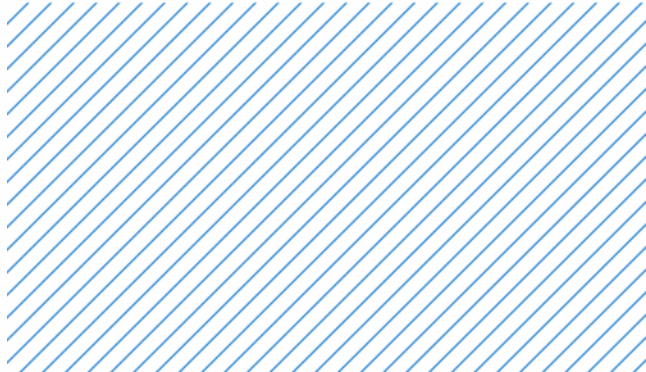
HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Лекция 7

Основы объектно-ориентированного программирования

Программирование на языке Java

Роман Гуров
ВШЭ БИ 2021



Класс

С понятием класса мы уже немного знакомы:

```
import java.util.Scanner;
import java.math.BigInteger;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BigInteger value = scanner.nextBigInteger();
        System.out.println(value.pow(1000));
    }
}
```

Переменные объектного типа хранят в себе ссылку на *объект класса*

Класс – это пользовательский тип данных

Объектно-ориентированное программирование – парадигма программирования, основывающаяся на концепции таких «объектов»

Инкапсуляция

Класс даёт возможность объединить в одной сущности
определённый набор данных и функций над ними

Это объединение является одним из основополагающих принципов ООП
и называется *инкапсуляцией*

Также, инкапсуляция включает в себя и понятие сокрытия – объект открывает только свой некоторый внешний интерфейс, но не выдаёт наружу всю внутреннюю кухню

BigInteger хранит внутри себя число в некотором виде (например, массив цифр), но не даёт нам возможности как-то на этот вид взглянуть или поменять данные напрямую

В этом плане инкапсуляция воплощает концепцию «черного ящика»

Класс и объект

Важно не путать понятия объекта и класса

```
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BigInteger value = scanner.nextBigInteger();
        System.out.println(value.pow(1000));
    }
}
```

Класс – описательный шаблон, по которому можно создать объект

Объект – конкретный экземпляр класса, хранящий в себе данные
и позволяющий вызывать над ними функции

Объект класса создаётся при помощи оператора `new`

Поля класса

Поле класса – переменная, которая будет храниться внутри объекта класса

```
public class Car {
    public float speed;
    public String manufacturer;
}
```

```
public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.speed = 200.f;
        car.manufacturer = "Ford";
        ...
    }
}
```

Поля описывают *данные* класса

Методы класса

Метод класса – функция, вызываемая у объекта класса и имеющая доступ к его полям

```
public class Car {
    public float speed_kph;
    public String manufacturer;

    public float getDistance(float time_seconds) {
        return speed_kph / 3.6f * time_seconds;
    }
}
```

```
Car car = new Car();
car.speed_kph = 200.f;
car.manufacturer = "Ford";
float distance_2_minute = car.getDistance(120);
System.out.println(distance_2_minute);
```

Методы описывают *действия* над данными класса

Конструктор

Не хочется каждый раз при создании класса вбивать в него все данные отдельно

Конструктор – специальный метод класса, вызываемый в момент создания объекта для его инициализации

```
public class Car {
    public float speed_kph;
    public String manufacturer;
    public String model;

    public Car(float speed_kph, String manufacturer, String model) {
        this.speed_kph = speed_kph;
        this.manufacturer = manufacturer;
        this.model = model;
    }
}
```

```
Car car = new Car(220.f, "BMW", "E36");
```

Ключевое слово `this` ссылается на текущий объект, в методе которого мы находимся
Пригождается, когда имя параметра метода совпадает с именем поля

Делегация конструкторов

Иногда, параметров много и хочется дать возможность не указывать значения некоторых из них

Для этого можно определить несколько версий конструктора для разных наборов параметров

```
public class Car {
    public float speed_kph;
    public String manufacturer;
    public String model;

    public Car(float speed_kph, String manufacturer, String model) {
        this.speed_kph = speed_kph;
        this.manufacturer = manufacturer;
        this.model = model;
    }

    public Car(float speed_kph, String manufacturer) {
        this(speed_kph, manufacturer, "unknown");
    }

    public Car(float speed_kph) {
        this(speed_kph, "unknown");
    }

    public Car() {
        this(0.f);
    }
}
```

Это ещё одно применение для ключевого слова `this`

Делегироваться можно только в один конструктор и он должен быть первой строчкой

Перегрузка методов

Возможность определять методы с одним именем и разными параметрами не ограничивается конструкторами и не обязательно связана с делегацией вызовов

Это понятие называют *перегрузкой методов*

```
public void println() { this.newLine(); }

public void println(boolean x) {...}

public void println(char x) {...}

public void println(int x) {...}

public void println(long x) {...}

public void println(float x) {...}

public void println(double x) {...}

public void println( @NotNull char[] x) {...}

public void println( @Nullable String x) {...}

public void println( @Nullable Object x) {...}
```

Модификаторы доступа

А что же с сокрытием?

