Программирование на Java
Тема 5.2Коллекции

Интерфейс Comparable

```
public interface Comparable <T>
      { public int compareTo(T obj); }
```

Интерфейс Comparable

- В интерфейсе Comparable объявлен всего один **метод compareTo(T obj),** предназначенный для реализации упорядочивания объектов класса. Его удобно использовать при сортировке упорядоченных списков или массивов объектов.
- Данный метод сравнивает объект, вызывающий метод с объектом obj. В отличие от метода equals, который возвращает true или false, compareTo возвращает:
  - 0, если значения равны;
  - Отрицательное значение, если вызывающий объект меньше параметра;
  - Положительное значение , если вызывающий объект больше параметра.
- Метод может выбросить исключение ClassCastException, если типы объектов не совместимы при сравнении.
- Необходимо помнить, что аргумент метода compareTo имеет тип сравниваемого объекта класса.
- Классы Byte, Short, Integer, Long, Double, Float, Character, String , Date уже реализуют интерфейс Comparable.
- См. на сл. слайде пример реализующий интерфейс Comparable.
-

Интерфейс Comparable

```
import java.util.*;

class SortableObject implements Comparable
{    private long id;
     private String value;
     public long getId() {return id;}
     public void setId(long id) { this.id = id;}
     public String getValue() {return value;}
     public void setValue(String value) { this.value = value;}
     public SortableObject(long id, String value)
      {    this.id = id;
           this.value = value;
       }
     @Override
     public int compareTo(Object o)
      {    //сортировка
          int res = 0;
          SortableObject compared = (SortableObject) o;
          if (compared.id < this.id) { res = 1;}
          if (compared.id > this.id) { res = -1;}
          return res;
     }
     @Override
     public String toString() {
          return "id="+this.id+" value="+this.value;
     }
 }
```

Интерфейс Comparable

```
public class IthqArraySort
{  public static void main(String[] args)
   {  //сортировка массива строк
      String[] strArray = new String[5];
      for (int j = 0; j < 5; j++)
        {int suff = 9-j;
         strArray[j]="Val"+suff;
        }
      System.out.println("До сортировки / массив строк");
      for (String tmp: strArray) { System.out.println(tmp);}
        Arrays.sort(strArray);
      System.out.println("После сортировки / массив строк");
      for (String tmp: strArray) { System.out.println(tmp);}

      //сортировка массива объектов
      SortableObject[] sArray = new SortableObject[5];
      for (int j = 0; j < 5; j++)
        {  long v = 9-j;
           SortableObject obj = new SortableObject(v, "Value "+v);
           sArray[j]=obj;
        }
      System.out.println("До сортировки / массив объектов");
      for (SortableObject tmp: sArray) {System.out.println(tmp);}
         Arrays.sort(sArray);
      System.out.println("После сортировки / массив объектов");
      for (SortableObject tmp: sArray) { System.out.println(tmp);}
```

Интерфейс Comparable

```
    //сортировка ArrayList строк
    List<String> stringsList = new ArrayList<String>();
      stringsList.add("xxxyyy");
      stringsList.add("eeefff");
      stringsList.add("ccdddd");
      stringsList.add("aaabbb");
    System.out.println("До сортировки / List строк");
    for (String tmp: stringsList) { System.out.println(tmp);}
    //сортировка в обратном порядке
      Collections.reverse(stringsList);
    System.out.println("После сортировки / List строк");
    for (String tmp: stringsList) { System.out.println(tmp);}
    //сортировка ArrayList объектов
    List<SortableObject> list = new ArrayList<SortableObject>();
    for (long j = 0; j < 6; j++)
      {   long v = 9-j;
          SortableObject obj = new SortableObject(v, "Value " + v);
          list.add(obj);
      }
     System.out.println("До сортировки / List объектов");
     for (SortableObject tmp: list) { System.out.println(tmp);}
       Collections.sort(list);
     System.out.println("После сортировки / List объектов");
     for (SortableObject tmp: list) { System.out.println(tmp);}
     Collections.reverse(list);
     System.out.println("После обратной сортировки / List объектов");
     for (SortableObject tmp: list) { System.out.println(tmp);}
  }
}
```

