Мультимедийный курс

Программирование на Java

Лекция 3.1

ОБЪЕКТНАЯ МОДЕЛЬ JAVA (ч.1)

- Классы и объекты
- Поля и методы

Разработал:
доцент каф. ЭВМ Борисенко В.П

Любая программа на Java – это **класс или набор классов**

Класс

- это **шаблон**, который определяет состав и поведение объекта, созданного с помощью этого класса
- с другой стороны - это программный **тип**: **п**еременные *типа* "класс" – это *ссылки* на **объекты (экземпляры)** класса

Общая форма объявления класса:

[<спецификатор доступа>][<модификатор>]
class <Имя_класса> {

<Объявление и инициализация полей >

<Методы класса>

}

Спецификаторы доступа класса

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

- default - по умолчанию

Модификаторы классов

- **final** - *классы не могут иметь подклассов*, например, классы используемые в качестве некоторого важного для разрабатываемой программной системы стандарта

- **abstract** – класс, в котором *хотя бы один метод не определен полностью*

Основные элементы

- Переменные, определенные в классе, называются **полями (переменными экземпляра)**
- Каждый объект (экземпляр класса) содержит свою собственную **копию** полей
- **Члены класса** – это поля и методы, определенные внутри класса

Пример объявления класса:

```
class Point extends Object {
    public double x;
    public double y;
}
```

Этот класс содержит только 2 поля x и y .

Что может содержать класс
(элементы/ члены класса)

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

Экземпляр класса

Если класс - это шаблон, то

экземпляр класса (объект)
реализация шаблона

new *- оператор создания экземпляра*

Создание и размещение объекта в динамической памяти *(куче)*

- Оператор **new**

```
class Rectangle {
  int x1, y1, x2, y2;
}
public class Try {
  public static void main(String[ ] args) {
        // создание объекта класса
   Rectangle r = new Rectangle();
   r.x1 = 10; r.y1 = 10; r.x2 = 100; r.y2 = 100;
  }
}
```

Специальная переменная **this** – ***предопределена*** в каждом классе

- Является стандартной ссылкой на объект, из которого вызывается метод
- Используется

а) когда реально необходима:

- в методе объекта выполняется обращение к члену того же объекта
- объект обращается к методу другого класса и передает ссылку на себя в качестве параметра

б) для улучшения читабельности кода

Пример использования переменной this

```
// Разрешение конфликтов пространства имен

Box(double width, double height, double depth) {

  this.width = width;
  this.height = height;
  this.depth = depth;
}
```

Конструктор

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

Конструктор – это особый метод, который автоматически вызывается при создании нового объекта

- **Вызывается** после выполнения явной или неявной инициализации полей
- Конструкторов в классе может быть **несколько**. Они должны отличаться друг от друга списком параметров

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**

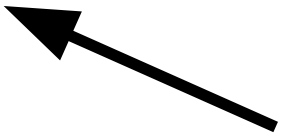
Т.о. любой класс **содержит конструктор**

Пример:

```
class Rectangle {
    int x1, y1, x2, y2;
// конструктор 1
    public Rectangle() {
        x1 = -1; y1 = -1; x2 = 0; y2 = 0;
    }
 // конструктор 2
    public Rectangle(int x1, int y1, int x2, int y2) {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }
}
```

- Если конструктор в классе не определен, компилятор создает **пустой конструктор** без параметров
- Из одного конструктора можно вызвать другой конструктор того же класса

```
this(<аргументы>) ;
```



Первый оператор в коде конструктора !!!

Определяют **состояние объекта.**

```
class Human {
    int  age = 30;
}
Human human = new Human();
System.out.println(human.age);
```

Объявление полей :

[<спецификатор_доступа>] [<модификатор>]
<тип-данных> <имя-переменной>
/<список-переменных>;

Модификатор **static** – поле существует только в классе и не принадлежит объекту

Модификатор **final** – запрещает изменение значения поля, т.е. определяет **константу**

```
int x1, x2;
static int x1;          // поле класса
final double PI = 3.1415926535898   // константа
                                    класса
```

Методы класса или соответствующего объекта - это функции, из которых доступны поля

Определяют **функциональность объектов.**

```
class Test {
    void m() {...}
}
Test t = new Test();
t.m();
```

Общий синтаксис для объявления методов :

[<спецификатор_доступа>] [<модификатор>]
<тип_данных> <имя_метода>(<список_параметров>)
{
тело метода
}

Тип данных, возвращаемых методом - любой допустимый тип, включая типы-классы либо **void**

Класс м. включать несколько ***одноименных*** методов!

Пример:

```
class Rectangle {
    int x1, y1, x2, y2;
    public Rectangle() { …}     // код конструктора

    // Передвигает прямоугольник на dx по оси x и на dy по оси y
    public void move(int dx, int dy) {
        x1 += dx;
        x2 += dx;
        y1 += dy;
        y2 += dy;
    }
// Возвращает true, если точка (x,y) находится внутри
                                    // прямоугольника
    public boolean isInside (int x, int y) {
        return x1 < x && x < x2 && y1 < y && y < y2;
    }
}
```

Статические поля и методы

- Принадлежат только классу, а не его экземплярам.
- Объявляются с помощью модификатора **static**
- Доступ к ним осуществляется с помощью имени класса

```
class Circle {
    final static double PI = 3.14159;
    …
}
…
d = Circle.PI * r;      // доступ через имя класса
```

- У класса, все методы могут быть статическими (например, класс java.lang.System из библиотеки Java)

Статические поля и методы (продолжение)

- Статическим методам не передается ссылка **this** !!!

- Из статических методов прямое обращение – только к статическим полям и методам своего класса!!!

Статические поля и методы (продолжение)

- Сложную инициализацию статических полей можно выполнить с помощью **статических блоков инициализации**

```
static {<операторы>};
```

Статические блоки инициализации **выполняются** сразу после загрузки класса и выполнения операторов явной инициализации полей

Пример:

```
class  UseStatic {
    static int a=3;
    static int b;

    static void meth (int  x) {
        System.out.println("x =  "+x);
       System.out.println("a =  "+a);
       System.out.println("b =  "+b);
     }
     static {
        System.out.println("Статический блок инициализирован");
              b=a*4;
     }
     public static void main (strings args[ ]) {
          meth (42);
      }
  }
```

Блоки инициализации

Инициализируют объект:

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

Правила создания файлов классов

- ✔ Исходные тексты классов должны сохраняться в файлах с расширением .java
- ✔ В файле может быть объявлено несколько классов,
- ✔ Один из классов должен иметь имя, совпадающее с именем файла и,возможно, спецификатор **public**
- ✔ Другие классы файла не могут иметь спецификатор **public**

Метод main()

- Должен присутствовать в каждом автономном приложении
- Имеет спецификатор доступа **public** и модификатор **static**
- Должен иметь формальный параметр в виде строкового массива (типа String).
- Часто используется для автономного тестирования классов

Применение метода main() для отладки

```
class Employee {
  public Employee (String n, double s,  int year,
  int month, int day)
  {
     name = n;
     salary = s;
     GregorianCalendar  calendar =
                new GregorianCalendar(year, month - 1, day);
    hireDay = calendar.getTime();
  }
. . .
    public static void main(String[ ] args) // отладочный модуль
    {
       Employee e = new  Employee ("Romeo", 50000);
       e.raiseSalary(10);
       System.out.println(e.getName() + " " + e.getSalary()) ;
     }
  }
```