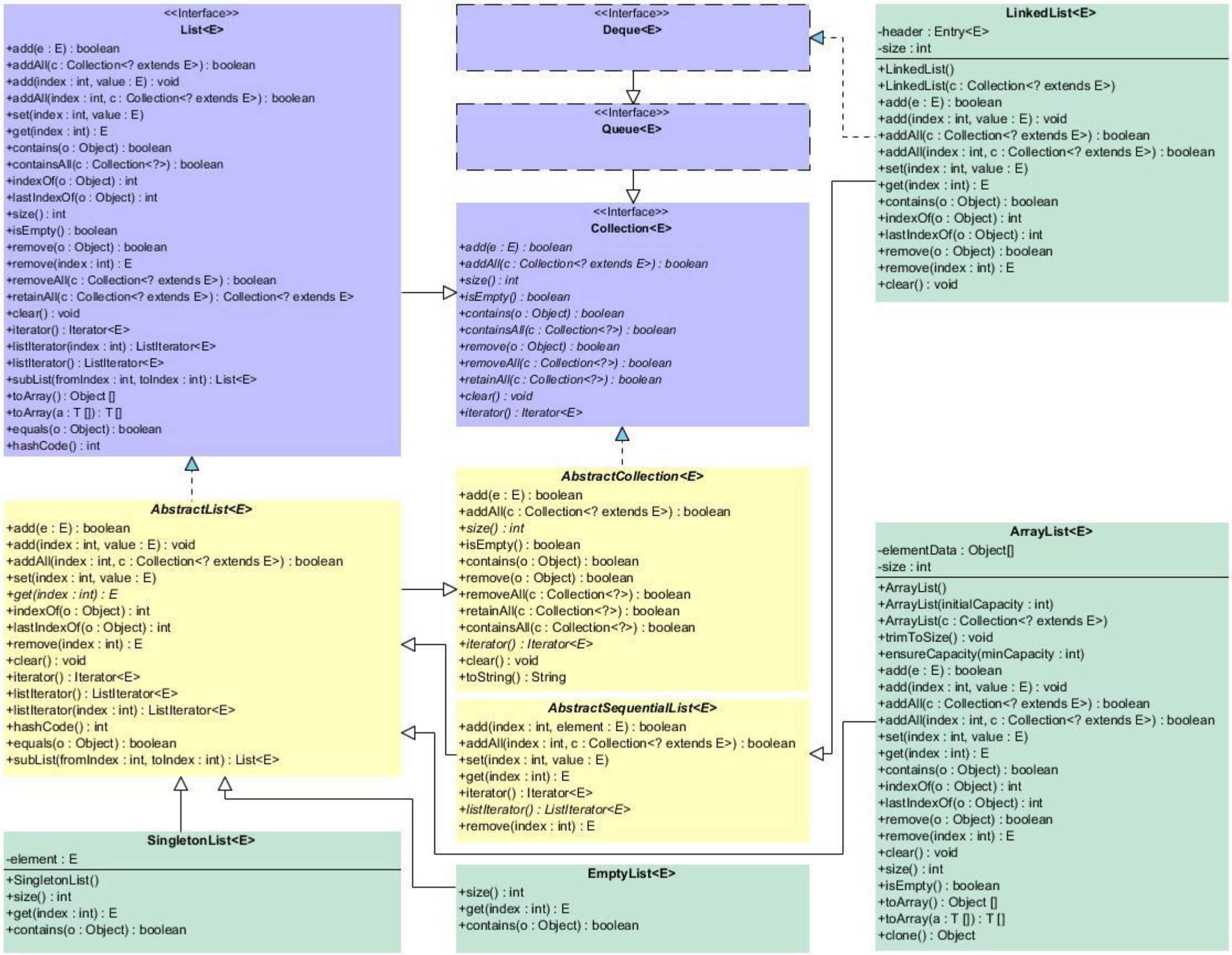

DIASOFT
всё по-настоящему

DIASOFT
всё по-настоящему

Коллекции

5. Списки

Списки


<<Interface>>
List<E>
+add(e : E) : boolean
+addAll(c : Collection<? extends E>) : boolean
+add(index : int, value : E) : void
+addAll(index : int, c : Collection<? extends E>) : boolean
+set(index : int, value : E)
+get(index : int) : E
+contains(o : Object) : boolean
+containsAll(c : Collection<?>) : boolean
+indexOf(o : Object) : int
+lastIndexOf(o : Object) : int
+size() : int
+isEmpty() : boolean
+remove(o : Object) : boolean
+remove(index : int) : E
+removeAll(c : Collection<? extends E>) : boolean
+retainAll(c : Collection<? extends E>) : Collection<? extends E>
+clear() : void
+iterator() : Iterator<E>
+listIterator(index : int) : ListIterator<E>
+listIterator() : ListIterator<E>
+subList(fromIndex : int, toIndex : int) : List<E>
+toArray() : Object []
+toArray(a : T []) : T []
+equals(o : Object) : boolean
+hashCode() : int
<<Interface>>
Deque<E>
<<Interface>>
Queue<E>
<<Interface>>
Collection<E>
+add(e : E) : boolean
+addAll(c : Collection<? extends E>) : boolean
+size() : int
+isEmpty() : boolean
+contains(o : Object) : boolean
+containsAll(c : Collection<?>) : boolean
+remove(o : Object) : boolean
+removeAll(c : Collection<?>) : boolean
+retainAll(c : Collection<?>) : boolean
+clear() : void
+iterator() : Iterator<E>
LinkedList<E>
-header : Entry<E>
-size : int
+LinkedList()
+LinkedList(c : Collection<? extends E>)
+add(e : E) : boolean
+add(index : int, value : E) : void
+addAll(c : Collection<? extends E>) : boolean
+addAll(index : int, c : Collection<? extends E>) : boolean
+set(index : int, value : E)
+get(index : int) : E
+contains(o : Object) : boolean
+indexOf(o : Object) : int
+lastIndexOf(o : Object) : int
+remove(o : Object) : boolean
+remove(index : int) : E
+clear() : void
AbstractList<E>
+add(e : E) : boolean
+add(index : int, value : E) : void
