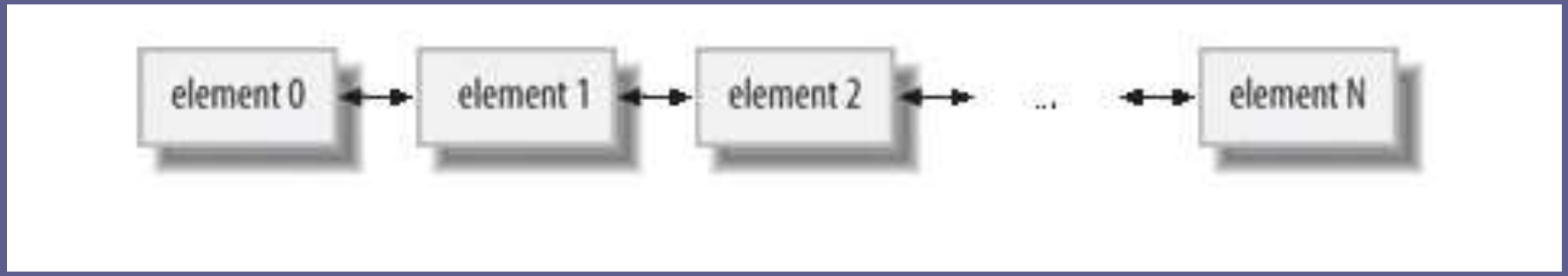
Тема 8. Колекции (Контейнери на Java) (продължение)

Съдържание

- Организация на информацията в колекциите;
- Алгоритми (Algorithms) - Същност и начин на реализация; Видове;
- Сортиране;
  - Сравнение на обектите;
  - Сравнители (естествен ред);
  - Класове за използване в колекции (непроменяеми);
  - Сравнители (интерфейс Comparator);
- Алгоритъм за случайно разместване;
- Алгоритми за манипулиране на данни;
- Алгоритми за търсене;
- Композиция;
- Намиране на екстремуми;
- Разширение на Iterator (ListIterator)

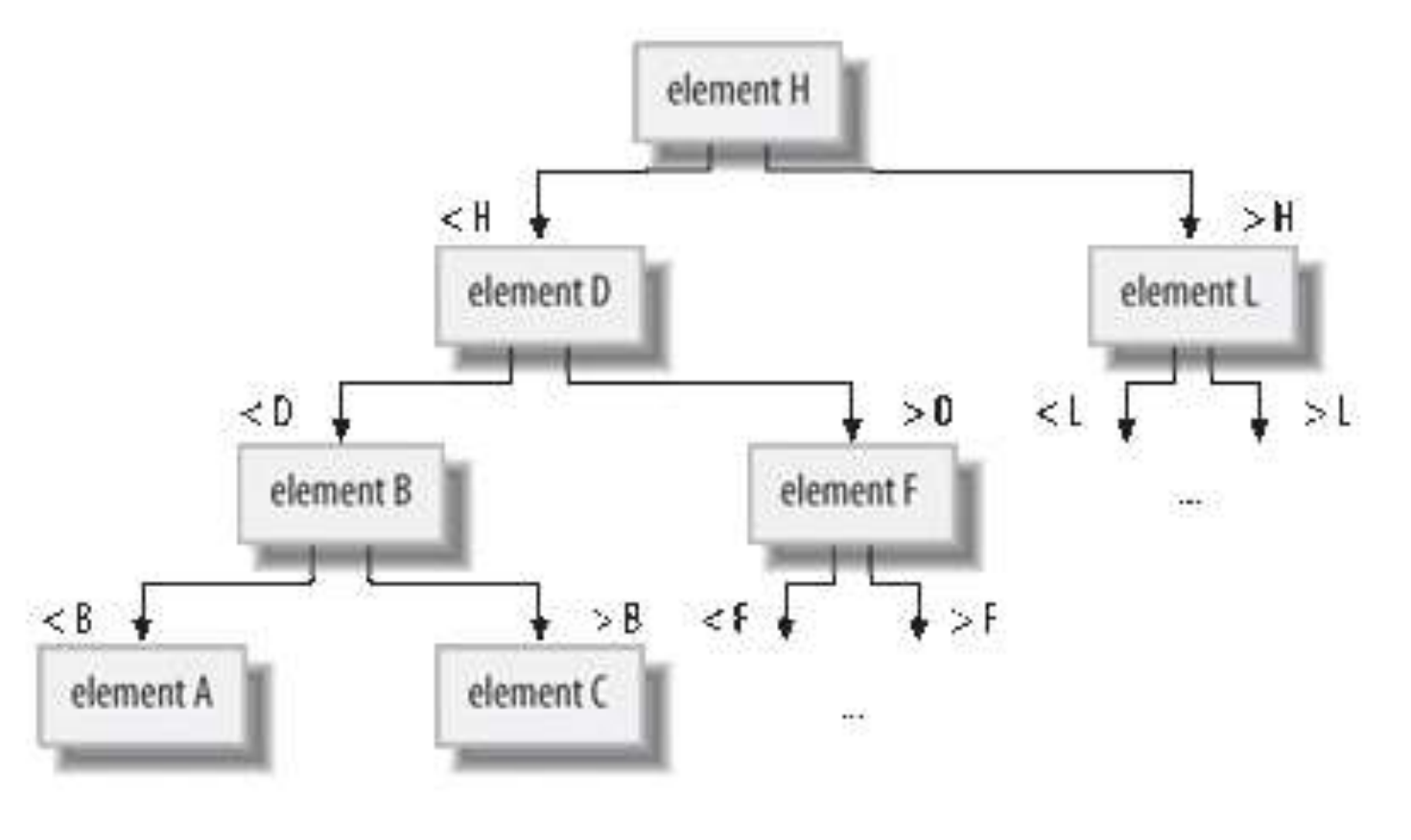
Организация на информацията в колекциите

