

# Fresher Android

## *Kotlin OOP Advance*





GLOBAL SMART  
TECHNOLOGIES

# Kotlin Object Oriented Programming Advance Concept

# 1. Generics

**Classes in Kotlin may have type parameters:**

```
class Box<T>(t: T) {  
    var value = t  
}
```

`val box: Box<Int> = Box<Int>(1)`  
`val box = Box(1)` // 1 has type Int, so the compiler figures out that we are talking about Box<Int>

# 1. Generics

## Generic functions

```
fun <T> singletonList(item: T): List<T> {  
    // ...  
}
```

```
fun <T> T.basicToString(): String { // extension function  
    // ...  
}
```

```
val l = singletonList<Int>(1)  
val l = singletonList(1)
```

# 1. Generics

## Generic constraints

Upper bounds

```
fun <T : Comparable<T>> sort(list: List<T>) { ...  
}
```

`sort(listOf(1, 2, 3))` // OK. `Int` is a subtype of `Comparable<Int>`

`sort(listOf(HashMap<Int, String>()))` // Error: `HashMap<Int, String>` is not a subtype of `Comparable<HashMap<Int, String>>`

# 1. Generics

Declaration-site variance

## Covariant

```
interface Source<out T> {  
    fun nextT(): T  
}
```

## Contravariant

```
interface Comparable<in T> {  
    operator fun compareTo(other: T): Int  
}
```

## 2. Nested and Inner Classes

```
class Outer {  
    private val bar: Int = 1  
    class Nested {  
        fun foo() = 2  
    }  
}
```

```
val demo = Outer.Nested().foo() // == 2
```

## 2. Nested and Inner Classes

### Inner classes

```
class Outer {  
    private val bar: Int = 1  
    inner class Inner {  
        fun foo() = bar  
    }  
}
```

```
val demo = Outer().Inner().foo() // == 1
```



# 3. Enum Classes

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

Initialization

```
enum class Color(val rgb: Int) {  
    RED(0xFF0000),  
    GREEN(0x00FF00),  
    BLUE(0x0000FF)  
}
```

Implementing Interfaces in Enum Classes

# 4. Sealed Classes

- Sealed classes are used for representing restricted class hierarchies, when a value can have one of the types from a limited set, but cannot have any other type.

```
sealed class Expr  
data class Const(val number: Double) : Expr()  
data class Sum(val e1: Expr, val e2: Expr) : Expr()  
object NotANumber : Expr()
```

- A sealed class is abstract by itself, it cannot be instantiated directly and can have abstract members.

```
fun eval(expr: Expr): Double = when(expr) {  
    is Const -> expr.number  
    is Sum -> eval(expr.e1) + eval(expr.e2)  
    NotANumber -> Double.NaN  
    // the `else` clause is not required because we've covered all the cases  
}
```

# 5. Object Expressions and Declarations

## Object expressions

```
class C {  
    // Private function, so the return type is the anonymous object type  
    private fun foo() = object {  
        val x: String = "x"  
    }  
  
    // Public function, so the return type is Any  
    fun publicFoo() = object {  
        val x: String = "x"  
    }  
  
    fun bar() {  
        val x1 = foo().x    // Works  
        val x2 = publicFoo().x // ERROR: Unresolved reference 'x'  
    }  
}
```

# 5. Object Expressions and Declarations

- **Object declarations**

```
object DataManager {  
    fun registerDataProvider(provider: DataProvider) {  
        // ...  
    }  
  
    val allDataProviders: Collection<DataProvider>  
        get() = // ...  
}
```

- **Companion Objects**

```
class MyClass {  
    companion object Factory {  
        fun create(): MyClass = MyClass()  
    }  
}  
  
val instance = MyClass.create()  
  
class MyClass {  
    companion object { }  
}  
  
val x = MyClass.Companion
```

- **object expressions are executed (and initialized) immediately, where they are used;**
- **object declarations are initialized lazily, when accessed for the first time;**

## 6. Type aliases

```
typealias NodeSet = Set<Network.Node>
```

```
typealias FileTable<K> = MutableMap<K, MutableList<File>>
```

```
typealias MyHandler = (Int, String, Any) -> Unit
```

```
typealias Predicate<T> = (T) -> Boolean
```

# Lesson Summary

- Generics
- Nested and Inner Classes
- Enum Classes
- Sealed Classes
- Object Expressions and Declarations
- Type aliases

CONFIDENTIAL

# Thank you

