Лекция 13а

Колекция от структури данни (Част I)

Основни теми

- Дефиниране на съвкупност от структури данни.
- Използване на ***class Arrays*** за работа с масиви.
- Използване на базисна система (***framework***) от библиотечни реализации на структури от данни.
- Използване на алгоритмите (***search***, ***sort*** и ***fill***) , реализирани в базисната система (***framework***) от библиотечни реализации на структури от данни.

Основни теми

- Използване на ***`interface`***- ите в базисната система (***`framework`***) от библиотечни реализации на структури от данни за тяхната полиморфична обработка.
- Използване на итератори за обхождане на съвъкупност от данни.

13a.7 *class* `Stack` в *package* `java.util`

13a.8 *class* `PriorityQueue` и `interface Queue`

13a.9 Видове приложения на интерфейс *Set*

13a.10 Видове приложения на интерфейс *Map*

Задачи

Литература:

Java How to Program, 10 Edition, глава 16

13а.1 Въведение

Масиви (*едномерни и многомерни*)- структури данни с фиксиран размер по време на изпълнение на програмата

Динамични структури данни- размерът им може да расте или да намалява по време на изпълнение на програмата.

Примери

- **Линейни структури**
  - **Свързани списъци**
  - **Стекови структури**
  - **Опашки**
- **Двоични дървета**

13a.1 Въведение

Java collections framework (`package java.util`)

- Съдържа базисни библиотечни реализации на класически структури от данни, интерфейси и съответни алгоритми
- Използва параметри за тип (*generics*)
- Реализации на структурите от данни изучавани дотук
  - Позволява повторно използване на код
  - Позволява повторно използване на готови компоненти
- Съвкупността от базисни компоненти на библиотеката позволява да се използват класическите структури от данни без да се губи време по програмиране на тези компоненти

13a.1 Въведение

Колекцията от структури данни на Java позволява:

- **оптимизирано бързодействие и качество на изпълнение-** ***максимум скорост при минимум използване на памет***

В тази лекция ще се запознаем с всеки тип структури данни на тази колекция и алгоритмите, реализирани в съответната структура и начина за обхождането им посредством итератори

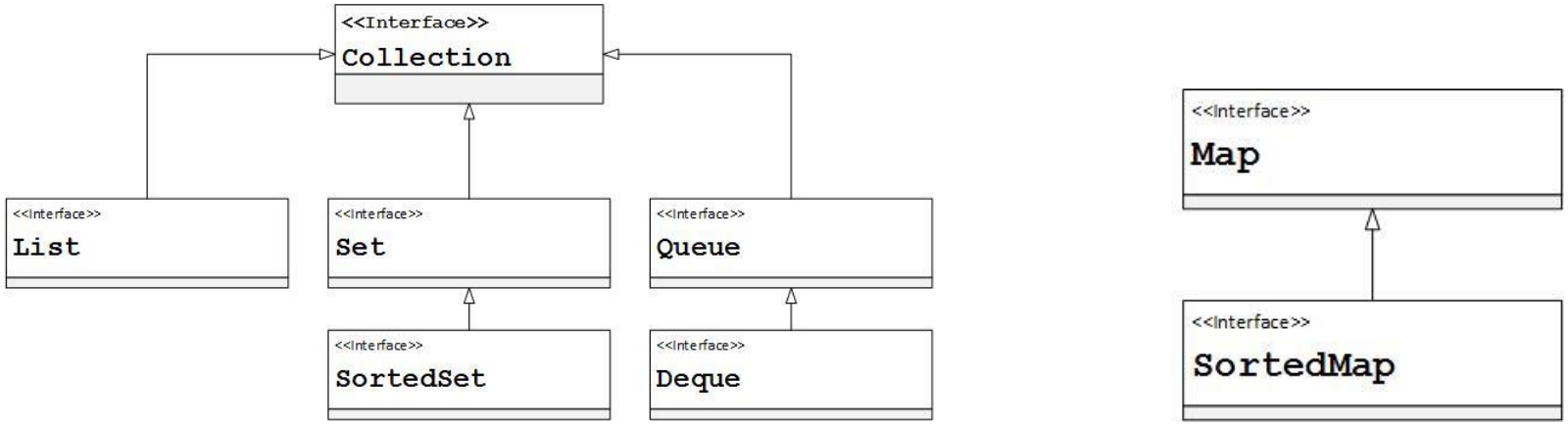
13а.2 Кратко описание на `Collections`

Колекция- ***това е структура от данни***

- ***Обект(контейнер), който може да съхранява референции към други обекти***

Базисна система от колекции (Collections framework)

- **Съдържа интерфейси, деклариращи операции за различните типове колекции**
- **Дават високо качество на изпълнение за класическите структури от данни**
- **Позволяват многократна употреба**
- **Обогатени възможности за реализация с използване на параметри за тип J2SE 5.0 (проверка за тип по време на компилация)**



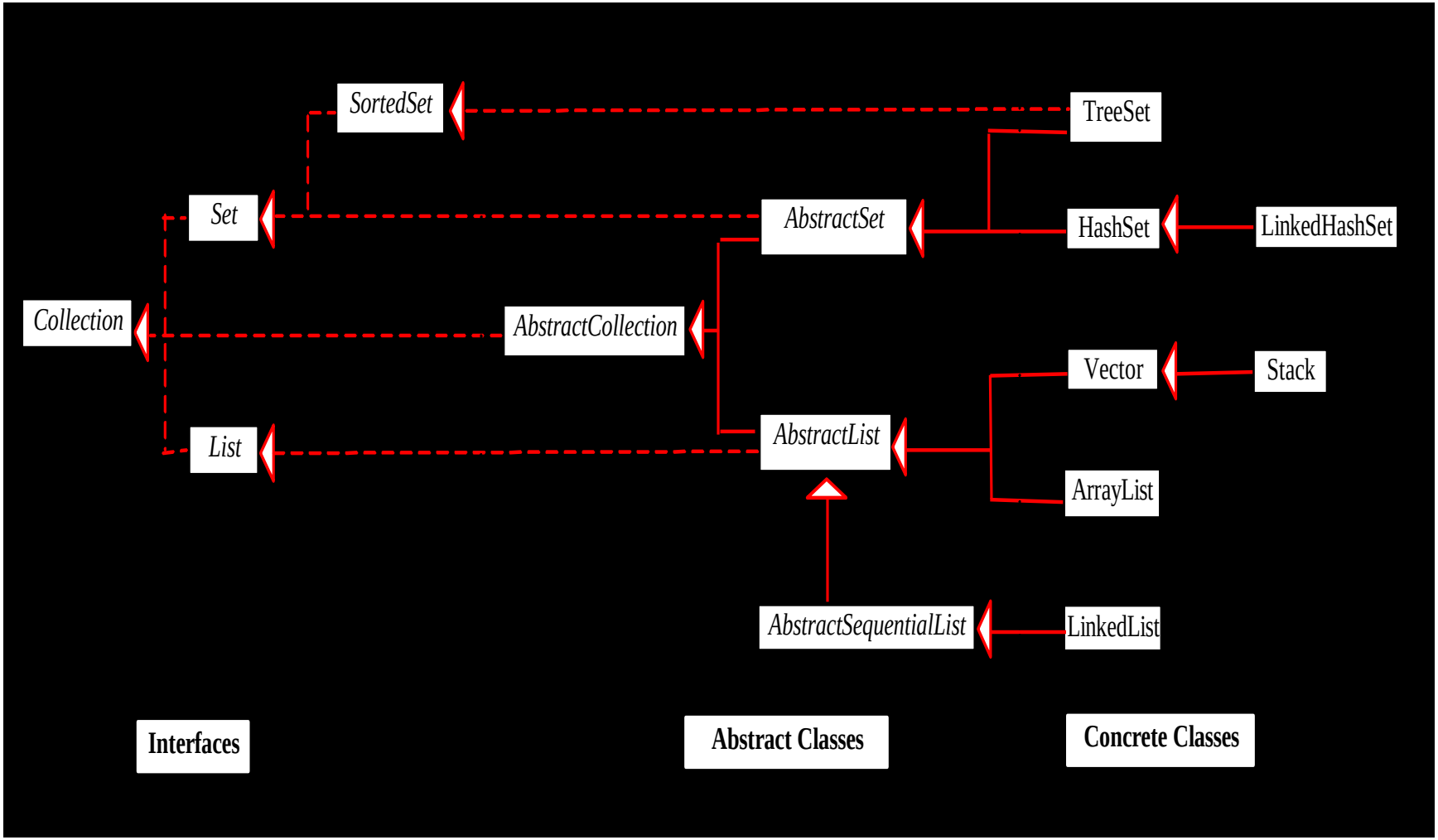
Interface	Description
Collection	The root interface in the collections hierarchy from which interfaces Set, Queue and List are derived.
Set	An unordered collection that does not contain duplicates.
List	An ordered collection that can contain duplicate elements.
Map	Associates keys to values and cannot contain duplicate keys.
Queue	Typically a first-in, first-out collection that models a waiting line; other orders can be specified.
Deque	A linear collection that supports element insertion and removal at both ends.

