kotlin OOP concepts

Sure — let's go over Object-Oriented Programming (OOP) concepts in Kotlin, with examples.

Kotlin is a fully object-oriented language (though it also supports functional programming), and it implements OOP concepts like:

🔹 1. Class and Object

A class is a blueprint, and an object is an instance of that class.

```
class Car(val brand: String, val model: String) {
    fun start() {
        println("$brand $model is starting...")
    }
}

fun main() {
    val car1 = Car("Tesla", "Model S") // Object creation
    car1.start()
}
```

🔹 2. Constructor

Kotlin supports primary and secondary constructors.

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

    constructor(name: String) : this(name, 18) // Secondary constructor
}
```

🔹 3. Encapsulation

Encapsulation means hiding internal data using access modifiers (private, protected, public, internal).

```
class Account {
    private var balance: Double = 0.0

    fun deposit(amount: Double) {
        if (amount > 0) balance += amount
    }

    fun getBalance(): Double = balance
}
```

🔹 4. Inheritance

Kotlin classes are final by default, so to allow inheritance, mark them with open.

```
open class Animal {
    open fun sound() {
        println("Animal makes a sound")
    }
}

class Dog : Animal() {
    override fun sound() {
        println("Dog barks")
    }
}
```

🔹 5. Polymorphism

Polymorphism allows using a single interface or parent class reference for multiple subclasses.

```
fun makeSound(animal: Animal) {
```

```
    animal.sound()
}

fun main() {
    val a: Animal = Dog()
    makeSound(a)  // Calls Dog's implementation
}
```

🔹 6. Abstraction

Achieved using abstract classes or interfaces.

Using abstract class:

```
abstract class Shape {
    abstract fun area(): Double
}

class Circle(val radius: Double) : Shape() {
    override fun area() = Math.PI * radius * radius
}
```

Using interface:

```
interface Drivable {
    fun drive()
}

class Car : Drivable {
    override fun drive() = println("Car is driving")
}
```

🔹 7. Data Classes

Kotlin provides data classes for holding data without writing boilerplate code like toString(), equals(), hashCode().

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

🔹 8. Object Keyword

The object keyword is used for singleton objects or companion objects.

```
object Database {
    fun connect() = println("Connected to database")
}


fun main() {
    Database.connect()  // Singleton instance
}
```

🔹 9. Companion Objects

Used for static-like members inside classes.

```
class Utils {
    companion object {
        fun printHello() = println("Hello from Utils!")
    }
}


fun main() {
    Utils.printHello() // No need to create an object
}
```

Would you like me to give you a summary diagram or chart of these OOP concepts for Kotlin (e.g., visual cheat sheet)?