
Объектно-ориентированное программирование

Особенности языка Java

Литература

- Bruce Eckel – Thinking in Java – 2000 г.
Философия Java



Языки Java и JavaScript

- Java – компилируемый язык в отличие от JavaScript
 - Java – типизированный
 - Java более сложный язык по сравнению с JavaScript
-

БЛОК static

```
static {
```

```
}
```

Создание и уничтожение объектов

- `new` – создание объекта
- `finalize()`



Массивы

- При объявлении массива не указывается количество элементов
- Для создания массива надо использовать `new` с указанием количества элементов
- Создание массива не инициализирует его значения
- У каждого объекта-массива есть поле `length`

Объявления:

```
String[] a; String a[];  
String[] b=new String[5];  
String[] c={"a", "b", "c"}  
String[] d=new String[] {"d", "e", "f"}
```

Класс `Arrays`

- `equals()`
- `fill(e1)`
- `sort()`
- `binarysearch(e1)`
- `asList()`

Строки

- Класс String
 - Невозможно изменить существующую строку, можно только создать новую
 - Методы
 - `concat(S)`
 - `append(S)`
 - `substring(Start,End)`
 - `indexOf(S)`
 - `lastIndexOf(S)`
 - `startsWith(S)`
 - `endsWith(S)`
 - `charAt(n)`
 - `replace(S1,S2)`
 - `toLowerCase()`
 - `toUpperCase()`
-
- `Integer.parseInt(S)`

Спецификаторы доступа

- никакого
 - public
 - private
 - protected
-

ПАКЕТЫ

Запрещение наследования

- `final`
 - Для полей
 - Для методов
 - Для классов

Абстрактные классы

- `abstract` для метода
 - `abstract` для всего класса
-

Интерфейсы

- interface
 - implements
-

Обработка ИСКЛЮЧИТЕЛЬНЫХ ситуаций, работа с файлами

-
- Исключительные ситуации
 - Классы File, InputStream, RandomAccessFile, FileReader, BufferedReader, BufferedWriter, FileWriter, InputStreamReader, OutputStreamWriter

try, catch

- `throw new Exception();`
 - `try {`
 `...`
} `catch(Exception e) {`
 `...`
}
 - `public void f() throws Exception { ... }`
-

Создание своих исключений

```
class SimpleException extends Exception { }
```



Работа с файловой системой.

Класс File

- Это скорее путь к файлу
 - Соответствует файлу или папке: `new File("test.txt")` `new File(".")`
 - Методы
 - `File.separator` - \ или /
 - `String[] list()` - список файлов папки
 - `boolean isDirectory()`
 - `String getPath()`
 - `String getAbsolutePath()`
 - `File getAbsoluteFile()`
 - `String getCanonicalPath()`
 - `File getCanonicalFile()`
 - `boolean exists()`
 - `boolean createNewFile()`
 - `boolean delete()`
 - `boolean renameTo(File f)`
 - `long length()`
-

Работа с двоичными файлами

- Чтение из файла. Класс `InputStream`

```
File file = new File("file.tst");
InputStream str = new FileInputStream(file);
long length = file.length();
byte[] bytes = new byte[(int)length];
int readed = str.read(bytes,0,length);
str.close();
```

- Запись в файл. Класс `RandomAccessFile`

```
try {
    File f = new File("file.tst");
    RandomAccessFile raf = new RandomAccessFile(f,"rw");
    raf.seek(f.length());
    raf.writeChars("The End");
    raf.close();
}
catch IOException e) {
    System.out.println("Ошибка при чтении или записи файла");
}
```

Работа с файлами

- Чтение из текстового файла

```
try {  
    BufferedReader in = new BufferedReader(new FileReader("file.txt"));  
    String str;  
    while( (str=in.readLine()) != null) {  
        //  
    }  
    in.close();  
} catch(IOException e) {  
    System.out.println("Ошибка при чтении");  
}
```

- Запись (дополнение) в текстовый файл

```
try {  
    BufferedWriter out = new BufferedWriter(new FileWriter("file.txt",true));  
    out.write("It's a new line");  
    in.close();  
} catch(IOException e) {  
    System.out.println("Ошибка при записи");  
}
```

