Язык программирования Java

Мультимедийный курс

автор: Васильев А.Н.

www.vasilev.kiev.ua

Киев 2017

Использование стандартных коллекций

- Знакомство с коллекциями
- Интерфейсы коллекций
- Интерфейсы Collection, List, Set и SortedSet
- Классы Collection: ArrayList, Vector, LinkedList, HashSet, TreeSet
- Итераторы
- Компараторы
- Алгоритмы

Знакомство с коллекциями

Коллекции - набор специальных утилит для работы с группами объектов

Основные операции, поддерживаемые для групп объектов:
- Добавление элемента (объекта) в группу
- Удаление элемента (объекта) из группы
- Перебор элементов (объектов) в группе

Механизмы реализации коллекций:
- Набор интерфейсов, определяющих стандарты работы с коллекциями
- Набор классов, реализующих основные интерфейсы, предназначенные для работы с коллекциями
- Набор алгоритмов (специальные статические методы) для работы с коллекциями
- Итераторы - специальные объекты, предназначенные для реализации доступа к элементам коллекций

Основные интерфейсы

`Collection` - основной интерфейс, содержит базовые методы, которыми должна обладать любая коллекция

`List` - расширяет интерфейс `Collection`. Интерфейс для реализации коллекции типа списка: упорядоченной последовательности элементов, к которым можно получить доступ по индексу

`Set` - расширяет интерфейс `Collection`. Интерфейс для реализации множества: набора уникальных элементов, порядок следования которых не важен

`SortedSet` - расширяет интерфейс `Set` . Интерфейс предназначен для реализации упорядоченных множеств

`Queue` - расширяет интерфейс `Collection`. Интерфейс для реализации очереди: коллекции, в которой элементы добавляются в один конец, а считываются с другого конца

`Map` - интерфейс для реализации коллекции, в которой используются ключи и значения

Дополнительные интерфейсы

`Comparator` - интерфейс для определения способа сравнения объектов вколлекции

`Iterator` - интерфейс для создания итераторов для работы с коллекциями (циклический перебор коллекции с возможностью добавления и удаления элементов)

`ListIterator` - расширяет интерфейс `Iterator`. Интерфейс для получения доступа к элементам коллекции типа списка. Позволяет выполнять двунаправленный обход списка и модификацию его элементов

Интерфейс Collection

Содержит объявляния основных методов, общих для всех коллекций

Метод	Описание
`boolean add(Object obj)`	Прибавляет объект `obj` к коллекции. Результат равен `true` если объект добавлен, и `false` если объект уже является эллементом коллекции
`boolean addAll(Collection c)`	Добавляет в исходную коллекцию (из которой вызывается метод) все элементы коллекции `c`. Результатом является `true` если операция прошла удачно, и `false` в противном случае
`void clear()`	Удаляет все элементы из коллекции
`boolean contains(Object obj)`	Возвращает `true` если элемент `obj` содержится в коллекции, и `false` если нет
`boolean containsAll(Collection c)`	Возвращает `true` если все элементы коллекции `c` содержатся в исходной коллекции (из которой вызывается метод), и `false` если нет
`boolean equals(Object obj)`	Проверка на предмет равенства объекта коллекции и объекта, переданного аргументом (две одинаковые коллекции)
`int hashCode()`	Возвращает хэш-код объекта коллекции
`boolean isEmpty()`	Проверка коллекции на предмет того, что она пуста
`Iterator iterator()`	Возвращает итератор для коллекции

Интерфейс Collection

Метод	Описание
`boolean remove(Object obj)`	Удаляет элемент `obj` из вызывающей коллекции. Результатом является `true` если элемент удален из коллекции, и `false` в противном случае
`boolean removeAll(Collection c)`	Удаляет все элементы коллекции `c` из вызывающей коллекции. Результатом является `true` если элементы удалены, и `false` в противном случае
`boolean retainAll(Collection c)`	Удаляет все элементы из вызывающей коллекции, за исключением элементов из коллекции `c`. Результатом является `true` если элементы удалены, и `false` в противном случае
`int size()`	Возвращает размер коллекции - количество элементов, содержащихся в коллекции
`Object[] toArray()`	Возвращает массив, который содержит все элементы, хранящиеся в вызывающей коллекции. Элементы массива являются копиями элементов коллекции

Интерфейс List

Интерфейс `List` расширяет интерфейс `Collection`.
Элемент можно вставлять и извлекать по индексу (отсчитывается от нуля).
Может содержать дублированные (совпадающие) элементы.

