

Лекции 3-4. Технология Java. Классы и объекты.

Березовская Юлия
Владимировна
myumla.myu@gmail.com

ОБЗОР ТЕХНОЛОГИИ JAVA

Язык Java

Язык программирования Java – язык программирования высокого уровня, обладающий характеристиками:

- объектно-ориентированный;
- простой;
- распределенный;
- многопоточный;
- динамичный;
- архитектурно (аппаратно) независимый;
- переносимый;
- высокопроизводительный;
- надежный (устойчивый к сбоям),
- безопасный.

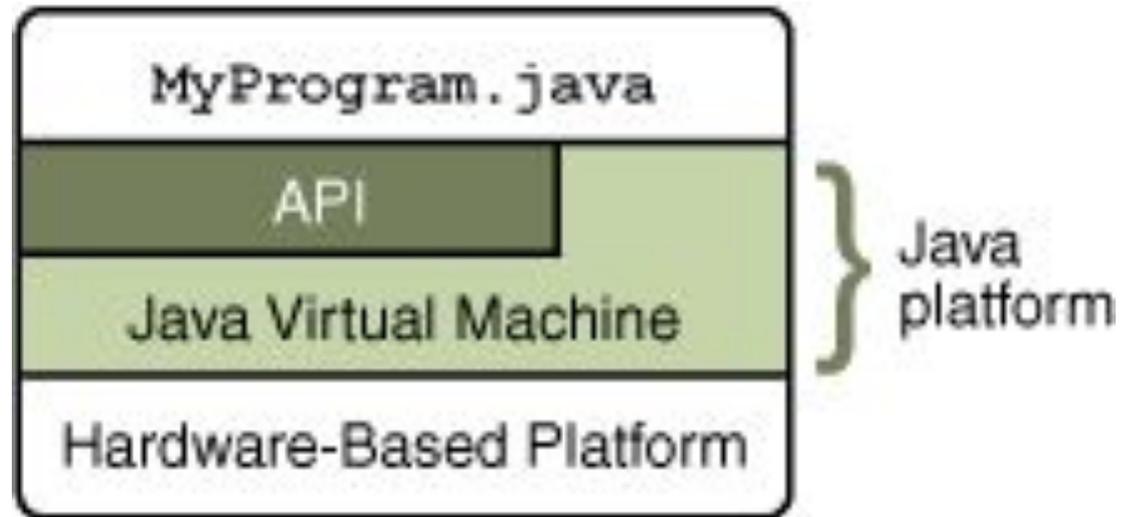
Подробнее: <http://java.sun.com/docs/white/langenv/>

Платформа Java

Платформа – окружение из аппаратного или программного обеспечения, в котором выполняется программа. В большинстве случаев платформа рассматривается как объединение ОС и аппаратного обеспечения (железа), на котором функционирует ОС.

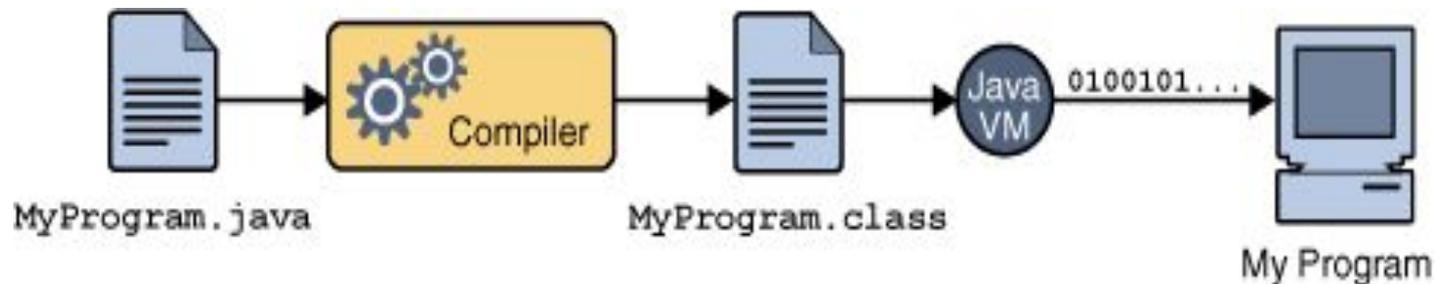
Java-платформа – исключительно программное обеспечение, функционирующее над платформой, основанной на аппаратном обеспечении, позволяющее исполнять Java-программы.