Примери на:

http://www.java2s.com/Code/Java/Collections-Data-Structure/CatalogCollections-Data-Structure.htm

Fig. 13a.1 | Някои от интерфейсите на базисната библиотека от колекции на Java.

13а.2 Кратко описание на `Collections`



13a.3 `class Arrays`

`class Arrays`

- Доставя `static` методи за работа с масиви
- Доставя методи на "*горно ниво*" (извикват се чрез името на `class Arrays`)
  - Метод `binarySearch` за търсене на сортирани масиви
  - Метод `equals` за сравняване на масиви
  - Метод `fill` за запълване на масиви със стойности
  - Метод `sort` за сортиране на масиви

Тези методи са *презаредени* с версии за работа с *масиви от примитивен тип*, а също и `Object` *масиви.*

Допълнително, методите `sort` и `binarySearch` имат `generic` версии

13a.3 class Arrays

Arrays	
+asList(a: Object[]): List	Returns a list from an array of objects
Overloaded binarySearch method for byte, char, short, int, long, float, double, and Object. +binarySearch(a: xType[], key: xType): int	Overloaded binary search method to search a key in the array of byte, char, short, int, long, float, double, and Object
Overloaded equals method for boolean, byte, char, short, int, long, float, double, and Object. +equals(a: xType[], a2: xType[]): boolean	Overloaded equals method that returns true if a is equal to a2 for a and a2 of the boolean, byte, char, short, int, long, float, and Object type
Overloaded fill method for boolean char, byte, short, int, long, float, double, and Object. +fill(a: xType[], val: xType): void +fill(a: xType[], fromIndex: int, toIndex: xType, val: xType): void	Overloaded fill method to fill in the specified value into the array of the boolean, byte, char, short, int, long, float, and Object type
Overloaded sort method for char, byte, short, int, long, float, double, and Object. +sort(a: xType[]): void +sort(a: xType[], fromIndex: int, toIndex: int): void	Overloaded sort method to sort the specified array of the char, byte, short, int, long, float, double, and Object type

```java
// Fig. 19.2: UsingArrays.java
// Using Java arrays.
import java.util.Arrays;

public class UsingArrays
{
   private int intArray[] = { 1, 2, 3, 4, 5, 6 };
   private double doubleArray[] = { 8.4, 9.3, 0.2, 7.9, 3.4 };
   private int filledIntArray[], intArrayCopy[];

   // constructor initializes arrays
   public UsingArrays()
   {
      filledIntArray = new int[ 10 ]; // create int array with 10 elements
      intArrayCopy = new int[ intArray.length ];

      Arrays.fill( filledIntArray, 7 ); // fill with 7s
      Arrays.sort( doubleArray ); // sort doubleArray ascending

      // copy array intArray into array intArrayCopy
      System.arraycopy( intArray, 0, intArrayCopy,
         0, intArray.length );
   } // end UsingArrays constructor

```

Използва **static** метод **fill** на **class Arrays** за **запълване** на масива със 7- ци

Използва **static** метод **sort** на **class Arrays** за **сортиране** на масив във възходящ ред

Използва **static** метод **arraycopy** на **class System** за **копиране** на масив **intArray** в масива **intArrayCopy**

```
25    // output values in each array
26    public void printArrays()
27    {
28       System.out.print( "doubleArray: " );
29       for ( double doubleValue : doubleArray )
30          System.out.printf( "%.1f ", doubleValue );
31
32       System.out.print( "\nintArray: " );
33       for ( int intValue : intArray )
34          System.out.printf( "%d ", intValue );
35
36       System.out.print( "\nfilledIntArray: " );
37       for ( int intValue : filledIntArray )
38          System.out.printf( "%d ", intValue );
39
40       System.out.print( "\nintArrayCopy: " );
41       for ( int intValue : intArrayCopy )
42          System.out.printf( "%d ", intValue );
43
44       System.out.println( "\n" );
45    } // end method printArrays
46
47    // find value in array intArray
48    public int searchForInt( int value )
49    {
50       return Arrays.binarySearch( intArray, value );
51    } // end method searchForInt
52
```

Използва **`static`** метод **`binarySearch`** на **`class Arrays`** за бинарно търсене на масив

```
53    // compare array contents
54    public void printEquality()
55    {
56       boolean b = Arrays.equals( intArray, intArrayCopy );
57       System.out.printf( "intArray %s intArrayCopy\n",
58          ( b ? "==" : "!=" ) );
59
60       b = Arrays.equals( intArray, filledIntArray );
61       System.out.printf( "intArray %s filledIntArray\n",
62          ( b ? "==" : "!=" ) );
63    } // end method printEquality
64
65    public static void main( String args[] )
66    {
67       UsingArrays usingArrays = new UsingArrays();
68
69       usingArrays.printArrays();
70       usingArrays.printEquality();
71
```

Използва **`static`** метод **`equals`** на **`class Arrays`** за определяне дали стойностите в двата масива са равни

```
72       int location = usingArrays.searchForInt( 5 );
73       if ( location >= 0 )
74          System.out.printf(
75             "Found 5 at element %d in intArray\n", location );
76       else
77          System.out.println( "5 not found in intArray" );
78
79       location = usingArrays.searchForInt( 8763 );
80       if ( location >= 0 )
81          System.out.printf(
82             "Found 8763 at element %d in intArray\n", location );
83       else
84          System.out.println( "8763 not found in intArray" );
85    } // end main
86 } // end class UsingArrays
```

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
intArray: 1 2 3 4 5 6
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

13a.3 `class Arrays`

Ред 17 извиква `static` метода `fill` на `class Arrays` за запълване на `10` елемента от `filledIntArray` със седмици.
Презаредени версии на `fill` позволяват масив *да се запълни зададен интервал от елементи* с една и съща стойност.

Ред 18 сортира елементите на масив `doubleArray`. `static` методът `sort` на `class Arrays` подрежда елементите във възходящ ред по подразбиране. Има презаредени версии за *сортиране само на зададено интервал* от елементи на масива

13a.3 `class Arrays`

Редове 21- 22 копират масив `intArray` в масива `intArrayCopy`. *Първият аргумент* (`intArray`) в метода `arraycopy` на клас `System` е масива от който ще копираме. *Вторият аргумент* (`0`) е началният индекс на интервала от елементи за копиране от масива. *Третият агрумент* (`intArrayCopy`) задава масива в който ще копираме елементите. *Четвъртият аргумент* (`0`) задава индекс в `intArrayCopy` в който ще попадне първият копиран елемент от `intArray`. *Последният аргумент* задава колко елемента да се копират.

13a.3 `class Arrays`

```
public static void arraycopy(Object src,
                int src_position, Object dst,
                int dst_position, int length)
```

Изключение `IndexOutOfBoundsException` се хвърля, когато :

- **Аргументът `srcOffset` е отрицателен .**
- **Аргументът `dstOffset` е отрицателен .**
- **Аргументът `length` е отрицателен .**
- **`srcOffset+length` е по- голям от `src.length`.**
- **`dstOffset+length` е по- голям от `dst.length`**

13a.3 `class Arrays`

Ред 50 извиква `static` метода `binarySearch` от `class Arrays` за бинарно търсене в сортирания масив `intArray`, използвайки `value` като ключ. Ако `value` бъде открит, `binarySearch` връща индекса на елемента на съвпадение в `intArray`; в противен случай, `binarySearch` връща отрицателна стойност.