+addAll(index : int, c : Collection<? extends E>) : boolean
+set(index : int, value : E)
+get(index : int) : E
+indexOf(o : Object) : int
+lastIndexOf(o : Object) : int
+remove(index : int) : E
+clear() : void
+iterator() : Iterator<E>
+listIterator() : ListIterator<E>
+listIterator(index : int) : ListIterator<E>
+hashCode() : int
+equals(o : Object) : boolean
+subList(fromIndex : int, toIndex : int) : List<E>
AbstractCollection<E>
+add(e : E) : boolean
+addAll(c : Collection<? extends E>) : boolean
+size() : int
+isEmpty() : boolean
+contains(o : Object) : boolean
+remove(o : Object) : boolean
+removeAll(c : Collection<?>) : boolean
+retainAll(c : Collection<?>) : boolean
+containsAll(c : Collection<?>) : boolean
+iterator() : Iterator<E>
+clear() : void
+toString() : String
AbstractSequentialList<E>
+add(index : int, element : E) : boolean
+addAll(index : int, c : Collection<? extends E>) : boolean
+set(index : int, value : E)
+get(index : int) : E
+iterator() : Iterator<E>
+listIterator() : ListIterator<E>
+remove(index : int) : E
ArrayList<E>
-elementData : Object[]
-size : int
+ArrayList()
+ArrayList(initialCapacity : int)
+ArrayList(c : Collection<? extends E>)
+trimToSize() : void
+ensureCapacity(minCapacity : int)
+add(e : E) : boolean
+add(index : int, value : E) : void
+addAll(c : Collection<? extends E>) : boolean
+addAll(index : int, c : Collection<? extends E>) : boolean
+set(index : int, value : E)
+get(index : int) : E
+contains(o : Object) : boolean
+indexOf(o : Object) : int
+lastIndexOf(o : Object) : int
+remove(o : Object) : boolean
+remove(index : int) : E
+clear() : void
+size() : int
+isEmpty() : boolean
+toArray() : Object []
+toArray(a : T []) : T []
+clone() : Object
SingletonList<E>
-element : E
+SingletonList()
+size() : int
+get(index : int) : E
+contains(o : Object) : boolean
EmptyList<E>
+size() : int
+get(index : int) : E
+contains(o : Object) : boolean

