**Section 47 — OOP in Kotlin**

**1. Introduction to OOP in Kotlin**

* **OOP (Object-Oriented Programming)** is about organizing code into **objects** that combine **data (properties)** and **behavior (functions)**.
* Kotlin supports all core OOP principles:
  1. **Encapsulation**
  2. **Inheritance**
  3. **Polymorphism**
  4. **Abstraction**

**2. Classes in Kotlin**

* **Class** is a blueprint for creating objects.
* **Basic class:**

kotlin

CopyEdit

// Class with primary constructor

class Person(val name: String, var age: Int) {

fun greet() {

println("Hello, my name is $name and I am $age years old.")

}

}

// Object creation

val p1 = Person("Harsh", 22)

p1.greet()

* **Primary Constructor** is part of the class header.
* **Secondary Constructors** → Alternative ways to create objects.

kotlin

CopyEdit

class Car(val brand: String) {

var model: String = "Unknown"

constructor(brand: String, model: String) : this(brand) {

this.model = model

}

}

**3. Properties in Kotlin**

* **val** → Read-only property (getter only).
* **var** → Mutable property (getter + setter).
* Custom getters & setters:

kotlin

CopyEdit

class Circle(val radius: Double) {

val area: Double

get() = Math.PI \* radius \* radius // calculated property

}

**4. Encapsulation**

* Restricting access to class members using **visibility modifiers**:
  + public (default) → accessible everywhere.
  + private → accessible only inside the class.
  + protected → accessible in class and subclasses.
  + internal → accessible within the same module.
* Example:

kotlin

CopyEdit

class BankAccount(private var balance: Double) {

fun deposit(amount: Double) {

if (amount > 0) balance += amount

}

fun getBalance() = balance

}

**5. Inheritance**

* **open** keyword → Allows class/method to be inherited.

kotlin

CopyEdit

open class Animal {

open fun sound() {

println("Animal makes a sound")

}

}

class Dog : Animal() {

override fun sound() {

println("Dog barks")

}

}

* **Best practice:** Keep classes final unless explicitly meant for inheritance.

**6. Polymorphism**

* Ability to use the **same interface/method name** for different implementations.
* Example with method overriding:

kotlin

CopyEdit

val animal: Animal = Dog()

animal.sound() // "Dog barks"

* **Runtime polymorphism** occurs when the actual implementation is chosen at runtime.

**7. Abstraction**

* **Abstract classes** → Cannot be instantiated, may have abstract methods (no body).

kotlin

CopyEdit

abstract class Shape {

abstract fun area(): Double

}

class Rectangle(val w: Double, val h: Double) : Shape() {

override fun area() = w \* h

}

* **Interfaces** → Define methods without implementation (multiple interfaces allowed).

kotlin

CopyEdit

interface Clickable {

fun onClick()

}

**8. Data Classes**

* For **model/data holding** without writing boilerplate (toString(), equals(), hashCode()).

kotlin

CopyEdit

data class User(val id: Int, val name: String)

* Useful in Android for **API responses, database entities**.

**9. Object Declaration & Singleton**

* Kotlin supports **singleton pattern** via object.

kotlin

CopyEdit

object Logger {

fun log(msg: String) {

println("Log: $msg")

}

}

Logger.log("App started")

**10. Companion Objects (Static Members)**

* Replaces Java’s static keyword.

kotlin

CopyEdit

class MathUtils {

companion object {

const val PI = 3.14

fun square(x: Int) = x \* x

}

}

println(MathUtils.square(5))

**11. Nested & Inner Classes**

* **Nested class** → No reference to outer class.
* **Inner class** → Can access outer class members.

kotlin

CopyEdit

class Outer {

private val secret = "hidden"

inner class Inner {

fun reveal() = secret

}

}

**12. Overloading**

* Kotlin supports **function overloading**:

kotlin

CopyEdit

class Calculator {

fun add(a: Int, b: Int) = a + b

fun add(a: Double, b: Double) = a + b

}

**13. Overriding**

* Use override keyword when modifying parent class method behavior.

**14. Constructors Recap**

* Primary constructor → Declared in header.
* Secondary constructor → Declared inside class with constructor.

**15. this Keyword**

* Refers to **current instance** of the class.
* Used when variable names clash or to call other constructors.

**16. super Keyword**

* Refers to **parent class** implementation.
* Example:

kotlin

CopyEdit

override fun sound() {

super.sound()

println("Extra sound")

}

**17. Access Modifiers Recap**

* **public** (default)
* **private**
* **protected**
* **internal**

**Tools, Libraries, APIs**

* Pure Kotlin OOP features (no extra dependencies).
* Often combined with:
  + **Room Database entities** (data classes)
  + **Retrofit DTOs** (data classes)
  + **MVVM architecture** (ViewModel classes, sealed classes for UI states)

**Best Practices**

* Keep classes small & focused (Single Responsibility Principle).
* Prefer val over var for immutability.
* Use **interfaces** for contracts, not inheritance unless needed.
* Use **data classes** for DTOs.
* Use object for singletons (thread-safe by default in Kotlin).
* Limit visibility with private or internal to reduce coupling.

**Part B — Missing but Important (Expanded)**

1. **Sealed Classes & Enums**
   * Sealed classes: Better for representing limited state sets (e.g., UI states: Loading, Success, Error).
   * Enums: For fixed constant values (e.g., Days of week).

kotlin

CopyEdit

enum class Status { SUCCESS, ERROR, LOADING }

1. **Interfaces with Default Implementation**
   * Kotlin allows methods in interfaces to have a body.

kotlin

CopyEdit

interface Clickable {

fun onClick() = println("Default Click")

}

1. **Abstract vs Interface in Android**
   * Abstract: Use when you need to share common behavior & state.
   * Interface: Use for contracts without state.
2. **Composition over Inheritance**
   * Favor **has-a** relationships over **is-a** when possible for flexibility.

kotlin

CopyEdit

class Engine

class Car(private val engine: Engine)

1. **Data Class Copy Function**
   * Create modified copies without changing original.

kotlin

CopyEdit

val user1 = User(1, "Harsh")

val user2 = user1.copy(name = "Raj")

1. **Operator Overloading**
   * Define custom meaning for operators (+, -).

kotlin

CopyEdit

data class Point(val x: Int, val y: Int) {

operator fun plus(other: Point) = Point(x + other.x, y + other.y)

}

1. **Delegation Pattern** (by keyword)
   * Delegate interface implementation to another object.

kotlin

CopyEdit

interface Printer { fun printMsg() }

class PrinterImpl: Printer { override fun printMsg() = println("Hello") }

class MyPrinter(printer: Printer) : Printer by printer