Лекция 3

Описание классов и объектов.
Статические методы.
Конструкторы.
Классы-обертки встроенных типов.

Структура файла программы

```
//Объявление пакета
[package имя_пакета;]

//Импорт пакетов или классов
[import имя_пакета.имя_подпакета.....*;]
[import имя_пакета.имя_подпакета.....Имя_класса;]

//Описание классов
public class Имя_класса {}
[class Имя_класса {}]*
```

Пакеты в Java

Автоимпортируемый пакет: java.lang

java.util – коллекции, служебные классы

java.io – классы потоков ввода и вывода

Стандартные пакеты располагаются в каталоге JDK. Месторасположение JDK должно быть указано с помощью системной переменной JAVA_HOME.

Также в переменной PATH должен быть прописан путь к каталогу bin установки JDK

Расположение остальных пакетов может быть задано с помощью системной переменной CLASSPATH

Если в файле имя пакета не указано, то используется пакет по умолчанию. Все члены такого пакета имеют пакетную область видимости.

Описание класса

[видимость] [модификатор] **class** имя_класса
 [**extends** имя_базового_класса]
 [**implements** список_имен_интерфейсов]
{
 //Описание атрибутов и методов
}

Описание классов

Видимость классов:

- не указана – пакетная видимость
- **public** – общедоступная видимость

ОГРАНИЧЕНИЕ: не более одного класса с видимостью **public** в одном файле. Имя файла должно совпадать с именем этого класса.

Модификаторы:

- **abstract** – абстрактный класс;
- **final** – класс не может быть суперклассом (нельзя наследовать)

Объявление объектов

Описание объекта в общем виде:
ИмяКласса имяОбъекта;

Создание объекта:
имяОбъекта = new ИмяКласса(параметры);

В языке Java при работе с объектными типами используются ссылки. Поэтому при присвоении объекта объекту происходит присвоение ссылки, а не создание копии объекта.

```
ClassA obj1 = new ClassA();
//obj2 ссылается на тот же объект, что и obj1
ClassA obj2 = obj1;
```

Описание атрибутов

[видимость] [модификаторы] тип имя [= значение]

Видимости:
private - закрытый,
protected - защищенный,
public - общедоступный,
не указана - пакетная.

Модификаторы:
final – константное (финальное) значение,
static – статический атрибут,
volatile – запрещена оптимизация.

Видимость

	private	-	protected	public
Один и тот же класс	Да	Да	Да	Да
Потомок класса этого же пакета	Нет	Да	Да	Да
Класс этого же пакета, не являющийся потомком	Нет	Да	Нет	Да
Потомок класса другого пакета	Нет	Нет	Да	Да
Класс другого пакета, не являющийся потомком класса этого пакета	Нет	Нет	Нет	Да

Пример

```
class TestClass{
  int a;  //Значение по умолчанию – 0
  protected final int b = a + 5;
  public static char ch  = 'X';
}
```

Обращение к атрибутам:

объекта: имя_объекта.имя_атрибута

класса: имя_класса.имя_атрибута

Описание методов

Описание методов:

[видимость] [модификаторы] тип имя(параметры)
[throws список_исключений]
{
  //Тело метода
}

Описание параметров:

тип имя [, тип имя […]]

Модификаторы:

- final – данный метод нельзя переопределять в потомке
- static – статический метод класса
- abstract – абстрактный метод (класс должен быть абстрактным)

Если метод не возвращает значений, то указывается ключевое слово **void**

Передача параметров

В языке Java передача параметров осуществляется по следующим правилам:

- передача параметров простейших типов (byte, short, int, long, char, float, double, boolean) осуществляется по значению.
- передача параметров всех ссылочных типов осуществляется по ссылке.

Пример:

```
class A{ int val;}

public class MyApp{
  public static void main(String argvp[]){
    A a = new A();
    a.val = 10;
    System.out.println("Val = " + a.val);    //10
    Inc(a);
    System.out.println("Val = " + a.val);    //11
  }
  private static void Inc(A a){
    a.val++;
  }
}
```

Возвращаемые значения

В языке Java метод может вернуть значение системного (встроенного) или ссылочного типа (включая массивы).

Пример:

```
class A { /* … */}
class B{
    public static A genAClass(){
      A val = new A(…);
      return val;
    }
    public int[] genArray(){
      int[] arr = new int[10];
      for(int i=0;i<10;i++) arr[i] = i;
      return arr;
    }
}
```

Перегрузка методов

В языке Java можно перегружать методы.
Перегруженные методы имеют одно и тоже имя, но различные типы, количество или порядок параметры.

Пример:

```
class MyClass{
  public int Meth(int v)              { /* … */ }
  public int Meth(String s)           { /* … */ }
  public int Meth(float b)            { /* … */ }
  public int Meth(int v, String s)    { /* … */ }
  public int Meth(String s, int v)    { /* … */ }
}
```

Неопределенное число параметров

В языке Java начиная с JDK5 стало возможным передача методам неопределенного числа параметров одного типа.

Описание такого метода имеет вид:

тип имяМетода (фикс_параметры, тип … имя)

```
{
  /*  */
}
```

Работа с такими параметрами осуществляется как с обычными массивами.