- Организация на свързан списък:



Организация на информацията в колекциите

Дървовидна организация



Алгоритми (Algorithms)

- Същност :

Алгоритмите в Java са функционалности за общо (полиморфно) използване, предоставяни от платформата на Java.

- Особености:
  - Всички са параметризирани от клас Collections и се предоставят като статични методи;
  - първият аргумент е колекцията, върху която се прилага операцията. По-голямата част от алгоритмите работят със списъци (инстанции на List);
  - част от тях оперират с инстанции на произволни колекции Collection.

Алгоритми

- Видове:
  - Сортиране;
  - Алгоритъм за случайно разместване;
  - Алгоритми за манипулиране на данни;
  - Алгоритми за търсене;
  - Композиция;
  - Намиране на екстремуми;

Алгоритми. Сортиране

Сортиране: Алгоритъмът за сортиране пренарежда списък List в съответствие с нарастващия ред на елементите му. Има две форми на операцията:

- Подразбираща се форма-Comparable;
- С интерфейс Comparator

Алгоритми. Сортиране

- Простата форма на операцията сортира подадения като аргумент списък в съответствие с **естествения ред** на неговите елементи. За сравнител се използва интерфейса **Comparable**.

Алгоритми. Сортиране

- **Естествен ред** на елементите на клас:
  - Интерфейс **Comparable;**
  - Предефиниране на **equals;**
  - Съответствие между тях:

  за всяка двойка обекти E1 и E2

  E1.compareTo (E2) == 0

  да има същата булева стойност като:

  E1.equals (E2)

Алгоритми. Сортиране

- **Естествен ред** на елементите на клас, **частен случай – инстанция null**:
  - null не е валидна инстанция на клас и e.compareTo (null) трябва да предизвика: **NullPointerException**
  - при дефиниране на еквивалентност:

  e.equals (null) връща false.

Алгоритми. Сортиране

Действие: Използва оптимизиран вариант на сортировка с *merge sort* в два варианта:

- Бърз: Гарантира стартиране за n log(n) време и се изпълнява по-бързо при предварително частично сортирани списъци. Имперично е доказана ефективност като на оптимизирания quicksort. Алгоритъмът quicksort е по конструкция по-бърз, но не гарантира n log(n) и не осигурява запазването на елементите.
- Със запазването на елементите : Не пренарежда еквивалентните обекти. Това е важно при последователни сортировки при промяна на атрибутите на сортировка.
- Пример:

Последователно сортиране на inbox по дата на писма, следвано от сортировка по имена ...