Отрицателната стойност определя мястото в масива `intArray`, където ключът за търсене би трябвало да се вмъкне, ако извършвахме операцията “`вмъкване`”.

13a.3 `class Arrays`

При определяне на точката за вмъкване `binarySearch` променя знака на този индекс на отрицателен и изважда `1` за да получи стойността за връщане.

На пример, на `Fig. 13a.2`, точката за вмъкване на `value=8763` е елемента с индекс `6` в масива. Методът `binarySearch` сменя индекса на `-6`, `после изважда 1` от този индекс и връща стойност `-7`.

13a.3 `class Arrays`

Редове 56 и 60 извикват `static` метода `equals` на `class Arrays` за определяне дали елементите на двата масива са еквивалентни.

Ако масивите имат същите елементи и в същия порядък, методът връща *`true`*; в противен случай- връща `false`.

Равенството на всеки елемент се проверява метода `equals` на `Object`. Много класове предефинират метода `equals` за извършване на сравнения в съответствия със спецификата на класа.

13a.3 `class Arrays`

Например, **`class String`** дефинира **`equals`** за посимволно сравнение на два **`String`** обекта.

Ако **`equals`** не е предефиниран, се използва версията на метода **`equals`** в **`class Object`**

Обичайна грешка при програмиране 13a.1

Предаване на несортиран масив като аргумент в метод `binarySearch` води до логическа грешка— връщаната стойност е неопределена.

13a.4 interface Collection и class Collections

interface Collection

- Базов интерфейс в йерархията от колекции
- Interface- и `Set`, `Queue`, `List` са производни на `interface Collection`
  - `Set` – колекция за представяне на данни без дублиране
  - `Queue` – колекция за представяне на опашки
  - `List` – наредена колекция евентуално с дублирани данни
- Съдържа *bulk operations* (*операции върху цялата колекция*)
  - добавяне, изтриване, сравняване и запазване на обекти
- Доставя метод, връщащ обект `Iterator`
  - Служи за обхождане на колекция и премахване на елементи от колекция

13a.4 interface Collection и class Collections

«interface» java.util.Collection<E>	
+add(o: E): boolean	Adds a new element o to this collection.
+addAll(c: Collection<? extends E): boolean	Adds all the elements in the collection c to this collection.
+clear(): void	Removes all the elements from this collection.
+contains(o: Object): boolean	Returns true if this collection contains the element o.
+containsAll(c: Collection<?>):boolean	Returns true if this collection contains all the elements in c.
+equals(o: Object): boolean	Returns true if this collection is equal to another collection o.
+hashCode(): int	Returns the hash code for this collection.
+isEmpty(): boolean	Returns true if this collection contains no elements.
+iterator(): Iterator	Returns an iterator for the elements in this collection.
+remove(o: Object): boolean	Removes the element o from this collection.
+removeAll(c: Collection<?>): boolean	Removes all the elements in c from this collection.
+retainAll(c: Collection<?>): boolean	Retains the elements that are both in c and in this collection.
+size(): int	Returns the number of elements in this collection.
+toArray(): Object[]	Returns an array of Object for the elements in this collection.

«interface» java.util.Iterator<E>	
+hasNext(): boolean	Returns true if this iterator has more elements to traverse.
+next(): E	Returns the next element from this iterator.
+remove(): void	Removes the last element obtained using the next method.

Software Engineering факти 13a.1

`Collection` обикновено се използва като параметър за тип на метод, позволяващ полиморфична обработка на всички обекти които имплементират `interface Collection`.

Software Engineering факти 13а.2

Повечето реализации на колекции доставят конструктор, който взима за аргумент референция към `Collection` обект , позволяващ да се създаде колекция съдържаща елементите на зададената като аргумент колекция.

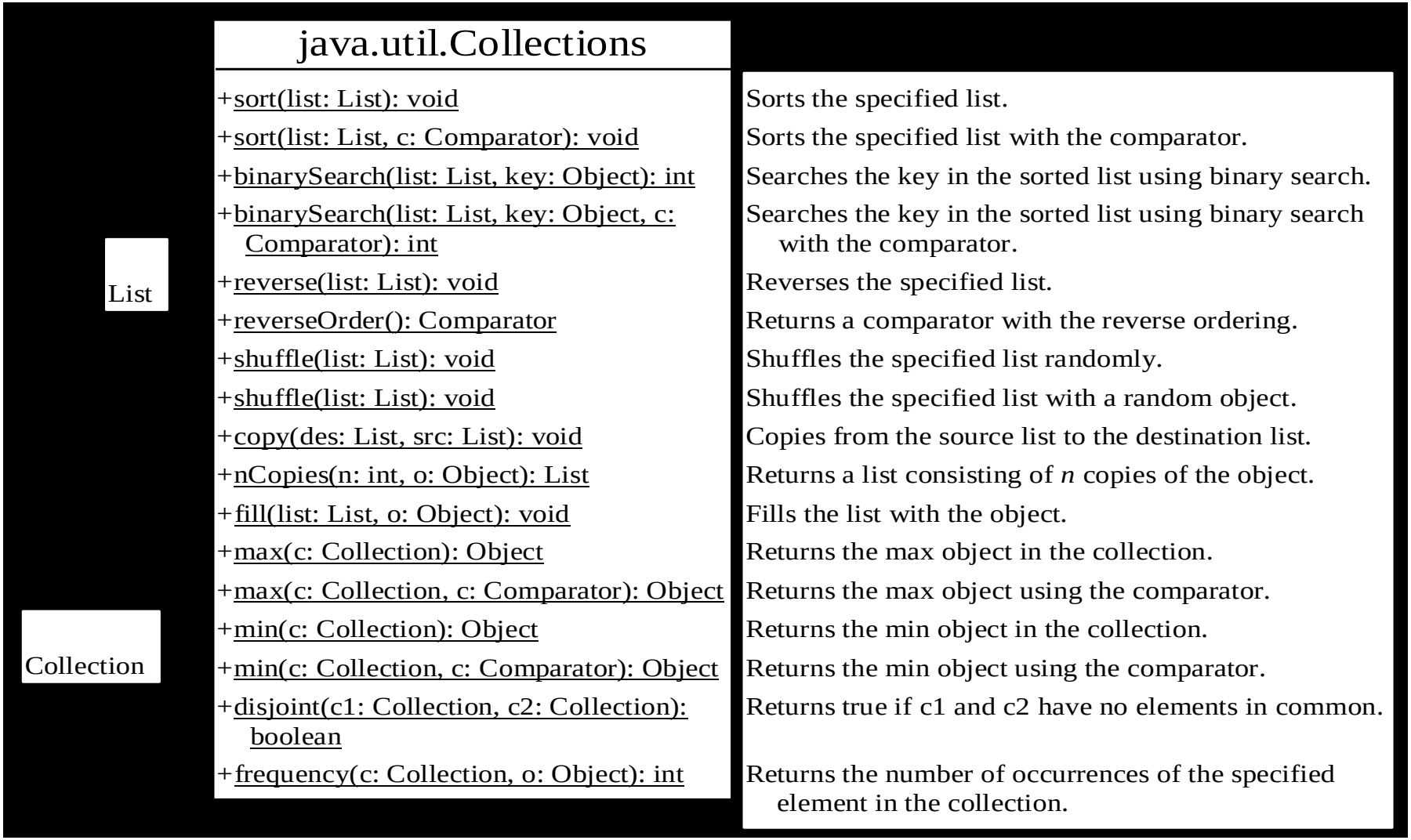
13a.4 `interface Collection` и `class Collections`

`class Collections`

- **Предоставя `static` методи за обработка на колекции**
  - **Имплементира алгоритми за търсене, сортиране и пр.**
- **Позволява полиморфична обработка на колекции**

Използват се "*опаковащи*" (`wrapper`) методи за гарантиране на неизменяемост (`Unmodifiable collection`) на елементите на колекцията (*потребителят може само да разглежда елементите на колекцията, но не и да премахва или добавя нови елементи*)

