**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.
* 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.

**NOTE:**

The most important method in java is **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.**

**Syntax:**

public int sum (int a, int b)

{

//method body

}

* public int sum(int a, int b)🡪Method Header
* public🡪Access specifier
* int🡪Return Type
* sum🡪Method Name
* (int a, int b)🡪Parameter List
* sum(int a, int b)🡪Method Signature

**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.
* 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.
* Single-word method name🡪add(), sub()
* Multi-word method name🡪stringComparison(), areaOfCircle()

**Types of Method:**

2 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**.

**Example:** length(), equals(), compareTo(), sqrt()

**NOTE:**

* Each and every predefined method is defined inside a class.
* **print()** method is defined in the **java.io.PrintStream** class.

**Example: predefined method**

public class StringExample{

public static void main(String[] args){

String text = “Hello World”;

System.out.println(“Text length: ”+text.length());

}

}

**Output:**

Text length: 11

**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.

**Example:**

public class Calculator{

public static int add(int a, int b){

return a+b;

}

public static void main(String[] args){

int sum = add(10, 5);

System.out.println(“Sum: ”+ sum);

}

}

**Output:**

Sum: 15

**How to call or invoke a user-defined method**

* Once we have defined a method, it should be called.

**Example:**

java.util.Scanner;

public class Calculator{

public static int add(int a, int b){

return a+b;

}

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

System.out.println(“Enter value1: ”);

int num1 = sc.nextInt();

System.out.println(“Enter value2: ”);

int num2 = sc.nextInt();

add(num1,num3);

}

}

**Static method:**

* A method that has static keyword is known as static method.
* The main advantage of a static method is that we can call it without creating an object.

**Example:**

public class StaticMethodExample{

static void display(){

System.out.println(“This is example of static method”);

}

public static void main(String[] args){

}

}

**Output:**

This is 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 Calculator{

public static int add(int a, int b){

s = a+b;

return s;

}

public static void main(String[] args){

Calculator cal = new Calculator();

System.out.println(“The sum is: ”+ obj.add(10, 20));

}

}

**Output:**

The sum is: 30

**2 types of instance method:**

* Accessor Method
* Mutator Method

**Accessor Method:**

* The method that reads the instance variable is known as the accessor method.
* It is also known as getters.
* Method denotion: get()

**Example:**

public int getId()

{

return Id;

}

**Mutator method:**

* The method read the instance variable and also modify the values.
* It is also known as setters
* Method denotion: set()

**Example:**

Public void setRoll(int roll){

this.roll = roll;

}

**Example: Accessor and mutator method**

public class Register{

private String name;

private int regno;

public String getName(){

return name;

}

public String setName(String name){

this.name = name;

}

public int getRegno(){

return regno;

}

public int setRegno(int regno){

this.regno = regno;

}

}

**Default Method:**

**Sample format**

public void display(){

System.out.println(“This is default method”);

}

**Parameterized Method:**

**Sample format**

public void display(int a, int b){

System.out.println(a+b);

}

**CONSTRUCTOR**

* In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method.
* It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) 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.

**2 types of Constructors**

Default constructor (no-argument constructor)

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 constructors**

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

**NOTE:**

* We can use access modifiers while declaring a constructor.
* It controls the object creation.
* In other words, we can have private, public, protected and default constructor in java.

**Default Constructor:**

**Syntax:**

Class\_Name(){

}

**Example:**

public class Person{

String name;

int age;

public person(){

name = “Mugilan”;

age = 26;

}

public void displayInfo(){

System.out.println(“Name: ”+name);

System.out.println(“Age: ”+age);

}

public static void main(String[] args){

Person p1 = new Person();

p1.displayInfo();

}

}

**Output:**

Name: Mugilan

Age: 26

**Example:**

public class Person{

String name;

int age;

public void displayInfo(){

System.out.println(“Name: ”+name);

System.out.println(“Age: ”+age);

}

public static void main(String[] args){

Person p1 = new Person();

p1.displayInfo();

}

}

**Output:**

Name: null

Age: 0

**Parameterized Constructor:**

**Syntax:**

Class\_Name(arg1, arg2){

//Statement

}

**Example:**

public class ConstructorExample{

public ConstructorExample(int regno, String name){

System.out.println(“Regno: ”+regno);

System.out.println(“Name: ”+name);

}

public void displayName(){

String name = “Mugilan”;

System.out.println(“Another person name: ”+name);

}

public static void main(){

ConstructorExample ce = new ConstructorExample(101, “Sudesh”);

Ce.displayName();

}

}

**Output:**

Regno: 101

Name: Sudesh

Another person name: Mugilan

**Constructor Overloading:**

* Constructor overloading in java is a technique of having more than one constructor with different parameter lists.
* They are arranged in a way that each constructor performs a different task.
* They are arranged in a way that each constructor performs a different task.