Метод	Описание
`void add(int index,Object obj)`	Вставляет объект `obj` в список вызова в позицию, определяемую индексом `index`. Существующие элементы сдвигаются
`boolean addAll(int index,Collection c)`	Добавляет (начиная с позиции `index`) в исходный список (из которого вызывается метод) все элементы коллекции `c`. Существующие элементы сдвигаются. Результатом является `true` если вызывающий список изменяется и `false` в противном случае
`Object get(int index)`	Возвращает объект, хранящийся в позиции с индексом `index`
`int indexOf(Object obj)`	Возвращает индекс первого экземпляра объекта `obj` в списке. Если такого объекта в списке нет, возвращается значение `-1`
`int lastIndexOf(Object obj)`	Возвращает индекс последнего экземпляра объекта `obj` в списке. Если такого объекта в списке нет, возвращается значение `-1`

Интерфейс List

Метод	Описание
`ListIterator listIterator()`	Возвращает итератор, установленный на начало списка
`ListIterator listIterator(int index)`	Возвращает итератор, установленный на элемент с индексом `index`
`Object remove(int index)`	Удаляет из списка элемент с индексом `index`. Результатом возвращается ссылка на удаленный из списка объект
`Object set(int index,Object obj)`	Устанавливает объект `obj` в позицию с индексом `int`. Результатом возвращается ссылка на объект
`List subList(int start,int end)`	Возвращает список из элементов в исходном списке (списке вызова) с индексами от `start` до `end-1` включительно

Интерфейс Set

Интерфейс `Set` расширяет интерфейс `Collection`.

Не допускает дублирования элементов - при добавлении элемента методом `add()` если такой элемент уже есть в коллекции, новый не добавится (метод вернет значение `false`).

Собственных методов не определяет.

Интерфейс SortedSet

Интерфейс `SortedSet` расширяет интерфейс `Set`.
Определяет отсортированное в прядке возрастания множество (набор).

Метод	Описание
`Comparator comparator()`	Возвращает компаратор для упорядоченного множества. Если для множества используется естественное (встроенное) упорядочивание, результатом возвращается пустая ссылка `null`
`Object first()`	Возвращает первый элемент упорядоченного множества
`SortedSet headSet(Object obj)`	Возвращается упорядоченное множество из элементов, которые меньше объекта `obj`
`Object last()`	Возвращает последний элемент в упорядоченном списке
`SortedSet subSet(Object start,Object end)`	Возвращает список из элементов в исходном списке (списке вызова), которые находятся между объектами `start` (включая) и `end` (не включая)
`SortedSet tailSet(Object obj)`	Возвращается упорядоченное множество из элементов, которые больше или равны объекту `obj`

Классы реализующие Collection

Некоторые классы абстрактные, некоторые - нет.

Класс	Описание
`AbstractCollection`	Реализует большую часть интерфейса `Collection`
`AbstractList`	Расширяет класс `AbstractCollection` и реализует большую часть интерфейса `List`
`AbstractSequentialList`	Расширяет класс `AbstractList` для реализации коллекции, которая использует последовательный, а не произвольный доступ к элементам
`LinkedList`	Реализует связанный список, расширяя класс `AbstractList`
`ArrayList`	Реализует динамический массив, расширяя класс `AbstractSequentialList`
`AbstractSet`	Расширяет класс `AbstractCollection` и реализует большую часть интерфейса `Set`
`HashSet`	Расширяет класс `AbstractSet` для реализации хэш-таблиц
`TreeSet`	Реализует набор, хранящийся в виде дерева (древовидный набор). Расширяет класс `AbstractSet`

Для работы с коллекциями также используются классы `Vector`, `Stack` и `Hashtable`

Класс ArrayList

- Расширяет класс `AbstractList` и реализует интерфейс `List`.

- Поддерживает динамические массивы, которые могут изменять размер в процессе выполнения программы.

- Список создается с некоторым начальным размером. При превышении размера список автоматически расширяется. При удалении объектов размер может уменьшаться.

- Динамический массив можно реализовать также с помощью класса `Vector`.