Интерфейс List

```
public interface List<E> extends Collection<E> {

    boolean add(E e);
    void add(int index, E element);
    boolean addAll(Collection<? extends E> c);
    void addAll(int index, Collection<? extends E> c);
    E set(int index, E element);
    E get(int index);
    boolean contains(Object o);
    boolean containsAll(Collection<?> c);
    int indexOf(Object o);
    int lastIndexOf(Object o);
    int size();
    boolean isEmpty();
    boolean remove(Object o);
    E remove(int index);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();
    Iterator<E> iterator();
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);
    List<E> subList(int fromIndex, int toIndex);
    Object[] toArray();
    <T> T[] toArray(T[] a);
    boolean equals(Object o);
    int hashCode();
}
```

Интерфейс List<E> определяет поведение коллекций, которые служат для хранения упорядоченного набора объектов с позиционным доступом. При простом добавлении порядок в котором хранятся элементы определяется порядком их добавления в список. Коллекции реализующие этот интерфейс могут хранить несколько копий одного и того же объекта. Есть возможность доступа к элементам и добавления элементов по индексу. Для получения индекса первого и последнего вхождения объекта в список можно использовать методы indexOf и lastIndexOf соответственно.

Интерфейс RandomAccess

```
public interface RandomAccess {

}
```

Интерфейс RandomAccess – маркерный интерфейс он говорит о том что список использует хранилище данных с произвольным доступом. Таким образом операции получения значения по индексу и изменения значения по индексу требуют постоянное время.

Реализации интерфейса List<E>

Тип коллекции	get(int i)	set(int i, E e)	add	remove	Iterator.remove
ArrayList<E>	O(1)	O(1)	O(1)	O(N)	O(N)
LinkedList<E>	O(N)	O(N)	O(1)	O(N)	O(1)
SingletonList<E>	O(1)	-	-	-	-
EmptyList<E>	-	-	-	-	-

Две часто используемые реализации интерфейса List<E> - ArrayList<E> и LinkedList<E>. Наиболее часто используется ArrayList<E>, поскольку он предоставляет позиционный доступ за постоянное время в то время как LinkedList<E> требует линейного времени. LinkedList<E> выгоднее использовать когда необходимо часто добавлять значения в начало списка или удалять из начала списка. Эти операции требуют постоянного времени в LinkedList<E> и линейного времени в ArrayList<E>.

Класс EmptyList<E>

Класс EmptyList<E>

```
public class Collections {

    public static final List EMPTY_LIST = new EmptyList();

    public static final <T> List<T> emptyList() {
        return (List<T>) EMPTY_LIST;
    }

    private static class EmptyList extends AbstractList<Object> implements RandomAccess, Serializable {

        private static final long serialVersionUID = 8842843931221139166L;

        public int size() {return 0;}

        public boolean contains(Object obj) {return false;}

        public Object get(int index) {
            throw new IndexOutOfBoundsException("Index: "+index);
        }

        // Preserves singleton property
        private Object readResolve() {
            return EMPTY_LIST;
        }
    }
}
```

Класс EmptyList<E> реализует интерфейс List для пустого списка. Для получение ссылки на объект класса EmptyList используется метод emptyList из класса Collections.

Использование EmptyList<E>

```
public class EmptyListDemo {

   public static void main(String[] args) {
      List<String> staff = Collections.emptyList();

      System.out.println("\nList contents: ");

      for (String value : staff) {
         System.out.println("name = " + value);
      }

      System.out.println("\nList size: " + staff.size());
   }
}
```

```
List contents:

List size: 0
```

Сравнение производительности для пустого List<E>

DIASOFT
всё по-настоящему

```
public class EmptyListsCompareDemo {

   private static final int ITERATIONS = 10000000;

   public static void main(String[] args) {

      List<String> theList;

      long now = System.currentTimeMillis();

      for (int i = 0; i < ITERATIONS; i++){
         theList = new ArrayList<String>();
      }

      System.out.println("Time using ArrayList(): " + (System.currentTimeMillis() - now) + " ms");
      now = System.currentTimeMillis();

         for (int i = 0; i < ITERATIONS; i++){
            theList = new ArrayList<String>(0);
         }

      System.out.println("Time using ArrayList(0): " + (System.currentTimeMillis() - now) + " ms");
      now = System.currentTimeMillis();

      for (int i = 0; i < ITERATIONS; i++){
         theList = new LinkedList<String>();
      }

      System.out.println("Time using LinkedList(): " + (System.currentTimeMillis() - now) + " ms");
      now = System.currentTimeMillis();

      for (int i = 0; i < ITERATIONS; i++){
         theList = Collections.emptyList();
      }
      System.out.println("Time using Collections.emptyList(): " + (System.currentTimeMillis() - now) + "
ms");
    }
}
```

