

Теория и технология программирования

Программирование на языке Java

Лекция 3. Проектирование классов в Java

Глухих Михаил Игоревич, к.т.н., доц.

[mailto: glukhikh@mail.ru](mailto:glukhikh@mail.ru)

Общий базовый класс

- ❑ В языке Java ЛЮБОЙ класс является неявным наследником класса Object (иначе говоря, экземпляр любого класса ЯВЛЯЕТСЯ объектом)
- ❑ Что такое Object?
 - `public boolean equals(Object o);`
 - `public int hashCode();`
 - `public String toString();`
 - `public Class getClass();`
 - `protected Object clone();`
 - `protected void finalize();`
 - + методы синхронизации потоков

Методы класса Object

- ❑ equals – сравнение двух ЛЮБЫХ ОБЪЕКТОВ на равенство СОДЕРЖИМОГО; по умолчанию – каждый объект равен ТОЛЬКО самому себе
- ❑ свойства операции сравнения на равенство:
 - рефлексивность – любой объект ВСЕГДА равен самому себе
 - симметричность – если `x.equals(y)`, то `y.equals(x)` и наоборот
 - транзитивность – если `x.equals(y)` и `y.equals(z)`, то `x.equals(z)`
 - никакой объект не равен `null`
 - консистентность
- ❑ сравнение на равенство используется в некоторых методах коллекций

Методы класса Object

- ❑ hashCode – формирование хэш-кода объекта; хэш-коды РАВНЫХ объектов (с точки зрения equals) ДОЛЖНЫ быть равны; хэш-коды НЕРАВНЫХ объектов ПО ВОЗМОЖНОСТИ должны различаться; по умолчанию хэш-код равен адресу объекта
- ❑ Если в некотором классе переопределен метод equals, НЕОБХОДИМО переопределить метод hashCode

Методы класса Object

- ❑ `toString` – формирование строкового представления объекта; по умолчанию формируется из адреса объекта
- ❑ `getClass` – возвращает объект типа `Class`, имеющий доступ к спискам полей и методов данного типа (Reflection, рефлексия, интроспекция – отслеживание собственной структуры)

Методы класса Object

- ❑ `clone()` – возвращает копию данного объекта
- ❑ `finalize()` – вызывается сборщиком мусора перед разрушением объекта

Зачем нужен класс Object?

- ❑ Мы получаем возможность создавать массивы (контейнеры) из объектов произвольного типа (на самом деле, контейнеры как раз хранят внутри себя ссылки типа Object)
- ❑ Мы получаем возможность сравнить два объекта любого типа на равенство
- ❑ Мы получаем возможность получить строковое представление любого объекта
- ❑ ...
- ❑ То есть, класс Object содержит общие свойства всех объектов Java

Проектирование класса – на примере класса Integer

```
public final class Integer {  
    public static final int MIN_VALUE=0x80000000;  
    public static final int MAX_VALUE=0x7fffffff;  
    public static int parseInt(String s)  
        throws NumberFormatException { ... }  
    private final int value;  
    public Integer(int value) {  
        this.value = value;  
    }  
    public Integer(String s) throws NumberFormatException {  
        this.value = parseInt(s);  
    }  
    public int intValue() {  
        return value;  
    }  
}
```


Проектирование класса – на примере класса Integer

```
public final class Integer {  
    @Override  
    public String toString() {  
        return String.valueOf(value);  
    }  
    @Override  
    public int hashCode() {  
        return value;  
    }  
    @Override  
    public boolean equals(Object obj) {  
        if (obj instanceof Integer)  
            return value==((Integer)obj).intValue();  
        return false;  
    }  
}
```