Пример метода с неопределенным числом параметров

```
//Описание метода
public void meth(int … vals)
{
  for(int val: vals){
    System.out.println(val);
  }
}

//Вызов метода
a.meth(1,2,3);
a.meth(1,2,3,4);
a.meth();
```

Проблемы varargs методов

Параметры с неопределенным количеством должен быть последним в описании метода:

void meth(**int** … vals, **boolean** flag) //Ошибка

Нельзя объявлять несколько параметров с неопределенным количеством:

void meth(**int** … vals, **float** … values) //Ошибка

Необходимо избегать возможных конфликтов при перегрузке методов:

void meth(**int** … vals) **{**/*…*/**}**

void meth(**boolean** … vals) **{**/*…*/**}**

При вызове:

a.meth(1,2,3);

a.meth(true, false);

a.meth(); //Ошибка! Но можно перегрузить метод без параметров

Проблемы varargs методов

Еще один пример неправильной перегрузки методов с неопределенным количеством параметров:

```
void meth(int i, int … v) {/*…*/}
void meth(int … v) {/*…*/}
```

Проблема при вызове

```
a.meth(1);
```

Конструкторы

В языке Java при объявлении класса у которого не определен ни один конструктор автоматически создается конструктор по умолчанию.

Пользователь может объявлять произвольное количество перегруженных конструкторов.

Имя конструктора должно совпадать с именем класса.

Конструктор не может возвращать какое-либо значение.

Конструкторы

Описание конструктора:

[видимость] ИмяКласса(параметры)

{

  //Тело конструктора

}

Конструкторы могут быть без параметров. В этом случае указываются пустые скобки.

Ключевое слово this

Ключевое слово this в языке Java используется двояко:

- для обращения к элементам класса внутри его методов (с целью разрешения конфликта имен с передаваемыми параметрами).
- для вызова перегруженного конструктора.

Вызов перегруженного конструктора должен осуществляться в самом начале.

Пример ключевого слова this

```
class A{
  private int value;
  private int index;
  A(int value){
    this.value = value;
  }
  A(int value, int index){
     this(value);
     this.index = index;
  }
  public String toString(){
    return index + ": " + value;
  }
}
```

Статический конструктор

Для инициализации статических атрибутов класса может использоваться статический конструктор.

Пример:

```
class A{
  public static int a;
  static{
    a = 10;
  }
}
```

Уничтожение объектов

В языке Java для уничтожение объектов осуществляется автоматически (garbage collector). Поэтому в языке Java отсутствуют деструкторы.

Некоторое подобие деструктора класса может быть создано путем описания метода finalize() в виде:

protected void finalize() **{** /*…*/ **}**

Этот метод нельзя вызывать явно. Он вызывается автоматически самой виртуальной машиной Java перед самым уничтожением объекта.

Вложенные и внутренние классы

В языке Java допускается определять класс внутри другого класса. Область определения вложенного класса ограничена областью определения внешнего класса.

Таким образом, если класс B определен внутри класса A, класс B не может существовать независимо от класса A.

Вложенный класс имеет доступ к членам, в том числе приватным, класса, в который он вложен. Внешний класс не имеет доступа к членам вложенного класса.

Вложенный класс, который объявлен непосредственно внутри области определения своего внешнего класса, является его членом. Можно также объявлять вложенные классы, являющиеся локальными для блока.

Вложенные и внутренние классы

Существует два типа вложенных класса: статические и нестатические.

Статический вложенный класс объявляется с модификатором **static**. Так как он является статическим, то он должен обращаться к своему внешнему классу посредством объекта. Используются редко.

Внутренний (нестатический) вложенный класс имеет доступ ко всем переменным и методам своего внешнего класса и может непосредственно ссылаться на них также, как это делают остальные нестатические члены внешнего класса.

Пример внутреннего класса

```
class Outer{
    int outer_x = 10;
    void test(){
        Inner inner = new Inner();
        inner.display();
    }
    private class Inner{
        int inner_x = 20;
        void display(){
            System.out.println(outer_x);
            System.out.println(inner_x);
        }
    }
}

class TestClass{
    public static void main(String argv[]){
        Outer outer = new Outer();
        outer.test();
    }
}
```

Пример внутреннего класса внутри блока

```
class Outer{
    int outer_x = 10;
    void test(){
        for(int i=0;i<5;i++){
            class Inner{
                void display(){
                    System.out.println(outer_x);
                 }
            }
            Inner inner = new Inner();
            inner.display();
        }
    }
}

class TestClass{
    public static void main(String argv[]){
        Outer outer = new Outer();
        outer.test();
    }
}
```

Классы-обертки

В языке Java в пакете java.lang содержится набор классов-оберток встроенных скалярных типов языка Java.

Символьный класс: Character
Конструктор: Character(char ch)
Получение значения: char charValue()

Логический класс: Boolean
Конструктор: Boolean(boolean val)
Boolean(String val)
Получение значения:
boolean booleanValue()

Классы-обертки

Классы Byte, Short, Integer, Float, Double наследуют абстрактный класс Number, который содержит следующие методы:

byte byteValue()

short shortValue()

int intValue()

long longValue()

float floatValue()

double doubleValue()

Для каждого класса определены два конструктора: по значению соответствующего типа, по строке.

А также методы parseByte(String) … parseDouble(String), которые осуществляют преобразование строки в определенный тип.

Автоупаковка / авторaспаковка

Старый подход:

```
Integer a = new Integer(10);
int b = 2*a.intValue();
Integer c = new Integer(b);
```

Современный подход (начиная с JDK5):

```
Integer a = 10, c;
c = 2*a;
```