Конструктор	Описание
`ArrayList()`	Создается пустой список
`ArrayList(Collection c)`	Создание списка на основе коллекции
`ArrayList(int capacity)`	Создание списка заданного размера (емкости)

Использование класса ArrayList

Программа: список на основе класса ArrayList - 1

```
// Импорт контейнерного класса:
import java.util.ArrayList;
class UsingArrayListDemo{
   // Статический метод для отображения содержимого контейнера:
   static void show(ArrayList lst){
      for(int k=0;k<lst.size();k++){
         System.out.print("| "+lst.get(k)+" ");
      }
      System.out.println("|");
   }
   // Главный метод:
   public static void main(String[] args){
      // Создание пустого списка:
      ArrayList<Integer> nums=new ArrayList<>();
      // Добавление элементов в список:
      nums.add(100);
      nums.add(200);
      nums.add(300);
      // Отображение содержимого списка:
      show(nums);
      // Добавление элементов в список:
      nums.add(0,-1);
      nums.add(2,0);
      // Отображение содержимого списка:
      show(nums);
      // Продолжение на следующем слайде!!!
```

Использование класса ArrayList

Программа: список на основе класса ArrayList - 2

```
        // Очистка списка:
        nums.clear();
        // Если список пустой:
        if(nums.isEmpty()) nums.add(123);
        // Отображение содержимого списка:
        show(nums);
        // Создание списка указанной емкости:
        ArrayList<Character> chars=new ArrayList<>(10);
        // Добавление элементов в список:
        chars.add('a');
        chars.add('b');
        chars.add(0,'z');
        // Отображение содержимого списка (неявный вызов метода toString()):
        System.out.println(chars);
        // Преобразование списка в массив:
        Object[] myarray=chars.toArray();
        // Поэлементное отображение элементов массива:
        for(int i=0;i<myarray.length;i++){
           System.out.print((Character)myarray[i]+" ");
        }
        System.out.println();
    }
}
```

Результат

```
| 100 | 200 | 300 |
| -1 | 100 | 0 | 200 | 300 |
| 123 |
[z, a, b]
z a b
```

Класс Vector

- Класс `Vector` Подобен классу `ArraytList`: наследует класс `AbstractList` и реализует интерфейс `List`.

- Поддерживает динамические массивы, которые могут изменять размер в процессе выполнения программы.

- Список создается с некоторым начальным размером. Когда размер исчерпан, он увеличивается. При создании контейнера можно указать размер "добавки" к размеру контейнера. Если "добавку" не указать, то каждый раз размер удваивается.

Конструктор	Описание
`Vector()`	Создается пустой вектор
`Vector(int size)`	Создание вектора указанного размера
`Vector(int size,ind delta)`	Создание вектора указанного размера и определение приращения для изменения размера
`Vector(Collection c)`	Создание вектора на основе коллекции

Использование класса Vector

Программа: список на основе класса Vector

```
// Импорт контейнерного класса:
import java.util.Vector;
class UsingVectorDemo{
   public static void main(String[] args){
      // Создание пустого вектора:
      Vector<Integer> nums=new Vector<>();
      // Добавление элементов в вектор:
      for(int i=1;i<=3;i++){
         nums.addElement(i*100);
      }
      // Отображение содержимого вектора:
      System.out.println(nums);
      // Добавление элементов в вектор:
      nums.add(0,-1);
      nums.add(2,0);
      // Отображение содержимого вектора:
      System.out.println(nums);
      // Изменение размера вектора:
      nums.setSize(nums.size()-2);
      // Отображение содержимого вектора:
      System.out.println(nums);
   }
}
```

Результат

```
[100, 200, 300]
[-1, 100, 0, 200, 300]
[-1, 100, 0]
```

Класс LinkedList

- Расширяет класс `AbstractSequentialList` и реализует интерфейс `List`.

- Поддерживает связный список.
- Список создается пустым или на основе существующей коллекции.
- В список можно добавлять элементы, и из списка можно удалять элементу.

Конструктор	Описание
`LinkedList()`	Создается пустой список
`LinkedList(Collection c)`	Создание списка на основе коллекции

Дополнительные методы

Метод	Описание
`addFirst(Object obj)`	Добавляется первый элемент
`addLast(Object obj)`	Добавляется последний элемент
`getFirst()`	Получить первый элемент
`getLast()`	Получить последний элемент
`removeFirst()`	Удалить первый элемент
`removeLast()`	Удалить последний элемент

Основное отличие классов `ArrayList` и `LinkedList` - в способе реализации списков.

