



Пакет `java.util`.

Пакет java.util.

- 1. Коллекции и Карты отображений**
- 2. Сервисные классы**
- 3. Наследованные классы и интерфейсы**

Хранение данных

Массивы — наиболее простой вид контейнера: эффективность и тип.

Массив является наиболее эффективным способом для хранения объектов и доступа к ним в случайном порядке.

Массив — это простая линейная последовательность, обеспечивающая быстрый доступ к элементам, но когда создается массив объектов, его размер фиксирован и не может изменяться в течение всей продолжительности жизни этого массива объектов.

Концепции хранения объектов

Коллекция (Collection): группа индивидуальных элементов. Список (List) должен хранить элементы в определенной последовательности, а Набор (Set) не может иметь дублирующиеся элементы.

Карта (Map): группа объектных пар ключ/значение. Карта может возвращать Набор своих ключевых значений, Коллекцию своих значений или Набор своих пар. Карты, как и массивы, могут иметь несколько измерений: создается Карта, чьими значениями являются другие карты (а значениями этих Карт тоже могут быть Карты и т.д.).

Обзор коллекций

Коллекция — группа объектов.

Коллекции — классы, позволяющие хранить и производить операции над множеством объектов. Коллекции используются для хранения, получения, манипулирования данными и обеспечивают агрегацию одних объектов другими.

Алгоритмы коллекций (статические методы класса Collections) — средство манипулирования объектами в коллекциях и картах отображений

Итератор(интерфейсы Iterator, ListIterator) — способ доступа к элементам коллекции по одному.

Карта отображений (map) хранит пары ключ/значение

Компаратор (интерфейс Comparator) — способ управления сортировкой элементов в коллекциях и картах

Интерфейсы Collections Framework

Интерфейсы играют ключевую роль — все классы коллекций унаследованы от различных интерфейсов, которые определяют поведение коллекции.

Интерфейс определяет *«что делает коллекция»*, а конкретная реализация — *«как коллекция делает то, что определяет интерфейс»*.

***Рекомендация разработчику:* использовать интерфейсы там, где это возможно:**

легко заменять реализацию интерфейса (с целью повышения производительности, например);

сконцентрироваться на задаче, а не на особенностях реализации.

