



Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Объектно-ориентированное программирование

Основные понятия и принципы объектно-ориентированного программирования

Язык программирования Java

Занятие 1

© Составление,
А.В. Гаврилов, 2014
А.П. Порфирьев, 2015

Самара
2015

План лекции

- Введение в объектно-ориентированное программирование
- Общие сведения о Java
- Пакеты и имена в Java
- Описание классов в Java
- Реализация инкапсуляции



Этапы программного решения задачи

- Создание модели, определение данных для предстоящей обработки
- Разработка алгоритма: определение операций над данными и последовательности шагов по преобразованию текущего состояния модели в следующее
- Формулировка модели и алгоритма на языке программирования



Развитие подходов

Инструкции

- Сплошные
- С операторами
- Процедуры
- Модули

Данные

- Ячейки памяти
- Переменные
- Массивы
- Объединения

ООП

Объединение данных и методов их обработки



Объектно-ориентированное программирование

- ООП – это парадигма программирования, в которой базовым является понятие объекта
- Объект имеет
 - Состояние
 - Поведение
 - Уникальность
- Объект умеет
 - Получать сообщения
 - Обработать данные
 - Отправлять сообщения
- Программа в ходе работы представляет собой набор взаимодействующих объектов



История ООП

- Около 1960
Появление близких к ООП идей в языках с другими парадигмами (Lisp, ALGOL)
- 1967
Simula – первый действительно объектно-ориентированный язык, типизация статическая
- 1969-1980
SmallTalk – динамическая типизация, рефлексия
- 1983
Objective-C, C++ – добавление идей ООП в язык C
- 1985
Eiffel – полностью объектно-ориентированный язык, взаимосвязанный с процессом разработки ПО



История ООП

■ 1995

Delphi – язык Object Pascal, компонентная модель, визуальное программирование

Java – кроссплатформенный язык, очень похожий на C++

■ 2000

C# - мультипарадигменный язык, составляющая платформы .Net



Откуда берутся объекты?..

- Прототипное программирование
 - Объект можно создать из ничего
 - Объект можно создать клонированием существующих объектов
 - Примеры языков: JavaScript, Perl
- Класс-ориентированное программирование
 - Объект создаётся как экземпляр класса
 - Примеры языков: C++, Java, C#



Класс

- Класс как сущность является объединением объектов с одинаковым набором свойств и общим поведением
- Класс как элемент программы описывает структуру состояния объектов и их поведение
 - Поля класса описывают элементы состояния объекта, по сути являются переменными
 - Методы класса описывают элементы поведения объекта, по сути являются функциями
- Объект принадлежит классу, является экземпляром класса
- Программа в ходе написания представляет собой набор классов



Обычные и статические элементы класса

- Описанные в классе элементы (поля и методы) принадлежат объекту (находятся в контексте объекта)
 - У каждого объекта класса будет свой экземпляр поля
 - Вызванный у объекта метод будет работать с данными именно этого объекта
- Статические элементы класса (поля и методы) принадлежат классу (находятся в контексте класса)
 - Каждого статического поля существует ровно один экземпляр
 - Статические методы находятся в классе, но не имеют объекта, с данными которого они работают



Конструкторы и деструкторы

■ Конструктор

- Особый метод класса, создающий объект и подготавливающий его для использования
- Обычно имя совпадает с именем класса
- Фактически возвращаемый тип – сам класс, формально это часто даже не пишут
- Может иметь параметры для инициализации состояния объекта

■ Деструктор

- Особый метод класса, вызывающийся при уничтожении объекта
- Предназначен для высвобождения ресурсов (выделенная память, открытые файлы и т.д.), занятых объектом, а также для изменения связей с другими объектами



Основные принципы ООП

■ Инкапсуляция

объединение данных и методов их обработки в одну сущность, приводящее к сокрытию реализации класса и отделению его внутреннего представления от внешнего