Интерфейс Comparable

```
До сортировки / массив строк
Val9
Val8
Val7
Val6
Val5
После сортировки / массив строк
Val5
Val6
Val7
Val8
Val9
До сортировки / массив объектов
id=9 value=Value 9
id=8 value=Value 8
id=7 value=Value 7
id=6 value=Value 6
id=5 value=Value 5
После сортировки / массив объектов
id=5 value=Value 5
id=6 value=Value 6
id=7 value=Value 7
id=8 value=Value 8
id=9 value=Value 9
```

```
До сортировки / List строк
xxxyyy
eeefff
cccddd
aaabbb
После сортировки / List строк
aaabbb
cccddd
eeefff
xxxyyy
До сортировки / List объектов
id=9 value=Value 9
id=8 value=Value 8
id=7 value=Value 7
id=6 value=Value 6
id=5 value=Value 5
id=4 value=Value 4
После сортировки / List объектов
id=4 value=Value 4
id=5 value=Value 5
id=6 value=Value 6
id=7 value=Value 7
id=8 value=Value 8
id=9 value=Value 9
После обратной сортировки / List объектов
id=9 value=Value 9
id=8 value=Value 8
id=7 value=Value 7
id=6 value=Value 6
id=5 value=Value 5
id=4 value=Value 4
```

Интерфейс Comparator

```
public interface Comparator <T>
    { int compare(T obj1, T obj2);
      equals(T obj).
    }
```

Интерфейс Comparator

- В интерфейсе Comparator объявлено два метода compare(T obj1, T obj2) и equals(T obj).
- compare(T obj1, T obj2) – так же, как и метод compareTo интерфейса Comparable, упорядочивает объекты класса. Точно так же на выходе получает 0, положительное значение и отрицательное значение.
- Метод может выбросить исключение ClassCastException, если типы объектов не совместимы при сравнении.
- Основным отличием интерфейса Comparator от Comparable является то, что вы можете создавать несколько видов независимых сортировок.
- equals(T obj) - сравнивает компараторы объектов. Переопределяется очень редко.
- См. на сл. слайде пример реализующий интерфейс Comparator.

Интерфейс Comparator

```
import java.util.Arrays;
import java.util.Comparator;

class Product
 {  private String name;
    private double price;
    private int quantity;
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
    public double getPrice() { return price;}
    public void setPrice(double price) { this.price = price;}
    public int getQuantity() { return quantity;}
    public void setQuantity(int quantity) { this.quantity = quantity;}
}
```

Интерфейс Comparator

```
// реализация интерфейса Comparator для сортировки по названию
class SortedByName implements Comparator<Product>
  {  public int compare(Product obj1, Product obj2)
        {  String str1 = obj1.getName();
           String str2 = obj2.getName();
           return str1.compareTo(str2);
        }
  }

// реализация интерфейса Comparator для сортировки по цене
class SortedByPrice implements Comparator<Product>
   {  public int compare(Product obj1, Product obj2)
        {  double price1 = obj1.getPrice();
           double price2 = obj2.getPrice();
           if(price1 > price2) { return 1;}
             else if(price1 < price2)  {return -1;}
                    else { return 0; }
        }
    }
```