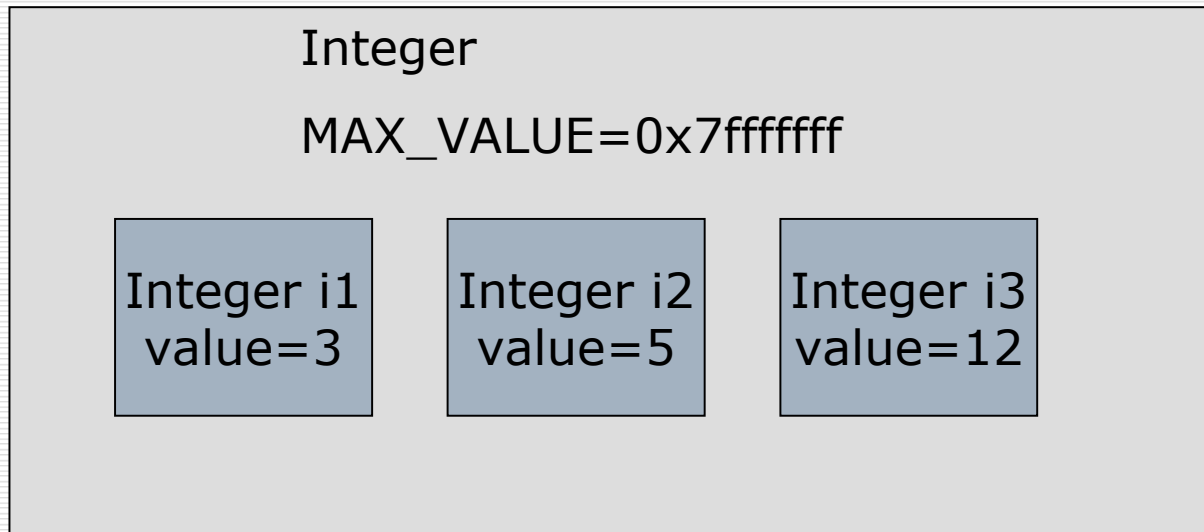
Спецификаторы класса

- ❑ `public` – класс доступен где угодно (без этого – класс доступен в том же пакете)
- ❑ `final` – классу запрещается иметь наследников

Спецификаторы полей

- ❑ `private` – поле доступно только в том же классе
- ❑ `public` – поле доступно где угодно
- ❑ `protected` – поле доступно в том же пакете и в классах-наследниках
- ❑ `private` / `public` / `protected` не указано – поле доступно в том же пакете
- ❑ `final` – поле принимает значение только один раз (в конструкторе) и более не изменяется
- ❑ `static` – поле общее для всех объектов данного класса (обращение через имя класса – `Integer.MAX_VALUE`)

Статические и обычные поля



Спецификаторы методов

- ❑ `private` – метод доступен только в том же классе
- ❑ `public` – метод доступен где угодно
- ❑ `protected` – метод доступен в том же пакете и в классах-наследниках
- ❑ `private` / `public` / `protected` не указано – метод доступен в том же пакете
- ❑ `final` – запрещается переопределять данный метод в классах-наследниках (если сам класс `final`, все его методы автоматически `final`)
- ❑ `static` – данный метод может работать только со статическими членами класса (обращение через имя класса: `Integer.parseInt("23")`)
- ❑ `@Override` – аннотация, показывает, что метод переопределяет метод базового класса; переопределять можно только нестатические методы

Другие ключевые слова

- `this` – доступ к объекту, для которого вызван метод (только для нестатических методов)
- `obj instanceof Integer` – является ли `obj` объектом класса `Integer`

Документирующие комментарии

```
□ /**
   * The <code>Integer</code> class wraps a value of the
   * primitive type <code>int</code> in an object.<p>
   * In addition, this class provides several methods for
   * converting an <code>int</code> to
   * a <code>String</code>, as well as other constants and
   * useful methods ...
   * @author Lee Boynton
   * @version 1.93
   * @since JDK1.0
   */
□ public final class Integer {
□     ...
□ }
```

