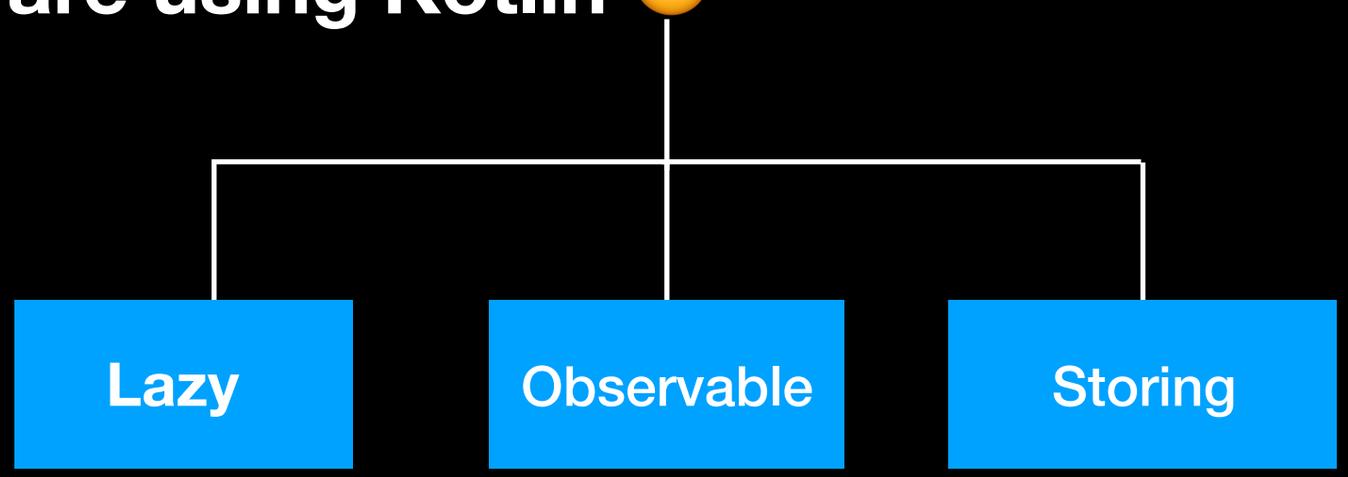
Kotlin-IV

Delegated Properties

- We can delegations establish designed in the projects.
- But wait….
- **We are using Kotlin** 😎



Inbuilt delegated properties present in Kotlin

Lazy Delegate

- Takes a lambda
- Returns a Lazy<T> property which can serve as a delegate to implement a lazy property.
- First call calculates value from lambda and for subsequent calls remembers the evaluated value
- By default synchronized
- For changing we have to change LazyThreadSafetyMode

```kotlin
val lazyValue: String by lazy {
    println("computed!")
}

fun main(args: Array<String>) {
    println(lazyValue)
}
```

```kotlin
data class Person(val name: String) {

    val books by lazy { BookManager.loadBooks(this) }
}
```

Delegation ‘by’ keyword

Observable delegation

- Listeners get notified about changes in this property.
- The *observable* delegate allows for a lambda to be triggered any time the value of the property changes.

```
import kotlin.properties.Delegates

class User {
    var name: String by Delegates.observable("<no name>") {
        prop, old, new ->
        println("$old -> $new")
    }
}

fun main(args: Array<String>) {
    val user = User()
    user.name = "first"
    user.name = "second"
}
```

Observable delegation

- Listeners get notified about changes in this property.
- The *observable* delegate allows for a lambda to be triggered any time the value of the property changes.

```kotlin
import kotlin.properties.Delegates

class User {
    var name: String by Delegates.observable("<no name>") {
        prop, old, new ->
        println("$old -> $new")
    }
}

fun main(args: Array<String>) {
    val user = User()
    user.name = "first"
    user.name = "second"
}
```

```
<no name> -> first
first -> second
```

Map Delegator

- One common use case storing values in map
- Common use case in Dynamic properties or JSON parsing

```
class User(val map: Map<String, Any?>) {
    val name: String by map
    val age: Int     by map
}


val user = User(mapOf(
    "name" to "John Doe",
    "age"  to 25
))

println(user.name) // Prints "John Doe"
println(user.age)  // Prints 25
```

Map Delegation

Collections

- Distinguishes between mutable and immutable collections
- List<T> is read only and have all the functions for get, size. But is read only and cannot be mutated
- MutableList<T> exists for mutable collection
- For set Set<T>/MutableSet<T>
- For map Map<K,V> /MutableMap<K,V>

```kotlin
val numbers: MutableList<Int> = mutableListOf(1, 2, 3)
val readOnlyView: List<Int> = numbers
println(numbers)        // prints "[1, 2, 3]"
numbers.add(4)
println(readOnlyView)   // prints "[1, 2, 3, 4]"
readOnlyView.clear()    // -> does not compile

val strings = hashSetOf("a", "b", "c", "c")
assert(strings.size == 3)
```