Алгоритми. Сортиране

Всеки клас, който имплементира интерфейс Comparable може да се сортира. В процеса на сортиране се използва задължителния за имплементиране метод:

int compareTo(...)

Алгоритми. Сортиране

Списъкът List (референция lst) се сортира с командата Collections.sort(lst);

Типове данни и принципи:

- Ако е от String, се сортира лексикографически;
- Ако е от дати – хронологично;
- Ако е от числа – като число със знак.

Алгоритми. Сортиране

Сортиране на класовете на езика:

- **Byte: числа със знак**
- **Character : числа без знак**
- **Long : числа със знак**
- **Integer : числа със знак**
- **Short : числа със знак**
- **Double : числа със знак**
- **Float : числа със знак**
- **BigInteger : числа със знак**
- **BigDecimal: числа със знак**
- **Boolean: Boolean.FALSE < Boolean.TRUE**
- **File: лексикографски по файлови имена**
- **String: лексикографски**
- **Date: хронологично**
- **CollationKey: лексикографски с локалните настройки по държави**

Алгоритми. Сортиране

Особености на класовете на езика:

- Един обект на заграждащ клас капсулира само една стойност на съответния му прост тип;
- Той е непроменяем (immutable) обект, който служи като контейнер за стойността и допуска само четене;
- Може да се създаде обект от прост тип или от стрингово представяне на стойността.

Непроменяемите обекти нямат методи за промяна на стойността (set) след конструиране.

Алгоритми. Сортиране

Пример: бързо сортиране на низове с клас CollationKey

```
public final class CollationKey extends Object {
    // няма конструктори
    // Методи:
  public int compareTo(CollationKey target);
  public boolean equals(Object target);
  public String getSourceString();
  public int hashCode();  // предефинира Object
  public byte[] toByteArray();
}
```

Алгоритми. Сортиране

```
// Пример Person с CollationKeys за сортиране с отчитане на
    регионалните настройки:
import java.text.CollationKey;
import java.text.Collator;
import java.util.Locale;
class Person implements Comparable {
  private final String firstName;
  private final String lastName;
  private static final Collator collator =
    Collator.getInstance(Locale.getDefault());
  private final CollationKey sortLastNameKey;
  private final CollationKey sortFirstNameKey;
  private int age;
```

(продължава)

Алгоритми. Сортиране

```
// конструктор:
public Person(String fName, String lName, int iage){
    firstName=fName;
    lastName=lName;
    age=iage;
    sortLastNameKey=collator.getCollationKey(
        lastName.toUpperCase()+firstName.toUpperCase());
    sortFirstNameKey=collator.getCollationKey(
        firstName.toUpperCase()+lastName.toUpperCase());
  }
// методи за четене:
public CollationKey getFirstNameKey() { return sortFirstNameKey; }
public CollationKey getLastNameKey() {  return sortLastNameKey;  }
```

(продължава)

Алгоритми. Сортиране

```
// дефиниция на сравнителни класове (методи)
class LastNameComparator implements Comparator {
  public int compare(Object person, Object anotherPerson) {
    return ((Person)person).getLastNameKey().
           compareTo(((Person)anotherPerson).getLastNameKey());
  }
}
class FirstNameComparator implements Comparator {
  public int compare(Object person, Object anotherPerson) {
return ((Person)person).getFirstNameKey().
           compareTo(((Person)anotherPerson).getFirstNameKey());
  }
}
```

Алгоритми. Сортиране

```
// main (test)
StudentGroup firstGroup=new StudentGroup(1);
firstGroup.insert( new Student("6106002", "ИВАН1", "ПЕТРОВ2", 21));
firstGroup.insert( new Student("6106003", "Иван3", "Петров3", 23));
firstGroup.insert( new Student("6106001", "Иван2", "Петров1", 21));
System.out.println("Natural Order");   System.out.println(firstGroup);
System.out.println("sortByLastFirstName:"); // LastNameComparator()
firstGroup.sortByLastName();   System.out.println(firstGroup);
System.out.println("sortByFirstLastName:"); // FirstNameComparator()
firstGroup.sortByFirstName();  System.out.println(firstGroup);
System.out.println("sortByFN:");
firstGroup.sortByFN();    System.out.println(firstGroup);
```