Платформа Java



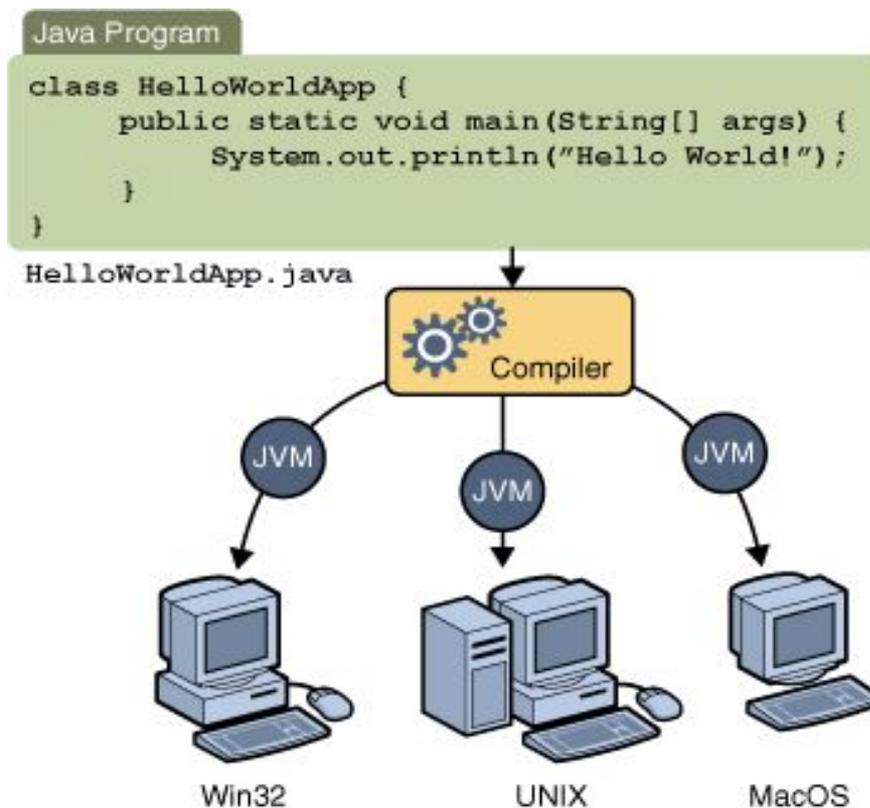
<http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html>

Создание и выполнение Java-программ:



<http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html>

Переносимость Java-программ



<http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html>

КЛАССЫ И ОБЪЕКТЫ

Классы основные строительные элементы Java-программы.

// заголовок класса

```
class MyClass
```

```
{
```

//тело класса:

// поля, конструкторы и методы

```
}
```

Заголовок класса

1. Модификатор управления доступом
2. Ключевое слово `class`
3. Название класса с большой буквы
4. Имя класса предка, предваренное словом `extends`
5. После слова `implements` через запятую имена интерфейсов, реализуемых классом

Тело класса

Члены класса:

- Поля – переменные и константы, характеризующие объект;
- Методы – процедуры, описывающие поведение объекта;
- Вложенные классы;
- Вложенные интерфейсы.

Модификаторы управления доступом

- **public** определяет, что следующие за ним определения доступны всем;
- **private** означает, что следующие за ним определения может использовать только создатель типа, внутри функций членов этого типа;
- **protected** по действию схож с **private**, за одним исключением: унаследованные классы имеют доступ к членам, помеченным **protected**, хотя и не имеют доступа к **private**-членам.

Модификаторы управления доступом

- Уровень класса - модификатор `public` или никакого;
- Уровень члена класса – модификаторы: `public`, `private`, `protected` или никакого.

Модификатор	Класс	Пакет	Класс-потомок	Все классы
<code>public</code>	+	+	+	+
<code>protected</code>	+	+	+	-
нет	+	+	-	-
<code>private</code>	+	-	-	-

Создание экземпляра класса

Этапы:

- Объявление объектов
`MyClass object1, object2;`
- Выделение памяти под объекты
`object1 = new MyClass();`
- Инициализация объектов
`MyClass()` – конструктор

Особенности конструктора:

- конструктор имеется в любом классе;
- конструктор выполняется автоматически при создании экземпляра класса;
- конструктор не возвращает никакого значения;
- конструктор нельзя наследовать и переопределить в подклассе;
- конструктор может содержать:
 - вызов конструктора суперкласса (`super`);
 - вызов другого конструктора того же класса (`this`).

