Visibility modifiers

Visibility modifiers in Kotlin are essential tools for controlling the access level of classes, properties, methods, and constructors. They help you achieve encapsulation by restricting access to certain parts of your code and making your codebase more secure, organized, and easier to maintain. Here’s a breakdown of the different visibility modifiers in Kotlin and how they can be applied:

**Visibility Modifiers Overview**

1. **public (default)**:
   * **Accessibility**: Everywhere, both within the module and from outside.
   * **Use case**: For properties and methods intended for use in any part of your application, even from other modules.
2. **private**:
   * **Accessibility**: Only within the class or the file where it is declared.
   * **Use case**: To hide the internal implementation details that shouldn't be exposed to other classes or modules.
3. **protected**:
   * **Accessibility**: Within the class and its subclasses.
   * **Use case**: When you want to allow access to properties and methods only within the class and its derived classes.
4. **internal**:
   * **Accessibility**: Within the same module.
   * **Use case**: When you want to restrict access to other parts of the application but still allow access within the same module.

**Examples and Use Cases**

**1. private modifier for properties and methods:**

* Example: A class where certain properties are intended to be managed internally.

open class SmartDevice(val name: String, val category: String) {

private var deviceStatus = "online"

}

In this example, the deviceStatus property is private and can only be accessed and modified within the SmartDevice class.

**2. protected modifier for setters:**

* Example: Controlling access to a property while allowing subclasses to modify it.

open class SmartDevice(val name: String, val category: String) {

var deviceStatus = "online"

protected set

}

Here, the deviceStatus property is publicly readable but only modifiable within the SmartDevice class and its subclasses.

**3. internal modifier for classes:**

* Example: A class that should only be accessible within the same module.

internal open class SmartDevice(val name: String, val category: String) {

// Class contents

}

The SmartDevice class is accessible only within the module it is declared in.

**4. Private properties in subclasses:**

* Example: Properties that should only be modified through specific methods within the class.

class SmartTvDevice(deviceName: String, deviceCategory: String) :

SmartDevice(name = deviceName, category = deviceCategory) {

private var speakerVolume = 2

set(value) {

if (value in 0..100) {

field = value

}

}

private var channelNumber = 1

set(value) {

if (value in 0..200) {

field = value

}

}

}

Here, speakerVolume and channelNumber are private properties of SmartTvDevice, ensuring they can only be modified internally.

**Best Practices**

* **Default to private**: Start by making properties and methods private. Only increase their visibility if necessary.
* **Use protected cautiously**: Use the protected modifier when you anticipate that subclasses will need to modify or access certain properties or methods.
* **Limit public access**: Make classes, properties, and methods public only when they need to be accessed from outside the module.

**Summary**

Visibility modifiers are crucial in Kotlin for enforcing encapsulation and protecting the internal state of your classes. By strategically applying public, private, protected, and internal modifiers, you can ensure that your code is both secure and maintainable, exposing only what is necessary to other parts of your application.