Алгоритми. Сортиране

Пример сортиране на списъка от параметри на главната функция на класа Sort :

```
import java.util.*;
public class Sort {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

Стартиране с примерни параметри: i walk the line
Изход:
[i, line, the, walk]

Алгоритми. Сравнение

Предефиниране на интерфейса в потребителски клас:

Пример:

public class Student extends Person implements Comparable

Функцията от този интерфейс, която трябва да се предефинира е **compareTo: compareTo е с резултат int и има смисъла на сравнител, връщащ -1 (по-малко), 1 (по-голямо) и 0 (равно).**

Алгоритми. Сравнение

Пример за предефиниране за сравняване на студенти по факултетен номер в нарастващ. Използва се сравнението на тип String, което е организирано на същия принцип. Добавят се методи за сравнението:

Примери.

```
public int compareTo(Object o) {
  return
   (strFacNumer.compareTo(((Student)o).strFacNu
   mer));
}
```

Алгоритми. Сравнение

Шаблонен вариант на интерфейса:

Шаблонно дефиниране на метода compareTo в интерфейса Comparable - шаблон **T**:

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

Класове за използване в колекции (непроменяеми)

```
import java.util.*;
// Пример за непроменяем клас
public class Name implements Comparable<Name> {
    private final String firstName, lastName;
// Класът не допуска нулеви референции ои сравняваните  данни!
    public Name(String firstName, String lastName) {
        if (firstName == null || lastName == null)
            throw new NullPointerException();
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String firstName() { return firstName; }
    public String lastName()  { return lastName;  }

//продължава
```

Класове за използване в колекции (непроменяеми)

```
public boolean equals(Object obj) {
        if (!(obj instanceof Name))
            return false;
        Name n = (Name)obj;
        return n.firstName.equals(firstName) &&
               n.lastName.equals(lastName);
    }
    public int hashCode() { //за използване от Hash…
        return 31*firstName.hashCode() + lastName.hashCode();
    }
    public String toString() {
        return firstName + " " + lastName;
    }
    public int compareTo(Name nm) { // каскадно сравнение
        int lastCmp = lastName.compareTo(nm.lastName);
        return (lastCmp != 0 ? lastCmp :
                firstName.compareTo(nm.firstName));
    }
}
```

Класове за използване в колекции (непроменяеми)

```
import java.util.*;

public class NameSort {
    public static void main(String[] args) {
        Name nameArray[] = {
            new Name("John", "Lennon"),
            new Name("Karl", "Marx"),
            new Name("Groucho", "Marx"),
            new Name("Oscar", "Grouch")
        };
        List<Name> names = Arrays.asList(nameArray);
        Collections.sort(names);
        System.out.println(names);
    }
}
```

[Oscar Grouch, John Lennon, Groucho Marx, Karl Marx]

Класове за използване в колекции (непроменяеми)

Изисквания:

- Обектите на Name са непроменяеми *immutable*. Класове, които могат да се използват като елементи на множество, карта, като ключ и стойност, да участват в сортировка и др. Не позволяват промяна на елементите след запис като елемент или ключ на колекция;
- Конструкторът проверява за нулеви аргументи. Това осигурява всички обекти да са правилно създадени и ако не е така да се продуцира изключение: NullPointerException.

Класове за използване в колекции (непроменяеми)

- Методът hashCode е предефиниран. Това е важно за всеки клас, който предефинира метода equals. (равните обекти трябва да имат равни кодове.)

При проверката за равенство на ключ се използва код:

```
if ((keyHashcode == storedKeyHashcode) && key.equals(storedKey))
    return object;
```

- Методът equals връща false ако специфицирания обект е null или не е инстанция на същия клас. Изисква се от общата конвенция за поведение на съответните методи;
- Предефиниран е методът toString за да се преобразува в читаема форма съдържанието на класа. Колекциите поддържат собствен метод, който циклично извежда елементите си – ключове, стойности.

Сравнители (интерфейс Comparator)

- Сравнители: Използват се за случаите, когато не може да се използва естествения ред на обектите.

За реализация се използва интерфейс Comparator — обект, който капсулира подредба. Подобно на интерфейса Comparable, интерфейса Comparator има само един метод.

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

Методът compare сравнява двата си аргумента и връща -1, 0, 1 ако левият му аргумент o1 е по-малък, равен или по-голям от десния. Продуцира ClassCastException ако има несъответствие на класове.

Сравнители (интерфейс Comparator)

Действие:

Подобно на сравнителя на класовете, като прилага операцията върху двата си аргумента.

Пример:

```
public class Employee implements Comparable<Employee>
{
  public Name name() { ... }
  public int number() { ... }
  public Date hireDate() { ... } ...
}
```

Нека естествения ред е основан на Name.

Сравнители (интерфейс Comparator)

Подредба по дата, без създаване на клас за сравнение:

```
import java.util.*;
public class EmpSort {
    static final Comparator<Employee> SENIORITY_ORDER =
        new Comparator<Employee>() {
                    public int compare(Employee e1, Employee e2) {
                    return e2.hireDate().compareTo(e1.hireDate());
                    }
            };
    // Employee collection
    static final Collection<Employee> employees = ... ;

    public static void main(String[] args) {
        List<Employee>e = new ArrayList<Employee>(employees);
        Collections.sort(e, SENIORITY_ORDER);
        System.out.println(e);
    }
}
```

Сравнители (интерфейс Comparator)

Особености на примерната дефиниция:

Не осигурява използването на обектите в сортирани колекции, например TreeSet. Ако се използва този сравнител, ще изключи дублираните по дата обекти от колекцията.

Решение:

Добавяне на допълнителни условия при равенство на сортиращия параметър – до изчерпване на всички възможности на еквивалентността:

“Каскадно сравняване” по атрибутите на класа.

Сравнители (интерфейс Comparator)

Пример:

```
static final Comparator<Employee> SENIORITY_ORDER =
                                  new Comparator<Employee>() {
    public int compare(Employee e1, Employee e2) {
        int dateCmp =
          e2.hireDate().compareTo(e1.hireDate());
        if (dateCmp != 0)
            return dateCmp;
        return (e1.number() < e2.number() ? -1 :
                (e1.number() == e2.number() ? 0 : 1));
    }
};
```

Еквивалент е “хакерски” код от вида:

```
return e1.number() - e2.number();
```

Алгоритъм за случайно разместване (shuffle)

Същност: Действието му е обратно на сортирането. Целта е да се размени всеки тип на подредба, който е приложен върху елементите на списък List. При разместването на входния списък се използват пермутации на стойностите. Допуска се че входа не е случаейно подреден. Използва се при имплементиране на игри и хазарт. Например при случайно разместване на списък от обекти Card, представящи тесте. Операцията има две форми:

- С един аргумент и подразбиращ се принцип на разместване;
- При предаден като аргумент обект, използващ се за случайното разместване обекта Random.

Алгоритъм за случайно разместване (shuffle)

Помощни статични методи в Collections:

Размяна на елементи **swap:**

```
public static <E> void swap(List<E> a, int i, int j) {
    E tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}
```

Случайно разместване

```
public static void shuffle(List<?> list, Random rnd) {
    for (int i = list.size(); i > 1; i--)
      swap(list, i - 1, rnd.nextInt(i));
}
```

Алгоритъм за случайно разместване (shuffle)

Примерна програма за разместване:

```
import java.util.*;