Класс DynArray

```
public class DynArray {
```

```
// класс имеет три поля
```

```
int size; // текущий размер массива
```

```
int maxSize; // размер отведенной  
памяти
```

```
int[] array; // сам массив
```

Класс DynArray

```
// аргумент указывает, сколько памяти  
    надо
```

```
// отвести под его элементы
```

```
public DynArray(int sz){  
    this(sz, sz, null);  
}
```

Класс DynArray

```
// аргументы указывают, сколько памяти  
// используется под элементы и сколько  
// отведено всего
```

```
public DynArray(int sz, int maxSz){  
    this(sz, maxSz, null);  
}
```

Класс DynArray

```
public DynArray(int sz, int maxSz, int[] iniArray) {  
    size = sz;  
    maxSize = (maxSz < sz ? sz : maxSz);  
    array = new int[maxSize];  
    if (iniArray != null){  
        for (int i=0; i < size && i < iniArray.length; i++)  
            array[i] = iniArray[i];  
    }  
}
```

Класс DynArray

// операция выборки элемента

```
public int elementAt(int i){  
    return array[i];  
}
```

Класс DynArray

```
// изменение текущего размера массива,  
// аргумент delta задает размер изменения
```

```
public void resize(int delta){  
    if (delta > 0) enlarge(delta);  
    else if (delta < 0) shrink(-delta);  
}
```

Класс DynArray

// операция расширения массива

```
void enlarge(int delta){  
    if ((size += delta) > maxSize) {  
        maxSize = size;  
        int[] newArray = new int[maxSize];  
        for (int i=0; i < size - delta; i++)  
            newArray[i] = array[i];  
        array = newArray;  
    }  
}
```

Класс DynArray

// операция уменьшения массива

```
void shrink(int delta){  
    size = (delta > size ? 0 : size - delta);  
}
```

Класс DynArray

```
// добавление одного нового элемента  
// (с возможным расширением массива)
```

```
void add(int e){  
    resize(1);  
    array[size-1]=e;  
}
```

Иерархия классов в Java

- Вершина иерархии классов Java - класс Object;
- Все классы наследники класса Object, т. е. ссылочная переменная типа Object может обращаться к объекту любого класса;
- Запрещено множественное наследование.

[Посмотреть:](#)

Класс Object

Методы:

- equals() сравнивает данный объект на равенство с объектом, заданным в аргументе, возвращает логическое значение.
- toString() пытается содержимое объекта преобразовать в строку символов и возвращает объект класса String.

Задание:

1. Внести изменения в класс `DynArray` так, чтобы элементы массива могли быть экземплярами произвольного класса.
(при определении массива используйте тип данных `Object`)
2. Переопределите методы `equals()` и `toString()` (класса `DynArray`).
3. Напишите класс `DynArrayTest`, тестирующий работу класса `DynArray`.

Интерфейсы в Java

Интерфейс – это явно указанная спецификация набора методов, которые должны быть представлены в классе, который реализует эту спецификацию.

Интерфейс – ссылочный тип данных, подобный классу, который может содержать только константы, заголовки методов и вложенные типы (классы и интерфейсы).

Определение интерфейса

```
public interface ИМЯ extends интерфейс1,  
    интерфейс2  
{  
    ТИП ИМЯ_КОНСТАНТЫ = значение;  
    тип_результата ИМЯ_МЕТОДА  
    (параметры_метода);  
}
```

Тело интерфейса

- Заголовки методов.

не содержат фигурных скобок (не определяют реализацию);

отделяются точкой с запятой;

методы являются `public`, поэтому этот модификатор не пишется.

Тело интерфейса

- Объявление констант.

Любая переменная, объявленная в интерфейсе является `public`, `static` и `final` поэтому эти модификаторы не пишутся.

Предполагается обязательное присвоение значения константе в теле интерфейса.

Реализация интерфейсов

```
class имя_класса
```

```
[extends суперкласс]
```

```
[implements интерфейс0 [, интерфейс1...]]
```

```
{ тело класса }
```

Реализация интерфейсов

Интерфейсы можно использовать для импорта в различные классы совместно используемых констант. В том случае, когда вы реализуете в классе какой-либо интерфейс, все имена переменных этого интерфейса будут видимы в классе как константы.

Реализация интерфейсов

Если интерфейс не включает в себя методы, то любой класс, объявляемый реализацией этого интерфейса, может вообще ничего не реализовывать. Для импорта констант в пространство имен класса предпочтительнее использовать переменные с модификатором `final`.

Реализация интерфейсов

Если интерфейс включает в себя заголовки методов, то любой класс, объявляемый реализацией этого интерфейса, должен содержать реализацию всех методов, описанных в интерфейсе.

Пример. Классы и интерфейсы.

```
public class IntList {  
    //внутренний класс  
    static class ListItem {  
        int item;  
        ListItem next;  
  
        public ListItem(int i, ListItem n){  
            item=i;  
            next=n;  
        }  
    };  
};
```

Пример. Классы и интерфейсы.

//поля класса

```
int count = 0;
```

```
ListItem first = null;
```

```
ListItem last = null;
```

Пример. Классы и интерфейсы.