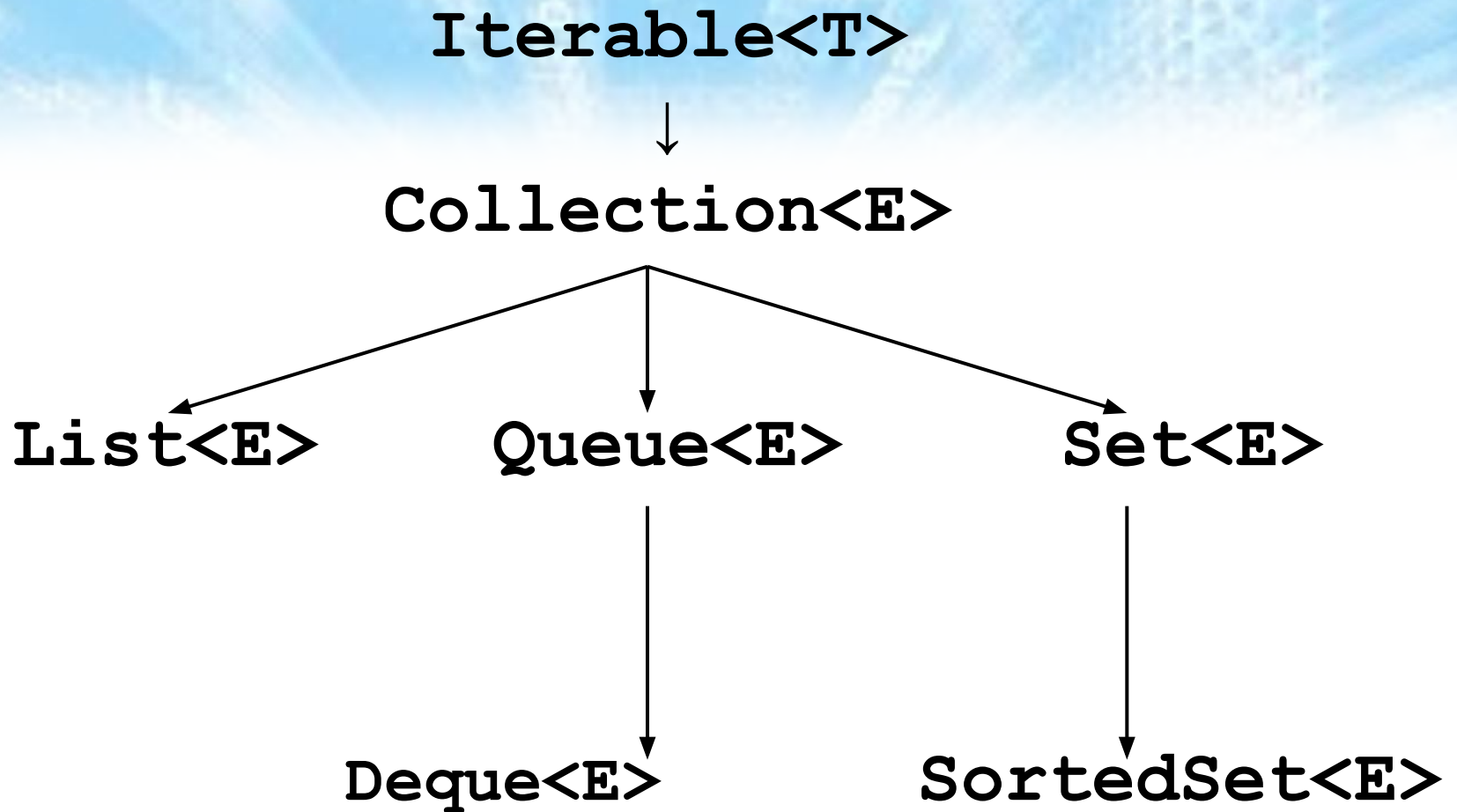
Интерфейсы Collections Framework

Интерфейс	Описание
Iterable<T>	коллекция может формировать объект-итератор → может быть использована в конструкции for (в виде for-each)
Collection<E>	<p>базовый интерфейс коллекций; определены основные методы для манипуляции с данными — вставка (add, addAll), удаление (remove, removeAll, clear), поиск (contains).</p> <p>В конкретной реализации часть методов может быть не определена → их использование вызовет исключение <code>UnsupportedOperationException</code>.</p>

Интерфейсы, наследующие Collection<E>

Интерфейс	Описание	Реализации
List<E>	Служит для описания списков; определяет поведение коллекций для хранения упорядоченного набора объектов; определена операция получения части списка.	ArrayList<E> , LinkedList<E>
Queue<E>	Реализует FIFO – буфер. Позволяет добавлять и получать объекты (в том порядке, в котором они были добавлены).	ArrayDeque<E> , LinkedList<E> .
Deque<E>	Наследует Queue<E>. Двухнаправленная очередь. Позволяет добавлять и удалять объекты с двух концов.	ArrayDeque<E> , LinkedList<E>
Set<E>	Обеспечивает уникальность хранимых объектов. Желательно переопределить метод equals хранимых объектов, т.к. он используется для определения их уникальности	HashSet<E> , LinkedHashSet<E> , TreeSet<E>
SortedList<E>	Наследует Set<E>, обеспечивая упорядочивание в порядке возрастания. Сортировка: - метод compareTo интерфейса Comparable<T> - класс, реализующий интерфейс Comparator<T>	TreeSet<E>

Иерархия интерфейсов коллекций



Интерфейс Collection

<i>Методы (некоторые)</i>	<i>Описание</i>
<code>boolean add(<E> o)</code>	Добавляет объект к вызывающей коллекции
<code>boolean addAll(Collection c)</code>	Добавляет все элементы <code>c</code> к вызывающей коллекции
<code>void clear()</code>	Удаляет все элементы
<code>boolean equals(Object o)</code>	Проверяет объекты на равенство
<code>boolean isEmpty(Object o)</code>	Возвращает истину, если коллекция пуста
<code>Iterator<T> iterator()</code>	Возвращает итератор для вызывающей коллекции
<code>boolean remove(<E> o)</code>	Удаляет объект из вызывающей коллекции
<code>boolean removeAll(Collection c)</code>	Удаляет все элементы <code>c</code> из вызывающей коллекции
<code>int size()</code>	Возвращает число элементов