13a.4 interface Collection и class Collections

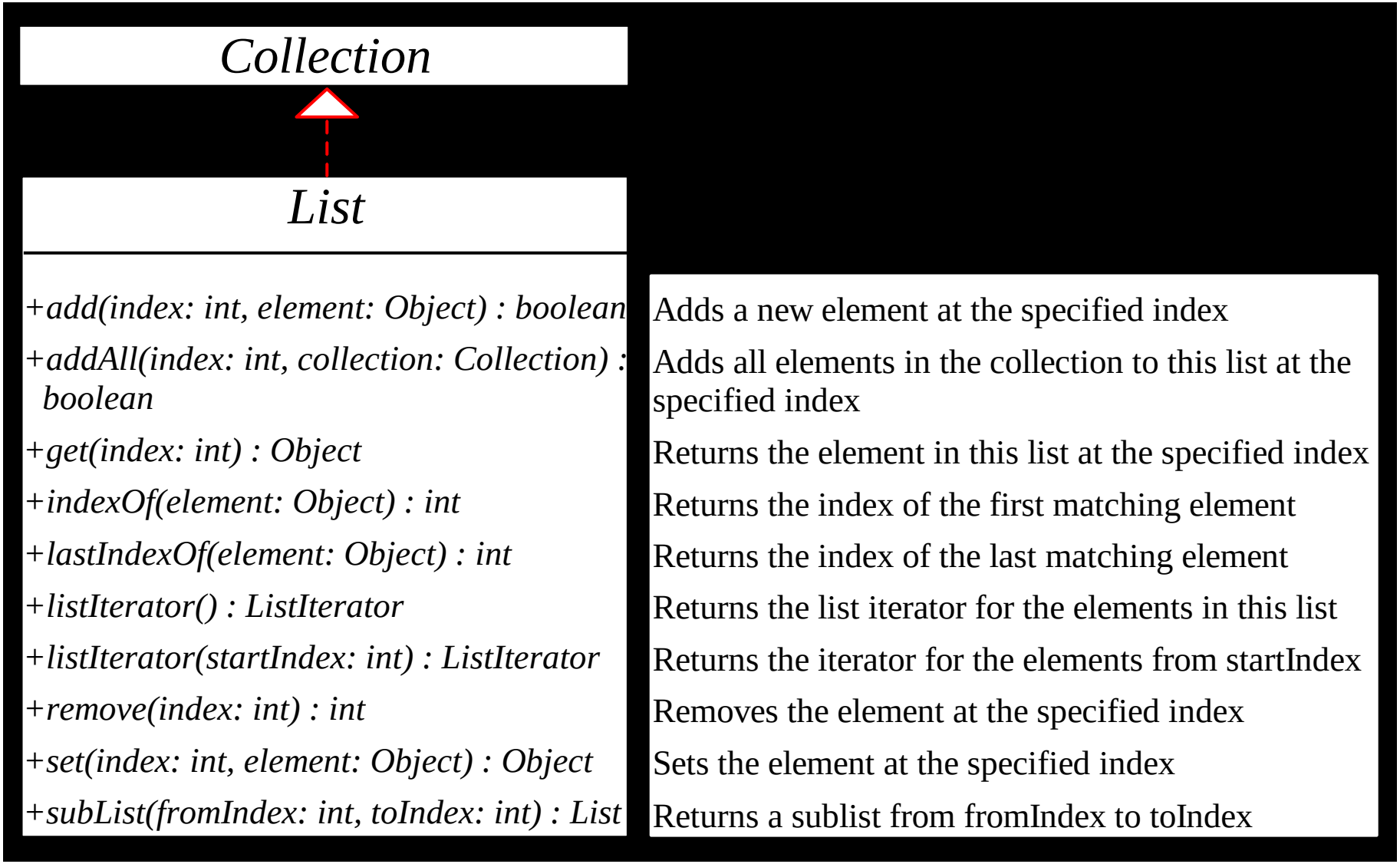


13а.5 Списъци

`interface List` е `Collection` (имплементира)

- **Наредена `Collection` от данни с евентуално дублирани стойности**
- **Понякога се нарича още *редица* (*sequence*)**
- **Както при масивите, индексите на `List` i започват от нула(т.е. Първият елемент има индекс нула). В допълнение на методите наследени от `Collection`, `List` доставя методи за обработка на елементи по индексите им, обработка на елементи в даден интервал от индекси, търсене на елементи и получаване на референция до `ListIterator` обект за обхождане на елементите на колекцията**

13a.5 Списъци



13а.5 Списъци

`interface List` се имплементира от няколко **класа**:

- `ArrayList`
- `LinkedList`
- `Vector`

Използват се ***автоматично преобразуване*** до `обвиващ` клас при добавяне на **примитивни данни**, понеже ***тези колекции съхраняват единствено референции към обекти***.

13а.5 Списъци

Класове `ArrayList` и `Vector` са реализирани на `List` като *масиви с променлива дължина*.

Клас `LinkedList` е реализация на interface `List` чрез *свързан списък*.

13а.5 Списъци

Класове `ArrayList` и `Vector` са реализирани на `List` като *масиви с променлива дължина*.

Клас `LinkedList` е реализация на interface `List` чрез *свързан списък*.

Съвет за по- добро качество 13а.1

Колекцията **`ArrayList`** се държи като колекцията **`Vector`** без да включва средства за синхронизация и затова е по-бърза при изпълнение от колекцията **`Vector`** .

Software Engineering факти 13а.3

Колекцията `LinkedList` може да се използва за реализация на `стекове`, `опашки`, `дървета` и `d- опашки` (“`deques` “двойно свързани опашки).

`Базовата библиотека от колекции` доставя реализации на някои от тези структури данни.

13a.5 Списъци

Примери

- **Fig. 1 демонстрира премахване на елементи от `ArrayList` с използване на `Iterator`.**
- **Fig. 2 и 3 демонстрират `ListIterator` и някои спесифични методи за `List` и `LinkedList`**
- **Fig. 4 демонстрира `List` методи и някои спесифични за клас `Vector` методи.**

13a.5.1 ArrayList и Iterator

ArrayList пример

Програмата въвежда два масива с имена на цветове в ArrayList колекции и използва Iterator да изтрие елементите от първата ArrayList колекция, които са дублират елементи във втората ArrayList колекция

- **Демонстрира възможностите на *interface Collection***
- **Въвежда *String* масиви в *ArrayList* колекции**
- **Използва *Iterator* за изтриване на елементи от *ArrayList* колекция**