- Контейнер на основе класса `ArrayList` создается с определенным размером "с запасом". Размер контейнера изменяется по мере необходимости.
- Список на основе класса `LinkedList` содержит цепочку элементов - ровно столько, сколько элементов добавлено в список

Использование класса LinkedList

Программа: список на основе класса LinkedList

```
// Импорт контейнерного класса:
import java.util.LinkedList;
class UsingLinkedListDemo{
   public static void main(String[] args){
      // Создание пустого списка:
      LinkedList<Integer> nums=new LinkedList<>();
      // Добавление элементов в список:
      nums.add(100);
      nums.add(0,200);
      nums.addLast(300);
      System.out.println(nums); // Отображение содержимого списка
      // Добавление и удаление элементов:
      nums.removeFirst();
      nums.add(0,400);
      nums.removeLast();
      nums.addLast(500);
      System.out.println(nums); // Отображение содержимого списка
      // Преобразование списка в массив:
      Object[] myarray=nums.toArray();
      // Поэлементное отображение элементов массива:
      for(int i=0;i<myarray.length;i++){
         System.out.print((Integer)myarray[i]+" ");
      }
      System.out.println();
   }
}
```

Результат

```
[200, 100, 300]
[400, 100, 500]
400 100 500
```

Класс HashSet

- Расширяет класс `AbstractSet` и реализует интерфейс `Set`.
- Поддерживает коллекцию на основе хэш-таблицы: информационное значение (элемент) используется для формирования уникального хэш-кода. Этот хэш-код используется как "индекс" элемента.

Конструктор	Описание
`HashSet()`	Создается пустая коллекция
`HashSet(Collection c)`	Создание коллекция на основе коллекции
`HashSet(int capacity)`	Создается коллекция указанной емкости
`HashSet(int capacity,float fillRatio)`	Создается коллекция указанной емкости и уровня заполнения (параметр от 0.0 до 1.0)

Важные методы

Метод	Описание
`add(Object obj)`	Добавляется элемент в коллекцию
`contains(Object obj)`	Проверка наличия элемента в коллекции
`remove(Object obj)`	Удаление элемента из коллекции
`size()`	Размер коллекции

Использование класса HashSet

Программа: коллекция на основе класса HashSet

```
// Импорт контейнерного класса:
import java.util.HashSet;
class UsingHashSetDemo{
   public static void main(String[] args){
      // Создание пустой коллекции:
      HashSet<String> objs=new HashSet<>();
      // Добавление элементов в коллекцию:
      objs.add("Первый");
      objs.add("Второй");
      objs.add("Третий");
      // Отображение содержимого коллекции:
      System.out.println(objs);
      // Удаление элемента из коллекции:
      objs.remove("Первый");
      // Добавление элемента в коллекцию:
      objs.add("Четвертый");
      // Количество элементов в коллекции:
      System.out.println("В коллекции "+objs.size()+" элемента:\n"+objs);
      // Наличие/отсутствие элементов в коллекции:
      if(!objs.contains("Первый"))
                        System.out.println("Элемента ПЕРВЫЙ в коллекции нет");
      if(objs.contains("Четвертый"))
                        System.out.println("Элемент ЧЕТВЕРТЫЙ в коллекции есть");
   }
}
```

Результат

```
[Первый, Третий, Второй]
В коллекции 3 элемента:
[Четвертый, Третий, Второй]
Элемента ПЕРВЫЙ в коллекции нет
Элемент ЧЕТВЕРТЫЙ в коллекции есть
```

Использование класса HashSet

Программа: коллекция на основе класса HashSet - 1

```
// Импорт контейнерного класса:
import java.util.HashSet;
// Пользовательский класс:
class MyClass{
    // Поле:
    int num;
    // Конструктор:
    MyClass(int n){
        num=n;
    }
    // Переопределение метода toString():
    public String toString(){
       return "Объект класса MyClass: "+num;
    }
}
// Главный класс:
class UsingTreeSetDemo{
   public static void main(String[] args){
      // Создание пустой коллекции:
      HashSet objs=new HashSet();
      // Добавление элементов в коллекцию:
      objs.add("Первый");
      objs.add(200);
      objs.add('R');
      objs.add(new MyClass(123));
      // Продолжение на следующем слайде!!!
```

Использование класса HashSet

Программа: коллекция на основе класса HashSet - 2

```
        // Отображение содержимого коллекции:
        System.out.println(objs);
        // Удаление элемента из коллекции:
        objs.remove("Первый");
        // Добавление элемента в коллекцию:
        objs.add("Четвертый");
        // Количество элементов в коллекции:
        System.out.println("В коллекции "+objs.size()+" элемента:\n"+objs);
        // Наличие/отсутствие элементов в коллекции:
        if(!objs.contains("Первый"))
                        System.out.println("Элемента ПЕРВЫЙ в коллекции нет");
        if(objs.contains("Четвертый"))
                        System.out.println("Элемент ЧЕТВЕРТЫЙ в коллекции есть");
    }
}
```

Результат

```
[R, Первый, Объект класса MyClass: 123, 200]
В коллекции 4 элемента:
[R, Объект класса MyClass: 123, 200, Четвертый]
Элемента ПЕРВЫЙ в коллекции нет
Элемент ЧЕТВЕРТЫЙ в коллекции есть
```

Класс TreeSet

- Реализует интерфейс `Set`.
- Для хранения данных используется древовидная структура.
- Подходит для хранения больших наборов сортированной информации.