Интерфейс List

<i>Методы (некоторые)</i>	<i>Описание</i>
<code>boolean add(int n, <E> o)</code>	Вставляет объект в вызывающий список в позицию <code>n</code>
<code>boolean addAll(int n, Collection c)</code>	Вставляет все элементы <code>c</code> в список с позиции <code>n</code> со сдвигом
<code><E> get(int n)</code>	Возвращает объект в позиции <code>n</code>
<code>int indexOf(Object o)</code>	Возвращает индекс первого объекта в списке
<code>int lastIndexOf(Object o)</code>	Возвращает индекс последнего объекта в списке
<code>ListIterator<E> listIterator()</code>	Возвращает итератор, установленный к началу списка
<code><E> remove(int n)</code>	Удаляет элемент в позиции <code>n</code>
<code><E> set(int n, Object o)</code>	Устанавливает объект в позицию <code>n</code>

Интерфейс SortedList

Объявляет поведение набора, отсортированного в возрастающем порядке.

Метод	Описание
<code>Comparator<E> comparator ()</code>	Возвращает компаратор вызывающего набора
<code><E> first ()</code>	Возвращает первый элемент
<code>SortedSet<E> headSet (<E> end)</code>	Возвращает объект, содержащий элементы вызывающего набора, которые меньше, чем end
<code><E> last ()</code>	Возвращает последний элемент
<code>SortedSet<E> subSet (<E> start, <E> end)</code>	Возвращает объект, содержащий элементы между start и end-1
<code>SortedSet<E> tailSet (<E> start)</code>	Возвращает объект, содержащий элементы вызывающего набора, которые больше, чем start

Итераторы

Назначение контейнера — хранение объектов:

- 1. Поместить объект;**
- 2. Извлечь объект;**

Концепция *итераторов* позволяет достичь абстракции при работе с разными типами контейнеров.

Итератор - это объект для перемещения по последовательности объектов и выборе каждого объекта (*перебор коллекции по одному*)

Доступ к коллекциям через итератор

Итератор – объект, реализующий один из интерфейсов:
`Iterator<E>`, `ListIterator<E>`.

Обеспечивает проход коллекции с получением или удалением ее элементов в одном или обоих направлениях.

Методы `Iterator`

Метод	Описание
<code>boolean hasNext()</code>	Возвращает <code>true</code> , если в коллекции присутствует следующий элемент. Иначе – <code>false</code> .
<code><E> next()</code>	Возвращает следующий элемент
<code>void remove()</code>	Удаляет текущий элемент.

ListIterator

Обеспечивает *двухнаправленный* обход коллекции и модификацию ее элементов.

Метод	Описание
<code>void add(<E> obj)</code>	Вставляет obj в список
<code>boolean hasNext()</code>	Возвращает true, если в коллекции присутствует следующий элемент. Иначе – false.
<code>boolean hasPrevious()</code>	Возвращает true, если в коллекции присутствует предыдущий элемент. Иначе – false.
<code><E> next()</code>	Возвращает следующий элемент
<code>int nextIndex()</code>	Возвращает индекс следующего элемента
<code>int previousIndex()</code>	Возвращает индекс предыдущего элемента
<code><E> previous()</code>	Возвращает предыдущий элемент
<code>void set(<E> obj)</code>	Назначает obj на текущий элемент
<code>void remove()</code>	Удаляет текущий элемент.

Использование итератора.

В каждом коллекционном классе определен метод

`iterator() / listIterator()` , который возвращает итератор к началу коллекции.

