**Session: Methods and Access Specifiers in Java**

**Introduction**

* **Methods**: Represent the behavior or functionality of a class.
* **Access Specifiers**: Control the visibility of methods and fields across different parts of a program.

**Topics Covered**

1. What are Methods?
2. Defining and Using Methods.
3. Types of Methods.
4. Access Specifiers in Java.
5. Combining Methods and Access Specifiers.
6. Best Practices.

**1. What are Methods?**

* A method is a block of code designed to perform a specific task.
* It enhances code reusability and modularity.

**Syntax**:

java

**returnType** methodName(parameters) {

// Method body

}

**Example**:

java

public **int** add(int a, int b) {

return a + b;

}

**2. Defining and Using Methods**

**Steps to Define a Method:**

1. Specify the **return type** (e.g., void, int, String).
2. Provide a **name** for the method.
3. Add optional **parameters** inside parentheses.
4. Write the method body with the logic.

**Calling a Method:**

* Use the method name followed by parentheses.
* Pass arguments if the method requires parameters.

**Example**:

java

class Calculator {

int add(int a, int b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

Calculator calc = new Calculator();

int result = calc.add(5, 10);

System.out.println("Result: " + result);

}

}

**Output**:

Result: 15

**3. Types of Methods**

**1. Instance Methods:**

* Belong to an object.
* Require an object to be invoked.

**2. Static Methods:**

* Belong to the class rather than objects.
* Can be called using the class name.

**Example**:

java

class Utility {

static int square(int x) {

return x \* x;

}

}

public class Main {

public static void main(String[] args) {

int result = Utility.square(4);

System.out.println("Square: " + result);

}

}

**Output**:

Square: 16

**3. Parameterized Methods:**

* Accept input values as parameters.

**4. Method Overloading:**

* Allows multiple methods with the same name but different parameters.

**4. Access Specifiers in Java**

Access specifiers define the **visibility** of a method or variable.

| **Access Specifier** | **Within Class** | **Within Package** | **Subclass** | **Everywhere** |
| --- | --- | --- | --- | --- |
| **public** | ✔ | ✔ | ✔ | ✔ |
| **protected** | ✔ | ✔ | ✔ | ✘ |
| **default** | ✔ | ✔ | ✘ | ✘ |
| **private** | ✔ | ✘ | ✘ | ✘ |

**Examples:**

1. **Public**:
   * Accessible everywhere.

java

public void display() {

System.out.println("This is public");

}

1. **Private**:
   * Accessible only within the class.

java

private void secret() {

System.out.println("This is private");

}

1. **Protected**:
   * Accessible in the same package and subclasses.

java

protected void show() {

System.out.println("This is protected");

}

1. **Default (no modifier)**:
   * Accessible only within the same package.

java

void packageMethod() {

System.out.println("This is package-private");

}

**5. Combining Methods and Access Specifiers**

**Example:**

java

class Person {

private String name;

// Public method to set name

public void setName(String name) {

this.name = name;

}

// Public method to get name

public String getName() {

return name;

}

// Private method for internal logic

private void validateName() {

System.out.println("Validating name...");

}

}

**Usage:**

java

public class Main {

public static void main(String[] args) {

Person person = new Person();

person.setName("Alice");

System.out.println("Name: " + person.getName());

}

}

**6. Best Practices**

1. **Encapsulation**: Keep fields private and use public getter/setter methods.
2. **Method Naming**: Use meaningful names (e.g., calculateArea instead of calc).
3. **Static Methods**: Use them for utility or helper functions.
4. **Access Specifiers**:
   * Use private for sensitive data.
   * Use protected for inheritance.
   * Use public for universal access.

**Hands-On Exercise**

1. Create a BankAccount class with:
   * Private fields: accountNumber, balance.
   * Public methods: deposit, withdraw, getBalance.
2. Write a main class to:
   * Create a BankAccount object.
   * Perform deposit and withdrawal operations.

**Conclusion**

* **Methods** allow you to define behaviors for your objects.
* **Access Specifiers** control how methods and fields are accessed, enabling secure and modular programming.

Encourage questions related to:

1. Overloading methods.
2. Static vs. instance methods.
3. Practical use cases of access specifiers.