Конструктор	Описание
`TreeSet()`	Создается пустая коллекция
`TreeSet(Collection c)`	Создание коллекция на основе коллекции
`TreeSet(Comparator cmp)`	Создается коллекция указанной емкости
`TreeSet(SortedSet st)`	Создается коллекция на основе сортированного списка

Важные методы

Метод	Описание
`add(Object obj)`	Добавляется элемент в коллекцию
`contains(Object obj)`	Проверка наличия элемента в коллекции
`remove(Object obj)`	Удаление элемента из коллекции
`size()`	Размер коллекции

Использование класса TreeSet

Программа: коллекция на основе класса TreeSet

```
// Импорт контейнерного класса:
import java.util.TreeSet;
class UsingTreeSetDemo{
   public static void main(String[] args){
      // Создание пустой коллекции:
      TreeSet<String> objs=new TreeSet<>();
      // Добавление элементов в коллекцию:
      objs.add("Первый");
      objs.add("Второй");
      objs.add("Третий");
      // Отображение содержимого коллекции:
      System.out.println(objs);
      // Удаление элемента из коллекции:
      objs.remove("Первый");
      // Добавление элемента в коллекцию:
      objs.add("Четвертый");
      // Количество элементов в коллекции:
      System.out.println("В коллекции "+objs.size()+" элемента:\n"+objs);
      // Наличие/отсутствие элементов в коллекции:
      if(!objs.contains("Первый"))
                        System.out.println("Элемента ПЕРВЫЙ в коллекции нет");
      if(objs.contains("Четвертый"))
                        System.out.println("Элемент ЧЕТВЕРТЫЙ в коллекции есть");
   }
}
```

Результат

```
[Второй, Первый, Третий]
В коллекции 3 элемента:
[Второй, Третий, Четвертый]
Элемента ПЕРВЫЙ в коллекции нет
Элемент ЧЕТВЕРТЫЙ в коллекции есть
```

Итератор

- Итератор - специальный объект, используемый для перебора элементов в коллекции.
- Итератор создается на основе класса, реализующего интерфейс `Iterator` или `ListIterator`.
- Интерфейс `Iterator` позволяет циклически пройти коллекцию, получать доступ к элементам или удалять элементы.
- Интерфейс `ListIterator` расширяет `Iterator`, обеспечивая двунаправленный обход коллекции и модификацию ее элементов.

Методы интерфейса Iterator

Метод	Описание
`boolean hasNext()`	Возвращает `true`, если в коллекции есть следующий элемент. Иначе возвращает `false`
`Object next()`	Возвращает ссылку на следующий элемент коллекции. Если следующего элемента нет, выбрасывается исключение класса `NoSuchElementException`
`void remove()`	Удаляет текущий элемент. Выбрасывает исключение класса `IllegalStateException` (если перед вызовом `remove()` не был вызван метод `next()`)

Итератор

Методы интерфейса ListIterator

Метод	Описание
`void add(Object obj)`	Вставляет `obj` в список перед элементом, который будет возвращен при следующем вызове методом `next()`
`boolean hasNext()`	Возвращает `true`, если в коллекции есть следующий элемент. Иначе возвращает `false`
`boolean hasPrevious()`	Возвращает `true`, если в коллекции есть предыдущий элемент. Иначе возвращает `false`
`Object next()`	Возвращает ссылку на следующий элемент коллекции. Если следующего элемента нет, выбрасывается исключение класса `NoSuchElementException`
`int nextIndex()`	Возвращает индекс следующего элемента в списке. Если следующего элемента нет, возвращается размер списка
`Object previous()`	Возвращает ссылку на предыдущий элемент коллекции. Если следующего элемента нет, выбрасывается исключение класса `NoSuchElementException`
`int previousIndex()`	Возвращает индекс предыдущего элемента в списке. Если следующего элемента нет, возвращается `-1`
`void remove()`	Удаляет текущий элемент. Выбрасывает исключение класса `IllegalStateException` (если перед вызовом `remove()` не был вызван метод `next()`)
`void set(Object obj)`	Назначает `obj` на текущий элемент (элемент, ссылка на который получена вызовом метода `next()` или `previous()`)

Итератор

Как используется итератор

- В каждом коллекционном классе (реализует интерфейс `Collection`) есть метод `iterator()`, возвращающий ссылку на объект-итератор, установленный на начало коллекции.

