# Week 2 Notes - Java OOPs

Object-Oriented Programming (OOP) is a core concept in Java and a frequent topic in technical interviews. Understanding its principles and applications is essential for designing maintainable and scalable code.

## 1. Core Principles of OOP

• Class and Object: A class is a blueprint; an object is an instance of a class. Use objects to access variables and methods.

• Encapsulation: Keeping data (variables) and code (methods) safe from outside interference. Use `private` access modifiers with public getters/setters.

• Inheritance: A mechanism to acquire properties and behavior from a parent class. Use `extends` keyword.

• Polymorphism: Ability to take many forms. Two types: Compile-time (method overloading) and Runtime (method overriding).

• Abstraction: Hiding complex logic and showing essential features. Use abstract classes or interfaces.

## 2. Access Modifiers

• private: Accessible only within the class.

• default (no modifier): Accessible within the package.

• protected: Accessible within package and subclass.

• public: Accessible everywhere.

## 3. Frequently Asked Concepts

• Constructor: Special method to initialize objects. Types: Default, Parameterized, Copy.

• ‘this’ keyword: Refers to the current object. Used to avoid confusion with local variables.

• ‘super’ keyword: Refers to the immediate parent class. Used to call parent constructor or method.

• Static vs Non-static: Static belongs to the class; non-static to instances.

• Final Keyword: Final class can’t be extended, final method can’t be overridden, final variable = constant.

• instanceof Operator: Checks whether an object is an instance of a class or subclass.

## 4. Abstract Classes vs Interfaces

• Abstract class can have both abstract and non-abstract methods; can have constructors.

• Interface can only have abstract methods (Java 7), can have default and static methods (Java 8+).

• A class can implement multiple interfaces but extend only one class.

## 5. Best Practices and Coding Tips

• Use meaningful class and method names.

• Keep methods short and focused on a single task.

• Use encapsulation to protect sensitive data.

• Apply OOP concepts to improve modularity and reduce code duplication.