**Session: Understanding Classes and Objects in Java**

**Introduction**

* Java is an **object-oriented programming (OOP)** language.
* The core concepts of OOP revolve around **classes** and **objects**.
* A **class** is a blueprint for creating objects.
* An **object** is an instance of a class, representing a real-world entity with state and behavior.

**Topics Covered**

1. What is a Class?
2. What is an Object?
3. Defining and Using Classes and Objects.
4. Member Variables and Methods.
5. Real-World Analogy.
6. Best Practices.

**1. What is a Class?**

* A class is a template or blueprint for creating objects.
* It defines the **properties (fields)** and **behaviors (methods)** that the objects will have.

**Syntax**:

java

class ClassName {

// Fields (properties)

// Methods (behaviors)

}

**Example**:

java

class Car {

// Fields

String brand;

int speed;

// Method

void displayDetails() {

System.out.println("Brand: " + brand);

System.out.println("Speed: " + speed);

}

}

**2. What is an Object?**

* An object is an instance of a class.
* It is created using the new keyword.

**Syntax**:

java

ClassName objectName = new ClassName();

**Example**:

java

public class Main {

public static void main(String[] args) {

Car myCar = new Car(); // Creating an object of the Car class

myCar.brand = "Toyota";

myCar.speed = 120;

myCar.displayDetails();

}

}

**Output**:

makefile

Brand: Toyota

Speed: 120

**3. Defining and Using Classes and Objects**

**Step-by-Step**

1. **Define a class** with fields and methods.
2. **Create objects** of the class using the new keyword.
3. Access fields and call methods using the object reference.

**4. Member Variables and Methods**

**Member Variables (Fields):**

* Represent the state or attributes of an object.
* Declared inside the class but outside any method.

**Methods:**

* Represent the behavior or actions of an object.
* Can operate on fields of the class.

**Example**:

java

class Person {

// Fields

String name;

int age;

// Method

void introduce() {

System.out.println("Hi, I'm " + name + " and I'm " + age + " years old.");

}

}

**Usage**:

java

public class Main {

public static void main(String[] args) {

Person person = new Person();

person.name = "Alice";

person.age = 30;

person.introduce();

}

}

**Output**:

css

Hi, I'm Alice and I'm 30 years old.

**5. Real-World Analogy**

* A **class** is like a blueprint for a house.
* An **object** is the actual house built using that blueprint.
* For example:
  + **Class**: Car
    - Fields: brand, speed.
    - Methods: start(), stop().
  + **Objects**: "Toyota Car", "Honda Car".

**6. Best Practices**

1. Use meaningful class and field names:
   * Good: Car, brand, speed.
   * Bad: C, b, s.
2. Keep fields private and access them through public methods (**Encapsulation**).
3. Initialize objects before accessing fields.
4. Keep methods focused on a single responsibility.

**Hands-On Exercise**

**Task**: Create a Book class with the following:

1. Fields: title, author, price.
2. Method: displayDetails() to print the book details.
3. Write a Main class to:
   * Create a Book object.
   * Set the fields.
   * Call displayDetails().

**Conclusion**

* Classes and objects form the backbone of Java programming.
* By understanding how to define and use them, you lay a strong foundation for mastering object-oriented programming concepts.

**Q&A Session**:  
Encourage learners to ask questions related to:

1. Multiple objects from a single class.
2. Real-world applications of classes and objects.