1. Получить итератор – вызов метода `iterator() / listIterator()` – старт коллекции
2. Установить цикл с обращением к методу `hasNext()`
3. Внутри цикла получать очередной элемент коллекции – `next()`

Пример итератора v.1

```
import java.util.*;

class IteratorDemo {
    public static void main(String args[]) {
        ArrayList al = new ArrayList(); //создание коллекционного
        объекта

        al.add("C"); //добавление элементов в коллекцию
        al.add("A");
        . . .
        al.add("F");

        //Получение итератора для просмотра al
        Iterator itr = al.iterator();
        while(itr.hasNext()) {
            Object element = itr.next();
            System.out.print(element + "/");
        }
        . . . } }
```

Пример итератора ListIterator

//модификация элементов коллекции на основе ListIterator

```
. . .  
ListIterator litr = al.listIterator();  
    while(litr.hasNext()) {  
        Object element = litr.next();  
        litr.set(element + "*");  
    }  
. . .
```

Пример итератора `ListIterator`

//проход коллекции в обратном направлении

. . .

```
while (litr.hasPrevious()) {  
    Object element = litr.previous();  
    System.out.print(element + ";");  
}
```

. . .

Пример for-each

```
ArrayList<String> al = new ArrayList<String>();
```

```
    al.add(new String("A"));
```

```
    al.add(new String("B"));
```

```
    al.add("F");
```

```
for (String s : al) { System.out.println(s); }
```

```
// =
```

```
    for (int i = 0; i < al.size(); i++) {
```

```
System.out.println(al.get(i)); }
```

Пример итератора v.2

```
LinkedList<Integer> ll = new  
LinkedList<Integer>();
```

```
    ll.add(new Integer(1));
```

```
    ll.add(new Integer(3));
```

```
    ll.add(new Integer(7));
```

```
for (Iterator<Integer> itr =  
ll.iterator(); itr.hasNext(); ) {  
    System.out.println(itr.next());
```

Компараторы.

Компаратор задает точное определение порядка сортировки.

Интерфейс Comparator

Методы:

```
int compare(Object obj1, Object obj2) ;  
boolean equals(Object obj) ;
```

Алгоритмы коллекций

Применяются к коллекциям и картам отображений.
Определены как *статические* методы класса `Collections`.

Некоторые методы:

```
public static <T> void copy(List<? super  
T> dest, List<? extends T> src);
```

```
public static <T> T max(Collection<? extends  
T> coll, Comparator<? super T> comp);
```

```
public static void shuffle(List<?> list);
```

```
public static <T> void sort(List<T> list,  
Comparator<? super T> c);
```

```
public static <T> Collection<T>  
synchronizedCollection(Collection<T> c);
```

Работа с картами отображений

Карта отображений – объект, хранящий ассоциации (связи) между ключами и значениями, или пары ключ/значение.

Ключи и значения являются объектами. Ключи уникальны.

Операции с картами:

Поместить в карту значение – `V put (K k, V v)` .

Получить значение – `V get (Object k)` .

Интерфейсы карт

Интерфейс	Описание
Map	Отображает уникальные ключи в значения
Map.Entry	Описывает элемент отображения. Работает со входами карт
SortedMap	Расширяет Map, поддерживает ключи в возрастающем порядке. Поддерживает входы в восходящем порядке ключей.

Интерфейс Map<K,V>

Описывает функциональность ассоциативных массивов.

Реализации:

HashMap<K, V>, LinkedHashMap<K, V>, TreeMap<K, V>, WeakHashMap<K, V>

(использует хэш-таблицу со "слабыми" связями - разрешена сборка мусора, когда ключи не используются).

Интерфейс SortedMap<K,V>

Наследует Map<K,V>. Реализации этого интерфейса обеспечивают хранение элементов множества ключей в порядке возрастания .

Реализации: TreeMap<K,V>.

Исключительные ситуации, возможные при работе с картами отображений:

Исключение	Условие
<code>NoSuchElementException</code>	В вызывающей карте элементов не существует
<code>ClassCastException</code>	Объект несовместим с элементами карты
<code>NullPointerException</code>	Попытка использовать <code>null</code> -указатель, если он недопустим
<code>IllegalArgumentException</code>	Вызываемые ключ или значение отсутствуют в карте
<code>UnsupportedOperationException</code>	Попытка применить неподдерживаемую картой операцию

