Лекция #9.7 Data-классы

Иногда классы бывают необходимы только для хранения некоторых данных. В Kotlin такие классы называются data-классы. Они определяются с модификатором **data**:

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

При компиляции такого класса компилятор автоматически добавляет в класс функции с определенной реализацией, которая учитывает свойства класса, которые определены в первичном конструкторе:

- **equals()**: сравнивает два объекта на равенство
- **hashCode()**: возвращает хеш-код объекта
- **toString()**: возвращает строковое представление объекта
- **copy()**: копирует данные объекта в другой объект

Например, возьмем функцию `toString()`, которая возвращает строковое представление объекта:

```
fun main() {

    val alice: Person = Person("Alice", 24)
    println(alice.toString())
}

class Person(val name: String, val age: Int)
```

Результатом программы будет следующий вывод:

```
Person(name=Alice, age=24)
```

По умолчанию строковое представление объекта нам практически ни о чем не говорит. Как правило, данная функция предназначена для вывода состояния объекта, но для этого ее надо переопределять. Однако теперь добавим модификатор data к определению класса:

```
data class Person(val name: String, val age: Int){
    override fun toString(): String {
        return "Name: $name  Age: $age"
    }
}
```

В этом случае для функции `toString()` компилятор не будет определять реализацию.

Другим показательным примером является копирование данных:

```
fun main() {

    val alice: Person = Person("Alice", 24)
    val kate = alice.copy(name = "Kate")
    println(alice.toString())   // Person(name=Alice, age=24)
    println(kate.toString())    // Person(name=Kate, age=24)
}

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

Опять же компилятор генерирует функцию копирования по умолчанию, которую мы можем использовать. Если мы хотим, чтобы некоторые данные у объкта отличались, то мы их можем указать в функции copy в виде именованных арументов, как в случае со свойством name в примере выше.

При этом чтобы класс определить как data-класс, он должен соответствовать ряду условий:

- Первичный конструктор должен иметь как минимум один параметр
- Все параметры первичного конструктора должны предваряться ключевыми словами **val** или **var**, то есть определять свойства. Свойства, которые определяются вне первичного конструктора, не используются в функциях toString, equals и hashCode
- Класс не должен определяться с модификаторами **open**, **abstract**, **sealed** или **inner**

Также стоит отметить, что несмотря на то, что мы можем определять свойства в первичном конструкторе и через val, и через var, например:

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

Но вообще в ряде ситуаций рекомендуется определять свойства через val, то есть делать их неизменяемыми, поскольку на их основании вычисляет хеш-код, который используется в качестве ключа объекта в такой коллекции как HashMap.

Декомпозиция data-классов

Kotlin предоставляет для data-классов возможность декомпозиции на переменные:

```
fun main() {

    val alice: Person = Person("Alice", 24)

    val (username, userage) = alice
    println("Name: $username  Age: $userage") // Name: Alice  Age: 24
}

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