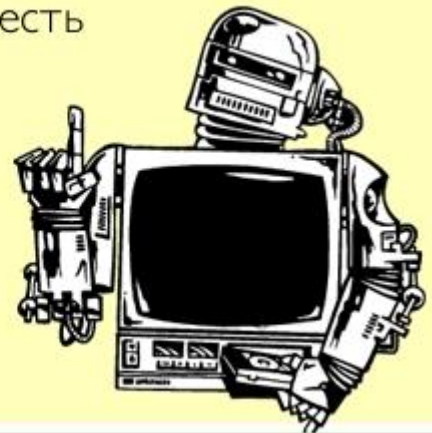
■ Наследование

отношение между классами, при котором один класс использует структуру или поведение другого (одиночное наследование) или других (множественное наследование) классов

■ Полиморфизм

способность объекта соответствовать во время выполнения двум или более возможным типам

- У меня в жизни есть
3 принципа
- Наследование,
инкапсуляция
и полиморфизм?



Atkritka.com

joyreactor.cc



Достоинства ООП

- **Упрощение разработки**
Разделение функциональности, локализация кода, инкапсуляция
- **Возможность создания расширяемых систем**
Обработка разнородных структур данных, изменение поведения на этапе выполнения, работа с наследниками
- **Легкость модернизации с сохранением совместимости**



Недостатки ООП

- Неэффективность на этапе выполнения
- Неэффективность в смысле распределения памяти
- Излишняя избыточность
- Психологическая сложность проектирования
- Техническая сложность проектирования и документирования

