**Lab 15 - Smart casting and Regular casting example in Kotlin**

In Kotlin, casting is the process of converting one type into another. There are two main types of casting:

* Regular Casting: This is done using the as keyword, and it is similar to casting in other languages like Java. However, it can throw a ClassCastException if the cast is not successful.
* Smart Casting: Kotlin automatically casts types when it can be proven that the cast is safe, eliminating the need for explicit casting in many situations.

Regular Casting Example

Regular casting is performed using the as keyword. If the object cannot be cast to the specified type, a ClassCastException will be thrown.

fun main() {

val obj: Any = "Kotlin String"

// Regular casting using 'as'

val str: String = obj as String

println(str) // Output: Kotlin String

// Attempting an invalid cast

try {

val num: Int = obj as Int // This will throw a ClassCastException

} catch (e: ClassCastException) {

println("Cast failed: ${e.message}")

}

}

**Explanation:**

* obj as String: The obj variable is cast to String. This works because obj actually contains a string.
* obj as Int: This cast fails because obj is a string, not an integer, and a ClassCastException is thrown.

**Smart Casting Example**

Smart casting occurs automatically in Kotlin. When the compiler can verify that a variable is of a certain type after a type check (e.g., using is), it will allow you to use the variable as that type without requiring an explicit cast.

fun printLength(obj: Any) {

if (obj is String) {

// Smart casting: 'obj' is automatically cast to 'String'

println("The length of the string is: ${obj.length}")

} else {

println("Not a string")

}

}

fun main() {

printLength("Hello, Kotlin!") // Output: The length of the string is: 14

printLength(12345) // Output: Not a string

}

**Explanation:**

* if (obj is String): The is keyword checks if obj is of type String. After this check, within the if block, obj is automatically treated as a String.
* obj.length: You can access the length property directly without casting obj to String because Kotlin smart casts it.

**Combining Smart Casting with Safe Calls**

* You can combine smart casting with safe calls to make your code even more concise and safe:

fun printUpperCase(obj: Any?) {

if (obj is String && obj.isNotEmpty()) {

println(obj.toUpperCase()) // Smart cast to 'String' within this block

} else {

println("Not a valid string")

}

}

fun main() {

printUpperCase("hello") // Output: HELLO

printUpperCase("") // Output: Not a valid string

printUpperCase(null) // Output: Not a valid string

}

**Explanation:**

* The smart cast occurs within the if block, and you can safely call toUpperCase() on obj because the type is guaranteed to be String after the check.

**Summary**

* Regular Casting (as): Used to explicitly cast an object to another type. If the cast fails, a ClassCastException is thrown.
* Smart Casting: Kotlin automatically casts a variable to a more specific type if it can be proven safe, typically after an is check. This reduces the need for explicit casting and makes the code safer and more readable.