//создание пустого списка

```
public IntList(){
```

//создание копии уже имеющегося списка

```
public IntList(final IntList src){
```

```
    addLast(src);
```

//добавляет список src в конец списка this

```
}
```

Пример. Классы и интерфейсы.

//добавление элементов в конец списка

```
public void addLast(final IntList src) {  
    for(ListItem cur = src.first; cur != null;  
        cur = cur.next)  
        addLast(cur.item);  
}
```

Пример. Классы и интерфейсы.

//добавление элемента в конец списка

```
public void addLast(int item){  
    ListItem newItem = new ListItem(item, null);  
    if (last == null){  
        first = newItem;  
    } else {  
        last.next = newItem;  
    }  
    last = newItem;  
    count++;  
}
```

Пример. Классы и интерфейсы.

//добавление элемента в начало списка

```
public void addFirst(int item){  
    ListItem newItem = new ListItem(item,first);  
    if (first == null){  
        last = newItem;  
    }  
    first = newItem;  
    count++;  
}
```

Пример. Классы и интерфейсы.

//удаление первого элемента из списка

```
public int remove(){
```

```
    int res = first.item;
```

```
    first = first.next;
```

```
    count--;
```

```
    return res;
```

```
}
```

Пример. Классы и интерфейсы.

```
public interface Visitor {  
    void visit(int item);  
}
```

В класс IntList добавим метод-итератор:

```
public void iterator(Visitor visitor){  
    for (ListItem cur = first; cur!=null; cur = cur.next){  
        visitor.visit(cur.item);  
    }  
}
```

Интерфейс – тип данных

Если тип переменной определен как интерфейс, то объект, присвоенный этой переменной, должен быть экземпляром класса, реализующего этот интерфейс.

Продолжаем пример:

```
public class Summator implements Visitor {  
    int sum = 0;  
    String s = "";  
    public void visit(int item){  
        s+=(item+" ");  
        sum+=item;  
    }  
    public int getSum(){return sum;}  
    public String getList(){return s;}  
}
```

Пример. Классы и интерфейсы.

```
public class IntListTest {  
    public static void main(String[] args) {  
        IntList lst = new IntList();  
        for(int i=0; i<10; i++){  
            lst.addFirst(2*i);  
            lst.addLast(20-2*i);  
        }  
        System.out.println(lst.getCount());  
        Summator summator = new Summator();  
        lst.iterator(summator);  
        System.out.println(summator.getList());  
        System.out.println(summator.getSum());  
    }  
}
```

Итак:

Интерфейс определяет протокол взаимодействия двух объектов.

Объявление интерфейса содержит сигнатуры методов, но не их реализации

Объявление интерфейса может содержать определение констант.

Класс, реализующий интерфейс должен реализовывать все методы объявленные в интерфейсе.

Имя интерфейса может использоваться везде, где может использоваться тип данных.

Пакет – объединение классов и интерфейсов.

- объединение родственных классов и интерфейсов;
- упрощение поиска классов и интерфейсов, выполняющих определенные функции;
- исключение конфликта имен, каждый пакет имеет свое пространство имен;
- неограниченный доступ классов и интерфейсов, объединенных в пакет, друг к другу.

Использование пакета

Создание пакета

оператор:

```
package имя_пакета;
```

Обращение к членам пакета из другого пакета

- имя_пакета.имя_класса; (полное имя класса)
- import имя-пакета.имя_класса; (импорт класса)
- import имя_пакета.*; (импорт целого пакета)

Особенности импорта пакетов

- импорт по умолчанию (java.lang, текущий пакет, безымянный пакет);
- `import имя_пакета.*;` (импорт только классов и интерфейсов данного пакета, но не подпакетов)
- `import имя_пакета.имя_подпакета.*;`
- если два импортированных пакета имеют классы с одинаковыми именами, то необходимо использовать полное имя класса;
- импорт статических полей и методов.

Иерархия пакетов

Древовидная структура (иерархия) пакетов и подпакетов в точности отображается на структуру файловой системы.

Все файлы с расширением `class` (содержащие байт-коды), образующие пакет, хранятся в одном каталоге файловой системы.

Подпакеты собраны в подкаталоги этого каталога.

Лирика

Структура исходного файла с текстом программы на языке Java:

Необязательный оператор `package`.

Необязательные операторы `import`.

Описания классов и интерфейсов.

Правила:

В файле только один открытый `public`-класс.

Имя файла совпадает с именем открытого класса.

Лирика

Cod Conventions

- Pascal naming convention – все слова в имени начинаются с заглавной буквы, используется для именованя классов;
- Camel naming convention – все слова в имени кроме первого начинаются с заглавной буквы, используется для именованя полей и методов.
- public-класс записывается первым в файле.