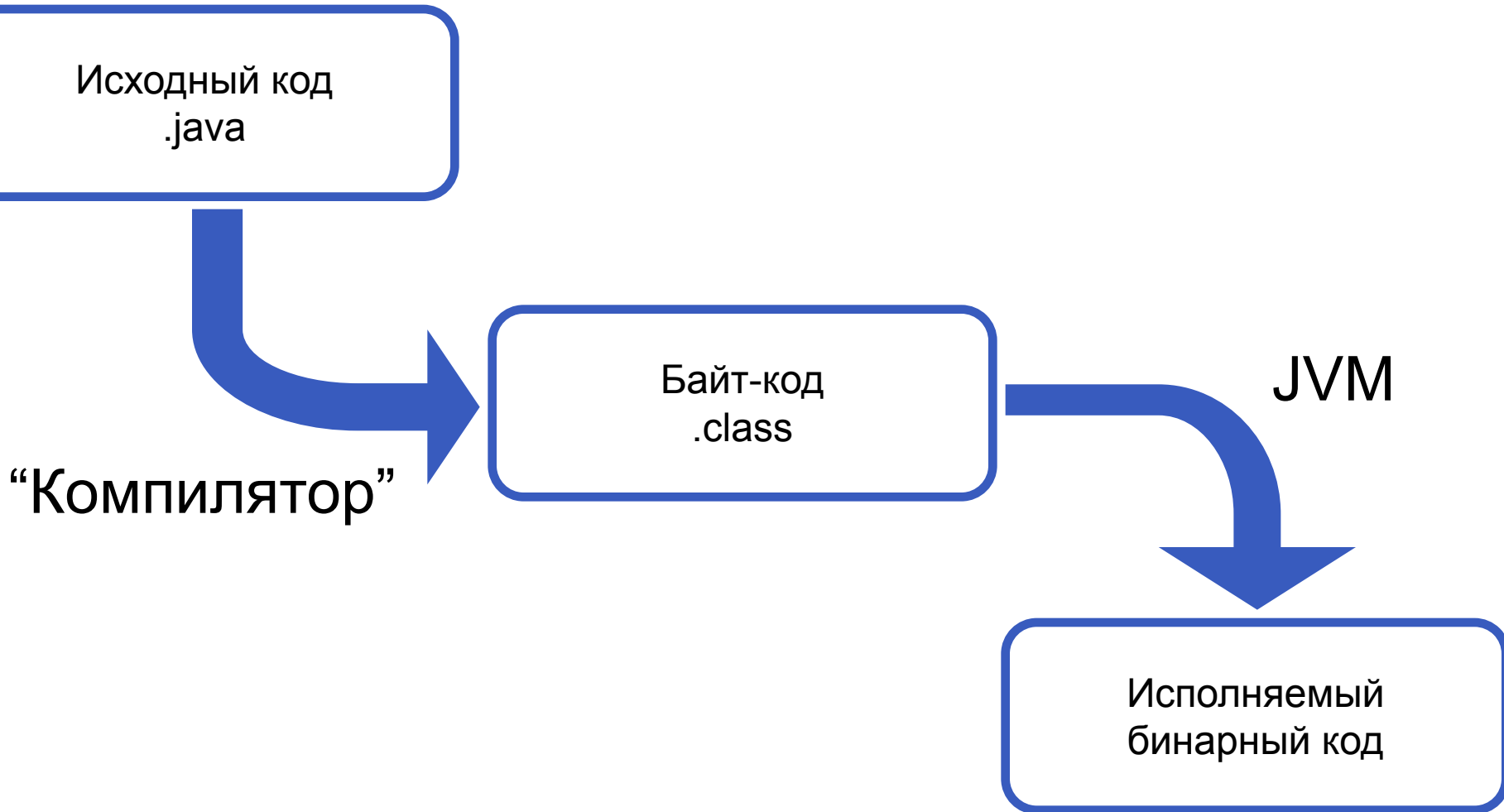


Особенности Java

- Кросс-платформенность
- Объектная ориентированность
- Привычный синтаксис (C++)
- Встроенная модель безопасности
- Ориентация на интернет-задачи, распределенные приложения
- Динамичность, легкость развития
- Легкость в освоении



Разработка и запуск



Объектный язык Java

- Все сущности в Java являются объектами, классами либо интерфейсами
- Строгая реализация инкапсуляции
- Реализовано одиночное наследование от класса и множественное от интерфейсов



Понятие о пакетах

- Способ логической группировки классов
- Комплект ПО, могущий распространяться независимо и применяться в сочетании с другими пакетами
- Членами пакетов являются:
 - классы
 - интерфейсы
 - вложенные пакеты
 - дополнительные файлы ресурсов



Функциональность пакетов

- Позволяют группировать взаимосвязанные классы и интерфейсы в единое целое
- Способствуют созданию пространств имен, позволяющих избежать конфликтов идентификаторов, относящихся к различным типам
- Обеспечивают дополнительные средства защиты элементов кода



Способы реализации и доступ к пакетам

- Пакеты могут быть реализованы:
 - в виде структуры каталогов с файлами классов
 - в виде jar-архива
- Путь к используемым пакетам указывается:
 - непосредственно при запуске JVM с помощью ключей
 - через переменную окружения CLASSPATH (по умолчанию CLASSPATH="")



Понятие имени

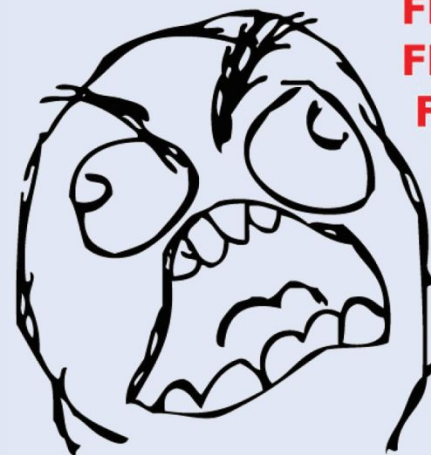
- **Имена** задаются посредством идентификаторов, указывают на компоненты программы
- **Пространства имен**
 - пакеты
 - типы
 - поля
 - методы
 - локальные переменные и параметры
 - метки
- **Имена типов** бывают
 - составные (`java.lang.Double`)
 - простые (`Double`)



Душераздирающий, но корректный код

Пример зависимости имени от контекста

```
package Reuse;  
  
class Reuse {  
    Reuse Reuse (Reuse Reuse) {  
        Reuse:  
        for(;;) {  
            if (Reuse.Reuse(Reuse) == Reuse)  
                break Reuse;  
        }  
        return Reuse;  
    }  
}
```



