Methods and Loops in Java

What are Methods in Java?

- Methods are blocks of code that perform specific tasks.
- A method can take input (parameters), process it, and optionally return a result.

Method Structure

```
returnType methodName(parameters) {
   // Method body
   return returnValue; // (if applicable)
}
```

Return Type: The data type of the value the method returns. Use void if the method doesn't return anything.
Method Name: The name of the method, which should be descriptive of the task it performs.
parameters: The input data that the method uses to perform its task.
Method body: The set of statements enclosed within curly braces {} that define what the method does.
return: If the method has a return type other than void, it should return a value of that type using the return keyword.

Example: Print Jai Shree RAM

```
public class RAM {

  public static void main(String[] args) {
    System.out.println("Jai Shree RAM");
  }
```

Example: Adding Two Numbers

```
public class Calculator {

  public static void main(String[] args) {
    Calculator calculator = new Calculator();
    int result = calculator.add(5, 7);
    System.out.println("Sum: " + result);
  }

  public int add(int num1, int num2) {
    int sum = num1 + num2;
    return sum;
  }

}
```

Functions with Modifiers in Java:

In Java, methods can have modifiers that control their accessibility with the help of Access modifiers.

Access Modifiers

Access modifiers determine the visibility of methods in different parts of your program. There are four main access modifiers in Java:

public: Methods with this modifier are accessible from anywhere.
private: Methods are accessible only within the same class.
protected: Methods are accessible within the same package and subclasses, even if they are in different packages.
default (no modifier): Methods are accessible within the same package.

Example:

```
public class Example {
   public void publicMethod() {
      // This method is accessible from anywhere.
   }

   private void privateMethod() {
      // This method is accessible only within this class.
   }

   protected void protectedMethod() {
      // This method is accessible within the package and by subclasses.
   }

   void defaultMethod() {
      // This method is accessible within the same package.
   }
}
```

Other Modifiers

static: Methods declared as static belong to the class rather than instances of the class. They can be called using the class name without creating objects.

final: A final method cannot be overridden by subclasses. It remains the same across the class hierarchy.

Example 2:

```
class Car {
   public void startEngine() {
      System.out.println("Engine started.");
   }

   private void stopEngine() {
      System.out.println("Engine stopped.");
   }

   protected void accelerate() {
      System.out.println("Car accelerating.");
   }

   void honkHorn() {
      System.out.println("Honk honk!");
   }
}

public class Main {
   public static void main(String[] args) {
      Car myCar = new Car();
      myCar.startEngine();    // Accessible (public)
      // myCar.stopEngine();  // Not accessible (private)
      myCar.accelerate();     // Accessible (protected)
      myCar.honkHorn();       // Accessible (default)
   }
}
```

static Method Examples:

```
class Test {
   public static int multiply(int a, int b) {
      return a * b;
   }
}

public class Main {
   public static void main(String[] args) {
      int result = Test.multiply(5, 3);
      System.out.println("Result: " + result);
   }
}
```

final Method Example:

```
class Shape {
    public final void displayInfo() {
        System.out.println("This is a shape.");
    }
}

class Circle extends Shape {
    // Cannot override displayInfo() due to final modifier
    // public void displayInfo() { /* ... */ }
}

public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle();
        circle.displayInfo();  // Will call the final method
    }
}
```

protected Method Example:

eg1:

```
class Animal {
    protected void makeSound() {
        System.out.println("Animal makes a sound.");
    }
}

class Dog extends Animal {
    protected void makeSound() {
        System.out.println("Dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound();  // Calls the overridden makeSound() in Dog class
    }
}
```

Eg 2:

```
// Package: com.example.animals
class Animal {
    protected void roam() {
        System.out.println("Animal roams around.");
    }
}

// Package: com.example.zoo
import com.example.animals.Animal;

class ZooKeeper {
    void feedAnimal(Animal animal) {
        animal.roam();  // Protected method can be accessed due to inheritance
    }
}
```

Loops in Java

Loops are control structures that allow you to execute a block of code repeatedly based on a condition.

Types of loops:

- for
- while
- do while
- foreach

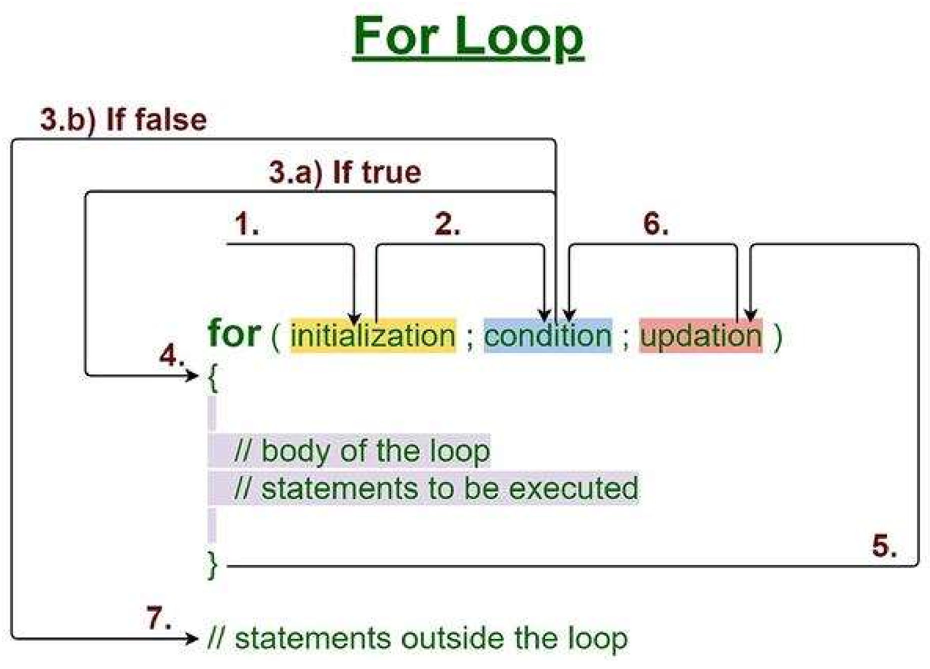
for loop syntax:

The for loop is used when you know how many times you want to execute a code block.

```
for (initialization; condition; update) {
    // Code to be executed repeatedly
}
```

Example: print number

```
for (int i = 1; i <= 5; i++) {
   System.out.print(i + " ");
}
// Output: 1 2 3 4 5
```


For Loop
3.b) If false
3.a) If true
1.
2.
6.
for (initialization ; condition ; updation)
4.
{
// body of the loop
// statements to be executed
}
5.
7.
// statements outside the loop