Интерфейс Comparator

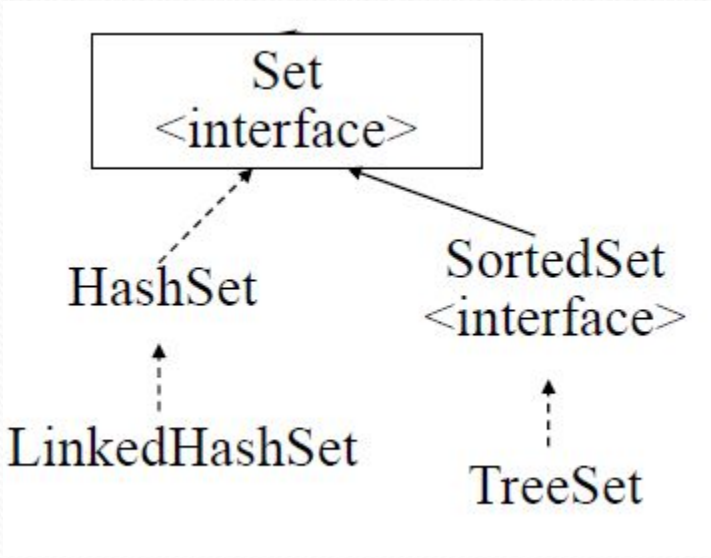
```
public class Example
{   public static void main(String[] args)
    {   Product[] p = new Product[3];
        // заполним объект Product содержимым
        p[0] = new Product(); p[0].setName("Milk");
        p[0].setPrice(7.56);  p[0].setQuantity(56);

        p[1] = new Product(); p[1].setName("Coffee");
        p[1].setPrice(17.00); p[1].setQuantity(32);

        p[2] = new Product(); p[2].setName("Tea");
        p[2].setPrice(12.50); p[2].setQuantity(0);
    System.out.println("============ no sorted ==============");
    for(Product i : p)
    { System.out.println("Name: " + i.getName() +
      " price: " + i.getPrice() +" quantity: " + i.getQuantity());
    }
    System.out.println("========== sorted by price===========");
    Arrays.sort(p, new SortedByPrice());
    for(Product i : p)
    { System.out.println("Name: " + i.getName() +
      " price: " + i.getPrice() + " quantity: " + i.getQuantity());
    }
    System.out.println("========== sorted by name ===========");
    Arrays.sort(p, new SortedByName());
    for(Product i : p)
    { System.out.println("Name: " + i.getName() +
      " price: " + i.getPrice() + " quantity: " + i.getQuantity());
    }
  }
}
```

Интерфейс Comparator

```
============ no sorted ==============
Name: Milk price: 7.56 quantity: 56
Name: Coffee price: 17.0 quantity: 32
Name: Tea price: 12.5 quantity: 0
========== sorted by price===========
Name: Milk price: 7.56 quantity: 56
Name: Tea price: 12.5 quantity: 0
Name: Coffee price: 17.0 quantity: 32
========== sorted by name ===========
Name: Coffee price: 17.0 quantity: 32
Name: Milk price: 7.56 quantity: 56
Name: Tea price: 12.5 quantity: 0
```

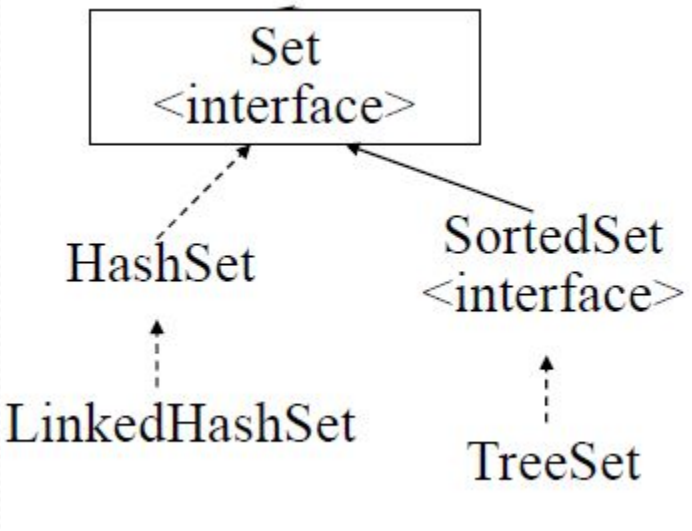
Интерфейс Set

```
public interface Set<E> extends Collection<E>
{ int size();


boolean isEmpty();
boolean contains(Object element);
boolean add(E element);
boolean remove(Object element);
Iterator<E> iterator();
boolean containsAll(Collection<?> c);

boolean addAll(Collection<? extends E> c);
boolean removeAll(Collection<?> c);
void clear();
Object[] toArray();
…
}
```