FFFFFFF
FFFFFFF
FFFFFFF
FFFUU
UUUU
UUUU
UUUU
UUUU
UUUU
UUUU-



Понятие модуля компиляции

- Модуль компиляции хранится в `.java` файле и является единичной порцией входных данных для компилятора
- Состоит из:
 - объявления пакета
(указывает принадлежность типов в модуле к пакету)
`package mypackage;`
 - выражений импортирования
(добавляют другие классы в область видимости)
`import java.net.Socket;`
`import java.io.*;`
 - объявлений верхнего уровня
(описания классов и интерфейсов)



Правила именования

- Пакеты
`java.lang`, `javax.swing`, `ru.ssau.tk`
- Типы
`Student`, `ArrayIndexOutOfBoundsException`
`Cloneable`, `Runnable`, `Serializable`
- Поля
`value`, `enabled`, `distanceFromShop`
- Методы
`getValue`, `setValue`, `isEnabled`, `length`, `toString`
- Поля-константы
`PI`, `SIZE_MIN`, `SIZE_MAX`, `SIZE_DEF`
- Локальные переменные



Описание класса

Класс может содержать:

- поля
- методы
- вложенные классы и интерфейсы

```
public class Body {  
    public long idNum;  
    public String name;  
    public Body orbits;  
  
    public static long nextID = 0;  
}
```



Модификаторы объявления класса

- **public**

Признак общедоступности класса (класс виден вне пакета)

- **abstract**

Признак абстрактности класса (класс не полностью реализует поведение)

- **final**

Завершенность класса (класс не допускает наследования)

- **strictfp**

Повышенные требования к операциям с плавающей точкой (результаты операций одинаковые на различных платформах)



Поля класса

- По сути являются переменными: обладают типом, именем и значением
- Объявление поля
`[модификаторы] <тип> {<имя> [= <инициализирующее выражение>] };`
- Примеры
`double sum = 2.5 + 3.7, a;`
`public double val = sum + 2 * Math.sqrt(2);`
- Если поле явно не инициализируется, ему присваивается значение по умолчанию его типа (`0`, `false` или `null`)



Модификаторы полей класса

- модификаторы доступа
- **static**
поле статично (принадлежит контексту класса, а не объекта)
- **final**
поле не может изменять свое значение после инициализации
- **transient**
поле не сериализуется (влияет только на механизмы сериализации)
- **volatile**
усиливает требования к работе с полем в многопоточных программах



Методы класса

- По сути являются функциями: обладают именем, параметрами и возвращаемым значением
- Объявление метода:
[модификаторы] <тип> <сигнатура> [throws исключения] {<тело>}
- Тело метода состоит из набора инструкций

```
class Primes {  
    static int nextPrime(int current) {  
        <Вычисление простого числа в теле метода>  
    }  
}
```



Модификаторы методов класса

- Модификаторы доступа
- **abstract**
абстрактность метода (метод объявляется, но тело при этом не описывается)
- **static**
статичность метода (метод принадлежит контексту класса, а не объекта)
- **final**
завершенность метода (метод не может быть переопределен при наследовании)



Модификаторы методов класса

- **synchronized**

синхронизированность метода (особенности вызова метода в многопоточных приложениях)

- **native**

«нативность» метода (тело метода не описывается, при вызове вызывается метод из native-библиотеки)

- **strictfp**

повышенные требования к операциям с плавающей точкой (результаты операций одинаковые на различных платформах)



Особенности методов

- Для нестатических методов вызов через ссылку на объект или в контексте объекта
`reference.method()` ;
`methodReturningReference().method()` ;
- Для статических методов вызов через имя типа, через ссылку на объект или в контексте класса
`ClassName.staticMethod()` ;
`reference.staticMethod()` ;
`staticMethodReturningReference().method()` ;
- Наличие круглых скобок при вызове **обязательно**, т.к. они являются оператором вызова метода



Особенности методов

- На время выполнения метода управление передается в тело метода
- Возвращается **одно** значение
`return someValue;`
- Аргументы передаются **по значению**, т.е. значения параметров копируются в стек:
 - для примитивных типов копируются сами значения
 - для ссылочных типов копируется значение ссылки
- Перегруженными являются методы с одинаковыми именами и различными по типу списками параметров



Что можно делать в методе?