Указание кодировки

- Чтение из текстового файла

```
try {
    BufferedReader in = new BufferedReader(
        new InputStreamReader(new FileInputStream("file.txt"),"windows-1251"));
    String str;
    while( (str=in.readLine()) != null) {
        //
    }
    in.close();
} catch(IOException e) {
    System.out.println("Ошибка при чтении");
}
```

- Запись в текстовый файл

```
try {
    BufferedWriter out = new BufferedWriter(
        new OutputStreamWriter(new FileOutputStream("file.txt"),"windows-1251"));
    out.write("Мы дописали этот текст");
    out.close();
} catch(IOException e) {
    System.out.println("Ошибка при записи");
}
```

Интерфейсы

- *interface* и *abstract*

- Использование *interface*

```
interface Instrument {  
    // Константа времени компиляции:  
    int i = 5; // static & final  
    // Не могут присутствовать определения методов:  
    void play(); // автоматически public  
    String what();  
}
```

```
class Wind implements Instrument {  
    public void play() { System.out.println("Wind.play()"); }  
    public String what() { return "Wind"; }  
}
```

- Интерфейсы можно наследовать

Интерфейсы

- Множественное наследование

```
interface CanSwim {
    void swim();
}
interface CanFly {
    void fly();
}
class Hero extends AnyClass
    implements CanSwim, CanFly {
    public void swim() {}
    public void fly() {}
}
```
-

Вложенные (внутренние) классы и интерфейсы

```
public class Parcel {
    class Contents {
        private int i = 11;
        public int value() { return i; }
    }
    class Destination {
        private String label;
        Destination(String whereTo) { label = whereTo; }
    }
    // Использование внутреннего класса похоже на использование обычного
    public void ship(String dest) {
        Contents c = new Contents();
        Destination d = new Destination(dest);
    }
    public static void main(String[] args) {
        Parcel p = new Parcel(); p.ship("Tanzania");
    }
}
```

Контейнеры

Сортировка

- Метод `sort()`
- Интерфейс `Comparable`
 - метод `int compareTo(Object o)`
 - вызов: `Arrays.sort(a)`
- Интерфейс `Comparator`
 - метод `int compare(Object o)`, метод `boolean equals(Object o)`
 - вызов: `Arrays.sort(a, экземпляр_класса_реал.Comparator)`
 - вызов: `Arrays.sort(a, Collections.reverseOrder())`
 - Пример:

```
public class CompType implements Comparable {  
    int i; int j;  
}
```

```
class CompTypeJ implements Comparator {  
    public int compare(Object o1, Object o2) {  
        int j1 = ((CompType)o1).j;  
        int j2 = ((CompType)o2).j;  
        return (j1 < j2 ? -1 : (j1 == j2 ? 0 : 1));  
    }  
}
```

```
} CompType[] a = new CompType[10];  
Arrays.sort(a, new CompTypeJ());
```

ДВОИЧНЫЙ ПОИСК

- `Arrays.binarySearch(Object a, Object o)`
Если элемент найден, возвращает его индекс
Иначе значение:
-индекс_первого_большего-1
-

Контейнерные классы

- List

Поддерживается порядок элементов

- Set

Элемент может присутствовать только один раз

- Map

Содержит ключи и соответствующие им значения

- Queue

Очереди

Методы контейнеров

- `boolean add(Object o)`
 - `boolean addAll(Collection b)`
 - `boolean contains(Object o)`
 - `boolean containsAll(Collection b)`
 - `void clear()`
 - `boolean isEmpty()`
 - `boolean remove(Object o)`
 - `int size()`
 - `Iterator iterator()`
 - `Object[] toArray()`
-

Методы для List

- `boolean add(int index, Object o)`
 - `Object get(int index)`
 - `Object set(int index, Object o)`
 - `int indexOf(Object o)`
 - `int lastIndexOf(Object o)`
 - `ListIterator listIterator()`
 - `ListIterator listIterator(int index)`
-