13a.5.1 ArrayList и Iterator

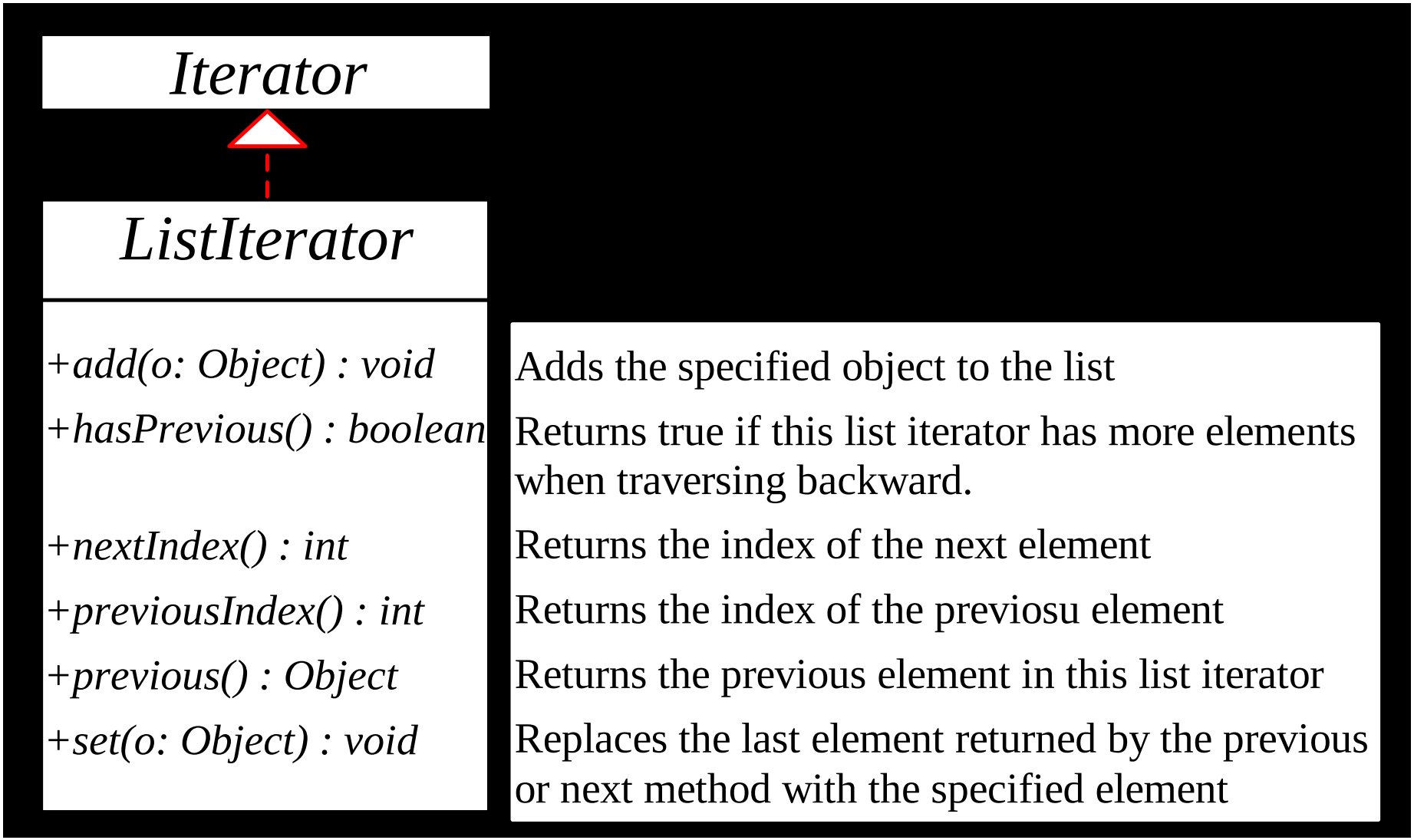
ArrayList

- **Подобна на масив структура данни, която може динамично да променя дължината си**
- **Има капацитет(*capacity*)- *когато броят елементи надхвърли капацитетът*, `Vector` обектът автоматично увеличава капацитета си**

  newCapacity = (oldCapacity * 3)/2 + 1

- **Нарастването на капацитета става на порции равни на текущия *капацитет. Начален капацитет се задава в конструктора на класа, по подразбиране е 16 елемента***
- **Ако не е зададено *нарастване на капацитет*, то по подразбиране дължината на `Vector` се удвоява (150% + 1) всеки път когато това е необходимо.**

13a.5.1 ArrayList и Iterator



```
1  // Fig. 1: CollectionTest.java
2  // Using the Collection interface.
3  import java.util.List;
4  import java.util.ArrayList;
5  import java.util.Collection;
6  import java.util.Iterator;
7
8  public class CollectionTest
9  {
10    private static final String[] colors =
11       { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
12    private static final String[] removeColors =
13       { "RED", "WHITE", "BLUE" };
14
15    // create ArrayList, add Colors to it and manipulate it
16    public CollectionTest()
17    {
18       List< String > list = new ArrayList< String >();
19       List< String > removeList = new ArrayList< String >();
20
```

Създава **`ArrayList`** обекти и присвоява референции към тях съответно на променливи **`list`** и **`removeList`**

```
21      // add elements in colors array to list
22      for ( String color : colors )
23         list.add( color );
24
25      // add elements in removeColors to removeList
26      for ( String color : removeColors )
27         removeList.add( color );
28
29      System.out.println( "ArrayList: " );
30
31      // output list contents
32      for ( int count = 0; count < list.size(); count++ )
33         System.out.printf( "%s ", list.get( count ) );
34
35      // remove colors contained in removeList
36      removeColors( list, removeList );
37
38      System.out.println( "\n\nArrayList after calling removeColors: " );
39
40      // output list contents
41      for ( String color : list )
42         System.out.printf( "%s ", color );
43   } // end CollectionTest constructor
44
```

Използва метод **add** на **List** за добавяне на елемент към **list** и **removeList**, съответно

Използва метод **size** на **List** за получаване на елементите на **ArrayList**

Използва метод **get** на **List** за четене на отделни елементи

Метод **removeColors** приема два аргумента от тип **Collection**; **Ред 36** предава като аргументи два **List** обекта, които се преобразуват до **Collection**

```
45    // remove colors specified in collection2 from collection1
46    private void removeColors(
47       Collection< String > collection1, Collection< String > collection2 )
48    {
49       // get iterator
50       Iterator< String > iterator = collection1.iterator();
51
52       // loop while collection has items
53       while ( iterator.hasNext() )
54
55          if ( collection2.contains( iterator.next() ) )
56             iterator.remove(); // remove current Color
57    } // end method removeColors
58
59    public static void main( String args[] )
60    {
61       new CollectionTest();
62    } // end main
63 } // end class CollectionTest
```

```
ArrayList:
MAGENTA RED WHITE BLUE CYAN

ArrayList after calling removeColors:
MAGENTA CYAN
```

Метод **removeColors** работи с **Collection** от **String аргументи**

Получава **итератор** за **Collection** обект

Iterator метода **hasNext** определя дали **Iterator** има **още елементи** за обхождане

Iterator метода **next** връща **референция към следващия** елемент в колекцията

Използва метода **remove** на **Iterator** за **изтриване** на **String** от **collection1**

Методът **contains** на **Collection** определя дали **collection2 съдържа** елемента върнат с **next**

Обичайна грешка при програмиране 13а.2

Ако една колекция се промени посредством някой от методите й, след като е създаден итератор за тази колекция, то този итератор веднага става невалиден и при всяка операция, изпълнявана от итератора се хвърля `ConcurrentModificationExceptions`.

Пример ConcurrentModificationException

```
1  import java.util.ArrayList;
2  import java.util.Iterator;
3
4  public class FailFastIteratorExample
5  {
6      public static void main(String[] args)
7      {
8          //Creating an ArrayList of integers
9
10         ArrayList<Integer> list = new ArrayList<Integer>();
11
12         //Adding elements to list
13
14         list.add(1452);
15
16         list.add(6854);
17
18         list.add(8741);
19
20         list.add(6542);
21
22         list.add(3845);
23
24         //Getting an Iterator from list
25
26         Iterator<Integer> it = list.iterator();
27
28         while (it.hasNext())
29         {
30             Integer integer = (Integer) it.next();
31
32             list.add(8457);      //This will throw ConcurrentModificationException
33         }
34     }
35 }
```


ListIterator<E>

Modifier and Type	Method and Description
void	add(E e) Inserts the specified element into the list (optional operation).
boolean	hasNext() Returns true if this list iterator has more elements when traversing the list in the forward direction.
boolean	hasPrevious() Returns true if this list iterator has more elements when traversing the list in the reverse direction.
E	next() Returns the next element in the list and advances the cursor position.
int	nextIndex() Returns the index of the element that would be returned by a subsequent call to next().
E	previous() Returns the previous element in the list and moves the cursor position backwards.
int	previousIndex() Returns the index of the element that would be returned by a subsequent call to previous().
void	remove() Removes from the list the last element that was returned by next() or previous() (optional operation).
void	set(E e) Replaces the last element returned by next() or previous() with the specified element (optional operation).

Iterator<E>

Modifier and Type	Method and Description
`boolean`	`hasNext()` Returns `true` if the iteration has more elements.
`E`	`next()` Returns the next element in the iteration.
`void`	`remove()` Removes from the underlying collection the last element returned by this iterator (optional operation).