Сравнение производительности для пустого List<E>

```
Time using ArrayList(): 312 ms
Time using ArrayList(0): 188 ms
Time using LinkedList(): 1187 ms
Time using Collections.emptyList(): 31 ms
```

Класс SingletonList<E>

Класс SingletonList<E>

```
public class Collections {

    public static <T> List<T> singletonList(T o) {
        return new SingletonList<T>(o);
    }

    private static class SingletonList<E> extends AbstractList<E> implements RandomAccess, Serializable {

        static final long serialVersionUID = 3093736618740652951L;

        private final E element;

        SingletonList(E obj) {element = obj;}

        public int size() {return 1;}

        public boolean contains(Object obj) {return eq(obj, element);}

        public E get(int index) {
            if (index != 0)
                throw new IndexOutOfBoundsException("Index: "+index+", Size: 1");
            return element;
        }
    }
    ...
}
```

Класс SingletonList<E> реализует интерфейс List для списка из одного значения. Для получение ссылки на объект класса SingletonList используется метод singletonList из класса Collections.

Использование SingletonList<E>

```
public class SingletonListDemo {

   public static void main(String[] args) {
      List<String> staff = Collections.singletonList("Harry Hacker");

      System.out.println("\nList contents: ");

      for (String value : staff) {
         System.out.println("name = " + value);
      }

      System.out.println("\nList size: " + staff.size());
   }
}
```

```
List contents:
name = Harry Hacker

List size: 1
```

Сравнение производительности для List<E> с одним э

DIASOFT
всё по-настоящему

```
public class OneValueListsCompareDemo {

   public static void main(String[] args) {

      List<String> theList;

      long now = System.currentTimeMillis();

      for (int i = 0; i < 10000000; i++){
         theList = new ArrayList<String>();
         theList.add("Harry Hacker");
      }

      System.out.println("Time using ArrayList(): " + (System.currentTimeMillis() - now) + " ms");
      now = System.currentTimeMillis();

      for (int i = 0; i < 10000000; i++){
         theList = new ArrayList<String>(1);
         theList.add("Harry Hacker");
      }

      System.out.println("Time using ArrayList(1): " + (System.currentTimeMillis() - now) + " ms");
      now = System.currentTimeMillis();

      for (int i = 0; i < 10000000; i++){
         theList = new LinkedList<String>();
         theList.add("Harry Hacker");
      }

      System.out.println("Time using LinkedList(): " + (System.currentTimeMillis() - now) + " ms");
      now = System.currentTimeMillis();

      for (int i = 0; i < 10000000; i++){
         theList = Collections.singletonList("Harry Hacker");
      }
      System.out.println("Time using SingletonList(): " + (System.currentTimeMillis() - now) + " ms");
   }
}
```

Сравнение производительности для List<E> с одним элементом

```
Time using ArrayList(): 359 ms
Time using ArrayList(1): 266 ms
Time using LinkedList(): 1406 ms
Time using SingletonList(): 78 ms
```

Класс ArrayList<E>

Класс ArrayList<E>

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable {

    private transient Object[] elementData;
    private int size;

    public ArrayList()
    public ArrayList(int initialCapacity)
    public ArrayList(Collection<? extends E> c)

    public void trimToSize()
    public void ensureCapacity(int minCapacity)

    public boolean add(E e)
    public void add(int index, E element)
    public boolean addAll(Collection<? extends E> c)
    public E set(int index, E element)
    public E get(int index)

    public boolean contains(Object o)
    public int indexOf(Object o)
    public int lastIndexOf(Object o)

    public E remove(int index)
    public boolean remove(Object o)
    public void clear()

    public int size()
    public boolean isEmpty()

    public Object[] toArray()
    public E[] toArray(E[])
    public Object clone()
}
```

Класс ArrayList<E> реализует интерфейс List<E> с помощью массива. Ссылка на массив хранится в поле elementData. Размер массива называется ёмкостью ArrayList<E>. Количество используемых элементов массива хранится в поле size. Если размер массива становится недостаточным выделяется новый массив. Ёмкость больше или равна числу хранимых значений.

Ёмкость и размер

Изменение ёмкости

DIASOFT
всё по-настоящему