Документирующие комментарии

```
public final class Integer {  
    /**  
     * A constant holding the minimum value an  
     * <code>int</code> can have, -2<sup>31</sup>.  
     */  
    public static final int MIN_VALUE=0x80000000;  
    /**  
     * Constructs a newly allocated <code>Integer</code>  
     * object that represents the specified <code>int</code>  
     * value.  
     * @param value the value to be represented  
     */  
    public Integer(int value) { ... }  
}
```


Проектирование своего класса – интервал

- Задача – спроектировать класс "интервал целых значений" $[min:max]$
- Необходимые методы:
 - объединение и пересечение
 - сложение и вычитание:
если $a=[1,2]$ и $b=[3,4]$,
то $a+b=[4,6]$, $b-a=[1,3]$
 - инвертирование:
если $a=[-2,-1]$, то $-a=[1,2]$
 - сравнение на равенство:
интервалы равны, если их границы равны

Проектирование своего класса – интервал

- Дополнительные методы:
 - конструкторы [0:0], [val:val], [min:max]
 - преобразование в строку
 - хэш-код

Проектирование своего класса – поля класса

```
/**
 * Интервал целых значений
 * @author Mikhail Glukhikh
 */
public class IntegerInterval implements Cloneable {
    /** Нижняя граница */
    private final int min;
    /** Верхняя граница */
    private final int max;
    /**
     * Конструктор нуля
     */
    public IntegerInterval() {
        this(0, 0);
    }
}
```

Проектирование своего класса – основной конструктор

```
/**
 * Конструктор полноценного интервала
 * @param min нижняя граница
 * @param max верхняя граница
 * @throws IllegalArgumentException
 * если нижняя граница больше верхней
 */
public IntegerInterval(int min, int max) throws
    IllegalArgumentException {
    if (min > max)
        throw new IllegalArgumentException(
            "Нижняя граница " + min +
            " больше верхней " + max);
    this.min = min;
    this.max = max;
}
```

Проектирование своего класса – клонирование, границы

```
/**
 * Клонирование
 * @return глубокая копия интервала
 */
@Override
public IntegerInterval clone() throws
    CloneNotSupportedException {
    return (IntegerInterval) super.clone();
}

/**
 * Получить нижнюю границу
 * @return нижняя граница
 */
public int getMin() {
    return min;
}
```

Проектирование своего класса – объединение интервалов

```
/**
 * Объединиться с интервалом
 * @param variant второй интервал
 * @return результат объединения
 */
public IntegerInterval disj(IntegerInterval variant) {
    final int min1 = this.getMin();
    final int min2 = variant.getMin();
    final int max1 = this.getMax();
    final int max2 = variant.getMax();
    return new IntegerInterval(
        min1 < min2 ? min1 : min2, max1 > max2 ? max1 : max2);
}
```

Проектирование своего класса – пересечение интервалов

```
/**
 * Пересечься с интервалом
 * @param variant второй интервал
 * @return результат пересечения
 * @throws IllegalArgumentException
 * если интервалы не пересекаются */
public IntegerInterval conj(IntegerInterval variant) throws
    IllegalArgumentException {
    final int min1 = this.getMin(), min2 = variant.getMin();
    final int max1 = this.getMax(), max2 = variant.getMax();
    final int resMin = min1 > min2 ? min1 : min2;
    final int resMax = max1 < max2 ? max1 : max2;
    if (resMin > resMax) throw new IllegalArgumentException(
        "Интервалы " + this + " и " + variant +
        " не пересекаются");
    return new IntegerInterval(resMin, resMax);
}
```

Проектирование своего класса – вычитание интервалов

```
/**
 * Операция вычитание (!ATTENTION!)
 * @param value второй аргумент
 * @return результат операции
 */
public IntegerInterval sub(IntegerInterval variant) {
    final int rmin = this.getMin() - variant.getMax();
    final int rmax = this.getMin() - variant.getMax();
    return new IntegerInterval(rmin, rmax);
}
```


Проектирование своего класса – преобразование в строку

```
/**
 * Преобразование в строку
 * @return строковое представление в форме a:b
 */
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append(min).append(':');
    return sb.append(max).toString();
}
```

