Enum Class

In Kotlin, an `enum class` (short for enumerated type) is used to represent a fixed set of constants. It is useful when you need a predefined list of values that are known at compile-time. Each constant in an enum is an object, and enum classes can have properties, methods, and implement interfaces.

Here's a simple example of how an `enum class` works in Kotlin:

```
enum class Direction {
    NORTH, SOUTH, EAST, WEST
}
```

Key Features of Kotlin Enums:

1. **Constant Values:** Enum constants are essentially instances of the enum class. Each constant can be accessed like a static field in Java.

   ```
   val direction = Direction.NORTH // data type of variable is Direction
   ```

2. **Properties and Methods:** Enum classes can have properties, constructors and methods like a regular class.

   ```
   enum class Direction(val degrees: Int) {
       NORTH(0),
       EAST(90),
       SOUTH(180),
       WEST(270);

       fun description(): String {
           return "Direction $name with angle $degrees degrees"
       }
   }

   fun main() {
       val direction = Direction.NORTH
       println(direction.description())  // Output: Direction NORTH with angle 0
   degrees
   }
   ```

3. **Custom Behavior:** You can override same method for each specific constants in an enum if necessary.

   ```
   enum class Operation {
       ADD {
           override fun apply(x: Int, y: Int): Int = x + y
       },
       SUBTRACT {
           override fun apply(x: Int, y: Int): Int = x - y
       };

       abstract fun apply(x: Int, y: Int): Int
   }

   fun main() {
       val operation = Operation.ADD
       println(operation.apply(3, 5))  // Output: 8
   }
   ```

Common Enum Methods:

- `name`: Returns the name of the enum constant.
- `ordinal`: Returns the position of the enum constant (starting from 0).
- `values()`: Returns an array of all enum constants.
- `valueOf()`: Returns the enum constant with the specified name.

```
kotlin
Copy code
fun main() {
    for (direction in Direction.values()) {
        println("${direction.name} is at index ${direction.ordinal}")
    }
    val east = Direction.valueOf("EAST")
    println(east)  // Output: EAST
}
```

Enums are powerful tools for defining constants and attaching behavior to them in a clean, type-safe way.

Q) Why we can't create multiple instances of enum class just like a simple class as val obj = Direction(), and why constructors can only be called in body of enum class for these constant values. Why they are behaving like static member of a class since they are refenced by class_name. ?

Ans- The reason we access enum constants using the class name (e.g., Direction.NORTH) rather than creating new objects (like Direction()), is because **enum constants are single, pre-defined instances** of the enum class that are created by the Kotlin compiler at compile time.

When you define an enum class, the compiler creates exactly one instance for each enum constant. These instances are created when the class is loaded, and they remain the same throughout the application's runtime.

Enums represent a **closed set** of known values/objects. Since these values are predefined (like `NORTH`, `SOUTH`, etc.), they don't need to be instantiated multiple times. All references to `Direction.NORTH` will point to the same, immutable object.

```
fun main() {
    //Access the enum constants
        val dir1 = Direction.NORTH
        val dir2 = Direction.NORTH
        println(dir1 == dir2) // true , both point to the same object
}
```

If you tried to create new instances using `Direction()` (like you would with regular classes), it would defeat the purpose of enums, which is to have a fixed, limited set of constant values. Kotlin enforces this by preventing you from instantiating new enum objects.

When you use Direction.NORTH, you're not creating a new object. Instead, you're accessing an existing, static-like instance. Since these constants are created by the compiler as part of the enum definition.

Enums in Kotlin (and Java) are designed to prevent the creation of new instances. Therefore, Kotlin doesn't allow constructors to be called externally for enum classes. This enforces the concept of a fixed set of constants, which can only be accessed through the class reference.