Способы хранения в Collections Framework

1. **Массивы;**
2. **Связные списки - цепочка из объектов, ссылающихся друг на друга;**
3. **Бинарные деревья - хранение и поиск упорядоченных объектов → для хранимых объектов необходимо определить отношение порядка при помощи метода `compareTo` интерфейса `Comparable<T>` или класса, реализующего интерфейс `Comparator<T>`. Скорость доступа к произвольному объекту пропорциональна логарифму размера контейнера.**
4. **Хэш-таблицы – контейнеры на основе массивов, в которых для поиска элемента в массиве используется не его индекс, а его хэш-функция, а в нужной позиции массива хранится указатель на связанный список элементов, у которых хэш-функции совпадают. Скорость доступа тем меньше, чем короче связанные списки хэш-таблицы, иными словами, когда хэш-функции различных объектов не совпадают.**

Реализации контейнеров

ArrayList<E>, ArrayDeque<E> - на основе массивов;

LinkedList<E> - на основе связного списка;
двунаправленный замкнутый список;

TreeSet<E>, TreeMap<K, V> - на основе бинарных
деревьев.

HashSet<E>, LinkedHashSet<E>, HashMap<K, V>, WeakHashMap<K, V> - на основе хэш-таблиц

Класс Arrays

Мост между массивами и коллекциями

Некоторые методы:

```
static List asList(Object[]  
array) //возвращает List-объект,  
поддерживаемый массивом объектов.  
static boolean equals(...)
```

Наследованные классы и интерфейсы

Dictionary

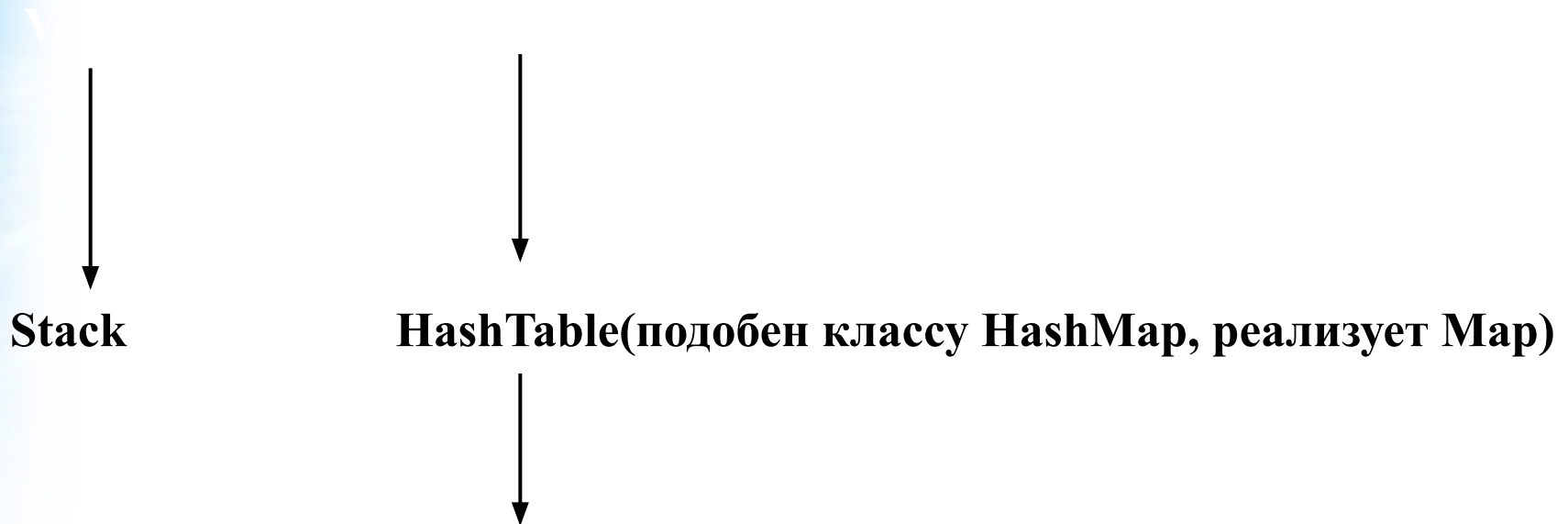
Stack

HashTable

Vector

Properties

(синхронизированы)