- Можно обращаться к данным
 - Параметры метода
 - Локальные переменные
 - Поля объекта
 - Статические поля классов
- Можно выполнять операции
- Можно объявлять переменные
- Можно создавать объекты
- Можно вызывать методы объектов и классов



Создание объектов

- Объявление переменной и создание объекта – различные операции
- Используется оператор `new`, он возвращает ссылку на объект
- После оператора указывается имя конструктора и его параметры

```
Body sun;  
sun = new Body();  
sun.idNum = Body.nextID++;  
sun.name = "Sun";  
sun.orbits = null;  
  
Body earth = new Body();  
earth.idNum = Body.nextID++;  
earth.name = "Earth";  
earth.orbits = sun;
```



Конструкторы

- Память для объекта выделяет оператор `new`
- Конструкторы предназначены для формирования начального состояния объекта
- Правила написания конструктора сходны с правилами написания методов
- Имя конструктора совпадает с именем класса



Особенности конструкторов

- Для конструкторов разрешено использование только модификаторов доступа
- При написании конструктор не имеет возвращаемого типа
- Оператор возврата `return` прекращает выполнение текущего конструктора
- Конструкторы могут быть перегружены
- Конструкторы могут вызывать друг друга с помощью ключевого слова `this` в первой строке конструктора



Особенности конструкторов

- Если в классе явно не описан ни один конструктор, автоматически создается т.н. **конструктор по умолчанию**, не имеющий параметров
- Если в классе описан хотя бы один конструктор, то автоматически конструктор по умолчанию не создается
- Также конструктором по умолчанию называют конструктор, не имеющий параметров



Конструкторы

```
class Body {  
    public long idNum;  
    public String name = "No Name";  
    public Body orbits = null;  
  
    private static long nextID = 0;  
  
    Body() {  
        idNum = nextID++;  
    }  
    Body(String name, Body orbits) {  
        this();  
        this.name = name;  
        this.orbits = orbits;  
    }  
}
```



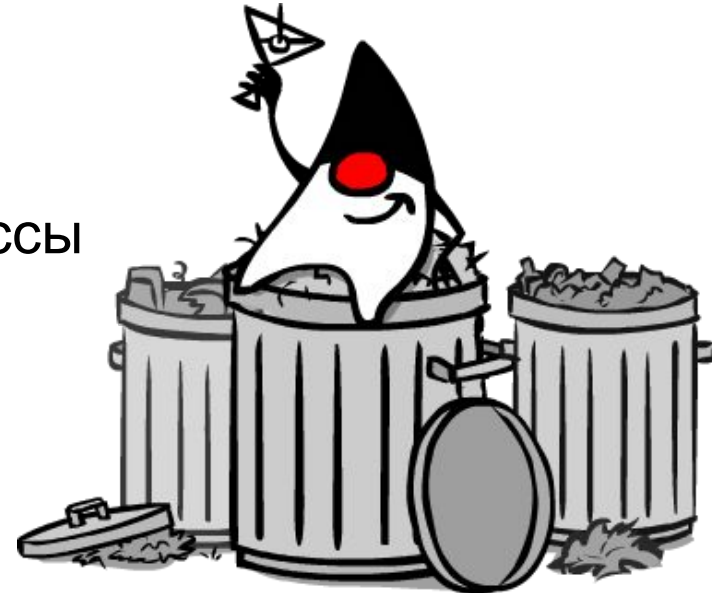
Деструкторы?

- Явное уничтожение объектов является серьёзным источником ошибок: если после вызова деструктора где-то осталась ссылка (указатель) на объект, ее использование и приведет к возникновению ошибки
- В Java деструкторов нет, вместо них применяется механизм автоматической сборки мусора
- Задачу высвобождения ресурсов обычно выполняет простой метод объекта с говорящим названием (`close()`, `dispose()` и т.д.)