```
public class ArrayListCapacityDemo {

   private static final int ITERATIONS = 25000000;

   public static void main(String[] args) {

      ArrayList<String> arrayList = new ArrayList<String>();

      long startTime = System.currentTimeMillis();

      arrayList.ensureCapacity(ITERATIONS);

      for (int i = 0; i < ITERATIONS; i++)
         arrayList.add("");

      long endTime = System.currentTimeMillis();
      System.out.println("ArrayList with ensureCapacity() time: " + (endTime - startTime));
      arrayList = new ArrayList<String>();

      startTime = System.currentTimeMillis();

      for (int i = 0; i < ITERATIONS; i++)
         arrayList.add("");

      endTime = System.currentTimeMillis();
      System.out.println("ArrayList without ensureCapacity() time: " + (endTime - startTime));
   }
}
```

```
ArrayList with ensureCapacity() time: 312
ArrayList without ensureCapacity() time: 641
```

Добавление элементов

Добавление элементов

```
public class SimpleListAddDemo {

   public static void main(String[] args) {

      List<String> fruits = new ArrayList<String>();

      fruits.add("apple");
      fruits.add("orange");
      fruits.add("kiwi");
      fruits.add("apple");
      fruits.add("apple");
      fruits.add("mango");
      fruits.add("pear");
      fruits.add("pear");
      fruits.add("apple");
      fruits.add("orange");

      System.out.println("List contents: " + fruits);
   }
}
```

```
List contents: [apple, orange, kiwi, apple, apple, mango, pear, pear, apple, orange]
```

Добавление элементов по индексу

DIASOFT
всё по-настоящему

```
public class ListInsertDemo {

   public static void main(String[] args) {

      List<String> fruits = new ArrayList<String>();

      fruits.add(0,"apple");
      fruits.add(0,"banana");
      fruits.add(0,"kiwi");
      fruits.add(0,"mango");
      fruits.add(0,"orange");
      fruits.add(0,"peach");
      fruits.add(0,"pear");

      System.out.println("List contents: " + fruits);

      fruits.clear();

      fruits.add(0,"apple");
      fruits.add(1,"banana");
      fruits.add(2,"kiwi");
      fruits.add(3,"mango");
      fruits.add(4,"orange");
      fruits.add(5,"peach");
      fruits.add(6,"pear");

      System.out.println("List contents: " + fruits);
   }
}
```

```
List contents: [pear, peach, orange, mango, kiwi, banana, apple]
List contents: [apple, banana, kiwi, mango, orange, peach, pear]
```

Удаление элементов

Удаление элементов

DIASOFT
всё по-настоящему

```
public class ListRemoveDemo {

   public static void main(String[] args) {

      String[] toAdd = { "apple", "cucumber", "carrot", "kiwi", "potato",
            "tomato", "cucumber", "orange", "carrot", "tomato" };
      String[] toRemove = { "carrot", "tomato", "onion", "cucumber", "potato" };

      List<String> produce = new ArrayList<String>();

      Collections.addAll(produce, toAdd);
      // List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("List size: " + produce.size());
      System.out.println("List contents: " + produce + "\n");

      for (String f : toRemove) {

         if (produce.remove(f))
            System.out.println(f + " was removed. List contents: " + produce);
         else
            System.out.println(f + " was not removed");
      }

      System.out.println("\nFinal list size: " + produce.size());
      System.out.println("Final list contents: " + produce + "\n");
   }
}
```

Удаление элементов

```
List size: 10
List contents: [apple, cucumber, carrot, kiwi, potato, tomato, cucumber, orange, carrot, tomato]

carrot was removed. List contents: [apple, cucumber, kiwi, potato, tomato, cucumber, orange, carrot, tomato]
tomato was removed. List contents: [apple, cucumber, kiwi, potato, cucumber, orange, carrot, tomato]
onion was not removed
cucumber was removed. List contents: [apple, kiwi, potato, cucumber, orange, carrot, tomato]
potato was removed. List contents: [apple, kiwi, cucumber, orange, carrot, tomato]

