Java Methods:

A **method** in Java is a block of code that performs a specific task. Methods help in code reusability and organization. You define a method once and can use it multiple times in your program. Methods can take inputs, perform operations, and return outputs.

Basic Structure of a Method:

```
returnType methodName(parameters) {
    // Method body
    // Code to perform the task
    return value; // Optional, depends on the return type
}
```

Types of Java Methods:

There are two type of functions there in Java Language. They are,

Library methods or Predefined methods: Library methods are those which are predefined in java compiler/libraries.

The implementation part of predefined methods is available in java library files(JAR) that are .class files are contained precompiled code. println(), nextInt(), next() etc. are predefined methods.

Limitations of predefined methods:

All predefined methods are contained limited task only. that is for what purpose method is designed for same purpose it should be used.

As a programmer we do not have any controls on predefined method implementation part, it is in machine readable(binary) format.

In implementation whenever a predefined method is not supporting user requirements then we go for user defined methods.

User defined methods:

Defining of method is nothing but give body of method that means write logic inside method body.

Syntax:

access-modifier non-access modifier return_type method_name(parameters)

{

//method body

//statements;

}

Return type: A method may return a value. The return_type is the data type of the value the method returns. Return type parameters and returns statement are optional.

Method name: Method name is the name of method it is decided by programmer.

Parameters: This is a value which is pass in method at the time of calling of method, A parameter is like a placeholder.

Method body: Method body is the collection of statements.

Recursion:

Recursion **i**s a process where a method calls itself. This is useful for tasks that can be broken down into simpler, repetitive tasks. However, recursion needs a base case to avoid infinite loops.

Example of Recursion:

```
public class FactorialExample {
    public static int factorial(int n) {
        if (n == 0) { // Base case
            return 1;
        } else {
            return n * factorial(n - 1); // Recursive call
        }
    }
```

```
    public static void main(String[] args) {
        int result = factorial(5);
        System.out.println("Factorial of 5 is: " + result);
    }
}
```

Output:

Factorial of 5 is: 120

Calling Methods:

To call a method, simply use the method's name followed by parentheses. If the method requires parameters, you need to pass them inside the parentheses.

Syntax:

method-name();

Note: At the time of method calling, method must be terminated with ' ; '.

Example of Calling a Method:

```
public class Calculator {
    public static int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int sum = add(10, 20); // Calling the add method
        System.out.println("Sum is: " + sum);
    }
}
```

Output:

Sum is: 30

About Java Object Oriented Programming:

Java is an object-oriented programming language, which means it uses objects to represent data and methods to manipulate that data. OOP focuses on four main principles:

1. **Encapsulation**: Bundling the data (attributes) and methods (functions) that operate on the data into a single unit called a class.
2. **Abstraction**: Hiding the complex implementation details and showing only the necessary features.
3. **Inheritance**: Allowing a new class to inherit the properties and methods of an existing class.
4. **Polymorphism**: Allowing methods to do different things based on the object it is acting upon.

Class:

A class is a blueprint for creating objects. It defines the properties (attributes) and methods (functions) that the created objects will have.

Example of a Class:

```
public class Car {
   // Class properties (attributes)
   String color;
   String model;
   int year;


   // Method (function)
   void displayDetails() {
      System.out.println("Model: " + model + ", Color: " + color + ", Year: " + year);
   }
}
```

Class Properties:

Class properties, also known as attributes or fields, are variables defined inside a class. They represent the state or data of the class.

Objects:

An object is an instance of a class. When a class is defined, no memory is allocated until an object of that class is created.

Example:

```
public class Main {
    public static void main(String[] args) {
        // Creating an object of the Car class
        Car myCar = new Car("Toyota", "Camry", 2021);

        // Accessing class properties using the object
        myCar.displayDetails();
    }
}
```

new Keyword:

The new keyword is used in Java to create a new instance (object) of a class. When you use new, memory is allocated for the object, and the constructor of the class is called to initialize the object.

Example:

```
public class Main {
    public static void main(String[] args) {
        // Creating an object of the Car class using the new keyword
        Car myCar = new Car("Toyota", "Camry", 2021);

        // Creating another object of the Car class
        Car anotherCar = new Car("Honda", "Civic", 2020);

        // Accessing properties and methods of the objects
```

```
        myCar.displayDetails();
        anotherCar.displayDetails();
    }
}
```

Object Creation:

Object creation in Java refers to the process of creating an instance of a class. This process involves allocating memory for the new object and initializing it using the class constructor. An object is a runtime entity that encapsulates data (attributes) and behaviour (methods) defined by its class.

Example of Object Creation:

```
public class Main {
    public static void main(String[] args) {
        // Creating an object of the Car class
        Car myCar = new Car();
        myCar.color = "Red";
        myCar.model = "Toyota";
        myCar.year = 2021;

        // Calling the method
        myCar.displayDetails();
    }
}
```

Output:

Model: Toyota, Color: Red, Year: 2021

Instance Of Keyword:

The instance of keyword is used to test whether an object is an instance of a specific class or subclass.

Example of instance Of Keyword:

```
if (myCar instanceof Car) {
    System.out.println("myCar is an instance of the Car class");
}
```

About Constructor:

A constructor in Java is a special type of method used to initialize objects. When an object of a class is created using the new keyword, the constructor is automatically called to set up the initial state of the object.

Default Constructor: Provided by the compiler if no constructor is defined. It initializes the object with default values.
Parameterized Constructor: Allows you to pass arguments to initialize an object with specific values.

Example of a Default Constructor:

```
public class Car {
    String model;
    String color;
    int year;

    // Default Constructor
    public Car() {
        model = "Unknown";
        color = "White";
        year = 2020;
    }

    void displayDetails() {
        System.out.println("Model: " + model + ", Color: " + color + ", Year: " + year);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Creating an object using the default constructor
        Car myCar = new Car();
        myCar.displayDetails();
    }
}
```

OUTPUT:

```
Model: Unknown, Color: White, Year: 2020
```

Example of a Parameterized Constructor:

```
public class Car {
    String model;
    String color;
    int year;

    // Parameterized Constructor
    public Car(String model, String color, int year) {
        this.model = model;
        this.color = color;
        this.year = year;
    }

    void displayDetails() {
        System.out.println("Model: " + model + ", Color: " + color + ", Year:
" + year);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Creating an object using the parameterized constructor
        Car myCar = new Car("Toyota", "Red", 2021);
        myCar.displayDetails();
    }
}
```

Output:

```
Model: Toyota, Color: Red, Year: 2021
```