Интерфейс SortedSet

```
public interface SortedSet<E> extends Set<E>
{ Comparator <? Super E> comparator();
  E first();
  E last();
  SortedSet<E> headSet(E toElement);
  SortedSet<E> subSet(E fromElement,
                      E toElement);
  SortedSet<E> tailSet(E fromElement);

}
```



Интерфейс Set и SortedSet

- **Интерфейс Set** из пакета java.util, расширяющий интерфейс Collection, описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию множества (set) . Новые методы в интерфейс Set не добавлены, просто метод *add()* не станет добавлять еще одну копию элемента, если такой элемент уже есть в множестве.
- **Интерфейс SortedSet** расширяющий интерфейс Set, описывает упорядоченное множество, отсортированное по естественному порядку возрастания его элементов или по порядку, заданному реализацией интерфейса Comparator. Элементы не нумеруются, но есть понятие первого, последнего, большего и меньшего элемента. Доп-е методы интерфейса отражают эти понятия:
  - *Comparator comparator ()* — возвращает способ упорядочения коллекции;
  - *object first ()*— возвращает первый, меньший элемент коллекции;
  - *SortedSet headset (Object toEiement)* — возвращает начальные, меньшие элементы до элемента toElement исключительно;
  - *object last ()* — возвращает последний, больший элемент коллекции;
  - *SortedSet subset(Object fromElement, Object toEiement)* — возвращает подмножество коллекции от элемента fromElement включительно до элемента toElement исключительно;
  - *SortedSet tailSet (Object fromElement)* — возвращает последние, большие элементы коллекции от элементаf romElement включительно.
- Платформа Java содержит три класса реализации интерофейса Set: **HashSe**t, **TreeSet** и **LinkedHashSet.**

Классы HashSet, TreeSet, LinkedHashSet

- **Конструкторы класса HashSet :**
  - `HashSet()`
  - `HashSet(int capacity)`
  - `HashSet(int capacity, float loadFactor)`
  - `HashSet(Collection c)`
- **Конструкторы класса TreeSet :**
  - `TreeSet ()`
  - `TreeSet (Comparator c)`
  - `TreeSet (Collection coll)`
  - `TreeSet (SortedMap sf)`

Классы HashSet, TreeSet, LinkedHashSet

- Класс **HashSet** наследуется от абстрактного суперкласса **AbstractSet** и реализует интерфейс **Set**, используя хэш-таблицу для хранения коллекции. Ключ (хэш-код) используется вместо индекса для доступа к данным, что значительно ускоряет поиск определенного элемента.
- В классе HashSet четыре конструктора:
  - *Hashset ()* — создает пустой объект с показателем загруженности 0,75;
  - *HashSet (int capacity)* — создает пустой объект с начальной емкостью capacity и показателем загруженности 0,75;
  - *HashSet (int capacity, float loadFactor)* — создает пустой объект с начальной емкостью capacity и показателем загруженности loadFactor ;
  - *HashSet (Collection coll)* — создает объект, содержащий все элементы коллекции coll , с емкостью, равной удвоенному числу элементов коллекции coll , но не менее 11, и показателем загруженности 0,75.
- Класс **TreeSet** реализован как бинарное дерево поиска, значит, его элементы хранятся в упорядоченном виде. Это значительно ускоряет поиск нужного элемента. Обработка операций удаления и вставки объектов происходит медленнее, чем в хэш-множествах, но быстрее, чем в списках. Класс TreeSet полностью реализует интерфейс **SortedSet**
- В классе TreeSet четыре конструктора:
  - *TreeSet ()* — создает пустой объект с естественным порядком элементов;
  - *TreeSet (Comparator c)* — создает пустой объект, в котором порядок зад-ся объектом сравнения с;
  - *TreeSet (Collection coll)* — создает объект, содержащий все элементы коллекции coll , с естественным порядком ее элементов;
  - *TreeSet (SortedMap sf)* — создает объект, содержащий все эл-ты отображения sf , в том же порядке.
- TreeSet коллекция которая позволяет хранить объекты в отсортированном виде. Для собственных классов необходимо реализовать интерфейс Comparable или Comparator, в противном случае, при добавлении второго элемента вы получите ошибку: <Class> cannot be cast to java.lang.Comparable