Final list size: 6
Final list contents: [apple, kiwi, cucumber, orange, carrot, tomato]
```

Удаление элементов по индексу

DIASOFT
всё по-настоящему

```
public class ListRemoveAtIndexDemo {

   public static void main(String[] args) {

      String[] toAdd = { "apple", "cucumber", "carrot", "kiwi", "potato",
            "tomato", "cucumber", "orange", "carrot", "tomato" };

      List<String> produce = new ArrayList<String>();

      Collections.addAll(produce, toAdd);
      // List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("List size: " + produce.size());
      System.out.println("List contents: " + produce + "\n");

      for (int i = 0; i < toAdd.length; i++) {

         String removed = produce.remove(0);
         System.out.println(removed + " was removed. List contents: "
               + produce);
      }

      System.out.println("\nFinal list size: " + produce.size());
      System.out.println("Final list contents: " + produce + "\n");
   }
}
```

Удаление элементов по индексу

```
List size: 10
List contents: [apple, cucumber, carrot, kiwi, potato, tomato, cucumber, orange, carrot, tomato]

apple was removed. List contents: [cucumber, carrot, kiwi, potato, tomato, cucumber, orange, carrot, tomato]
cucumber was removed. List contents: [carrot, kiwi, potato, tomato, cucumber, orange, carrot, tomato]
carrot was removed. List contents: [kiwi, potato, tomato, cucumber, orange, carrot, tomato]
kiwi was removed. List contents: [potato, tomato, cucumber, orange, carrot, tomato]
potato was removed. List contents: [tomato, cucumber, orange, carrot, tomato]
tomato was removed. List contents: [cucumber, orange, carrot, tomato]
cucumber was removed. List contents: [orange, carrot, tomato]
orange was removed. List contents: [carrot, tomato]
carrot was removed. List contents: [tomato]
tomato was removed. List contents: []

Final list size: 0
Final list contents: []
```

Позиционный доступ

Позиционный доступ

DIASOFT
всё по-настоящему

```
public class ListGetSetDemo {

   public static void main(String[] args) {

      String[] toAdd = { "apple", "carrot", "kiwi", "potato", "tomato", "pear",
            "cucumber", "orange" };

      List<String> produce = new ArrayList<String>();

      Collections.addAll(produce, toAdd);
      // List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("Set size: " + produce.size());
      System.out.println("Set contents: " + produce + "\n");

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

         String to = produce.get(i).toUpperCase();
         String prev = produce.set(i, to);
         System.out.println("Changing " + prev + " to " + to);
      }
      System.out.println("\nSet size: " + produce.size());
      System.out.println("Set contents: " + produce + "\n");
   }
}
```

Позиционный доступ

```
Set size: 8
Set contents: [apple, carrot, kiwi, potato, tomato, pear, cucumber, orange]

Changing apple to APPLE
Changing carrot to CARROT
Changing kiwi to KIWI
Changing potato to POTATO
Changing tomato to TOMATO
Changing pear to PEAR
Changing cucumber to CUCUMBER
Changing orange to ORANGE

Set size: 8
Set contents: [APPLE, CARROT, KIWI, POTATO, TOMATO, PEAR, CUCUMBER, ORANGE]
```

Поиск

Поиск первого и последнего вхождения

DIASOFT
всё по-настоящему

```
public class ListIndexOfDemo {

   public static void main(String[] args) {

      String[] toAdd = { "apple", "cucumber", "carrot", "kiwi", "potato", "tomato",
            "pear", "cucumber", "orange", "carrot" ,"tomato" };
      String[] toFind = { "carrot", "tomato", "onion", "cucumber", "potato" };

      List<String> produce = new ArrayList<String>();

      Collections.addAll(produce, toAdd);
      //List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("Set size: " + produce.size());
      System.out.println("Set contents: " + produce + "\n");

      for (String f : toFind) {

         int first = produce.indexOf(f);
         int last = produce.lastIndexOf(f);

         if (first != -1)
            System.out.println(f + " is on the list, first index: " + first + ", last index: " + last);
         else
            System.out.println(f + " is not on the list");
      }
   }
}
```

Поиск первого и последнего вхождения

