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.

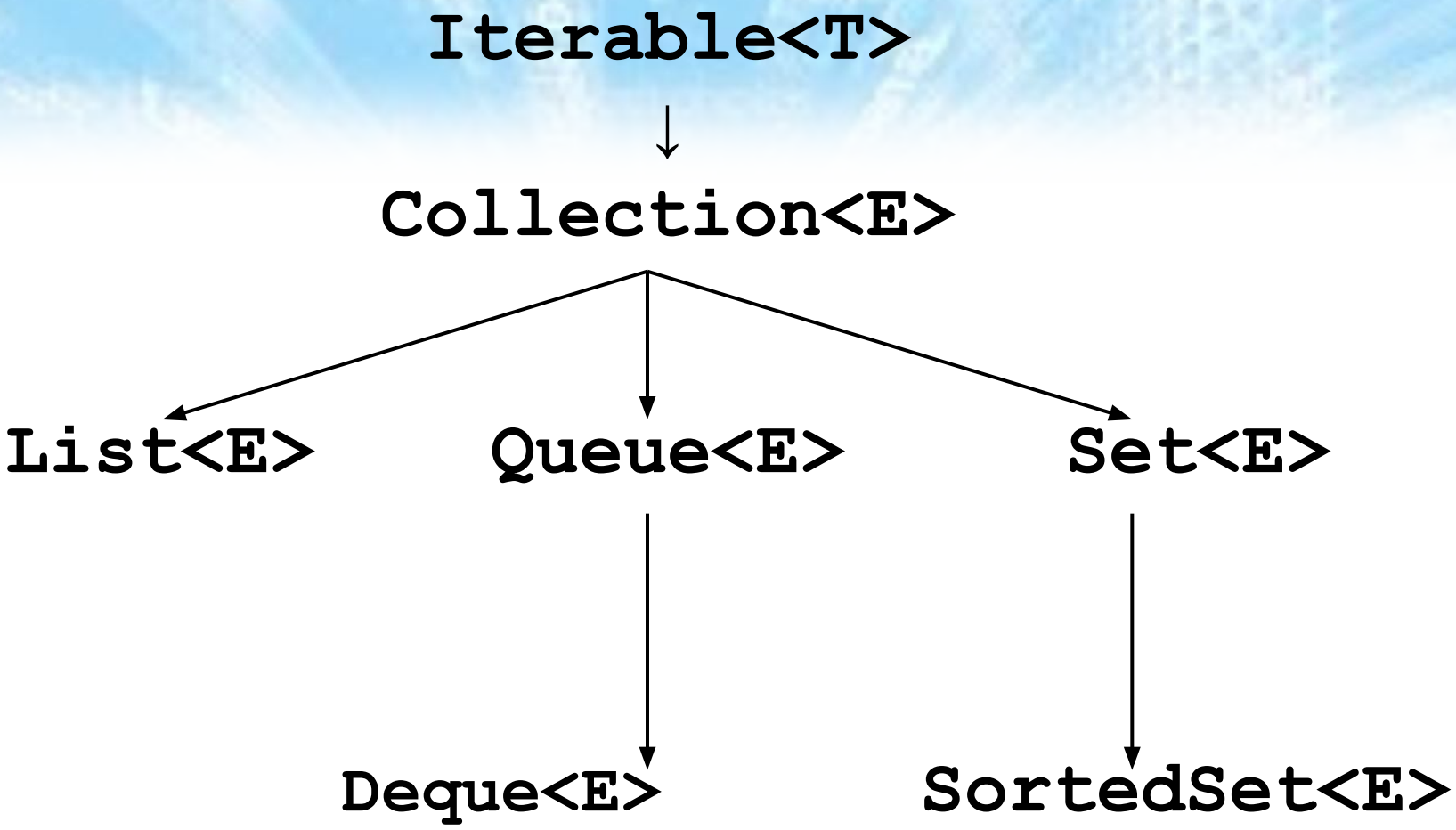
Интерфейсы Collections Framework

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

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>. Двунаправленная очередь. Позволяет добавлять и удалять объекты с двух концов.	**ArrayDeque<E>, LinkedList<E>**
Set<E>	Обеспечивает уникальность ранимых объектов. Желательно переопределить метод **equals** хранимых объектов, т.к. они 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

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

Интерфейс List

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

Интерфейс SortedList

Объявляет поведение набора, отсортированного в возрастающем порядке.

Метод	Описание
`Comparator<E> comparator()`	Возвращает компаратор вызывающего набора
`<E> first()`	Возвращает первый элемент
`SortedSet<E> headSet(<E> end)`	Возвращает объект, содержащий элементы вызывающего набора, которые меньше, чем **`end`**
`<E> last()`	Возвращает последний элемент
`SortedSet<E> subSet(<E> start, <E> end)`	Возвращает объект, содержащий элементы между **`start`** и **`end-1`**
`SortedSet<E> tailSet(<E> start)`	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
`boolean hasNext()`	Возвращает true, если в коллекции присутствует следующий элемент. Иначе – false.
`<E> next()`	Возвращает следующий элемент
`void remove()`	Удаляет текущий элемент.

ListIterator

Обеспечивает *двунаправленный* обход коллекции и модификацию ее элементов.

Метод	Описание
`void add(<E> obj)`	Вставляет obj в список
`boolean hasNext()`	Возвращает true, если в коллекции присутствует следующий элемент. Иначе – false.
`boolean hasPrevious()`	Возвращает true, если в коллекции присутствует предыдущий элемент. Иначе – false.
`<E> next()`	Возвращает следующий элемент
`int nextIndex()`	Возвращает индекс следующего элемента
`int previousIndex()`	Возвращает индекс предыдущего элемента
`<E> previous()`	Возвращает предыдущий элемент
`void set(<E> obj)`	Назначает obj на текущий элемент
`void remove()`	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
`NoSuchElementException`	В вызывающей карте элементов не существует
`ClassCastException`	Объект несовместим с элементами карты
`NullPointerException`	Попытка использовать `null`-указатель, если он недопустим
`IllegalArgumentException`	Вызываемые ключ или значение отсутствуют в карте
`UnsupportedOperationException`	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)

Stack **HashTable(подобен классу HashMap, реализует Map)**

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`

`Grigorian Calendar`

`TimeZone`

`SimpleTimeZone`

`Local` - географический или культурный регион

`Random`

Интерфейс наблюдателя и наблюдаемый класс

К
со
н

Н
о

Класс Observable

Метод	Описание
void addObserver(Observer ob)	Добавляет объект к списку объектов, которые наблюдают вызывающий объект
protected void clearChanged()	Меняет состояние вызывающего объекта на "неизменяемое"
int countObservers()	Возвращает число объектов, которые наблюдают вызывающий объект
void deleteObserver(Observer ob)	Удаляет объект из списка объектов, которые наблюдают вызывающий объект
void deleteObservers()	Удаляет всех наблюдателей
boolean hasChanged()	Возвращает истину, если вызывающий объект был изменен
void notifyObservers()	Уведомляет всех наблюдателей вызывающего объекта о том, что он был изменен с помощью метода update(). В качестве второго аргумента update() передается null.
void notifyObservers(Object obj)	Уведомляет всех наблюдателей вызывающего объекта о том, что он был изменен с помощью метода update(). В качестве второго аргумента update() передается obj.
protected void setChanged()	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
`Map<String, List<String>> anagrams = new HashMap<String, List<String>>();`	`Map<String, List<String>> anagrams = new HashMap<>();`
оператор <> diamond (брилиант) получает тип от описания ссылки.	

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
```