

Классы. ООП в Java.
Конструкторы. Блоки
инициализации.

Пакеты

- Определяют пространства имен типов.
- Могут быть вложенными.

```
package com.my;  
class A {...}
```

Полное имя класса: com.my.A

Использование пакетов

Импортирование пакета

```
import com.my;
```

```
A a = new A();
```

Использование полного имени типа

```
com.my.A a = new com.my.A();
```

Подпакеты не импортируются!

Примеры пакетов

- `java.lang` базовые типы
- `java.util` структуры данных
- `java.io` потоки ввода/вывода
- `java.sql` JDBC
- `javax.swing` GUI

Виды классов по объявлению

- class
- enum

Виды классов по расположению

- Верхнего уровня
- Вложенные
 - Анонимные
 - Локальные
 - Внутренние
 - Элементы классов

Экземпляр класса

Класс - это шаблон

Экземпляр класса - реализация шаблона

`new` - оператор создания экземпляра

Что может содержать класс (элементы/члены класса)

- Конструкторы
- Блоки инициализации
- Методы
- Поля
- Вложенные классы

static ЭЛЕМЕНТЫ КЛАССА

Принадлежат классу, но не его экземплярам

```
class A {  
    static int x;  
    static void m() {...}  
    static {...}  
    static class B {...}  
}
```

Конструкторы класса

Предназначены для создания объектов.

```
class Test {  
    Test() {...}  
}
```

```
Test t = new Test();
```

Методы класса

Определяют функциональность объектов.

```
class Test {  
    void m() {...}  
}
```

```
Test t = new Test();  
t.m();
```

Поля класса

Определяют состояние объекта.

```
class Human {  
    int age = 30;  
}
```

```
Human human = new Human();  
System.out.println(human.age);
```

Блоки инициализации

Инициализируют объект.

```
class Test {  
    {...}  
}
```

Классы - элементы классов

Объект может содержать (агрегировать) другие объекты.

```
class Student {  
    class Brain {...}  
    Brain brain;  
}
```

Наследование

```
class A extends B {...}
```

Потомок - **всегда** частный случай предка.

Наследуются **все** элементы класса B.

Потомок может заменить предка в любом контексте.

Инкапсуляция

Ограничение доступа к элементам класса.

Соккрытие деталей внутренней реализации.

Цель: целостность объекта.

Полиморфизм

```
class Base { void m() {...} }  
class A extends Base { void m() {...} }  
class B extends Base { void m() {...} }
```

Потомок может переопределить
функциональность предка

```
Base base = new A(); Base base = new B();  
base.m();           base.m();
```

Уровни доступа к элементам класса

- private внутри класса
- **default** **внутри пакета**
- protected внутри пакета и потомков
- public любой внешний код

default - по умолчанию

Уровни доступа к классам

- Классы верхнего уровня:
 - public **default**
- Вложенные классы:
 - public protected **default** private
- Локальные классы:
 - **default**

Конструктор

- Создает экземпляр класса
- Имя совпадает с именем класса
- Не может быть наследован
- Не имеет типа возвращаемого результата
- Может иметь любой уровень доступа

Конструктор по умолчанию

Конструктор без параметров:

```
class A { public A(){...} }
```

Если в классе не определен ни один конструктор, то компилятор создаст и вставит в байт код конструктор по умолчанию.

Т.о. любой класс содержит конструктор

Ключевое слово this

- Ссылка на экземпляр класса, который ее использует
- Способ вызова одного конструктора из другого

Ключевое слово super

- Способ обратиться к элементу класса предка
- Способ вызова конструктора класса предка

Вызов конструктора предка из конструктор потомка

Любой конструктор всегда содержит первой строкой вызов конструктор предка.

```
public A(int x) {  
    super(9, "abcd");  
}
```

Если вызов явно не прописан, то компилятор вставит в байт код вызов **super();**

Перегрузка методов

```
class A {  
    void m() {...}  
    void m(int x) {...}  
}
```

Конструкторы класса всегда перегружены.