Проектирование своего класса – сравнение на равенство

```
/**
 * Сравнение на равенство
 * @param obj сравниваемый объект
 * @return true, если интервалы равны
 * (обе границы совпадают)
 */
@Override
public boolean equals(Object obj) {
    if (obj==this)
        return true;
    else if (obj instanceof IntegerInterval) {
        final IntegerInterval var = (IntegerInterval)obj;
        return this.min == var.min && this.max == var.max;
    } else return false;
}
```

Проектирование своего класса – ХЭШ-КОД

```
/**
 * Хэш-код интервала
 * @return хэш-код
 */
@Override
public int hashCode() {
    int hash = 5;
    hash = 29 * hash + this.min;
    hash = 29 * hash + this.max;
    return hash;
}
```

Проектирование тестов класса

- ❑ Тесты предназначены для проверки правильности функционирования спроектированного класса
- ❑ Классическая структура теста – выполняем некоторую операцию, например, `[1:2]+[3:4]`, и сравниваем результат с ожидаемым `[4:6]`
- ❑ Тесты находятся в группе пакетов `test` (Test Packages); как правило, тест класса располагается в том же пакете, что и сам класс, и называется как `<ClassName>Test`

JUnit

- ❑ Библиотека для тестирования Java-классов
- ❑ Отдельный jar-файл, то есть её надо подключать к проекту
- ❑ В IDEA – Project Structure / Libraries
- ❑ Добавлять проще через Maven (глобальный репозиторий Java-библиотек + зависимости между ними + система их сборки): `junit:junit:4.12`

Тестирующий класс

```
package intervals;

import org.junit.Test;
import static org.junit.Assert.*;

/**
 * Тесты интервала целых значений
 * @author Mikhail Glukhikh
 */
public class IntegerIntervalTest {
}
```

Тестирующий класс – примеры тестов

```
@Test
public void testAdd() {
    final IntegerInterval var1 = new IntegerInterval(-6, 4);
    final IntegerInterval var2 = new IntegerInterval(2, 5);
    final IntegerInterval res = new IntegerInterval(-4, 9);
    assertEquals(res, var1.add(var2));
    assertEquals(res, var2.add(var1));
}

@Test
public void testSub() {
    final IntegerInterval var1 = new IntegerInterval(-6, 4);
    final IntegerInterval var2 = new IntegerInterval(2, 5);
    final IntegerInterval res = new IntegerInterval(-11, 2);
    assertEquals(res, var1.sub(var2));
    assertEquals(res, var2.sub(var1));
}
```

Тестирующий класс – примеры тестов

```
@Test
public void testClone() {
    try {
        final IntegerInterval var =
            new IntegerInterval(10, 20);
        final IntegerInterval res = var.clone();
        assertEquals(var, res);
        assertNotSame(var, res);
    } catch (CloneNotSupportedException ex) {
        fail("Клонирование не поддерживается: " +
            ex.getMessage());
    }
}
```


Тестирующий класс – примеры тестов

```
@Test
public void testConj2() {
    try {
        final IntegerInterval var1 =
            new IntegerInterval(1, 2);
        final IntegerInterval var2 =
            new IntegerInterval(3, 4);
        var1.conj(var2);
        fail("Должно было произойти исключение");
    } catch (IllegalArgumentException ex) {}
}
```

Построение Java-документации

- ❑ Выполняется командой `Generate JavaDoc` из контекстного меню проекта (NetBeans) или из меню Tools (IDEA)
- ❑ Документация формируется в формате `html` и помещается в каталог `dist/javadoc` (`out/javadoc`)
- ❑ Просматривается любым браузером

Итоги

- ❑ Рассмотрели класс Object и его методы
- ❑ А также примерную структуру класса на Java
- ❑ Основные связанные ключевые слова (public, protected, private, final, static)
- ❑ Примеры и способы тестирования