Method Overloading:

Java supports method overloading, which allows you to define multiple methods with the same name in the same class. These methods must have different parameter lists (number or types of parameters). Overloaded methods are differentiated by their parameter lists.

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

    public static double add(double a, double b) {
        return a + b;
    }

    public static String add(String s1, String s2) {
        return s1 + s2;
    }

    public static void main(String[] args) {
        int sum1 = add(5, 10);
        double sum2 = add(3.5, 2.5);
        String combined = add("Hello, ", "World!");

        System.out.println("Sum of integers: " + sum1);
        System.out.println("Sum of doubles: " + sum2);
        System.out.println("Combined strings: " + combined);
    }
}
```

In this example, the `add` method is overloaded to handle different types of data and parameters. Java determines which version of the method to call based on the arguments passed.

Method Recursion:

Recursion is a technique where a method calls itself to solve a problem. It's commonly used for problems that can be broken down into smaller sub-problems. Recursion must have a base case to prevent infinite looping.

```java
```

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

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

In this example, the `factorial` method calculates the factorial of a number using recursion. The base case is when `n` is 0 or 1. The method calls itself with a smaller value of `n` until the base case is reached.

Varargs (Variable-Length Arguments):

Java allows you to define methods with variable-length argument lists using the varargs feature. This allows you to pass a variable number of arguments of the same type.

```java
public class VarargsExample {
    public static int sum(int... numbers) {
        int total = 0;
        for (int num : numbers) {
            total += num;
        }
        return total;
    }

    public static void main(String[] args) {
        int sum1 = sum(1, 2, 3);
        int sum2 = sum(5, 10, 15, 20);
        System.out.println("Sum 1: " + sum1);
        System.out.println("Sum 2: " + sum2);
    }
}
```

In this example, the `sum` method accepts a variable number of integers as arguments. You can call the method with different numbers of arguments, and Java packs them into an array for you.

Method Access Modifiers:

Java methods can have access modifiers that control their visibility and accessibility from other classes. The common access modifiers are `public`, `private`, `protected`, and package-private (no modifier).

```java
public class AccessModifierExample {
    public void publicMethod() {
        // Code
    }

    private void privateMethod() {
        // Code
    }

    protected void protectedMethod() {
        // Code
    }

    void packagePrivateMethod() {
        // Code
    }
}
```

public class checking balance {
    public void publicMethod() {
        /code
}

In this example, methods have different access modifiers, affecting where they can be accessed from. `public` methods can be accessed from anywhere, while `private` methods are limited to the same class, and `protected` methods allow access within the same package and subclasses.

These additional concepts should provide you with a deeper understanding of Java methods and how to use them effectively in various scenarios.