Kotlin 치트시트

1. 변수, 타입, Null 안정성

- 변수 선언:
  - `val`: 읽기 전용 (immutable) 변수. Java의 `final`과 유사.
  - `var`: 재할당 가능한 (mutable) 변수.
- 타입 추론: `val name = "Kotlin"` (컴파일러가 `String`으로 추론)
- 기본 타입: `Int`, `Double`, `Boolean`, `Char`, `String` 등. Java와 달리 모든 것이 객체.
- Null 안정성 (Null Safety): Kotlin의 핵심 기능. Null Pointer Exception(NPE)을 방지.
  - `String?`: Null이 될 수 있는 타입.
  - `String`: Null이 될 수 없는 타입.
- 안전한 호출 (Safe Call): `val length = name?.length` (name이 null이면 null 반환)
- 엘비스 연산자 (Elvis Operator): `val length = name?.length ?: -1` (name이 null이면 -1 반환)
- Not-null 단언 (Not-null assertion): `val length = name!!.length` (name이 null이면 NPE 발생, 주의해서 사용)

2. 제어 흐름

- `if-else`: Java와 유사. 표현식으로 사용 가능.

```
val max = if (a > b) a else b
```

- `when`: Java의 `switch`를 대체하는 강력한 기능.

```
when (x) {
    1 -> print("x == 1")
    2, 3 -> print("x is 2 or 3")
    in 4..7 -> print("x is in the range")
    is String -> print("x is a String")
    else -> print("otherwise")
}
```

- `for` 루프:

```
for (item in collection) print(item)
for (i in 1..5) { ... } // 1, 2, 3, 4, 5
for (i in 1 until 5) { ... } // 1, 2, 3, 4
for (i in 5 downTo 1 step 2) { ... } // 5, 3, 1
```

- `while` / `do-while`: Java와 동일.

3. 함수

- 함수 정의: `fun` 키워드 사용.

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

- 단일 표현식 함수: `fun sum(a: Int, b: Int) = a + b`
- 기본 인자 (Default Arguments): `fun greet(name: String, message: String = "Hello") { ... }`
- 명명된 인자 (Named Arguments): `greet(message = "Hi", name = "Bob")`

4. 클래스와 객체

- 클래스 정의:

```
class Person(val name: String) { // 주 생성자
    var age: Int = 0
}
```

- 데이터 클래스 (Data Class): `equals()`, `hashCode()`, `toString()`, `copy()` 등을 자동으로 생성.

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

- 상속: `open` 키워드로 상속 허용. `:` 사용.

```
open class Shape
class Rectangle : Shape()
```

- 인터페이스: `interface MyInterface { fun myMethod() }`
- 객체 (Object): 싱글턴(Singleton)을 쉽게 생성.

```
object DataProviderManager {
    fun registerDataProvider(...) { ... }
}
```

5. 확장 함수 및 프로퍼티 (Extensions)

기존 클래스를 수정하지 않고 새로운 함수나 프로퍼티를 추가.

```
fun String.initials(): String {
    return this.split(' ').map { it.first() }.joinToString("")
}
val name = "John Doe"
println(name.initials()) // JD
```

6. 고차 함수와 람다

- 람다 표현식: `{ 인자 -> 본문 }`

```
val sum = { x: Int, y: Int -> x + y }
println(sum(1, 2)) // 3
```

- 고차 함수: 함수를 인자로 받거나 함수를 반환하는 함수.
- 컬렉션 함수: `filter`, `map`, `forEach`, `reduce` 등.

```
val numbers = listOf(1, 2, 3, 4, 5)
val evens = numbers.filter { it % 2 == 0 } // [2, 4]
val squared = numbers.map { it * it } // [1, 4, 9, 16, 25]
```

7. 코루틴 (Coroutines)

비동기 코드를 쉽게 작성하기 위한 기능.

- `suspend` 함수: 일시 중단 가능한 함수.
- 코루틴 빌더:
  - `launch`: 결과를 반환하지 않는 코루틴 시작.
  - `async`: 결과를 `Deferred` 객체로 반환하는 코루틴 시작. `await()`로 결과 수신.
  - `runBlocking`: 코루틴이 완료될 때까지 현재 스레드를 블로킹.

```
import kotlinx.coroutines.*

fun main() = runBlocking { // this: CoroutineScope
    launch { // 새로운 코루틴을 시작하고 계속 진행
        delay(1000L) // 1초간 논블로킹 지연
        println("World!")
    }
    println("Hello,")
}
```

8. 스코프 함수 (Scope Functions)

객체의 컨텍스트 내에서 코드 블록을 실행. `let`, `run`, `with`, `apply`, `also`.

- `let`: non-null 값에 대해 코드 블록을 실행하고 람다 결과를 반환.

```
val name: String? = "Kotlin"
name?.let { println("Name is $it") }
```

- `apply`: 객체를 설정하는 데 사용. 객체 자신을 반환.

```
val person = Person("John").apply {
    age = 30
    city = "New York"
}
```