**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.

• **Syntax of Methods**



* **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.

**•** **Static**

* 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.

**• Call static methods & variables**

* The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class). The static keyword belongs to the class than an instance of the class.
* The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

**Java Static Variable**

* If you declare any variable as static, it is known as a static variable.
* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

class Student{

int rollno: //instance specific

String name;

Static String college=”ITS”;

//common to class specific

}



**Java Static Method**

* If you apply static keyword with any method, it is known as static method.
* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

**Restrictions for the static method**

There are two main restrictions for the static method. They are:

1. The static method cannot use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

**When to use static variables and methods?**

* Use the static variable for the property that is common to all objects.
* For example, In class Student, all students share the same college name.
* Use static methods for changing static variables.

**Syntax to declare the static method:**

**Access\_modifier static void methodName()**

**{**

**// Method body.**

**}**

The name of the class can be used to invoke or access static methods.

**Syntax to call a static method:**

**className.methodName();**

• **Invoking Parameterized Methods & different Return types**

* **Parameterizedmethods**: These methods contain a parameter list or an argument list which receives a value from the calling method.
* The **parameter** is the variable definition in the method definition. Their type is specified in the parameter and is separated with commas. It is important to understand what type of **parameter** a function has and how it operates on them. There are two types of parameters: Formal Parameter and Actual Parameter*.*
* **Formal Parameter**: **Formal parameter** is the one that is present in the function definition. This parameter receives the arguments that are passed to the method.

For example:

* public int sum(int a, int b){
* //body
* }
* **Actual Parameter**: **Actual parameter** is the one that is present in the function call statement. This parameter sends the arguments that are passed to the method.

For example:

* ob.sum(num1,num2); //this is the actual parameter

• **Method Overloading with static methods**

We can have two or more static methods with the same name, differences in input parameters.

**Overloading:**

Overloading is also a feature of OOP languages like Java that is related to compile-time (or static) polymorphism.

This feature allows different methods to have the same name, but different signatures, especially the number of input parameters and type of input parameters.

**• Instance (explain briefly)**

* In Java, an "instance" typically refers to an object created from a class.
* A class is a blueprint or template that defines the structure and behaviour of objects, while an instance is a specific realization of that class.
* The properties (variables) and behaviours (methods) that objects created from a class in Java will have are similarly defined by the class.
* Instances are the name for these things.
* Every instance has a unique state that describes the values of its characteristics at any given moment.
* Although an instance's state may vary as the programme executes, it continues to exist independently of other instances of the same class.
* **Instance Variables:** Instances have their own set of instance variables, also known as member variables or attributes. These variables hold specific values for each instance and can be accessed and modified within the instance's methods.
* **Instance Methods:**Along with instance variables, instances also have associated instance methods. These methods define the behaviour of the instance and can access and manipulate the instance's variables. Instance methods can be invoked on specific instances to perform operations related to that instance.

**Syntax:**

**modifier return\_type method\_name( )**

**{**

**method body ;**

**}**

**modifier:** It defines the access type of the method, and it is optional to use.

**return\_type:** Method may return a value. Ex:- int, void, String, char, float, etc.

**method\_name:** This is the method name you can write anything as you write the variable name.

**method body:** The method body describes what the method does with statements.

**Difference Between the Static method and Instance method**

| **Instance Methods** | **Static Methods** |
| --- | --- |
| It requires an object of the class. | It doesn’t require an object of the class. |
| It can access all attributes of a class. | It can access only the static attribute of a class. |
| The methods can be accessed only using object reference. | The method is only accessed by class name. |
| **Syntax: Objref.methodname()** | **Syntax: className.methodname()** |
| It’s an example of pass-by-value programming. | It is an example of pass-by-reference programming. |