Тема 2. Знакомство с объектно-ориентированными возможностями языка Kotlin

Романов Владимир Юрьевич
МГУ им. М.В.Ломоносова, ф-т ВМК
romanov.rvy@yandex.ru

Ноябрь 7, 2020

1.2. ЗНАКОМСТВО С ОБЪЕКТНО-ОРИЕНТИРОВАННЫМИ ВОЗМОЖНОСТЯМИ

Объявление интерфейсов

```
interface Move {
    val piece: Piece?

    @Throws(GameOver::class)
    fun doMove()

    fun undoMove() {}
}
```

- значение для свойства **piece** в интерфейсе не хранится

- метод интерфейса в Kotlin может иметь *реализацию по умолчанию.*
  Ключевое слово ***default*** (как в Java) не требуется.
  Достаточно наличия тела метода.

Наследование интерфейсов

```
interface ITransferMove : Move {
    /**
     * Вернуть клетку откуда пошла фигура.
     */
    val source: Square

    /**
     * Вернуть клетку куда пошла фигура.
     */
    val target: Square
}
```

- в интерфейсе-потомке могут быть объявлены новые свойства и функции
- часть свойств и функций в интерфейсе-потомке может быть реализована

Реализация интерфейсов

В языке Kotlin применение модификатора ***override*** обязательно

```
open class SimpleMove(
        override val source: Square,
        override val target: Square,
) : ITransferMove {
    override var piece: Piece? = null

    init { piece = source.getPiece()!! }

    override fun doMove() {
        piece?.moveTo(target)
    }
    override fun undoMove() {
        piece?.moveTo(source)
    }
}
```

Реализация свойств интерфейсов

- свойство интерфейса может быть реализовано как *переменная* ИЛИ как *метод доступа*

```
interface IPlayer {
    val name: String

    val authorName: String
}
class Neznaika : MovePiecePlayer() {
  override val name: String
            get() = "Незнайка"
  override val authorName: String
            = "Романов В.Ю."
}
```

Модификатор класса abstract

```
abstract
class Piece(var square: Square,
            var color: PieceColor) {
  abstract
  fun isCorrectMove(vararg squares: Square)
      : Boolean

  abstract
  fun makeMove(vararg squares: Square): Move
}
```

Модификатор класса abstract

```
abstract
class ChessPiece(square: Square,
                 color: PieceColor)
        : Piece(square, color)
{
  override fun isCorrectMove(
           vararg squares: Square): Boolean {
    val target = squares[0]
    val piece = target.getPiece() ?: return true

    return color !== piece.color
  }
}
```

Модификатор класса open

```
open class Capture(source: Square, target: Square)
    : SimpleMove(source, target), ICaptureMove {
    var capturedSquare: Square = target
    var capturedPiece: Piece = target.piece!!

    override val captured: List<Square>
        get() = listOf(capturedSquare)

    override fun doMove() {
        capturedPiece.remove()
        super.doMove()
    }
    override fun undoMove() {
        super.undoMove()
        capturedSquare.setPiece(capturedPiece)
    }
}
```

Модификатор класса final

Класс по умолчанию с модификатором ***final***

```
class Promotion(source: Square, target: Square)
    : SimpleMove(source, target), ICaptureMove {
    private val pawn: Piece = source.piece!!
    private var capturedPiece: Piece? = null
    private var promotedPiece: Piece? = null

    // ...
}
```