Программирование на Java

2. Разработка классов

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Общие принципы

- Любой класс описывает некоторое понятие (существительное), относящееся к решаемой задаче

Общие принципы

- Любой класс описывает некоторое понятие (существительное), относящееся к решаемой задаче
- Класс объединяет в себе данные и функции (методы) для работы с ними

Общие принципы

- Любой класс описывает некоторое понятие (существительное), относящееся к решаемой задаче
- Класс объединяет в себе данные и функции (методы) для работы с ними
- Данные, как правило, закрыты для пользователей класса, а методы, как правило, открыты (инкапсуляция)

Общие принципы

- Любой класс описывает некоторое понятие (существительное), относящееся к решаемой задаче
- Класс объединяет в себе данные и функции (методы) для работы с ними
- Данные, как правило, закрыты для пользователей класса, а методы, как правило, открыты (инкапсуляция)
- Неизменяемые классы предпочтительнее

Пример: рациональное число

- Содержит числитель и знаменатель
- А также описание ряда операций над подобными числами

- См. пример part1.Rational в проекте FromJavaToKotlin

Рациональное число: поля

```
public final class Rational {
    private final int numerator;

    private final int denominator;
}
```

Модификатор final

- Поле (или переменная) получает значение только один раз
  - = val в Котлине

Модификатор final

- Поле (или переменная) получает значение только один раз
  - = val в Котлине
- Класс не может иметь разновидности (наследников)

Модификатор final

- Поле (или переменная) получает значение только один раз
  - = val в Котлине
- Класс не может иметь разновидности (наследников)
- Модификатор крайне важен, поскольку обеспечивает неизменяемость

Почему private & final?

- Это позволяет защититься от случайных изменений объекта

Почему private & final?

- Это позволяет защититься от случайных изменений объекта
- Это позволяет в будущем изменить представление, не меняя основной код (скажем, заменить int на long)

Стоит ли делать public поля?

- Если они также изменяемые (не final), это очень опасно – кто угодно может изменить внутреннее состояние объекта

Стоит ли делать public поля?

- Если они также изменяемые (не final), это очень опасно – кто угодно может изменить внутреннее состояние объекта

- Если они final, это менее опасно, но всё равно не рекомендуется (если класс использует кто-то внешний, поле придётся оставить навсегда без изменений)

Рациональное число: конструктор

```
public final class Rational {
    public Rational(int numerator,
                    int denominator) {
        if (denominator == 0)
            throw new ArithmeticException("");
        int gcd = gcd(abs(numerator),
                      abs(denominator));
        if (denominator < 0) gcd = -gcd;
        this.numerator = numerator / gcd;
        this.denominator = denominator / gcd;
    }
}
```

Рациональное число: геттеры

```
public final class Rational {
    private final int numerator;

    private final int denominator;

    public int getNumerator() {
        return numerator;
    }
}
```

Конструктор vs метод

- Конструктор = специальный метод
- У него нет результата, но могут быть параметры
- Имя конструктора всегда совпадает с именем класса
- Вызывается при создании объекта класса: new Rational(…)

Рациональное число: пример метода

```
public final class Rational {
    private final int numerator;

    private final int denominator;

    public Rational plus(Rational other) {
        return new Rational(
                numerator * other.denominator +
                denominator * other.numerator,

                denominator * other.denominator
        );
    }
}
```

Рациональное число: пример теста

```
public class RationalTest {
    @Test
    public void plus() {
        assertEquals(new Rational(1, 2),
                new Rational(1, 6).plus(
                        new Rational(1, 3))
                );
    }
}
```

Тип void

- void = пустота (англ)
- Специальный тип «отсутствие результата»
- Тестовые функции JUnit результата не имеют

Рациональное число: equals

```
public final class Rational {
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj instanceof Rational) {
            Rational other = (Rational) obj;
            return numerator == other.numerator
             && denominator == other.denominator;
        }
        return false;
    }
}
```

Рациональное число: hashCode + toString

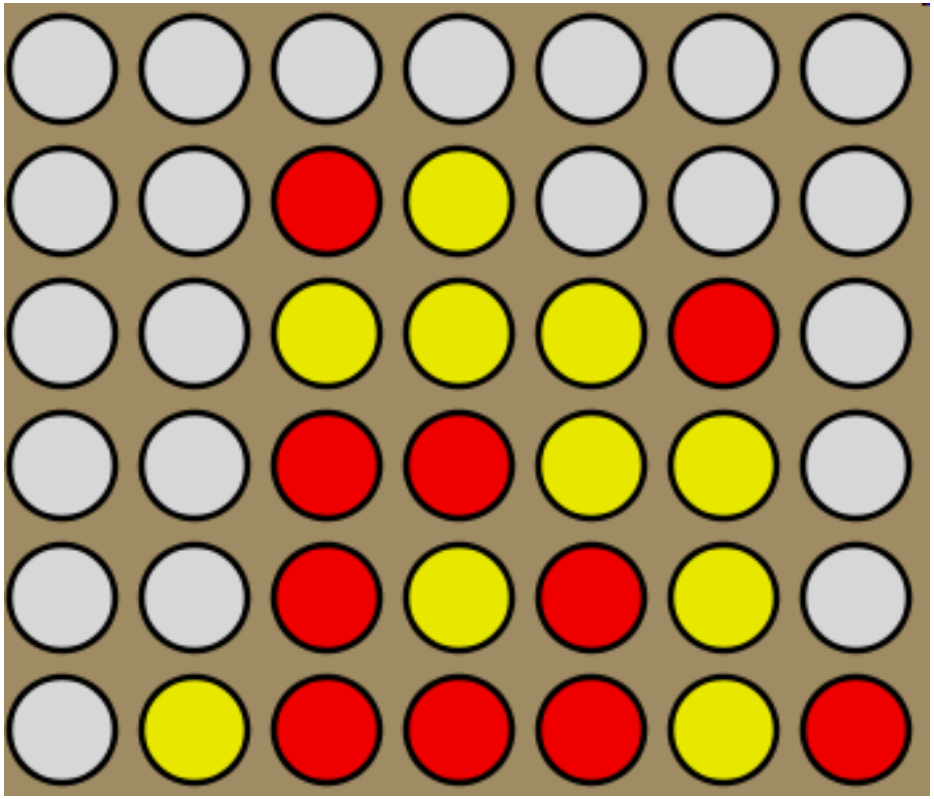
```
public final class Rational {
    @Override
    public int hashCode() {
        int result = numerator;
        result = 31 * result + denominator;
        return result;
    }

    @Override
    public String toString() {
        return "" + numerator + "/" + denominator;
    }
}
```