```
Set size: 11
Set contents: [apple, cucumber, carrot, kiwi, potato, tomato, pear, cucumber, orange, carrot, tomato]

carrot is on the list, first index: 2, last index: 9
tomato is on the list, first index: 5, last index: 10
onion is not on the list
cucumber is on the list, first index: 1, last index: 7
potato is on the list, first index: 4, last index: 4
```

Поиск

DIASOFT
всё по-настоящему

```
public class ListContainsDemo {

   public static void main(String[] args) {

      String[] toAdd = { "apple", "carrot", "kiwi", "potato", "tomato",
            "pear", "cucumber", "orange" };
      String[] toFind = { "carrot", "tomato", "onion", "cucumber", "potato" };

      List<String> produce = new ArrayList<String>();

      Collections.addAll(produce, toAdd);
      //List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("Set size: " + produce.size());
      System.out.println("Set contents: " + produce + "\n");

      for (String f : toFind) {
         if (produce.contains(f))
            System.out.println(f + " is on the list");
         else
            System.out.println(f + " is not on the list");
      }
   }
}
```

Поиск

```
Set size: 8
Set contents: [apple, carrot, kiwi, potato, tomato, pear, cucumber, orange]

carrot is on the list
tomato is on the list
onion is not on the list
cucumber is on the list
potato is on the list
```

Класс LinkedList<E>

Класс LinkedList<E>

```
public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>,..., Cloneable, Serializable
{

    private transient Entry<E> header = new Entry<E>(null, null, null);
    private transient int size = 0;

    public LinkedList()
    public LinkedList(Collection<? extends E> c)

    public boolean add(E element)
    public void add(int index, E element)
    boolean addAll(int index, Collection<? extends E> c)

    public E set(int index, E element)
    public E get(int index)

    public boolean contains(Object o)
    public int indexOf(Object o)
    public int lastIndexOf(Object o)

    public E remove(int index)
    public boolean remove(Object o)
    public void clear()

    public  ListIterator<E> listIterator()
    public  ListIterator<E> listIterator(int index)

    List<E> subList(int from, int to)
    private static class Entry<E> {
        ...
    }
}
```

Класс LinkedList реализует интерфейс List с помощью двусвязного списка. Для элементов двусвязного списка используется внутренний класс Entry<E>.

Позиционный доступ

Позиционный доступ

DIASOFT
всё по-настоящему

```
public class ListGetSetDemo {

   public static void main(String[] args) {

      String[] toAdd = { "apple", "carrot", "kiwi", "potato", "tomato", "pear",
            "cucumber", "orange" };

      List<String> produce = new LinkedList<String>();

      Collections.addAll(produce, toAdd);
      // List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("Set size: " + produce.size());
      System.out.println("Set contents: " + produce + "\n");

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

         String to = produce.get(i).toUpperCase();
         String prev = produce.set(i, to);
         System.out.println("Changing " + prev + " to " + to);
      }
      System.out.println("\nSet size: " + produce.size());
      System.out.println("Set contents: " + produce + "\n");
   }
}
```

Позиционный доступ

```
Set size: 8
Set contents: [apple, carrot, kiwi, potato, tomato, pear, cucumber, orange]

Changing apple to APPLE
Changing carrot to CARROT
Changing kiwi to KIWI
Changing potato to POTATO
Changing tomato to TOMATO
Changing pear to PEAR
Changing cucumber to CUCUMBER
Changing orange to ORANGE

Set size: 8
Set contents: [APPLE, CARROT, KIWI, POTATO, TOMATO, PEAR, CUCUMBER, ORANGE]
```

Позиционный доступ в LinkedList лучше не использовать. Он требует затрат O(N).

Сортировка списков

Сортировка

Списки можно сортировать используя метод sort из класса Collections. Для сортировки можно использовать естественный порядок или объект класса реализующего интерфейс Comparator<E>.

Сортировка

```
public class ListSortDemo {

   public static void main(String[] args) {

      String[] toAdd = { "orange", "apple", "carrot", "kiwi", "potato", "banana", "tomato",
            "pear", "cucumber"};

      List<String> produce = new ArrayList<String>();

      Collections.addAll(produce, toAdd);
      //List<String> produce = new ArrayList<String>(Arrays.asList(toAdd));

      System.out.println("Set contents: " + produce + "\n");

      System.out.println("Sorting ...\n");

      Collections.sort(produce);

      System.out.println("Set contents: " + produce + "\n");
   }
}
```

```
Set contents: [orange, apple, carrot, kiwi, potato, banana, tomato, pear, cucumber]

Sorting ...

