1. Method and Constructor.

A **method** in Java is a block of code that performs a specific task. It is defined within a class and can be called to execute its code. Methods can take parameters and can return a value. They help in organizing code into reusable components, making programs more modular and easier to maintain.

Characteristics of Methods:

- **Name**: Each method has a unique name within its class.
- **Parameters**: Methods can take input parameters, which allow passing data into the method.
- **Return Type**: Methods can return a value of a specified type, or they can be declared as `void` if they do not return anything.
- **Body**: The method contains a body where the actual code is executed.

Constructor

A **constructor** is a special type of method that is called when an instance (object) of a class is created. Its primary purpose is to initialize the object's attributes. Constructors do not have a return type and must have the same name as the class.

Characteristics of Constructors:

- **Same Name as Class**: The constructor must have the same name as the class.
- **No Return Type**: Constructors do not have a return type, not even `void`.
- **Called Automatically**: A constructor is invoked automatically when an object is created.
- **Can be Overloaded**: Like regular methods, constructors can also be overloaded.

2. Pass by Value and Pass by Reference.

In programming, especially in languages like Java, understanding how data is passed to methods is crucial. The two main concepts are **pass by value** and **pass by reference**. Here's a breakdown of each:

Pass by Value

Pass by Value means that when you pass a variable to a method, a copy of that variable is made. Any changes made to the parameter inside the method do not affect the original variable outside of the method.

- **Primitive Data Types**: In Java, primitive types (like `int`, `float`, `char`, etc.) are passed by value. This means that when you pass a primitive type to a method, the method operates on a copy of the original value.

Pass by Reference

Pass by Reference means that when you pass an object to a method, you are passing a reference (or address) to that object, not the object itself. This allows the method to modify the object's state, affecting the original object.

- **Reference Data Types**: In Java, reference types (like objects and arrays) are passed by reference. This means that changes made to the object inside the method affect the original object.

3. **Public, Private and Protected.**

1. Public

- **Definition**: The `public` modifier allows the class, method, or variable to be accessible from any other class in any package.
- **Usage**: Use `public` when you want to expose functionality that can be accessed from anywhere in your application.

2. Private

- **Definition**: The `private` modifier restricts access to the class, method, or variable to only within the defining class. No external classes can access private members.
- **Usage**: Use `private` to encapsulate data and ensure that it can only be modified through methods within the class, promoting data hiding.

3. Protected

- **Definition**: The `protected` modifier allows access to the class, method, or variable from the same package and by subclasses (even if they are in different packages).
- **Usage**: Use `protected` when you want to allow access to subclasses while still restricting access from unrelated classes.

4 . Instance variable and Static Variable.

In Java, variables can be classified into two main types: **instance variables** and **static variables**. Each type serves a different purpose and has distinct characteristics. Here's a detailed explanation of both:

Instance Variables

- **Definition**: Instance variables are non-static variables that are declared within a class but outside any method. Each object (instance) of the class has its own copy of instance variables.
- **Scope**: They are accessible within all methods of the class and are tied to a specific instance of the class.
- **Lifetime**: Instance variables are created when an object is instantiated and destroyed when the object is destroyed.
- **Usage**: Use instance variables to represent the state or attributes of an object.

Static Variables

- **Definition**: Static variables are variables that are declared with the `static` keyword. They are shared among all instances of a class. There

is only one copy of a static variable, regardless of how many objects are created.

- **Scope**: Static variables can be accessed directly by static methods and by instance methods. They belong to the class rather than to any specific instance.
- **Lifetime**: Static variables are created when the class is loaded and destroyed when the class is unloaded.
- **Usage**: Use static variables to represent properties or data that should be shared among all instances of a class.