Алгоритмизация и программирование

7. Классы и интерфейсы

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Что? Каким образом?

- “Что умеет делать объект?” → интерфейс

Что? Каким образом?

- “Что умеет делать объект?” → интерфейс
- Примеры интерфейсов
  - Iterable, Collection, List, Set, Map, CharSequence, …

Что? Каким образом?

- “Что умеет делать объект?” → интерфейс
- Примеры интерфейсов
  - Iterable, Collection, List, Set, Map, CharSequence, …

- “Каким образом он это делает?” → класс

Что? Каким образом?

- “Что умеет делать объект?” → интерфейс
- Примеры интерфейсов
  - Iterable, Collection, List, Set, Map, CharSequence, …

- “Каким образом он это делает?” → класс
- Примеры классов
  - ArrayList, LinkedList, HashSet, HashMap, String, …

Пример: матрица

```
data class Cell(val row: Int, val column: Int)

interface Matrix<E> {
    val height: Int
    val width: Int

    operator fun get(row: Int, column: Int): E
    operator fun get(cell: Cell): E

    operator fun set(row: Int, column: Int, value: E)
    operator fun set(cell: Cell, value: E)
}
```

Настраиваемый тип <…>

- Пример: Matrix<E>
- Матрица из элементов типа E

Настраиваемый тип <…>

- Пример: Matrix<E>
- Матрица из элементов типа E

- Аналоги: List<E>, Set<E>, Map<K, V>, …

Пример: использование матрицы

```
fun invertMatrix(matrix: Matrix<Int>) {
    for (row in 0..matrix.height - 1) {
        for (column in 0..matrix.width - 1) {
            matrix[row, column] = -matrix[row, column]
        }
    }
}
```

Пример: использование матрицы

```
fun invertMatrix(matrix: Matrix<Int>) {
    for (row in 0..matrix.height - 1) {
        for (column in 0..matrix.width - 1) {
            matrix.set(row, column,
                       -matrix.get(row, column))
        }
    }
}
```

Функции-создатели

- Интерфейс НЕ может иметь конструктор

Функции-создатели

- Интерфейс НЕ может иметь конструктор
- Вместо них часто используют factory methods

Функции-создатели

- Интерфейс НЕ может иметь конструктор
- Вместо них часто используют factory functions

```
fun <E> createMatrix(
        height: Int, width: Int, e: E
): Matrix<E> = TODO()
```

Функции-создатели

- Интерфейс НЕ может иметь конструктор
- Вместо них часто используют factory functions

```
fun <E> createMatrix(
        height: Int, width: Int, e: E
): Matrix<E> = TODO()

// Аналоги: listOf(), setOf(), ...
```

Пример использования функции-создателя

```
fun <E> transpose(matrix: Matrix<E>): Matrix<E> {
    if (matrix.width < 1 || matrix.height < 1) {
        return matrix
    }
    val result = createMatrix(
            height = matrix.width,
            width = matrix.height, e = matrix[0, 0])
    for (i in 0..matrix.width - 1) {
        for (j in 0..matrix.height - 1) {
            result[i, j] = matrix[j, i]
        }
    }
    return result
}
```

Реализация: скелет

```
class MatrixImpl<E> : Matrix<E> {
    override val height: Int = TODO()
    override val width: Int = TODO()

    override fun get(row: Int, column: Int): E = TODO()
    override fun get(cell: Cell): E = TODO()
    override fun set(row: Int, column: Int, value: E) {
        TODO()
    }
    override fun set(cell: Cell, value: E) {
        TODO()
    }
    override fun equals(other: Any?) = TODO()
    override fun toString(): String = TODO()
}
```

Матрица = …

- Высота + Ширина + Набор элементов
- Высота + Ширина:
  - class MatrixImpl<E>(val height: **Int**, val width: **Int**) …
- Как представить элементы?

Матрица = …

- Высота + Ширина + Набор элементов
- Высота + Ширина:
  - class MatrixImpl<E>(val height: **Int**, val width: **Int**) …
- Как представить элементы?