Set contents: [apple, banana, carrot, cucumber, kiwi, orange, pear, potato, tomato]
```

Класс String реализует интерфейс Comparable. Эта реализация использует лексикографический порядок.

Сортировка с Comparable

DIASOFT
всё по-настоящему

```
public class AnotherListSortDemo {

   public static void main(String[] args) {

      List<Item> items = new ArrayList<Item>();

      items.add(new Item("Toaster", 1234));
      items.add(new Item("Kettle", 4562));
      items.add(new Item("Microwave oven", 9912));
      items.add(new Item("Coffemaker", 2912));
      items.add(new Item("Blender", 1231));

      System.out.println("Set contents: ");

      for(Item item : items)
      System.out.println(item);

      System.out.println("\nSorting ...\n");

      Collections.sort(items);

      System.out.println("Set contents: ");
      for(Item item : items)
         System.out.println(item);
   }
}
```

Сортировка с Comparable

```
Set contents:
[name=Toaster, number=1234]
[name=Kettle, number=4562]
[name=Microwave oven, number=9912]
[name=Coffemaker, number=2912]
[name=Blender, number=1231]

Sorting ...

Set contents:
[name=Blender, number=1231]
[name=Toaster, number=1234]
[name=Coffemaker, number=2912]
[name=Kettle, number=4562]
[name=Microwave oven, number=9912]
```

Сортировка с Comparator

DIASOFT
всё по-настоящему

```
public class ComparatorListSortDemo {

   public static void main(String[] args) {

      List<Item> items = new ArrayList<Item>();

      items.add(new Item("Toaster", 1234));
      items.add(new Item("Kettle", 4562));
      items.add(new Item("Microwave oven", 9912));
      items.add(new Item("Coffemaker", 2912));
      items.add(new Item("Blender", 1231));

      System.out.println("Set contents: ");

      for(Item item : items)
      System.out.println(item);

      System.out.println("\nSorting ...\n");

      Collections.sort(items, new NameComparator());

      System.out.println("Set contents: ");
      for(Item item : items)
         System.out.println(item);
   }
}
```

Сортировка с Comparator

```
Set contents:
[name=Toaster, number=1234]
[name=Kettle, number=4562]
[name=Microwave oven, number=9912]
[name=Coffemaker, number=2912]
[name=Blender, number=1231]

Sorting ...

Set contents:
[name=Blender, number=1231]
[name=Coffemaker, number=2912]
[name=Kettle, number=4562]
[name=Microwave oven, number=9912]
[name=Toaster, number=1234]
```

Сравнение производительности

Сравнение производительности

DIASOFT
всё по-настоящему

```
public class ListsPerformanceDemo {

   private static final int ITERATIONS = 2000000;

   public static void main(String[] args) {

      List<String> linkedList = new LinkedList<String>();

      for (int i = 0; i < ITERATIONS; i++) {
         linkedList.add("" + (i % 10));
      }

      long startTime = System.currentTimeMillis();

      for (int i = 0; i < 1000; i++)
         linkedList.add(0,"");

      long endTime = System.currentTimeMillis();
      System.out.println("LinkedList time: " + (endTime - startTime));

      List<String> arrayList = new ArrayList<String>();

      for (int i = 0; i < ITERATIONS; i++) {
         arrayList.add("" + (i % 10));
      }

      startTime = System.currentTimeMillis();

      for (int i = 0; i < 1000; i++)
         arrayList.add(0,"");

      endTime = System.currentTimeMillis();
      System.out.println("ArrayList time: " + (endTime - startTime));
   }
}
```

Сравнение производительности

```
LinkedList time: 0
ArrayList time: 10484
```

Как правило использование ArrayList быстрее чем LinkedList. В данном примере специально был выбран самый неподходящий вариант использования ArrayList.

Спасибо

Россия, 127018,
Москва, ул. Полковая 3, стр. 14
Тел.: +7(495) 780 7575, 789 9339
Факс: +7(495) 780 7576, 789 9338
info@diasoft.ru, www.diasoft.ru