public class Shuffle {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        for (String a : args)
            list.add(a);
        Collections.shuffle(list, new Random());
        System.out.println(list);
    }
}
```

Алгоритми за манипулиране на данни

В Java 5.0 се допуска да се обработват данните на List с 5 алгоритъма:

- reverse — обратен ред на елементите на List.
- fill — запълване с определена стойност. Променя всеки елемент на списъка, заменяйки го със предадената стойност. Използва се за реинициализация на List.
- copy — предават се два аргумента, destination List и source List, копира елементите на source в destination, презаписвайки съдържанието му. Трябва destination List да е равен или по-голям от source. Ако е по-голям, остатъчните елементи на destination List не се променят.

Алгоритми за манипулиране на данни

- swap — разменя елементите на указани позиции в List.
- addAll — добавя всички специфицирани елементи в Collection. Елементите се задават поединично или като масив.

Алгоритми за търсене (Searching)

Алгоритъм binarySearch търси специфицирания елемент в сортирана колекция List. Има две форми:

- Със списък като аргумент и елемент за търсене реализира търсене на “ключ”. Тази форма предполага нарастваща подредба на елементите относно търсения.
- С допълнителен Comparator със списъка List и търсения ключ и предполага нарастваща подредба на List в съответствие със специфицирания Comparator. Алгоритъмът sort може да се използва преди да се стартира binarySearch.

Алгоритми за търсене (Searching)

Резултатът е еднакъв за двете форми:

- Ако се намери ключа- връща се индекс.
- Ако не се намери резултатът е (-(insertion point) - 1), където insertion point е индексът на списъка, по който стойността ще бъде записана в съответствие с подредбата на елементите в List, или индекса на първият елемент, по-голям от стойността или list.size() ако всички елементи са по-малки от търсения. Формулата гарантира положителна или нулева (>= 0) стойност при откриване на ключа. Тя е "хакерска комбинация" между индекс и логическа стойност boolean (found) и int (index) в една връщана стойност.

Алгоритми за търсене (Searching)

```
int pos = Collections.binarySearch(list, key);
    if (pos < 0)
        list.add(-pos-1);
```

Алгоритми за композиция

Алгоритми за композиция проверяват аспекти на композицията на две или повече колекции Collections:

- frequency — брои колко пъти даден елемент се съдържа в колекцията;
- disjoint — определя дали две колекции са независими, т.е. Нямат общи елементи.

Намиране на екстремуми

Два алгоритъма:

- min;
- max.

Връщат респективно минималния или максималния елемент в специфицираната колекция Collection.

Намиране на екстремуми

И двата имат две форми:

- Първата е с параметър Collection и връща minimum (или maximum) елемент в съответствие с нарастващ ред на елементите в съответствие с подразбиращата се (естествена) подредба.
- Втората има допълнителен сравнител - тип Comparator заедно с Collection и връща елемента minimum (или maximum) в съответствие със сравнението, специфицирано в Comparator

Разширение на Iterator (ListIterator)

Дефиниция в пакета:java.util

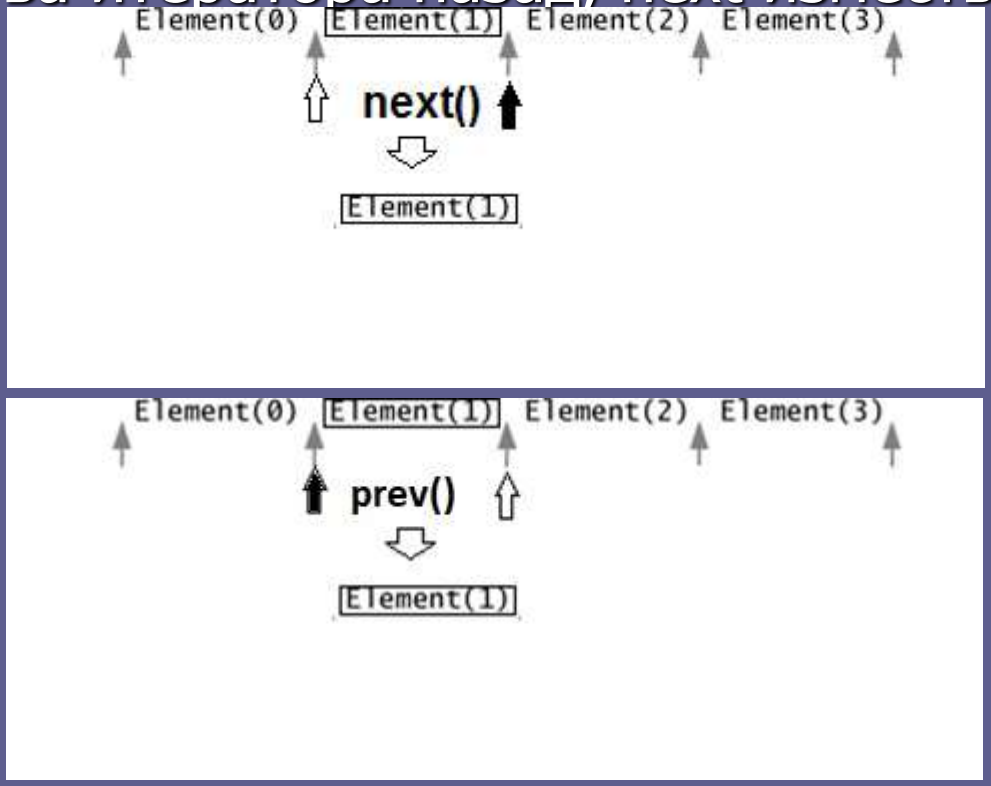
```
public interface ListIterator extends Iterator {
    boolean hasNext();
    Object next();