List/Set

```kotlin
val readWriteMap = hashMapOf("foo" to 1, "bar" to 2)
println(readWriteMap["foo"])  // prints "1"
val snapshot: Map<String, Int> = HashMap(readWriteMap)
```

Map

Coroutines

- Perform async and parallel
- Non blocking calls made easy
- Structure of sequential code becomes same as parallel code.
- No Callback hell
- LightWeight
- Lighter than threads

Basic Keywords and Terms

- **launch:** Creates a new coroutine, fires and forgets it. If exception occurs and is uncaught it can abrupt flow.
- **async:** This fires and wait for the response.
- **Deferred<T>:** We invoke await to receive response on deferred objects.
- **run:** Creates a new Coroutine.
- Thread.sleep ==> delay

```kotlin
// Start a coroutine
launch {
    delay(1000)
    println("Hello")
}
```

```kotlin
launch(CommonPool) {
 sendMail()
}

private suspend fun sendMail() {
 val mailId = async(CommonPool) {
    getMailFromDB()
 }
val mailMsg = async(CommonPool) {
    getMessageFromDB()
 }
val msg = async(CommonPool) {
    async(UI) {
       mail_id_text_view.text = String.format(getString(R.string.email_id_string),
                        mailId.await())
       msg_text_view.text = String.format(getString(R.string.email_msg),
                        mailMsg.await())
    }
    sendMsgFromApi(mailId.await(), mailMsg.await())
}

}
```

```kotlin
// Start a coroutine
launch {
    delay(1000)
    println("Hello")
}
```

```kotlin
launch(CommonPool) {
  sendMail()
}

private suspend fun sendMail() {
 val mailId = async(CommonPool) {
    getMailFromDB()
 }
val mailMsg = async(CommonPool) {
    getMessageFromDB()
 }
val msg = async(CommonPool) {
    async(UI) {
       mail_id_text_view.text = String.format(getString(R.string.email_id_string),
                        mailId.await())
       msg_text_view.text = String.format(getString(R.string.email_msg),
                        mailMsg.await())
    }
    sendMsgFromApi(mailId.await(), mailMsg.await())
}

}
```

```kotlin
// Start a coroutine
launch {
    delay(1000)
    println("Hello")
}
```

```kotlin
launch(CommonPool) {
  sendMail()
}

private suspend fun sendMail() {
 val mailId = async(CommonPool) {
    getMailFromDB()
 }
val mailMsg = async(CommonPool) {
    getMessageFromDB()
 }
val msg = async(CommonPool) {
    async(UI) {
       mail_id_text_view.text = String.format(getString(R.string.email_id_string),
                        mailId.await())
       msg_text_view.text = String.format(getString(R.string.email_msg),
                        mailMsg.await())
    }
    sendMsgFromApi(mailId.await(), mailMsg.await())
}

}
```

```kotlin
// Start a coroutine
launch {
    delay(1000)
    println("Hello")
}
```

```kotlin
launch(CommonPool) {
  sendMail()
}

private suspend fun sendMail() {
 val mailId = async(CommonPool) {
    getMailFromDB()
 }
val mailMsg = async(CommonPool) {
    getMessageFromDB()
 }
val msg = async(CommonPool) {
    async(UI) {
       mail_id_text_view.text = String.format(getString(R.string.email_id_string),
                        mailId.await())
       msg_text_view.text = String.format(getString(R.string.email_msg),
                        mailMsg.await())
    }
    sendMsgFromApi(mailId.await(), mailMsg.await())
}

}
```

```kotlin
// Start a coroutine
launch {
    delay(1000)
    println("Hello")
}
```

```kotlin
launch(CommonPool) {
  sendMail()
}

private suspend fun sendMail() {
 val mailId = async(CommonPool) {
    getMailFromDB()
 }
val mailMsg = async(CommonPool) {
    getMessageFromDB()
 }
val msg = async(CommonPool) {
    async(UI) {
       mail_id_text_view.text = String.format(getString(R.string.email_id_string),
                        mailId.await())
       msg_text_view.text = String.format(getString(R.string.email_msg),
                        mailMsg.await())
    }
    sendMsgFromApi(mailId.await(), mailMsg.await())
}

}
```

Thank you 🙋🏻‍♂️