Лекция #9.8 Перечисления enums

Enums или перечисления представляют тип данных, который позволяет определить набор логически связанных констант. Для определения перечисления применяются ключевые слова **enum class**. Например, определим перечисление:

```
enum class Day{
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
```

Данное перечисление Day представляет день недели. Внутри перечисления определяются константы. В данном случае это названия семи дней недели. Константы определяются через запятую. Каждая константа фактически представляет объект данного перечисления.

```
fun main() {

    val day: Day = Day.FRIDAY
    println(day)            // FRIDAY
    println(Day.MONDAY)     // MONDAY
}
```

Классы перечислений, как и обычные классы, также могут иметь конструктор. Кроме того, для констант перечисления также может вызываться конструктор для их инициализации.

```
enum class Day(val value: Int){
    MONDAY(1), TUESDAY(2), WEDNESDAY(3),
    THURSDAY(4), FRIDAY(5), SATURDAY(6), SUNDAY(100500)
}

fun main() {

    val day: Day = Day.FRIDAY
    println(day.value)        // 5
    println(Day.MONDAY.value) // 1
}
```

В примере выше у класса перечисления через конструктор определяется свойство value. Соответственно при определении констант перечисления необходимо каждую из этих констант инициализировать, передав значение для свойства value.

При этом перечисления - это не просто список значений. Они могут определять также свойства и функции. Но если класс перечисления содержит свойства или функции, то константы должны быть отделены точкой с запятой.

```
enum class Day(val value: Int){
    MONDAY(1), TUESDAY(2), WEDNESDAY(3),
    THURSDAY(4), FRIDAY(5), SATURDAY(6),
    SUNDAY(7);
    fun getDuration(day: Day): Int{
        return value - day.value;
    }
}

fun main() {

    val day1: Day = Day.FRIDAY
    val day2: Day = Day.MONDAY
    println(day1.getDuration(day2))        // 4
}
```

В данном случае в перечислении определена функция getDuration(), которая вычисляет разницу в днях между двумя днями недели.

Встроенные свойства и вспомогательные методы

Все перечисления обладают двумя встроенными свойствами:

- **name**: возвращает название константы в виде строки
- **ordinal**: возвращает порядковый номер константы

```
enum class Day(val value: Int){
    MONDAY(1), TUESDAY(2), WEDNESDAY(3),
    THURSDAY(4), FRIDAY(5), SATURDAY(6),
    SUNDAY(7)
}

fun main() {

    val day1: Day = Day.FRIDAY
    println(day1.name)        // FRIDAY
    println(day1.ordinal)     // 4
}
```

Кроме того, в Kotlin нам доступны вспомогательные функции:

- `valueOf(value: String)`: возвращает объект перечисления по названию константы
- `values()`: возвращает массив констант текущего перечисления

```
fun main() {

    for(day in Day.values())
        println(day)

    println(Day.valueOf("FRIDAY"))
}
```

Анонимные классы и реализация интерфейсов

Константы перечисления могут определять анонимные классы, которые могут иметь собственные методы и свойства или реализовать абстрактные методы класса перечисления:

```
enum class DayTime{
    DAY{
        override val startHour = 6
        override val endHour = 21
        override fun printName(){
            println("День")
        }
    },
    NIGHT{
        override val startHour = 22
        override val endHour = 5
        override fun printName(){
            println("Ночь")
        }
    };
    abstract fun printName()
    abstract val startHour: Int
    abstract val endHour: Int
}

fun main() {

    DayTime.DAY.printName()     // День
    DayTime.NIGHT.printName()   // Ночь

    println("Day from ${DayTime.DAY.startHour} to ${DayTime.DAY.endHour}")

}
```

В данном случае класс перечисления DayTime определяет абстрактный метод `printName()` и две переменных - `startHour` (начальный час) и `endHour` (конечный час). А константы определяют анонимные классы, которые реализуют эти свойства и функцию.

Также, классы перечислений могут применять интерфейсы. Для этого для каждой константы определяется анонимный класс, который содержат все реализуемые свойства и функции:

```
interface Printable{
    fun printName()
}
enum class DayTime: Printable{
    DAY{
        override fun printName(){
            println("День")
        }
    },
    NIGHT{
        override fun printName(){
            println("Ночь")
        }
    }
}

fun main() {

    DayTime.DAY.printName()     // День
    DayTime.NIGHT.printName()   // Ночь
}
```

Хранение состояния

Нередко перечисления применяются для хранения состояния в программе. И в зависимости от этого состояния мы можем направить действие программы по определенному пути. Например, определим перечисление, которое представляет арифметические операции, и функцию, которая в зависимости от переданной операции выполняет то или иное действие:

```
fun main() {

    println(operate(5, 6, Operation.ADD))         // 11
    println(operate(5, 6, Operation.SUBTRACT))   // -1
    println(operate(5, 6, Operation.MULTIPLY))   // 30
}
enum class Operation{

    ADD, SUBTRACT, MULTIPLY
}
fun operate(n1: Int, n2: Int, op: Operation): Int{

    when(op){
        Operation.ADD -> return n1 + n2
        Operation.SUBTRACT -> return n1 - n2
        Operation.MULTIPLY -> return n1 *n2
    }
}
```

Функция `operate()` принимает два числа - операнды операции и тип операции в виде перечисления Operation. И в зависимоси от значения перечисления возвращает либо сумму, либо разность, либо произведение двух чисел.