- Для доступа к очередному элементу коллекции используют метод `next()`.

- Метод `next()` можно вызывать, если метод `hasNext()` возвращает значение `true`.

- Коллекции, созданные на основе класса, реализующего интерфейс `List`, имеют метод `listIterator()`, возвращающий ссылку на объект-итератор, реализующий интерфейс `ListIterator`.

Использование итератора

Программа: использование итератора

```
import java.util.HashSet;
import java.util.Iterator;
import java.util.NoSuchElementException;
class UsingIteratorDemo {
   public static void main(String[] args){
      HashSet<String> cls=new HashSet<>();
      cls.add("Красный");
      cls.add("Желтый");
      cls.add("Зеленый");
      Iterator<String> pnt; // Переменная для записи ссылки на итератор
      pnt=cls.iterator();   // Получение ссылки на итератор
      while(pnt.hasNext()){
         System.out.println(pnt.next()); // Доступ к следующему элементу
      }
      pnt.remove();                      // Удаление элемента через итератор
      pnt=cls.iterator();                // Ссылка на итератор
      System.out.println("После удаления элемента");
      try{
         while(true){
            System.out.println(pnt.next()); // Доступ к элементу через итератор
         }
      }
      catch(NoSuchElementException e){       // Обработка исключения
         System.out.println("Элементов больше нет!");
      }
}}
```

Результат

```
Зеленый
Желтый
Красный
После удаления элемента
Зеленый
Желтый
Элементов больше нет!
```

Использование итератора

Программа: использование итератора

```
import java.util.ArrayList;
import java.util.ListIterator;
class UsingListIteratorDemo {
   public static void main(String[] args){
      ArrayList<Integer> nums=new ArrayList<>();
      int k=0;
      nums.add(100);
      nums.add(200);
      nums.add(300);
      ListIterator<Integer> pnt=nums.listIterator(); // Итератор
      while(pnt.hasNext()){
         System.out.print(pnt.next()+" ");           // Прямой перебор
      }
      System.out.println();
      pnt.set(400);                                  // Изменение элемента
      while(pnt.hasPrevious()){
         if(k%2==0) pnt.add(k+1);                    // Добавление элемента
         k++;
         System.out.print(pnt.previous()+" ");       // Обратный перебор
      }
      System.out.println();
      for(k=0;k<nums.size();k++){
         System.out.print(nums.get(k)+" ");          // Проверка содержимого
      }
      System.out.println();
}}
```

Результат

```
100 200 300
1 400 3 200 5 100
100 5 200 3 400 1
```

Компаратор

- Компаратор - специальный объект, используемый для сравнения элементов коллекции для их упорядочивания.

- Компаратор создается на основе класса, реализующего интерфейс `Comparator`.

Методы интерфейса Comparator

Метод	Описание
`int compare(Object a,Object b)`	Метод для сравнения объектов `a` и `b`. Если метод возвращает `0`, то объекты равны. При положительном результате объект `a` больше объекта `b`. При отрицательном результате объект `b` больше объекта `a`
`boolean equals(Object obj)`	Метод используется для сравнения компараторов: выполняется проверка, равен ли переданный аргументом компараторный объект тому компараторному объекту, из которого вызывается метод. Метод обычно не используется и его можно не определять

Использование компаратора

Программа: использование компаратора

```
import java.util.TreeSet;
import java.util.ArrayList;
import java.util.Comparator;
class MC implements Comparator{ // Класс компаратора
   public int compare(Object a,Object b){
        int x,y;
        x=(Integer)a/10%10;
        y=(Integer)b/10%10;
        return x-y;
}}
class UsingComparatorDemo {
   public static void main(String[] args){
      ArrayList<Integer> nums=new ArrayList<>();
      int[] n={350,140,530,220,410};
      // int[] n={300,100,500,200,400};
      for(int k=0;k<n.length;k++){
         nums.add(n[k]);
      }
      System.out.println(nums);
      TreeSet<Integer> A=new TreeSet<>(nums);
      System.out.println(A);
      // Использование компаратора:
      TreeSet<Integer> B=new TreeSet<>(new MC());
      B.addAll(nums);
      System.out.println(B);
}}
```

Результат

```
[350, 140, 530, 220, 410]
[140, 220, 350, 410, 530]
[410, 220, 530, 140, 350]
```

