2. User-defined Method

1. Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries 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. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Example: -

Demo.java

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

Output:

The maximum number is: 9

2. 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.

i. 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.

Example of static method

Display.java

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

Output:

It is an example of a static method.

ii. 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.

InstanceMethodExample.java

```
class InstanceMethodExample
{
    public static void main(String [] args)
    {
        //Creating an object of the class
        InstanceMethodExample obj = new InstanceMethodExample();
        //invoking instance method
        obj.add();
    }
    int s;
    //user-defined method because we have not used static keyword
```

```

void add()
{
    int a=10,b=5;
    s = a+b;
    System.out.println("sum="+s);
}

```

Output:

The sum is: 15

Types of Methods(Method Categories)

Functions can be divided in 4 categories based on arguments and return value.

Example:- No argument no return type

```

import java.util.*;
class Addition
{
    // No argument no return type
    void add()
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number 1:");
        int n1 = sc.nextInt();
        System.out.println("Enter number 2:");
        int n2 = sc.nextInt();
        int result = n1 + n2;
        System.out.println("Addition: "+result);
    }
    public static void main(String args[])
    {
        Addition a = new Addition();
        a.add();
    }
}

```

Example:- With argument no return type

```

import java.util.*;
class Addition
{

```

```

// With argument no return type
void add(int n1, int n2)
{
    int result = n1 + n2;
    System.out.println("Addition: "+result);
}
public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter number 1:");
    int n1 = sc.nextInt();
    System.out.println("Enter number 2:");
    int n2 = sc.nextInt();
    Addition a = new Addition();
    a.add(n1,n2);
}

```

Example:- With argument with return type

```

import java.util.*;
class Addition
{
    // With argument with return type
    int add(int n1, int n2)
    {
        int result = n1 + n2;
        return result;
    }
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number 1:");
        int n1 = sc.nextInt();
        System.out.println("Enter number 2:");
        int n2 = sc.nextInt();
        Addition a = new Addition();
        int res = a.add(n1,n2);
        System.out.println("Addition: "+res);
    }
}

```

```

        }
    }

Example:- No argument with return type
import java.util.*;
class Addition
{
    // No argument with return type
    int add()
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number 1:");
        int n1 = sc.nextInt();
        System.out.println("Enter number 2:");
        int n2 = sc.nextInt();
        int result = n1 + n2;
        return result;
    }
    public static void main(String args[])
    {
        Addition a = new Addition();
        int res = a.add();
        System.out.println("Addition: "+res);
    }
}

```

7.4 Recursion in Java: -

Recursion in java is a process in which a method calls itself continuously.

A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Syntax:

```

returntype methodname(){
    //code to be executed
    methodname();//calling same method
}

```

Example 1: -

```

class RecursionExample1 {
    void p(){
        System.out.println("hello");
    }
}

```

```
        p();  
    }  
  
public static void main(String[] args) {  
    RecursionExample1 R = new RecursionExample1();  
    R.p();  
}  
}
```

Output:-

```
hello  
hello  
...  
java.lang.StackOverflowError
```

Example 2:-

```
class RecursionExample2 {  
    static int count=0;  
    void p(){  
        count++;  
        if(count<=5){  
            System.out.println("hello "+count);  
            p();  
        }  
    }  
    public static void main(String[] args) {  
        RecursionExample2 R = new RecursionExample2();  
        R.p();  
    }  
}
```

Output:-

```
hello 1  
hello 2  
hello 3  
hello 4  
hello 5
```

Example 3:- Write a program to calculate factorial using recursion in Java

```
import java.util.*;
```

```

class RecursionExample {
    int factorial(int n){
        if (n == 1)
            return 1;
        else
            return(n * factorial(n-1));
    }

    public static void main(String[] args) {
        RecursionExample r = new RecursionExample();
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter Number: ");
        int n = sc.nextInt();
        System.out.println("Factorial of 5 is: "+r.factorial(n));
    }
}

```

Output: -

Enter Number: 4

Factorial of 5 is: 24

Example 4:- Write a Java program to calculate the power of a number like power(int number, int power) like power(2, 3) should return 8

```

import java.util.*;
class RecursionExample {
    int pow(int number, int power){
        if (power == 1)
            return number;
        else
            return(number * pow(number, power-1));
    }
}

```

```

public static void main(String[] args) {
    RecursionExample r = new RecursionExample();
    Scanner sc = new Scanner (System.in);
    System.out.print("Enter Number: ");
    int n = sc.nextInt();
    System.out.print("Enter Power: ");

```

```

        int p = sc.nextInt();
        System.out.println("Answer: "+r.pow(n,p));
    }
}

```

Output: -

```

Enter Number: 2
Enter Power: 3
Answer: 8

```

Example 5: - Write a Java program to convert Decimal to binary using recursion

```

import java.util.*;
class DecimalToBinary
{
    // Decimal to binary conversion
    int find(int decimal_number)
    {
        if (decimal_number == 0)
            return 0;
        else
            return (decimal_number % 2 + 10 *
                    find(decimal_number / 2));
    }

    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Decimal number:");
        int decimal_number = sc.nextInt();
        DecimalToBinary b = new DecimalToBinary();
        System.out.println(b.find(decimal_number));
    }
}

```