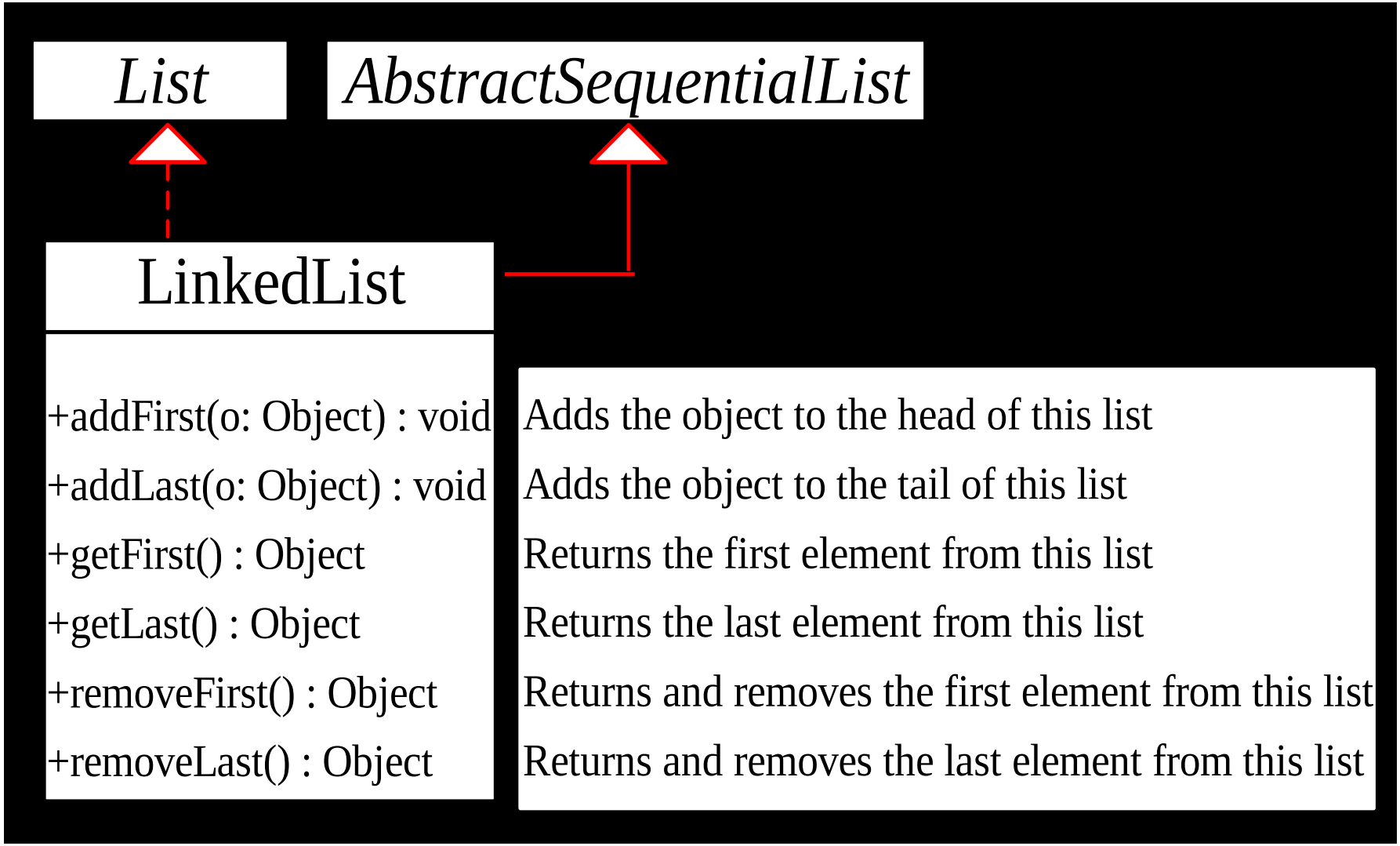
13a.5.2 LinkedList

`LinkedList` пример

Програмата създава два `LinkedList`-а които съдържат `String`- ове. Елементите на единия от `List` – овете се добавя към другия. Накрая, всички `String`- ове се преобразуват до главни букви, и се изтрива дадено подмножество от елементи.

- **Добавяне на елементи от един `List` към друг**
- **Преобразува `String`-ове в главни букви uppercase**
- **Изтриване на интервал от елементи на списъка**

13a.5.2 LinkedList



```
1  // Fig. 2: ListTest.java
2  // Using LinkLists.
3  import java.util.List;
4  import java.util.LinkedList;
5  import java.util.ListIterator;
6
7  public class ListTest
8  {
9     private static final String colors[] = { "black", "yellow",
10       "green", "blue", "violet", "silver" };
11    private static final String colors2[] = { "gold", "white",
12       "brown", "blue", "gray", "silver" };
13
14    // set up and manipulate LinkedList objects
15    public ListTest()
16    {
17       List< String > list1 = new LinkedList< String >();
18       List< String > list2 = new LinkedList< String >();
19
20       // add elements to list link
21       for ( String color : colors )
22          list1.add( color );
23
```

Създава два **`LinkedList`** обекта

Използва метода **`add`** на **`List`** за **добавяне** на елементи от масив **`colors`** в края на **`list1`**

```
24       // add elements to list link2
25       for ( String color : colors2 )
26          list2.add( color );
27
28       list1.addAll( list2 ); // concatenate lists
29       list2 = null; // release resources
30       printList( list1 ); // print list1 elements
31
32       convertToUppercaseStrings( list1 ); // convert to upper case string
33       printList( list1 ); // print list1 elements
34
35       System.out.print( "\nDeleting elements 4 to 6..." );
36       removeItems( list1, 4, 7 ); // remove items 4-7 from list
37       printList( list1 ); // print list1 elements
38       printReversedList( list1 ); // print list in reverse order
39    } // end ListTest constructor
40
41    // output List contents
42    public void printList( List< String > list )
43    {
44       System.out.println( "\nlist: " );
45
46       for ( String color : list )
47          System.out.printf( "%s ", color );
48
49       System.out.println();
50    } // end method printList
51
```

Използва метод **`add`** на **`List`** за добавяне на елементи от масив **`colors2`** към края на **`list2`**

Използва метод **`addAll`** на **`List`** за добавяне на **всички** елементи от **`list2`** към края на **`list1`**

Метод **`printList`** позволява **всеки `List` от `String`** да се разпечата на стандартен изход

```
52    // locate String objects and convert to uppercase
53    private void convertToUppercaseStrings( List< String > list )
54    {
55       ListIterator< String > iterator = list.listIterator();
56
57       while ( iterator.hasNext() )
58       {
59          String color = iterator.next();  // get item
60          iterator.set( color.toUpperCase() ); // convert to
61       } // end while
62    } // end method convertToUppercaseStrings
63
64    // obtain sublist and use clear method to delete sublist items
65    private void removeItems( List< String > list, int start, int end )
66    {
67       list.subList( start, end ).clear();  // remove items
68    } // end method removeItems
69
70    // print reversed list
71    private void printReversedList( List< String > list )
72    {
73       ListIterator< String > iterator = list.listIterator( list.size() );
74
```

`convertToUppercaseStrings` позволява `List` от `String` като аргументи

Вика метод `listIterator` на `List` за получаване на двойно свързан итератор за List

Използва `метод next` за намиране на следващия `String` в `List`

Вика метод `set` на `ListIterator` за заместване на текущия `String` указван с `iterator`със `String` върнат от `toUpperCase`

Метод `removeItems` позволява произволен `List` съдържащ низове да се предава като аргумент на метода

`printReversedList` има аргумент `List` от string- ове

Вика `subList` на `List` за извличане на подмножество от `List`

Вика метода `listIterator` на `List` с един аргумент, задаващ начало за обхождане на `List`

```
75       System.out.println( "\nReversed List:" );
76
77       // print list in reverse order
78       while ( iterator.hasPrevious() )
79          System.out.printf( "%s ", iterator.previous() );
80    } // end method printReversedList
81
82    public static void main( String args[] )
83    {
84       new ListTest();
85    } // end main
86 } // end class ListTest
```

`while` условието вика метод **`hasPrevious`** за определяне дали има още елементи при обратното обхождане н асписъка

Вика метод **`previous`** на **`ListIterator`** за намиране на предишния елемент при обратното обхождане на списъка

```
list:
black yellow green blue violet silver gold white brown blue gray silver

list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY SILVER

Deleting elements 4 to 6...
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER

Reversed List:
SILVER GRAY BLUE BROWN WHITE BLUE GREEN YELLOW BLACK
```