Существует три модификатора доступа для полей и методов:
`public`, `private` и `protected`

- `public` доступен снаружи
- `private` доступен только в методах своего класса, снаружи невидим
- `protected` как `private`, но доступен ещё и в классах-наследниках (об этом в следующий раз)

```
public class TestClass {
    public int a;
    private int b;

    public int GetB() {
        return b;
    }
    public void SetB(int value) {
        b = value;
    }

    private float HiddenMethod() {
        return 42.f;
    }
    public float AccessibleMethod() {
        return HiddenMethod();
    }
}
```

```
TestClass test = new TestClass();
test.a = 5;
test.b = 10;
test.SetB( value: 10);
System.out.println(test.GetB());
System.out.println(test.HiddenMethod());
System.out.println(test.AccessibleMethod());
```

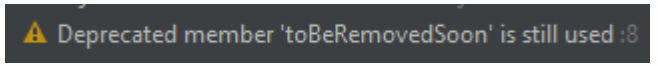
Аннотации

Аннотации – специальные метки, которые можно применять к разным сущностям языка

```java
public class Main {
    public static void main(String[] args) {
        String result = toBeRemovedSoon();
        System.out.println(result);
    }

    @Deprecated
    public static String toBeRemovedSoon() {
        return "ded";
    }
}
```

@Deprecated – аннотация, используемая, чтобы сообщить о том, что нечто устарело и может быть удалено в следующей версии



@SuppressWarnings – аннотация, используемая, чтобы подавлять некоторые предупреждения

```java
public class Main {
    public static void main(String[] args) {
        String result = iDontCare();
        System.out.println(result);
    }

    @SuppressWarnings("deprecated")
    public static String iDontCare() {
        return toBeRemovedSoon();
    }

    @Deprecated
    public static String toBeRemovedSoon() {
        return "ded";
    }
}
```

Ключевое слово static

Ключевое слово `static` может быть указано у поля или метода и делает его не привязанным к конкретному объекту класса.

```
public class CountingNumber {
    private int number;
    CountingNumber(int num) {
        number = num;
        ++created_count;
    }

    static private int created_count = 0;

    static int getCreatedCount() {
        return created_count;
    }
}
```

```
CountingNumber num = new CountingNumber(10);
System.out.println(CountingNumber.getCreatedCount());
num = new CountingNumber(15);
System.out.println(CountingNumber.getCreatedCount());
```

```
1
2
```

Статичное поле общее и одно для всех объектов класса
Статичный метод не имеет `this` и является по смыслу обыкновенной функцией

Теперь мы наконец-то понимаем класс `Main`! :)

Immutable классы

Если сделать все поля класса приватными и не изменять их ни в одном методе,
то класс получится *неизменяемым*

Объекты такого класса можно передавать в любые функции и быть уверенным,
что сам объект никак не изменится

Также, это открывает возможность кэширования некоторых объектов, например, значений BigInteger:

```
public static BigInteger valueOf(long val) {
    if (val == 0L) {
        return ZERO;
    } else if (val > 0L && val <= 16L) {
        return posConst[(int)val];
    } else {
        return val < 0L && val >= -16L ? negConst[(int)(-val)] : new BigInteger(val);
    }
}
```

Вложенные классы

Внутри класса можно объявить *вложенный класс:*

```
public class ManagedString {
    private String value;

    public String getValue() {
        return value;
    }

    public void setValue(String new_value) {
        value = new_value;
    }

    public ManagedStringView getReadonlyView() {
        return new ManagedStringView();
    }

    public class ManagedStringView {
        public String getValue() {
            return value;
        }
    }
}
```

```
public static void main(String[] args) {
    ManagedString ms = new ManagedString();
    ms.setValue("Hello");

    ManagedString.ManagedStringView view = ms.getReadonlyView();
    System.out.println(view.getValue());

    ms.setValue("world!");
    System.out.println(view.getValue());

    view.setValue("Incorrect!");
    ManagedString.ManagedStringView view2 = new ManagedString.ManagedStringView();
}
```

```
Hello
world!
```

Объект вложенного класса может быть создан только из нестатичного метода внешнего класса, потому что он запоминает ссылку на внешний объект и может использовать его поля

Вложенные классы

Но если объявить вложенный класс как static, то он станет самостоятельным классом, объект которого можно создать отовсюду, ссылок на внешний объект он захватывать уже не будет

Но ничто не мешает сделать то же самое вручную:

```
public class ManagedString {
    private String value;

    public String getValue() {
        return value;
    }

    public void setValue(String new_value) {
        value = new_value;
    }

    public ManagedStringView getReadonlyView() {
        return new ManagedStringView(this);
    }

    public static class ManagedStringView {
        ManagedString owner;

        private ManagedStringView(ManagedString owner) {
            this.owner = owner;
        }

        public String getValue() {
            return owner.value;
        }
    }
}
```

```
ManagedString ms = new ManagedString();
ms.setValue("Hello");

ManagedString.ManagedStringView view = ms.getReadonlyView();
System.out.println(view.getValue());

ms.setValue("world!");
System.out.println(view.getValue());

view.setValue("Incorrect!");
ManagedString.ManagedStringView view2 = new ManagedString.ManagedStringView(ms);
```

```
Hello
world!
```

