**Sealed Classes in Kotlin**

**Purpose**:

Sealed classes are primarily used to represent restricted class hierarchies in Kotlin. This means that the class hierarchy is closed, and all subclasses must be declared within the same file as the sealed class itself. This restriction enables the Kotlin compiler to provide exhaustive `when` expression checks.

**Declaration**:

A sealed class is declared using the `sealed` keyword before the `class` keyword.

```kotlin

sealed class MySealedClass {

// subclasses go here

}

```

**Subclasses:**

All subclasses of a sealed class must be declared within the same file where the sealed class is declared. These subclasses can be either data classes, regular classes, or object declarations.

```kotlin

sealed class Result

data class Success(val data: String) : Result()

data class Error(val message: String) : Result()

```

**Usage:**

Sealed classes are often used with `when` expressions, where each possible subclass can be handled in a separate branch. Since all subclasses are known at compile time, the compiler can ensure that all possible cases are handled.

```kotlin

fun handleResult(result: Result) {

when (result) {

is Success -> println("Success: ${result.data}")

is Error -> println("Error: ${result.message}")

}

}

```

**Exhaustiveness:**

The Kotlin compiler enforces that all possible subclasses are covered in a `when` expression. This means that if there's a sealed class with three subclasses, all three must be handled explicitly. This prevents accidental bugs due to missing cases.

Inheritance:

Sealed classes can have subclasses within their own file, but they cannot be subclassed outside of their declared file. This ensures that the hierarchy remains closed and exhaustively checkable.

**Benefits:**

- Enhanced safety: Sealed classes enforce handling of all possible cases, reducing the chance of runtime errors.

- Readability: Sealed classes make code more readable by clearly defining a limited set of possible types.

- Pattern matching: Sealed classes work well with pattern matching constructs like `when`, making code concise and expressive.

Overall, sealed classes are a powerful feature in Kotlin for modeling restricted class hierarchies, ensuring type safety, and improving code maintainability.