13a.5.2 LinkedList

Пример за `static` метода `asList` на class `Arrays`

- Позволява масив да се разглежда като `List` колекция
- Позволява да се *обработва масив като списък*
- Всяка *промяна* на `List` изгледа на масива променя и масива(!!)
- Всяка *промяна* в масива , променя и `List` изгледа на масива (!!)
- Единствената операция, позволена на `List` изгледа на масива , върнат с `asList` е *set*(!!)
- Обратно, `List` колекция се преобразува в масив с метод `toArray()` на `LinkedList`

```
1  // Fig. 3: UsingToArray.java
2  // Using method toArray.
3  import java.util.LinkedList;
4  import java.util.Arrays;
5
6  public class UsingToArray
7  {
8     // constructor creates LinkedList, adds elements and converts to array
9     public UsingToArray()
10    {
11       String colors[] = { "black", "blue", "yellow" };
12
13       LinkedList< String > links =
14          new LinkedList< String >( Arrays.asList( colors ) );
15
16       links.addLast( "red" );   // add as last item
17       links.add( "pink" );      // add to the end
18       links.add( 3, "green" );  // add at 3rd index
19       links.addFirst( "cyan" ); // add as first item
20
```

Вика метод **asList** за създаване на **List** изглед от масива **colors**, който после се използва за създаване на **LinkedList**

Вика метод **addLast** на **LinkedList** за добавяне на низа “**red**” към края на **links**

Вика метод **add** на **LinkedList** за добавяне на низа “**pink** като **последен елемент** и низа “**green**” като **елемент с индекс 3**

Вика метод **addFirst** на **LinkedList** за добавяне на низа“**cyan**” като първи елемент на **LinkedList**

```
21       // get LinkedList elements as an array
22       colors = links.toArray( new String[ links.size() ] );
23
24       System.out.println( "colors: " );
25
26       for ( String color : colors )
27          System.out.println( color );
28    } // end UsingToArray constructor
29
30    public static void main( String args[] )
31    {
32       new UsingToArray();
33    } // end main
34 } // end class UsingToArray
```

```
colors:
cyan
black
blue
yellow
green
red
pink
```

Вика метод **toArray** на **List** за намиране на представяне на **LinkedList като масив**

toArray използва за аргумент **масив с точно толкова елементи,** колкото са елементите на списъка

Обичайна грешка при програмиране 13а.3

Всеки опит за промяна (добавяне, изтриване) на елементите в `List`, получен с `asList()` води до `UnsupportedOperationException`. Всяка промяна на масива, използван като аргумент на `asList()` води до промяна на `List` обекта, получен с `asList()` . Единствената операция, която е позволена за `List` обекта, получен с `asList()` е операцията `set()`.

Обичайна грешка при програмиране 13а.3

Предаване на масив, съдържащ данни като аргумент на `toArray` може да доведе до логически грешки. Ако броят елементи в масива е по- малък от броя елементи на списъка, спрямо който `toArray` се вика, то се създава нов масив с достатъчно елементи да побере елементите на списъка, но данните в предишния масив се изтриват.

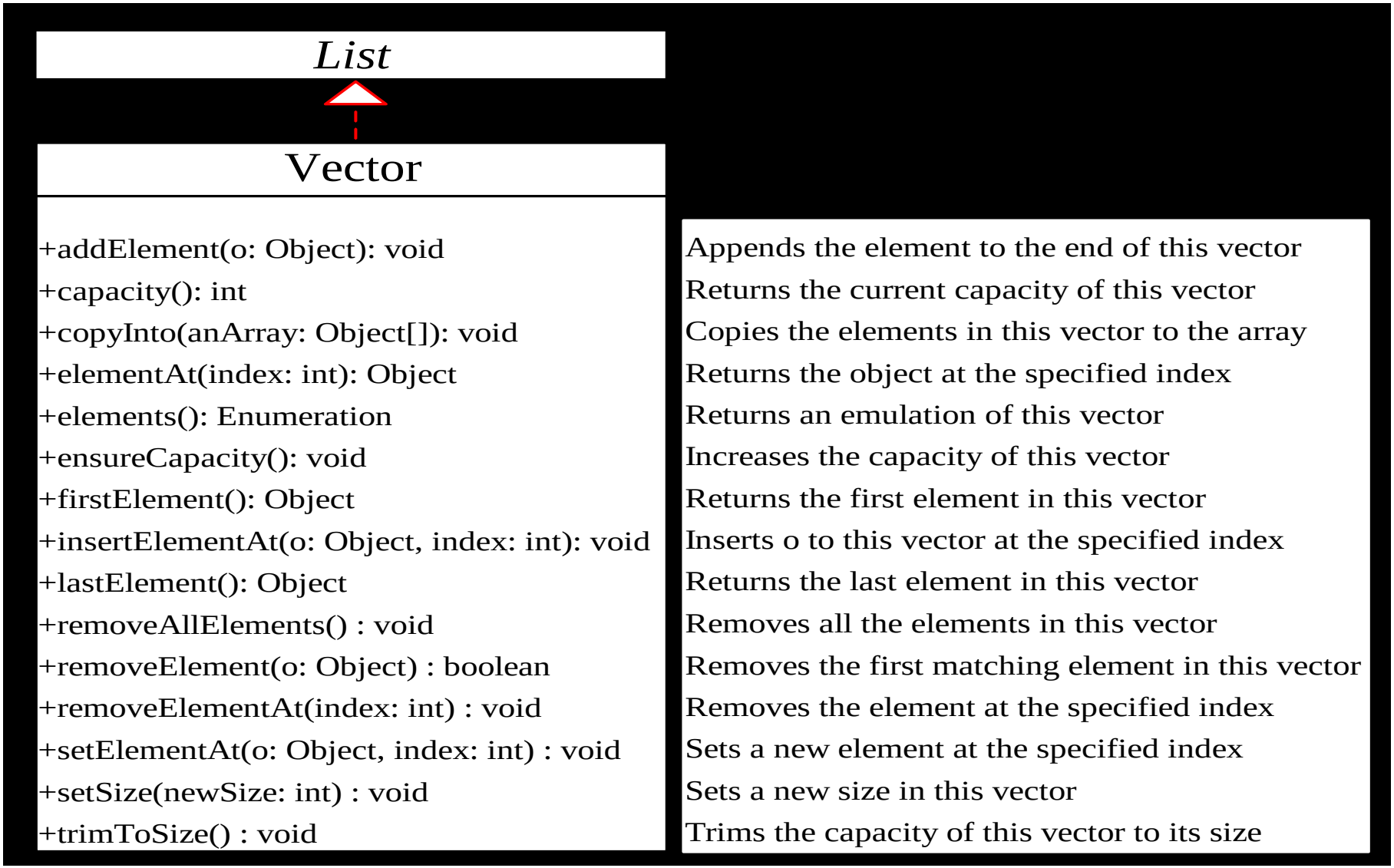
13a.5.3 `Vector`

class `Vector`

- **Подобна на масив структура данни, която може динамично да променя дължината си**
- **Има капацитет(*capacity*)- *когато броят елементи надхвърли капацитетът*, `Vector` обектът автоматично увеличава капацитета си**

  newCapacity = (oldCapacity * 3)/2 + 1
- **Нарастването на капацитета става на порции равни на зададено *нарастване на капацитет***
- **Ако не е зададено *нарастване на капацитет*, то по подразбиране дължината на `Vector` се удвоява (150% + 1) всеки път когато това е необходимо.**

13a.5.3 Vector



Съвет за по- добро качество 13a.2

Вмъкването на елемент във `Vector` чиито размер е по- малък от капацитета е относително бърза операция.