Классы HashSet, TreeSet, LinkedHashSet

- **LinkedHashSet** - упорядоченная версия HashSet, которая обслуживает дважды связанный список и его элементы. Следует использовать данный класс вместо HashSet, когда Вам важен порядок итерации. Когда вы проходите через HashSet - порядок непредсказуем, в то время как LinkedHashSet позволяет Вам проходить через элементы в том порядке, в котором они были вставлены.

Интерфейс Set и классы HashSet, TreeSet, LinkedHashSet

```
import java.util.*;

class SetsSamples {
    private HashSet<Integer> hashSet;
    private TreeSet<Integer> treeSet;
    private LinkedHashSet<Integer> linkedHashSet;
    public SetsSamples(int count){
        hashSet = new HashSet<Integer>();
        treeSet = new TreeSet<Integer>();
        linkedHashSet = new LinkedHashSet<Integer>();
        fillSets(count);
    }
    public void fillSets(int count){
        Random rand = new Random();
        for(int i = 0; i < count; i++){
            Integer element = rand.nextInt(100);
            hashSet.add(element);
            treeSet.add(element);
            linkedHashSet.add(element);
        }
    }
    public void print(){
      System.out.println("HashSet: \t" + hashSet.toString() + " (" +
         hashSet.size() + ")");
      System.out.println("TreeSet: \t" + treeSet.toString() + " (" +
        treeSet.size() + ")");
     System.out.println("LinkedHashSet: \t" + linkedHashSet.toString() +
        " (" + linkedHashSet.size() + ")");
     System.out.println("");
    }
```

Интерфейс Set и классы HashSet, TreeSet, LinkedHashSet

- Не смотря на то, что все три рассматриваемые коллекции реализуют один тот же интерфейс, они отличаются друг от друга. Главное различие — это порядок хранения элементов. HashSet хранит элементы в случайном (на первый взгляд) порядке. Дело в том, что для быстрого поиска HashSet расчитывает для каждлого элемента hashCode и именно по этому ключу ищет и упорядочивает элементы внутри себя. TreeSet, в отличии от HashSet, хранит элементы упорядоченно, то есть в каком бы порядке мы не добавляли и не удаляли элементы, коллекция останется строго упорядоченной. LinkedHashSet используется в том случае, если нам необходимо помнить порядок добавления элементов. Поиск по этой коллекции происходит также по hashCode, но порядок будет всегда совпадать с очерёдностью добавления.
- В конструкторе класса SetsSamples создаются объекты наших коллекции и вызывается метод fillSets, который и добавит count случайных чисел в только что созданные объекты. В этом методе мы используем класс Random (который, кстати, также содержится в пакете java.utils). Этот класс используется для получения последовательностей случайных чисел. В этом примере мы используем только один из его методов — это nextInt. Данный метод каждый раз будет возвращать случайное число в пределах от 0 до переданного параметром значения.
- Для каждой коллекции вызываются методы toString() и size(). Метод toString() возвращает строку, содержащую элементы коллекции, перечисленные через запятую. Список справа и слева ограничивается квадратными скобками «[" и "]» (например [1, 2, 5]). size() возвращает количество элементов в коллекции.
- Метод removeMax удаляет максимальный элемент из коллекции treeSet. Так, как TreeSet хранит элементы упорядоченно, метод last() (возвращает последний элемент коллекции) вернёт нам наибольший элемент. С помощью метода remove мы удалим этот элемент из коллекции.
- У нас есть возможность создавать коллекцию, сразу добавляя в неё элементы из другой. Этим и воспользуемся в методе create для копирования коллекции.
- removeAll удалит все элементы linkedHashSet из hashSet.
- retainAll — наоборот, оставит в hashSet все элементы, которые есть в linkedHashSet