Класс Vector

Конструкторы.

```
Vector()  
Vector(int size)  
Vector(int size, int incr)  
Vector(Collection c)
```

Некоторые методы:

```
final int capacity()  
final boolean contains(Object element)  
final Enumeration elements()  
final void addElement(Object element)  
final Object elementAt(int index)  
final void setElementAt(Object element, int index)  
final boolean removeElement(Object element)
```

Интерфейс Enumeration<E> (перечисление)

```
boolean hasMoreElements ();
```

```
<E> nextElement ();
```

```
Vector<E> v;
```

```
. . .
```

```
for (Enumeration<E> e = v.elements ();
```

```
e.hasMoreElements ();)
```

```
System.out.println(e.nextElement ());
```

Сервисные классы

Не входят в структуру коллекций

StringTokenizer - синтаксический анализатор строк

StringTokenizer(String str)

Разделители по умолчанию – пробельные символы(пробел, символ табуляции, символ новой строки) и перевод каретки

StringTokenizer(String str, String delimiters)

StringTokenizer(String str, String delimiters, boolean delimAsToken)

Разделители возвращаются как лексемы, если **delimAsToken** – true

BitSet - специальный тип массива, содержит битовые значения

Date

Calendar

GregorianCalendar

TimeZone

SimpleTimeZone

Local - географический или культурный регион

Random

Интерфейс наблюдателя и наблюдаемый класс

Класс

содержит

наблюдателя

и

наблюдаемый

Класс Observable

Метод	Описание
<code>void addObserver(Observer ob)</code>	Добавляет объект к списку объектов, которые наблюдают вызывающий объект
<code>protected void clearChanged()</code>	Меняет состояние вызывающего объекта на "неизменяемое"
<code>int countObservers()</code>	Возвращает число объектов, которые наблюдают вызывающий объект
<code>void deleteObserver(Observer ob)</code>	Удаляет объект из списка объектов, которые наблюдают вызывающий объект
<code>void deleteObservers()</code>	Удаляет всех наблюдателей
<code>boolean hasChanged()</code>	Возвращает истину, если вызывающий объект был изменен
<code>void notifyObservers()</code>	Уведомляет всех наблюдателей вызывающего объекта о том, что он был изменен с помощью метода <code>update()</code> . В качестве второго аргумента <code>update()</code> передается <code>null</code> .
<code>void notifyObservers(Object obj)</code>	Уведомляет всех наблюдателей вызывающего объекта о том, что он был изменен с помощью метода <code>update()</code> . В качестве второго аргумента <code>update()</code> передается <code>obj</code> .
<code>protected void setChanged()</code>	Вызывается при изменении объекта, инициализирующего обращение

Интерфейс Observer

```
void update (Observable observOb,  
Object arg) ;
```

observOb – наблюдаемый объект

**arg – значение, передаваемое методом
notifyObservers ()**

**Метод update вызывается при изменении
наблюдаемого объекта.**

Расширенная поддержка коллекций

До JAVA 7

В JAVA 7

```
List list = new ArrayList();  
list.add("item");  
String item = list.get(0);
```

```
List list = ["item"];  
String item = list[0];
```

```
Set set = new HashSet();  
set.add("item");
```

```
Set set = {"item"};
```

```
Map map = new HashMap();  
map.put("key", 1);  
int value = map.get("key");
```

```
Map map = {"key" : 1};  
int value = map["key"];
```

Улучшенное вычисление типов при создании коллекций

До ...	Теперь
<pre>Map<String, List<String>> anagrams = new HashMap<String, List<String>> ();</pre>	<pre>Map<String, List<String>> anagrams = new HashMap<> ();</pre>
<p>оператор <code><></code> diamond (бриллиант) получает тип от описания ссылки.</p>	

Объектные оболочки простых типов

Абстрактный класс

Number

Character

Boolean



Управление специальными значениями (“бесконечность” и “не число”)

Double **Float**
`isInfinite()` `isNaN()`

Преобразование чисел в строки и обратно

`parseByte()` `parseShort()` `parseInt()` `parseLong()`
`parseFloat()` `parseDouble()`

`throws` **NumberFormatException**