Перечисления

Предположим, что хотим передать день недели в функцию:

```
public static void fun(int dayOfTheWeek) {
    if (dayOfTheWeek < 0 || dayOfTheWeek > 6) {
        // error
    }
    // ...
}
```

Но можно задать дни и более явно, при помощи *перечисления* (enumeration):

```
public static void main(String[] args) {
    fun(DayOfTheWeek.THURSDAY);
}

public enum DayOfTheWeek {
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
}

public static void fun(DayOfTheWeek dayOfTheWeek) {
    // no need to check
    System.out.println("The day " + dayOfTheWeek.name() + " has number " + (dayOfTheWeek.ordinal() + 1));
}
```

```
The day THURSDAY has number 4
```

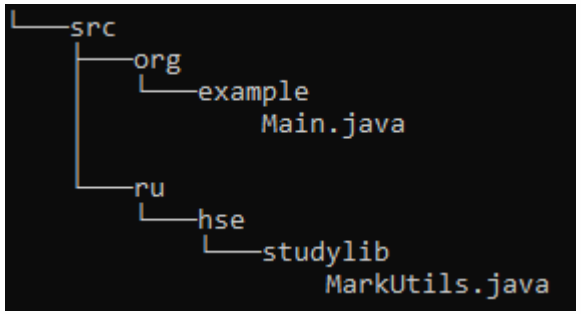
Перечисление полезно в любом месте, где нужно хранить выбор из нескольких альтернатив

Пакеты

Как вообще уживаются несколько классов внутри большого проекта ещё и с библиотеками?

Для этого несколько классов упаковываются в *пакет* – фактически общую директорию

Иерархия директорий в проекте может выглядеть примерно так:




```java
package org.example;
import ru.hse.studylib.MarkUtils;


public class Main {
    public static void main(String[] args) {
        MarkUtils.GenerateMark();
    }
}
```

```java
package ru.hse.studylib;
import java.util.concurrent.ThreadLocalRandom;


public class MarkUtils {
    public static int GenerateMark() {
        return ThreadLocalRandom.current().nextInt(3, 10);
    }
}
```

Для каждого файла с кодом нужно указать имя пакета, которому он принадлежит

Имя пакета соответствует его реальному пути на диске

Пакеты

Если не указать ключевое слово `package`, то будет считаться, что файл принадлежит корневому пакету и доступен всегда

Самый простой способ обратиться к классу из пакета – написать его полный путь:

```
public static void main(String[] args) {
    java.math.BigInteger bigint;
}
```

Если хочется писать только имя класса, то нужно помочь компилятору найти нужный пакет при помощи слова `import`

```
import java.math.BigInteger;

public class Main {
    public static void main(String[] args) {
        BigInteger bigint;
    }
}
```

Пакеты

Также, можно импортировать сразу все классы из пакета звёздочкой:

```
import java.math.*;

public class Main {
    public static void main(String[] args) {
        BigInteger bigint;
        BigDecimal bigdec;
    }
}
```

Ещё существует статический import, который импортирует *статические методы* класса:

```
import static java.lang.Math.*;
import static java.lang.System.out;

public class Main {
    public static void main(String[] args) {
        double val = asin(1./sqrt(2));
        out.println(val);
    }
}
```

Использовать его рекомендуется очень умеренно

Пакеты

В одном файле может быть объявлено не более одного public класса,
при этом имя этого класса обязано совпадать с именем файла

Но может быть объявлено неограниченное число классов без какого-либо модификатора доступа

```
package org.example;


public class Main {
    public static void main(String[] args) {
        System.out.println(Internal.ANTIPERFECT_NUMBER);
    }
}


class Internal {
    public static int ANTIPERFECT_NUMBER = -58;
}
```

Отсутствие модификатора доступа – является ещё одним, четвёртым, модификатором доступа,
который называется package-private

Пакеты

Отсутствие модификатора доступа – является ещё одним, четвёртым, модификатором доступа, который называется package-private

Сущность, объявленная без модификаторов, будет доступна всем классам, содержащимся в пакете с тем же именем

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

`private` и `protected` классов не существует

`public` классы видны из любого пакета
`package-private` классы видны только из этого же пакета

```java
package org.example;


public class Main {
    public static void main(String[] args) {
        System.out.println(Internal.ANTIPERFECT_NUMBER);
    }
}


class Internal {
    public static int ANTIPERFECT_NUMBER = -58;
}
```

Названия пакетов

В качестве начала названия пакета официально предлагается использовать доменное имя веб-сайта вашего проекта или организации в обратном порядке

Например: `com.google.somelibrary`
Или: `ru.hse.studylib`

Встроенные пакеты стандартной библиотеки Java начинаются с java и javax:

- `java.lang`
- `java.io`
- `java.math`
- `java.time`
- `java.util`
- `java.util.regex`
- `javax.xml.parsers`