10 cool features in Kotlin

Satya Ram Twanabasu

1. Null-safe and Elvis operation

- Null References: [The Billion Dollar Mistake]

  - **Kotlin provides a way to think beforehand to avoid that mistake with nullable references.**

```
var s1: String  = "abc"   // s1 is not-nullable
var s2: String? = "abc"   // s2 is nullable

s1 = null    // compilation error
s1 = s2      // compilation error

s2 = s1      // ok
s2 = null    // ok

println(s1.length)     //s1 is guaranteed to be not-null
println(s2?.length)    //safe call : returns length of s2 or null otherwise but does not throw a NPE
```

  - **Elvis Operator (?:)**

```
println(s2?.length ?: 0) //returns length of s2, if s2 is not null or 0 otherwise
```

2. Data class

- Kotlin knows how to "Walk the talk", FTW!

```kotlin
data class User(var name: String, var age: Int)

// Equivalent in Java
public final class User
{
    private String  name;
    private Integer  age;

    ...   // constructor with every field
    ...   // get()/set() methods for each field
    ...   // method toString()
    ...   // methods hashCode() and equals()
    ...   // copy() to make a new instance from existing one
}
```

3. Multiple class in a single File

- File name in Java is tightly connected to the class name
- Only one public class can be defined in a file. No such restriction in Kotlin.

```kotlin
// Student.kt
data class Student(var name: String, var age: Int, var homeAdress: Address)

data class Address(var streetName:String, var blockNumber: Int, postNumber: Int)
```

4. Switch construct with when

- A switch with Superpowers

```
val x = // value
val result = when (x) {
    0, 11 -> "0 or 11"
    in 1..10 -> "from 1 to 10"
    !in 12..14 -> "not from 12 to 14"
    else -> if (isOdd(x)) { "is odd" } else { "otherwise" }
}
```

  - Arbitrary Condition Branches

```
var result = when(number) {
        0 -> "Invalid number"
        1, 2 -> "Number too low"
        3 -> "Number correct"
        in 4..10 -> "Number too high, but acceptable"
        !in 100..Int.MAX_VALUE -> "Number too high, but solvable"
        else -> "Number too high"
    }
```

4. Switch construct with when (Cont.)

- When without an argument

```
val y = // value
val result = when {
  isNegative(y) -> "is Negative"
  isZero(y) -> "is Zero"
  isOdd(y) -> "is odd"
  else -> "otherwise"
}
```

5. Object deconstruction

- unpacking with style

```
data class Book(val author: String, val title: String, val year: Int)
val book = Book("Gillian Flynn", "Gone girl", 2012)

val (author, title, year) = book
if (year > 2010) {
    // ...
}

//Simplifying loops on Collections and Maps:
val books = listOf<Book>()
for ((author, title, year) in books) {
    // …
}
```

5. Object deconstruction (Cont.)

- Deconstruction in lambdas

```
books.map { (author, title, year) ->
    // ...
}
```

- Skipping unnecessary variables

```
val (_, _, year) = book

for ((_, title, _) in books) {
    // ...
}

books.map { (_, _, year) ->
    // …
}
```

6. Extension functions

- Work smart, not work hard

```
fun <T> List<T>.midElement(): T {
    if (isEmpty())
        throw NoSuchElementException("List is empty.")
    return this[size / 2]
}

//usage
var list = listOf(1, 2, 3, 4, 5)
var mid = list.midElement()
```

7. optional arguments & named arguments

- Swiss knife of Kotlin.

```java
// Java
public void hello(String name) {
  if (name == null) {
    name = "World";
  }

  System.out.print("Hello, " + name + "!");
}
```

```kotlin
// Kotlin
fun hello(name: String = "World") {
    println("Hello, $name!")
}


//usage
hello()
hello("Kotlin")
hello(name = "Kotlin")


fun connect(url: String, connectTimeout: Int = 1000, enableRetry: Boolean = true) {
    println("The parameters are url = $url, connectTimeout = $connectTimeout, enableRetry = $enableRetry")
}

//use of parameters
connect(
    connectTimeout = 500,
    url = "www.google.com"
)
```

8. Operator overloading

- Less known feature; makes various operations intuitive

```
data class Point(val x: Int, val y: Int)

// Here's how to define operator overloading for `+` operator
operator fun Point.plus(p: Point) = Point(x + p.x, y + p.x)

//similarly..
operator fun Point.minus(p: Point) = Point(x - p.x, y - p.y)
operator fun Point.times(p: Point) = Point(x * p.x, y * p.y)
operator fun Point.div(p: Point) = Point(x / p.x, y / p.y)
operator fun Point.unaryPlus() = Point(x + 1, y + 1)
operator fun Point.unaryMinus() = Point(x - 1, y - 1)

fun Point.equals(obj: Any?) : Boolean {
    if (obj === this) return true
    if (obj !is Point) return false
    return obj.x == x && obj.y == y
}

//usage
println(Point(1,1) + Point(2,2))  //Point(x=3, y=3)
println(+Point(1,1))              //Point(x=2, y=2)
println(Point(1,1) == Point(1,1)) //true
```

9. Smart Casting

- Casting done smoothly and painlessly

```java
// Java
if(s instanceof String){
  final String result = ((String) s).substring(1);
}
```

```kotlin
// Kotlin
if (s is String) {
  val result = s.substring(1)
}
```

```kotlin
//Smart Cast with when
var x:Any = getResult()

when (x) {
    is Int -> print(x + 1)
    is String -> print(x.length + 1)
    is IntArray -> print(x.sum())
}
```

10. String template

- Simply a string literal containing embedded expressions.

```
// Java
String message = "n = " + n;


// Kotlin
val message = "n = $n"

//String template can contain any valid expression
```kotlin
val message1 = "n + 1 = ${n + 1}"
val message2 = "$n is ${if(n > 0) "positive" else "not positive"}
```

//Can be nested
val message = "$n is ${if (n > 0) "positive" else
       if (n < 0) "negative and ${if (n % 2 == 0) "even" else "odd"}" else "zero"}"
```

10. String template (Cont.)

- Raw String

```java
// Java
String path = "C:\\Repository\\read.me";
```

```kotlin
// Kotlin
val path = """C:\Repository\read.me"""

//Multiline String
val receipt = """Item 1: $1.00
            >Item 2: $0.50""".trimMargin(">")
println(receipt)

//outputs
//Item 1: $1.00
//Item 2: $0.50
```