Матрица = …

- Высота + Ширина + Набор элементов
- Высота + Ширина:
  - class MatrixImpl<E>(val height: **Int**, val width: **Int**) …
- Как представить элементы?
  - Сквозной список
  - Массив массивов
  - Ассоциативный массив

Сквозной список

```
class MatrixImpl<E>(
        override val height: Int,
        override val width: Int,
        e: E
) : Matrix<E> {
    private val list = mutableListOf<E>()

    init {
        for (i in ...) {
            list.add(e)
        }
    }
    override fun get(row: Int, column: Int): E =
            list[...]
    // Other functions...
}
```

Массив массивов

```
class MatrixImpl<E>(
        override val height: Int,
        override val width: Int,
        e: E
) : Matrix<E> {
    private val array = Array(height) {
        row -> Array ... // element #row
    }

    override fun get(row: Int, column: Int): E =
            array[...][...]
    // Other functions...
}
```

Ассоциативный массив (Map)

```
class MatrixImpl<E>(
        override val height: Int,
        override val width: Int,
        e: E
) : Matrix<E> {
    private val map = mutableMapOf<Cell, E>()

    init {
        this[0, 0] = e
        this[0, 1] = e // loop!
    }

    override fun get(cell: Cell): E = list[cell] // null!
    // Other functions...
}
```

Any

```
class Any {
    open fun equals(other: Any?): Boolean = ...

    open fun hashCode(): Int = ...

    open fun toString(): String = ...
}
```

Функция equals: свойства

- Что угодно равно самому себе
- Если A равно B, то B равно A
- Если A равно B и B равно C, то A равно C

Функция equals: свойства

- Что угодно равно самому себе
- Если A равно B, то B равно A
- Если A равно B и B равно C, то A равно C
- Никакое значение из типа **Any** не может быть равно null

Функция equals: свойства

- Что угодно равно самому себе
- Если A равно B, то B равно A
- Если A равно B и B равно C, то A равно C
- Никакое значение из типа **Any** не может быть равно null
- Результат сравнения A и B не должен меняться при повторном вызове equals, ЕСЛИ внутреннее состояние A и B не изменилось между вызовами

Функция hashCode: свойства

- Если A равно B,
  то A.hashCode() равно B.hashCode()
- Если A не равно B,
  то, КАК ПРАВИЛО (но не всегда),
  A.hashCode() не равно B.hashCode()

Реализация equals / hashCode

```
class MatrixImpl<E> : Matrix<E> {
    override val height: Int = TODO()
    override val width: Int = TODO()
    // ... Other functions ...

    override fun equals(other: Any?) =
            other is MatrixImpl<*> &&
            height == other.height &&
            width == other.width // && elements comparison

    override fun hashCode(): Int {
        var result = 5
        result = result * 31 + height
        result = result * 31 + width
        // Something for elements...
        return result
    }
}
```

Реализация toString

```
class MatrixImpl<E> : Matrix<E> {
    override val height: Int = TODO()
    override val width: Int = TODO()
    // ... Other functions ...

    override fun toString(): String {
        val sb = StringBuilder()
        sb.append("[")
        for (row in 0..height - 1) {
            sb.append("[")
            for (column in 0..width - 1) {
                sb.append(this[row, column]) // Spaces!
            }
            sb.append("]")
        }
        sb.append("]")
        return "$sb" // or, sb.toString()
    }
}
```

Упражнения к лекции

- См. lesson7/task1 и tasks2 в обучающем проекте
  - Task1 = на реализацию матрицы (выберите любую)
  - Task2 = на операции с матрицей
  - Необходимо решить все задачи из Task1
  - + хотя бы одно из Task2
- Протестируйте решения с помощью готовых тестов
- Добавьте ещё хотя бы один тестовый случай
- Добавьте коммит в свой репозиторий
- Создайте Pull Request и убедитесь в правильности решения