Съвет за по- добро качество 13a.3

Вмъкването на елемент във `Vector` чиито размер е по- голям от капацитета е относително бавна операция

```
1  // Fig. 4: VectorTest.java
2  // Using the Vector class.
3  import java.util.Vector;
4  import java.util.NoSuchElementException;
5
6  public class VectorTest
7  {
8     private static final String colors[] = { "red", "white", "blue" };
9
10    public VectorTest()
11    {
12       Vector< String > vector = new Vector< String >();
13       printVector( vector ); // print vector
14
15       // add elements to the vector
16       for ( String color : colors )
17          vector.add( color );
18
19       printVector( vector ); // print
20
```

Създава **Vector**от тип **String** с начален капацитет от **10** елемента и инкремент за нарастване на капацитета **0**

Вика метода **add** на **Vector** за добавяне на **String**- ове към края на **Vector**

```
21      // output the first and last elements
22      try
23      {
24         System.out.printf( "First element: %s\n", vector.firstElement());
25         System.out.printf( "Last element: %s\n", vector.lastElement() );
26      } // end try
27      // catch exception if vector is empty
28      catch ( NoSuchElementException exception )
29      {
30         exception.printStackTrace()
31      } // end catch
32
33      // does vector contain "red"?
34      if ( vector.contains( "red" ) )
35         System.out.printf( "\n\"red\" found at
36            vector.indexOf( "red" ) );
37      else
38         System.out.println( "\n\"red\" not found\n" );
39
40      vector.remove( "red" ); // remove the string "red"
41      System.out.println( "\"red\" has been removed" );
42      printVector( vector ); // print vector
43
```

Вика метод **firstElement** на **Vector** за получаване на първия елемент на **Vector**

Вика метода **lastElement** на **Vector** получаване на последния елемент на **Vector**

Метод **contains** на **Vector** връща **boolean** указващ дали **Vector** има даден **Object**

Метод **indexOf** на **Vector** връща индекса на първия елемент в **Vector** съвпадащ с аргумента на метода

Метод **remove** на **Vector** изтрива на първия елемент в **Vector** съвпадащ с аргумента на метода

```
44       // does vector contain "red" after remove operation?
45       if ( vector.contains( "red" ) )
46          System.out.printf(
47             "\"red\" found at index %d\n", vector.indexOf( "red" ) );
48       else
49          System.out.println( "\"red\" not found" );
50
51       // print the size and capacity of vector
52       System.out.printf( "\nSize: %d\nCapacity: %d\n", vector.size(),
53          vector.capacity() );
54    } // end Vector constructor
55
56    private void printVector( Vector< String > vectorToOutput )
57    {
58       if ( vectorToOutput.isEmpty() )
59          System.out.print( "vector is empty
60       else  // iterate through the elements
61       {
62          System.out.print( "vector contains
63
64          // output elements
65          for ( String element : vectorToOutput )
66             System.out.printf( "%s ", element );
67       } // end else
68
```

Vector методите **size** и **capacity** връщат броя на елементи на **Vector** и **Vector** капацитета

Методът **printVector** позволява само аргументи, които са **Vector** от **String**

Vector метода **isEmpty** връща **true**, когато няма елементи в ъв **Vector**

```
69       System.out.println( "\n" );
70    } // end method printVector
71
72    public static void main( String args[] )
73    {
74       new VectorTest(); // create object and call its constructor
75    } // end main
76 } // end class VectorTest
```

```
vector is empty

vector contains: red white blue

First element: red
Last element: blue

"red" found at index 0

"red" has been removed
vector contains: white blue

"red" not found

Size: 2
Capacity: 10
```

Обичайна грешка при програмиране 13а.4

Ако не е предефиниран метод `equals`, програмата извършва сравнения на базата на оператора == за определяне дали две референции указват един и същ обект в паметта.

Съвет за по- добро качество 13а.6

Методите `contains` и `indexOf` на `Vector` изпълняват последователно претърсване на съдържанието на `Vector` обекта.

Тези методи могат да се окажат бавни при прилагането им към големи `Vectors` обекти

13a.5.4 Deque

```
public interface Deque<E>
                    extends Queue<E>
```

A linear collection that supports element insertion and removal at both ends. The name `deque` is short for "double ended queue" and is usually pronounced "deck". Most `Deque` implementations place no fixed limits on the number of elements they may contain, but this `interface` supports capacity-restricted deques as well as those with no fixed size limit. Implemented in classes `LinkedList`, `ArrayDeque`, `ConcurrentLinkedDeque`, `LinkedBlockingDeque`

13a.5.4 Deque

This `interface` defines methods to access the elements at both ends of the `deque`. Methods are provided to insert, remove, and examine the element. Each of these methods exists in two forms: one throws an exception if the operation fails, the other returns a special value (either `null` or `false`, depending on the operation). The latter form of the insert operation is designed specifically for use with capacity-restricted `Deque` implementations

13a.5.4 Deque

Summary of Deque methods

	First Element (Head)		Last Element (Tail)	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

Comparison of Queue and Deque methods

Queue Method	Equivalent Deque Method
`add(e)`	`addLast(e)`
`offer(e)`	`offerLast(e)`
`remove()`	`removeFirst()`
`poll()`	`pollFirst()`
`element()`	`getFirst()`
`peek()`	`peekFirst()`

Comparison of Stack and Deque methods

Stack Method	Equivalent Deque Method
`push(e)`	`addFirst(e)`
`pop()`	`removeFirst()`
`peek()`	`peekFirst()`

13a.5.4 Deque

```
Deque<String>  dequeB = new ArrayDeque<>();
Deque<String>  dequeA = new LinkedList<>();
dequeA.add      ("element 1"); //add element at tail
dequeA.addFirst("element 2"); //add element at head

dequeA.addLast ("element 3"); //add element at tail
//access via Iterator
Iterator<String> iterator = dequeA.iterator();
while(iterator.hasNext(){
     String element = iterator.next();
}
//access via for-loop
for(Object object : dequeA) {
      String element = object;
}
```


13a.5.4 Deque

```
Deque dequeA = new LinkedList();

dequeA.add      ("element 1"); //add element at tail
dequeA.addFirst("element 2"); //add element at head
dequeA.addLast ("element 3"); //add element at tail
```

```
Object firstElement = dequeA.remove();
Object firstElement = dequeA.removeFirst();
Object lastElement  = dequeA.removeLast();
```


13a.6 Local Variable Type Inference

Variable type inference, the ability of the compiler to automatically deduce the type of a variable, has become an essential part of most strongly typed programming languages. While many languages such as C++ and C# have introduced rich type inference mechanics over the years, Java has been noticeably absent in this department.

Prior to JDK 10, a vast majority of the type inference capability of Java was focused on the right-hand side of expressions and was largely seen in lambda argument type inference, the diamond operator, and generic method argument type inference:

```
List<Integer> ints = new ArrayList<>();
public <T> Bar foo(List<T> elements) { /* … */ }
Bar bar = foo(new ArrayList<Integer>());
```


13a.6 Local Variable Type Inference

In JDK 10, Java now includes the **`var`** reserved type name which allows **the type of an initialized local variable** to **be inferred** by the Java compiler:

```
var i = 0;
var students = new ArrayList<Student>();
var iter = list.listIterator();
```

This may seem to be a trivial improvement in the language, but its true usefulness becomes evident when comparing the following declarations:

```
Iterator<List<SomeLongBeanName>> iterator =
                                        someList.iterator();
// write instead just the following
var iterator = someList.iterator();
```


13a.6 Local Variable Type Inference

The **scope and applicability of the `var`** reserved type is applicable to a few specific contexts:

- ✓ Local variables with initializers
- ✓ Indices in the enhanced for-loop
- ✓ Locals declared in traditional for-loops

Therefore, **all of the following are valid uses** of `var` as a reserved type name:

```
var foo = 1;
for (var student: students) { /* … */ }
for (var i = 0; i < 10; i++) { /* … */ }
```


Задачи

Задача 1.

Дефинирайте следните термини в Collections библиотеката на Java:

a) `Collection`

b) `Collections`

c) `List`

Задача 2.

Обяснете как работи всеки от следните методи на `Iterator`:

a) `iterator`

b) `hasPrevious`

c) `next`

d) `remove`

Задачи

Задача 3.

Нека

```
LinkedList<String> list = new LinkedList<String>();
ListIterator<String> iter;
```

След изпълнение на всяка следните команди , какви ще бъдат елементите на `list`?

```
list.addFirst("Tom");
list.addFirst("Ann");
iter = list.listIterator();
iter.next();
iter.add("Mike");
iter.next();
iter.add("Vic");
```


Задачи

Задача 4.

Какво прави следния метод `f()` при зададен списък `aList`?

```
 public static <T> void f(LinkedList<T> aList)
{
  ListIterator<T> iter = aList.listIterator();
  int pos = 0;
  while(iter.hasNext())
  {
        aList.addFirst(iter.next());
        pos++;
        iter = aList.listIterator(pos);
        iter.next();
        iter.remove();
  }
}
```