1. Pengertian Class dan Objek

- **Class** adalah blueprint atau cetak biru untuk membuat objek. Di dalam class, kita mendefinisikan properti (atribut) dan fungsi (metode) yang menggambarkan karakteristik dan perilaku dari objek.
- **Objek** adalah instansiasi dari class. Dengan kata lain, objek adalah hasil dari implementasi class.

---

2. Membuat Class Sederhana

Berikut adalah cara mendefinisikan class di Kotlin:

```
class Person {
    var name: String = ""
    var age: Int = 0

    // Fungsi untuk mencetak detail objek
    fun printDetails() {
        println("Name: $name, Age: $age")
    }
}
```

3. Membuat Objek dari Class

Untuk membuat objek dari class, kita menggunakan kata kunci `val` atau `var` diikuti dengan nama class.

```
fun main() {
    val person = Person() // Membuat objek dari class Person
    person.name = "John"
    person.age = 25

    person.printDetails() // Output: Name: John, Age: 25
}
```

4. Constructor di Kotlin

Kotlin memiliki dua jenis konstruktor:

- Primary Constructor
- Secondary Constructor

a. Primary Constructor

Primary constructor didefinisikan di deklarasi class dan merupakan cara yang umum digunakan.

```
class Person(val name: String, var age: Int) {
    fun printDetails() {
        println("Name: $name, Age: $age")
    }
}

fun main() {
    val person = Person("Alice", 30)
    person.printDetails() // Output: Name: Alice, Age: 30
}
```

Dalam contoh ini, properti `name` dan `age` dideklarasikan dan diinisialisasi langsung dalam primary constructor.

b. Secondary Constructor

Secondary constructor digunakan jika Anda memerlukan lebih dari satu cara untuk menginisialisasi objek atau ingin memberikan beberapa logika tambahan dalam proses inisialisasi.

```
class Person (var name: String, var age: Int) {
    var gender: String = "Male"

    // Secondary constructor
    constructor(name: String, age: Int, gender: String ): this(name=name, age=age)
    {
        this.name = name
        this.age = age
        this.gender = gender
    }

    fun printDetails() {
        println("Name: $name, Age: $age, Gender: $gender")
    }
}

fun main() {
    val person = Person("Bob", 40)
    person.printDetails() // Output: Name: Bob, Age: 40, Gender: Male
    val person2 = Person("Susi", 25, "Female")
    person2.printDetails() // Output: Name: Agus, Age: 25, Gender: Female

}
```

5. Inisialisasi Blok (`init`)

Kotlin memiliki blok `init` yang dieksekusi segera setelah konstruktor dipanggil. Blok ini sering digunakan untuk inisialisasi atau validasi.

```
class Person(val name: String, var age: Int) {
    init {
        println("Person object created with name: $name and age: $age")
    }
```

```
    fun printDetails() {
        println("Name: $name, Age: $age")
    }
}

fun main() {
    val person = Person("Charlie", 35)
    person.printDetails()
    // Output:
    // Person object created with name: Charlie and age: 35
    // Name: Charlie, Age: 35
}
```

---

6. Getter dan Setter

Di Kotlin, **getter** dan **setter** secara otomatis dihasilkan untuk setiap properti dalam class. Namun, Anda juga bisa membuat getter dan setter kustom jika diperlukan.

```
class Person {
    var name: String = "Unknown"
        get() = field.uppercase() // getter kustom yang mengubah name menjadi huruf besar
        set(value) {
            field = value
        }

    var age: Int = 0
        set(value) {
            if (value > 0) {
                field = value
            }
        }
}

fun main() {
    val person = Person()
    person.name = "john"
    println(person.name) // Output: JOHN

    person.age = -5 // Tidak akan mengubah nilai karena kondisi setter
    println(person.age) // Output: 0
}
```

---

7. Inheritance (Pewarisan)

Di Kotlin, kita dapat mewariskan class dari class lain menggunakan kata kunci `open` pada class yang dapat diwariskan. Secara default, semua class di Kotlin adalah **final** (tidak bisa diwariskan).

```
open class Person(val name: String, var age: Int) {
    fun printDetails() {
        println("Name: $name, Age: $age")
    }
}

class Student(name: String, age: Int, val studentId: Int) : Person(name,
age) {
    fun printStudentId() {
        println("Student ID: $studentId")
    }
}

fun main() {
    val student = Student("Dave", 20, 12345)
    student.printDetails() // Output: Name: Dave, Age: 20
    student.printStudentId() // Output: Student ID: 12345
}
```

Overriding Fungsi

Jika Anda ingin melakukan overriding terhadap fungsi di class induk, gunakan kata kunci `open` pada fungsi di class induk dan `override` pada class turunan.

```
open class Person {
    open fun greet() {
        println("Hello!")
    }
}

class Student : Person() {
    override fun greet() {
        println("Hello, I am a student!")
    }
}

fun main() {
    val student = Student()
    student.greet() // Output: Hello, I am a student!
}
```

---

8. Interface di Kotlin

Selain pewarisan, Kotlin juga mendukung **interface**. Interface hanya mendefinisikan kontrak perilaku yang harus diimplementasikan oleh class.

```
interface Drivable {
    fun drive()
}

class Car : Drivable {
    override fun drive() {
        println("The car is driving")
    }
}

fun main() {
    val car = Car()
    car.drive() // Output: The car is driving
}
```

---

9. Data Class

Kotlin menyediakan **data class**, yang secara otomatis menghasilkan metode seperti `toString()`, `equals()`, dan `hashCode()` berdasarkan properti yang ada di class. Ini sangat berguna untuk class yang hanya digunakan untuk menyimpan data.

```
data class Person(val name: String, val age: Int)

fun main() {
    val person1 = Person("John", 30)
    val person2 = Person("John", 30)

    // Perbandingan berbasis nilai
    println(person1 == person2) // Output: true

    // toString otomatis
    println(person1) // Output: Person(name=John, age=30)
}
```

---

10. Singleton Object

Jika Anda hanya membutuhkan satu instance dari class, Anda bisa menggunakan **singleton object** di Kotlin.

```
object Database {
    val name = "MyDatabase"

    fun connect() {
```

```
        println("Connected to $name")
    }
}

fun main() {
    Database.connect() // Output: Connected to MyDatabase
}
```

Kesimpulan

- **Class** di Kotlin digunakan untuk membuat blueprint dari objek.
- **Primary constructor** dan **secondary constructor** membantu dalam inisialisasi objek.
- **Getter dan setter** dapat dihasilkan secara otomatis atau dikustomisasi.
- Kotlin mendukung **inheritance**, **interface**, dan **data class**.
- **Singleton object** di Kotlin digunakan untuk membuat satu instance yang bisa diakses di seluruh aplikasi.