Результат

```
[300, 100, 500, 200, 400]
[100, 200, 300, 400, 500]
[300]
```

Алгоритмы коллекций

Некоторые статические методы класса Collections

Метод	Описание
`int binarySearch(List L, Object V, Comparator C)`	Поиск объекта `V` из списка `L`, упорядоченного с помощью компаратора `C`. Возвращается позиция объекта в списке или `-1` если значение не найдено
`void copy(List L1,List L2)`	Копирует элементы `L2` в `L1`
`Object max(Collection L, Comparator C)`	Возвращает максимальный элемент в коллекции `L` используя для сравнения элементов компаратор `C`
`Object min(Collection L, Comparator C)`	Возвращает максимальный элемент в коллекции `L` используя для сравнения элементов компаратор `C`
`void reverse(List L)`	Реверсирует последовательность из списка `L`
`Comparator reverseOrder()`	Возвращает компаратор, обратный к используемому по умолчанию (компаратор, который реверсирует результат сравнения объектов)
`void sort(List L, Comparator C)`	Сортирует элементы списка `L` по правилам компаратора `C`

Использование алгоритмов

Программа: использование алгоритмов - 1

```
import java.util.Collections;
import java.util.Comparator;
import java.util.ArrayList;
// Класс для объектов из списка:
class MyClass{
   private int code;
   private String name;
   MyClass(String name,int code){
      this.name=name;
      this.code=code;
   }
   int get(){
      return code;
   }
   public String toString(){
      return name+"-"+code;
   }
}
// Класс компаратора:
class MC implements Comparator{
   public int compare(Object a,Object b){
      return ((MyClass)a).get()-((MyClass)b).get();
   }
}
// Продолжение на следующем слайде!!!
```

Использование алгоритмов

Программа: использование алгоритмов - 2

```
class UsingCollectionsDemo{
   public static void main(String[] args){
      int[] nums={12,5,17,8,21,25,9};              // Числовой и текстовый массивы
      String[]
txt={"ПЕРВЫЙ","ВТОРОЙ","ТРЕТИЙ","ЧЕТВЕРТЫЙ","ПЯТЫЙ","ШЕСТОЙ","СЕДЬМОЙ"};
      ArrayList<Integer> al=new ArrayList<>();          // Числовой список
      for(int i=0;i<nums.length;i++) al.add(nums[i]);   // Заполнение списка
      System.out.println(al);                           // Отображение списка
      Collections.sort(al);                             // Сортировка списка
      System.out.println(al);                           // Отображение списка
      Collections.reverse(al);                          // Обратный порядок
      System.out.println(al);                           // Отображение списка
      for(int k=1;k<=3;k++) al.add(2*k,k*100);          // Добавление элементов
      System.out.println(al);                           // Отображение списка
      Collections.sort(al,Collections.reverseOrder());  // Сортировка списка
      System.out.println(al);                           // Отображение списка
      ArrayList<MyClass> ml=new ArrayList<>();          // Список объектов
      // Добавление объектов в список:
      for(int k=0;k<nums.length;k++) ml.add(new MyClass(txt[k],nums[k]));
      System.out.println(ml);                           // Отображение списка
      MC cmp=new MC();                                  // Компаратор
      System.out.println("Наибольшее значение: "+Collections.max(ml,cmp));
      System.out.println("Наименьшее значение: "+Collections.min(ml,cmp));
      Collections.sort(ml,cmp);                         // Сортировка списка
      System.out.println(ml);                           // Отображение списка
}}
```

Использование алгоритмов

Программа: использование алгоритмов - результат

```
[12, 5, 17, 8, 21, 25, 9]
[5, 8, 9, 12, 17, 21, 25]
[25, 21, 17, 12, 9, 8, 5]
[25, 21, 100, 17, 200, 12, 300, 9, 8, 5]
[300, 200, 100, 25, 21, 17, 12, 9, 8, 5]
[ПЕРВЫЙ-12, ВТОРОЙ-5, ТРЕТИЙ-17, ЧЕТВЕРТЫЙ-8, ПЯТЫЙ-21, ШЕСТОЙ-25, СЕДЬМОЙ-9]
Наибольшее значение: ШЕСТОЙ-25
Наименьшее значение: ВТОРОЙ-5
[ВТОРОЙ-5, ЧЕТВЕРТЫЙ-8, СЕДЬМОЙ-9, ПЕРВЫЙ-12, ТРЕТИЙ-17, ПЯТЫЙ-21, ШЕСТОЙ-25]
```

Домашнее задание

- Напишите программу, в которой создается список с числами Фибоначчи. Используйте классы `ArrayList`, `LinkedList`, `Vector`. Предложите вариант программы с использованием класса `Stack` (см. далее).

- Напишите программу, в которой на основе контейнерного класса формируется список из объектов пользовательского класса. Каждый объект содержит числовое и символьное поле.