Интерфейс Set и классы HashSet, TreeSet, LinkedHashSet

```
    public void clear(){ hashSet.clear();}
    public void removeMax(){ Integer max = treeSet.last();
                             treeSet.remove(max);}
    public void iterate(){
        Iterator<Integer> it = linkedHashSet.iterator();
        while(it.hasNext()){
            Integer element = it.next();
            if(element % 2 == 0){ it.remove();}
        } }
    public void create(){ hashSet = new HashSet<Integer>(treeSet);}
    public void removeAll(){ hashSet.removeAll(linkedHashSet);}
    public void retainAll(){ hashSet.retainAll(linkedHashSet);}
}
public class Sets {
  public static void main(String args[])
  { int count = 5;
    SetsSamples samples = new SetsSamples(count);
    System.out.println("-- Begin --"); samples.print();
      samples.clear();
    System.out.println("-- Clear --"); samples.print();
      samples.removeMax();
    System.out.println("-- Remove max --"); samples.print();
      samples.iterate();
    System.out.println("-- Iterate --"); samples.print();
      samples.create();
    System.out.println("-- Create --"); samples.print();
      samples.removeAll();
    System.out.println("-- Remove all --"); samples.print();
      samples.create();
    System.out.println("-- Retain all --");
      samples.retainAll();  samples.print();
    } }
```

Интерфейс Set и классы HashSet, TreeSet, LinkedHashSet

```
-- Begin --
HashSet:          [70, 64, 41, 29, 30] (5)
TreeSet:          [29, 30, 41, 64, 70] (5)
LinkedHashSet:    [70, 64, 41, 29, 30] (5)

-- Clear --
HashSet:          [] (0)
TreeSet:          [29, 30, 41, 64, 70] (5)
LinkedHashSet:    [70, 64, 41, 29, 30] (5)

-- Remove max --
HashSet:          [] (0)
TreeSet:          [29, 30, 41, 64] (4)
LinkedHashSet:    [70, 64, 41, 29, 30] (5)

-- Iterate --
HashSet:          [] (0)
TreeSet:          [29, 30, 41, 64] (4)
LinkedHashSet:    [41, 29] (2)

-- Create --
HashSet:          [64, 41, 29, 30] (4)
TreeSet:          [29, 30, 41, 64] (4)
LinkedHashSet:    [41, 29] (2)

-- Remove all --
HashSet:          [64, 30] (2)
TreeSet:          [29, 30, 41, 64] (4)
LinkedHashSet:    [41, 29] (2)

-- Retain all --
HashSet:          [41, 29] (2)
TreeSet:          [29, 30, 41, 64] (4)
LinkedHashSet:    [41, 29] (2)
```

Интерфейс Set и класс TreeSet

- **Пример .**

```
import java.util.Set;
import java.util.TreeSet;

class Book implements Comparable<Book>
{   private String name; // имя
    private int nPages; // количество страниц

    public Book(String bookname, int npages)
    {  this.name=bookname;
       this.nPages=npages; }

    public String toString()
    { return "[Book: "+name + " Pages: " + nPages+"]";}

    public int compareTo(Book  obj)
    {  int otherNPages = ((Book) obj).getNPages();
        if (nPages == otherNPages) return 0;
        else return (nPages < otherNPages) ? 1 : -1;
    }
    public String getBookName()
    { return name;}

    public int getNPages()
    { return nPages;}
}
```

Интерфейс Set и класс TreeSet

- **Пример .**

```
class BookList{
  private Set<Book> al = new TreeSet<Book>();

  public void add(Book obj) {
        al.add(obj);
  }
  public int count() {
        return al.size();
  }

  public void  print()
  { System.out.println(al);}
}

public class Main {
   public static void main(String[] args)
   {   BookList bl = new BookList();
       Book book1 = new Book("C#",345);
       Book book2 = new Book("C++",1232);
       Book book3 = new Book("Java",2);
       bl.add(book1);
       bl.add(book2);
       bl.add(book3);
       bl.print();
   }
}
```

```
[[Book: C++ Pages: 1232], [Book: C# Pages: 345], [Book: Java Pages: 2]]
```