**Example:**

package com.demo.JavaCode;  
  
public class StudentDetails {  
 private String name;  
 private int regno;  
 private String location;  
 public StudentDetails(String name, int regno, String location){  
 this.name = name; // 'this'-->refer to the instance variable  
 this.regno = regno; // 'this'-->refer to the instance variable  
 this.location = location;  
  
 }  
 public StudentDetails(String name, String location){  
 this(name,0,location); //"this" to call another constructor  
 }  
 public StudentDetails(String name, int regno){  
 this(name,regno,"null"); //"this" to call another constructor  
 }  
 public void displayinfo(){  
 System.out.println("Name:"+this.name);  
 System.out.println("Regno:"+this.regno);  
 System.out.println("Location:"+this.location);  
 }  
 public static void main(String[] args){  
 StudentDetails s1 = new StudentDetails("Mugilan", 4128,"Vellore");  
 StudentDetails s2 = new StudentDetails("Sudesh","tirupur");  
 StudentDetails s3 = new StudentDetails("Vasanth",52);  
 s1.displayinfo();  
 s2.displayinfo();  
 s3.displayinfo();  
 }  
}

**Output:**

Name:Mugilan

Regno:4128

Location:Vellore

Name:Sudesh

Regno:0

Location:tirupur

Name:Vasanth

Regno:52

Location:null

**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++.

**Ways to copy the values of one object into another**

* By constructor.
* By assigning the values of one object into another.
* By clone() method of object class.

**Example:**

class Person {

private String name;

private int age;

// Regular constructor

public Person(String name, int age) {

this.name = name;

this.age = age;

}

// Copy constructor

public Person(Person other) {

this.name = other.name;

this.age = other.age;

}

// Getter methods for demonstration

public String getName() {

return name;

}

public int getAge() {

return age;

}

// Method to display person's details

public void displayInfo() {

System.out.println("Name: " + name + ", Age: " + age);

}

}

public class Main {

public static void main(String[] args) {

// Create an original Person object

Person originalPerson = new Person("Mugilan", 30);

// Create a copy of the original Person object using the copy constructor

Person copiedPerson = new Person(originalPerson);

// Display the information of both objects

System.out.println("Original Person:");

originalPerson.displayInfo();

System.out.println("Copied Person:");

copiedPerson.displayInfo();

}

}

**Output:**

Original Person:

Name: Alice, Age: 30

Copied Person:

Name: Alice, Age: 30

***this* keyword:**

In Java, this is a ***reference variable***that refers to the current object.

A diagram of a variable

Description automatically generated

**Usage of java this keyword:**

6 Usage of java this keyword

* this can be used to refer current class instance variable.
* this can be used to invoke current class method (implicitly).
* this() can be used to invoke current class constructor.
* this can be passed as an argument in the method call.
* this can be passed as an argument in the constructor call.
* this can be used to return the current class instance from the method.

**Example:**

class Person {

private String name;

private int age;

// Constructor with parameters

public Person(String name, int age) {

this.name = name; // 'this.name' refers to the instance variable

this.age = age; // 'this.age' refers to the instance variable

}

// Method to set the name

public void setName(String name) {

this.name = name; // 'this.name' refers to the instance variable

}

// Method to set the age

public void setAge(int age) {

this.age = age; // 'this.age' refers to the instance variable

}

// Method to display person information

public void displayInfo() {

System.out.println("Name: " + this.name + ", Age: " + this.age);

}

}

public class Main {

public static void main(String[] args) {

// Create a Person object

Person person = new Person("Mugilan", 26);

// Display initial information

person.displayInfo();

// Modify person details

person.setName("Sudesh");

person.setAge(29);

// Display updated information

person.displayInfo();

}

}

**Output:**

Name: Mugilan, Age: 26

Name: Sudesh, Age: 29