Перекрытие методов

Позволяет реализовать полиморфизм

```
class A {  
    void m() {...}  
}  
class B extends A {  
    void m() {...}  
}
```

Соккрытие статических МЕТОДОВ

```
class A {  
    static void m() {...}  
}  
class B extends A {  
    static void m() {...}  
}
```

Полиморфизма нет.

Ограничения при перекрытии

- Нельзя сужать уровень доступа;
- Нельзя расширять множество выбрасываемых проверяемых исключений;
- Тип возвращаемого результата:
 - для примитивных типов и `void`: такой же
 - для ссылочных должен быть **автоматически приводим** к типу возвращаемого результата метода предка

Значения полей по умолчанию

- примитивные типы числовые ==> 0
- boolean ==> false
- ссылочные ==> null

Инициализация полей

- При объявлении
- В конструкторе
- В блоках инициализации
- В методах

Ключевое слово final

Четыре контекста:

- класс - нельзя наследовать
- метод - нельзя перекрыть
- поле - константа
- локальная переменная - константа

final поля

Константы. Должны быть определены одним из следующих образом:

- 1) при объявлении
- 2) в конструкторе
- 3) в блоке инициализации

Если константа статическая, то пишут в верхнем регистре через подчеркивание:

```
static final int SOME_CONST = 2;
```


Локальные константы

```
void m() {  
    final int x;  
    final String s = "ABC";  
}
```

Абстрактные классы

Класс объявленный со спецификатором
abstract

Может содержать абстрактные методы
(методы без реализации).

```
abstract class A {  
    abstract void m();  
}
```

Свойства абстрактного класса

- Нельзя создать экземпляр, но можно объявить переменную данного типа
- Может иметь конструкторы
- Может иметь не абстрактные методы
- Может не содержать абстрактных методов

Предназначение абстрактных классов

Определить частичную функциональность, оставив часть методов не реализованными.

При наследовании абстрактного класса, класс потомок наполняет функциональностью нереализованные методы.

Процесс создания объекта.

A extends B extends C

При создании объекта: `new A()`

- 1) выполняются **статические блоки инициализации** C, B, A (если эти классы еще не загружены в JVM);
- 2) для классов C, B, A последовательно выполняются:
 - а) **блок инициализации**
 - б) **конструктор**

Практическое задание

См. далее, на следующих слайдах.

Это задание "для себя", т.е. не обязательно его делать, **НО ЖЕЛАТЕЛЬНО!**

Первое из 4-х заданий, кот. нужно сделать, будет дано позже через багтреккер.

Если будут какие-то вопросы (конкретные и лаконичные!), могу ответить вконтакте:
<http://vk.com/id32721652>

1. Класс "Окружность".

Класс должен иметь следующие поля:

- 1) x , y - координаты центра окружности;
- 2) `radius` - радиус окружности.

Класс должен иметь следующие методы:

- 1) передвинуть окружность на dx и dy ;
- 2) проверить попадание заданной точки внутрь данной окружности;
- 3) проверить попадание другой окружности внутрь данной;
- 4) вывести на экран параметры окружности.

2. Класс "Вектор" для хранения ссылок на объекты.

Класс должен иметь следующие поля:

- 1) массив ссылок, который может расти;
- 2) количество ссылок в массиве.

Класс должен иметь следующие методы:

- 1) очистить весь массив;
- 2) добавить ссылку в массив;
- 3) Получить j -й элемент;
- 4) Удалить j -й элемент;
- 5) вывести значения массива на экран.

3. Класс "Матрица".

Класс должен иметь следующие поля:

- 1) двумерный массив вещественных чисел;
- 2) количество строк и столбцов в матрице.

Класс должен иметь следующие методы:

- 1) сложение с другой матрицей;
- 2) умножение на число;
- 3) умножение на другую матрицу;
- 4) транспонирование;
- 5) вывод на печать.