- То же, что и в предыдущих пунктах, но с использованием итератора.

- Напишите программу, в которой на основе контейнерного класса формируется список из объектов пользовательского класса. Каждый объект содержит текстовое поле. Выполните сортировку списка с использованием компаратора, который сравнивает объекты путем сравнения длины текстовых строк (поля объекта). При одинаковой длине строк сравниваются первые символы в каждой строке.

Класс Stack

- Является подклассом класса `Vector`.
- Реализует стандартный стек по форме LIFO (Last In First Out).

Дополнительные (кроме из Vector) методы класса Stack

Метод	Описание
`boolean empty()`	Возвращает `true` если стек пуст, и `false` если стек содержит элементы
`Object peek()`	Возвращает элемент на вершине стека (но не удаляет этот элемент!)
`Object pop()`	Возвращает элемент, находящийся на вершине стека. При этом элемент из стека удаляется
`Object push(Object obj)`	Помещает элемент в стек (в вершину стека) и возвращает ссылку на него
`int search(Object obj)`	Выполняет поиск элемента, переданного аргументом методу, в стека, из которого вызывается метод. Если элемент найден, то возвращается его смещение от вершины стека. Если элемент не найден, возвращается значение `-1`

Использование стека

Программа: использование стека

```
import java.util.Stack;
import java.util.Random;
class UsingStackDemo{
   public static void main(String[] args){
      // Для генерирования случайных чисел:
      Random rnd=new Random();
      // Пустой стек:
      Stack<Integer> st=new Stack<>();
      if(st.empty()){
         // Заполнение стека:
         for(int k=1;k<=5;k++){
            System.out.println("Добавлен "+k+"-й элемент:
"+st.push(rnd.nextInt(101)));
         }
      }
      // Содержимое стека:
      System.out.println("Сформирован стек: "+st);
      // Извлечение элементов:
      while(!st.empty()){
         System.out.println("Прочитан элемент: "+st.pop());
         System.out.println("Содержимое стека: "+st);
      }
   }
}
```

Использование стека

Программа: использование стека - результат

```
Добавлен 1-й элемент: 85
Добавлен 2-й элемент: 84
Добавлен 3-й элемент: 21
Добавлен 4-й элемент: 61
Добавлен 5-й элемент: 82
Сформирован стек: [85, 84, 21, 61, 82]
Прочитан элемент: 82
Содержимое стека: [85, 84, 21, 61]
Прочитан элемент: 61
Содержимое стека: [85, 84, 21]
Прочитан элемент: 21
Содержимое стека: [85, 84]
Прочитан элемент: 84
Содержимое стека: [85]
Прочитан элемент: 85
Содержимое стека: []
```

Класс Hashtable

- Наследует абстрактный класс `Dictionary` и реализует интерфейс `Map`.
- Хранит пары ключ/значение в хэш-таблице.

Конструктор	Описание
`Hashtable()`	Создается пустая таблица
`Hashtable(int size)`	Создание таблицы заданного размера
`Hashtable(int size,float fillRatio)`	Создание таблицы заданного размера и долей заполнения (по умолчанию 0.75)
`Hashtable(Map m)`	Создается таблица, инициализируемая картой

Некоторые методы класса Hashtable

Метод	Описание
`void clear()`	Очистка таблицы
`Object get(Object key)`	Возвращает элемент по ключу
`Object put(Object k,Object v)`	Добавляет элемент с ключом в таблицу
`Object remove(Object key)`	Удаляет элемент по ключу
`int size()`	Размер таблицы

Использование Hashtable

Программа: использование Hashtable

```
import java.util.Hashtable;
class UsingHashtable{
   public static void main(String[] args){
      // Пустая таблица:
      Hashtable<String,Integer> ht=new Hashtable<>();
      // Добавление элементов в таблицу:
      ht.put("Иван Иванов",35);
      ht.put("Петр Петров",27);
      ht.put("Полиграф Шариков",43);
      // Содержимое таблицы:
      System.out.println("Содержимое таблицы: "+ht);
      // Запрос элемента по ключу:
      String name="Полиграф Шариков";
      // String name="полиграф шариков";
      System.out.println("Пользователь: "+name);
      if(ht.containsKey(name))
         System.out.println("Возраст: "+ht.get(name));
      else
         System.out.println("Такого пользователя нет");
   }
}
```

Результат

```
Содержимое таблицы: {Иван Иванов=35, Петр Петров=27, Полиграф Шариков=43}
Пользователь: Полиграф Шариков
Возраст: 43
```