Пример: игра «4 в ряд»

- См. part1.fourinrow.Chip, Cell, Board

Пример: игра «4 в ряд»



4 в ряд: правила

- Прямоугольное поле 7 x 6 клеток
- 2 игрока ходят по очереди
- Ход: добавление своей фишки в одну из колонок (она падает на дно колонки)
- 4 фишки своего цвета в ряд = победа, по горизонтали, вертикали или диагонали
- Поле заполнено, никто не выиграл = ничья

4 в ряд: понятия

- Фишка (Chip)
- Клетка (Cell)
- Доска (Board)

Chip

- Бывает двух цветов
- Типичный пример «перечисления»
  - = enum …

Chip

```
public enum Chip {
    YELLOW,
    RED;

    public Chip opposite() {
        if (this == YELLOW) return RED;
        else return YELLOW;
    }
}
```

this

- = “Этот” самый объект, для которого вызван метод
- = Объект-получатель

Cell (класс с данными)

```
public class Cell {
    private final int x;

    private final int y;

    public Cell(int x, int y) {
        this.x = x;
        this.y = y;
    }
    // + getters, equals, hashCode, toString
}
```

Board: поля

```
public class Board {

    static public final int TO_WIN_LENGTH = 4;

    private final int width;

    private final int height;

    private final Map<Cell, Chip> chips =
        new HashMap<>();

    private Chip turn = Chip.YELLOW;
}
```

static final…

- Выполняют роль констант
- Именуются прописными буквами

Статические поля / методы

- Статическое поле ОДНО на ВСЕ объекты класса

Статические поля / методы

- Статическое поле ОДНО на ВСЕ объекты класса
  - А нестатическое – одно на каждого

Статические поля / методы

- Статическое поле ОДНО на ВСЕ объекты класса
  - А нестатическое – одно на каждого
- Статическому методу не нужен объект класса
  - Он имеет доступ только к статическим полям

Статические поля / методы

- Статическое поле ОДНО на ВСЕ объекты класса
  - А нестатическое – одно на каждого
- Статическому методу не нужен объект класса
  - Он имеет доступ только к статическим полям
- Нестатическому методу нужен объект класса
  - Он имеет доступ ко всем полям

Map / HashMap

- Map – интерфейс (ассоциативный массив)
  - NB: Java не различает неизменяемые и изменяемые контейнеры!

Map / HashMap

- Map – интерфейс (ассоциативный массив)
  - NB: Java не различает неизменяемые и изменяемые контейнеры!
  - NB: в Java также отсутствуют часть методов…

Map / HashMap

- Map – интерфейс (ассоциативный массив)
  - NB: Java не различает неизменяемые и изменяемые контейнеры!
  - NB: в Java также отсутствуют часть методов…
- HashMap – класс (реализация на основе хэш-таблицы)

Board: выполнение хода

```
public Cell makeTurn(int x) {
    if (x < 0 || x >= width) return null;
    for (int y = 0; y < height; y++) {
        Cell cell = new Cell(x, y);
        if (!chips.containsKey(cell)) {
            chips.put(cell, turn);
            turn = turn.opposite();
            return cell;
        }
    }
    return null;
}
```

NotNull / Nullable

- Java сама по себе не различает такие вещи
- Есть, однако, аннотации @NotNull и @Nullable

NotNull / Nullable

- Java сама по себе не различает такие вещи
- Есть, однако, аннотации @NotNull и @Nullable

- NB: примитивные типы в Java не могут содержать null

Board: проверка возможности хода

```
public boolean hasFreeCells() {
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            if (get(x, y) == null)
                return true;
        }
    }
    return false;
} // NB: можно проще, как?
```

Board: поиск победителя

- См. в IDE
- Необходимо проверить все горизонтали, вертикали и диагонали из 4 элементов

Board: пример теста

```
@Test public void winner() {
    Board field = new Board(7, 6);
    int[] turns = new int[] {
        3, 4, 2, 5, 1, 6, 0
    };
    for (int turn: turns) {
        assertNotNull(field.makeTurn(turn));
    }
    assertEquals(YELLOW, field.get(0, 0));
    assertEquals(YELLOW, field.get(2, 0));
    assertEquals(YELLOW, field.winner());
    assertTrue(field.hasFreeCells());
}
```

Board: пример использования

- См. в IDE

Итоги

- Рассмотрели порядок проектирования и тестирования класса
- Далее: классы и интерфейсы в Java