Автоматическая сборка мусора

- В случае нехватки памяти для создания очередного объекта виртуальная машина находит недостижимые объекты и удаляет их
- Процесс сборки мусора можно инициировать принудительно
- Для явного удаления объекта следует утратить все ссылки на этот объект и инициировать сбор мусора
- Взаимодействие со сборщиком осуществляется через системные классы `java.lang.System` и `java.lang.Runtime`



Блоки инициализации

- Если некоторые действия по инициализации должны выполняться в любом варианте создания объекта, удобнее использовать блоки инициализации
- Тело блока инициализации заключается в фигурные скобки и располагается на одном уровне с полями и методами
- При создании объекта сначала выполняются инициализирующие выражения полей и блоки инициализации (в порядке их описания в теле класса), а потом тело конструктора



Блоки инициализации

```
class Body {
    public long idNum;
    public String name = "No Name";
    public Body orbits = null;

    private static long nextID = 0;

    {
        idNum = nextID++;
    }

    Body(String name, Body orbits) {
        this.name = name;
        this.orbits = orbits;
    }
}
```



Статическая инициализация

```
class Primes {
    static int[] knownPrimes = new int[4];

    static {
        knownPrimes[0] = 2;
        for (int i=1; i<knownPrimes.length; i++)
            knownPrimes[i] = nextPrime(i);
    }

    //nextPrime() declaration etc.
}
```

- Статический блок инициализации выполняет инициализацию контекста класса
- Вызов статического блока инициализации происходит в процессе загрузки класса в виртуальную машину



Точка входа программы



- Метод
- Статический
- Доступный
- С параметрами-аргументами
- Без возвращаемого значения

```
class Echo {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++)  
            System.out.println(args[i] + " ");  
        System.out.println();  
    }  
}
```



Пришли как-то в бар `public`, `static`,
`void` и `main`. `Void` просит у `public` в
долг на пиво, а `static` шепчет `public`:
"Ты дурак, она же ничего не возвращает!"

сегодня в 19:20

Поделиться  149 Мне нравится  724

Показать все 36 комментариев



без `main` шутка не компилируется

три часа назад | Ответить

 21
pikeburu



Модификаторы доступа

- Ключевые слова языка
- Определяют видимость (область доступа) элементов класса
- Являются средством обеспечения инкапсуляции
- В разных языках могут присутствовать в разном количестве
- В разных языках один и тот же модификатор может обозначать разные вещи



Модификаторы доступа

- **private**

Доступ только в контексте класса

- **package-private** (package, default, none)

Доступ для самого класса и классов в том же пакете

- **protected**

Доступ в пределах самого класса, классов-наследников и классов пакета

- **public**

Доступ есть всегда, когда доступен сам класс



Реализация инкапсуляции

```
class Rectangle {  
    public int width, height;  
}
```

- Значения публичных полей могут быть изменены извне объекта без его контроля
- Само по себе публичное поле – не нарушение инкапсуляции, пока...
 - С его элементом состояния не связано поведение
 - На значение не накладываются ограничения
 - Его значение не связано со значениями других полей

```
class Rectangle {  
    public int width, height;  
    public int area;  
}
```



Реализация инкапсуляции

```
class Rectangle {
    private int width, height;
    private int area;
    public int getWidth() {
        return width;
    }
    public int getHeight() {
        return height;
    }
    public void setWidth(int width) {
        this.width = width;
        area = width * height;
    }
    public void setHeight(int height) {
        this.height = height;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

- Приватные поля
- Публичные методы доступа
- Разделение внутреннего состояния и внешнего представления
- Соккрытие реализации (например, поля `area` может и не быть, а площадь может вычисляться прямо в методе `getArea()`)



Спасибо за внимание!

Дополнительные источники

- Object-oriented programming [Электронный ресурс]. – Режим доступа: http://en.wikipedia.org/wiki/Object-oriented_programming, дата доступа: 08.02.13.
- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 816 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: <http://www.oracle.com/technetwork/java/javase/overview/index.html>, дата доступа: 08.02.2013.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: <http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html>, дата доступа: 08.02.2013.