Методы для Map

- `boolean containsKey(Object key)`
 - `boolean containsValue(Object value)`
 - `Set entrySet()`
 - `Object get(Object key)`
 - `Object put(Object key, Object value)`
 - `Set keySet()`
 - `Collection values()`
-

Особенности использования контейнеров

Плюсы

- Поддержка очень многих методов
- Скорость критичных операций
- Универсальность

Минусы

- Некоторая громоздкость реализации
 - Неизвестный тип результата
-

Итераторы

Класс Iterator

- Любой контейнер имеет метод `iterator()`, возвращающий итератор

Методы класса Iterator

- `next()`
 - `hasNext()`
 - `remove()`
-

Итераторы

```
Collection c = new ArrayList();
```

```
...
```

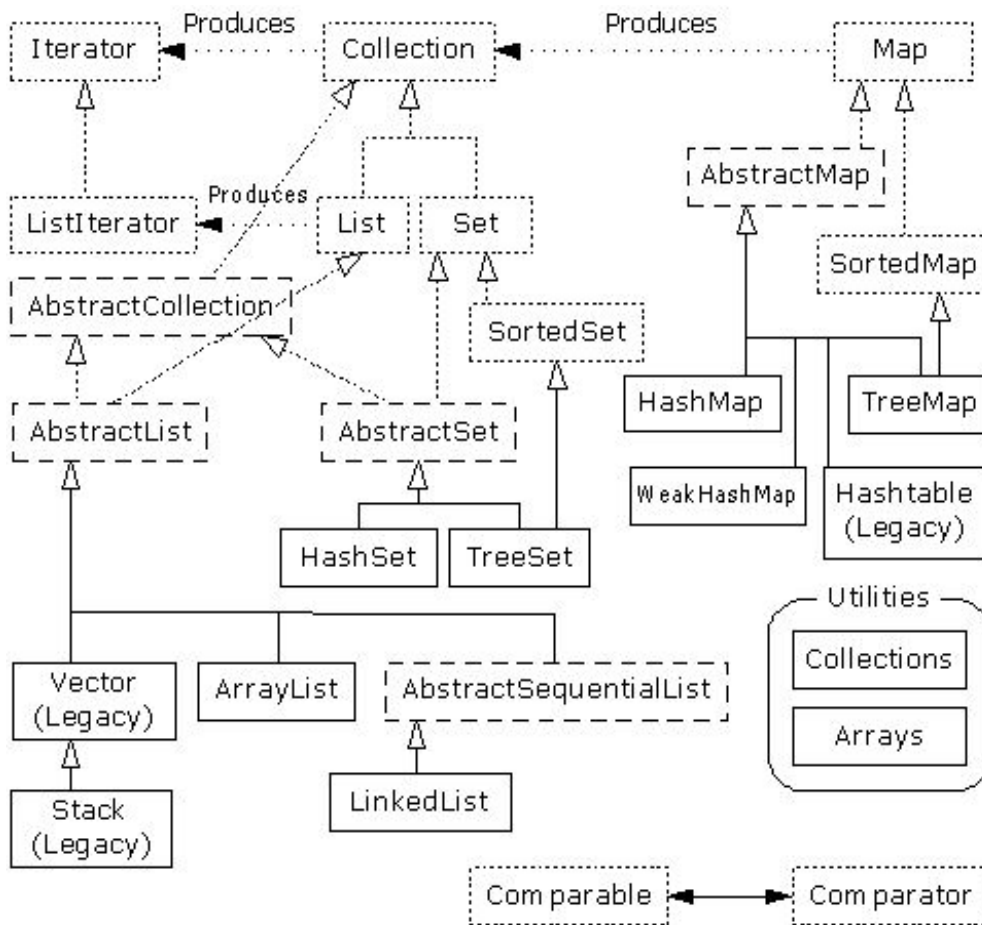
```
Iterator it = c.iterator();
```

```
while(it.hasNext()){
```

```
    System.out.print(it.next(), " ");
```

```
}
```

Иерархия контейнеров



`Iterator` - интерфейсы

`AbstractList` - абстрактные классы

`Vector` - реальные классы