    boolean hasPrevious();
    Object previous();

    int nextIndex();
    int previousIndex();

    void remove(); //с варианти
    void set(Object o); //с варианти
    void add(Object o); //с варианти
}
```

Разширение на Iterator (ListIterator)

- Методът add вмъква новият елемент в списъка, непосредствено преди текущата позиция на итератора;
- Резултатът който връща nextIndex е винаги с 1 по-голям от този върнат от previousIndex. Оттук следват граничните случаи:
  - Резултат от previousIndex когато курсора е в начално положение е -1;
  - Резултат от nextIndex когато курсорът е след последния елемент връща list.size().

Разширение на Iterator (ListIterator)

ListIterator наследява от Iterator методи (hasNext, next и remove). Действието е единтично за двата интерфейса. Методът hasPrevious и previous са аналози на hasNext и next. Операция previous премества итератора назад, next измества напред.



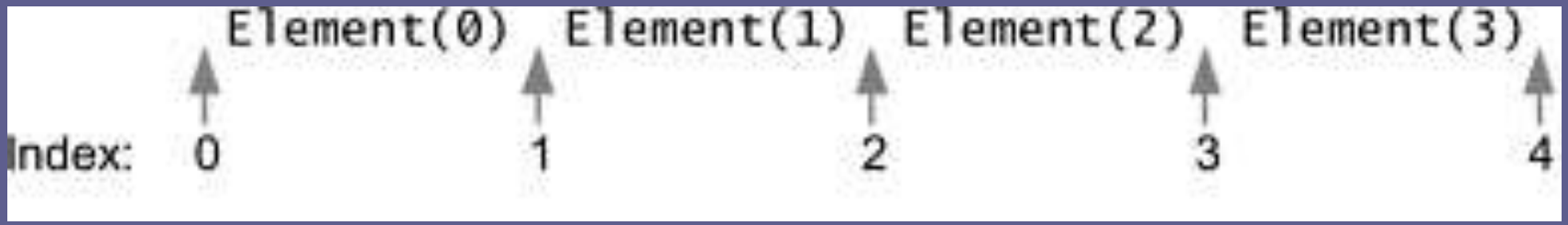
Разширение на Iterator (ListIterator)

Преминаване през списък от обекти в обратен ред с използване на итератори:

```
for (ListIterator iter = lst.listIterator(lst.size());
     iter.hasPrevious(); ) {
     Rectangle prev = (Rectangle) iter.previous();
 }
```

Разширение на Iterator (ListIterator)

Валидни индекси на списъчния итератор:



Разширение на Iterator (ListIterator)

Имплементация на indexOf:

```
public int indexOf(Object o) {
    for (ListIterator listit = listIterator(); listit.hasNext(); )
        if (o==null ? listit.next()==null : o.equals(listit.next()))
            return listit.previousIndex();
    return -1; // не е намерен
}
```

Разширение на Iterator (ListIterator)

Имплементация на replace за промяна на всички срещнати в списъка обекти val с newVal, реализирана чрез итераторен set:

```
public void replace(List l, Object val, Object newVal) {
    for (ListIterator i = l.listIterator(); i.hasNext(); )
        if (val==null ? i.next()==null : val.equals(i.next()))
            i.set(newVal);
}
```

Разширение на Iterator (ListIterator)

Имплементация на replace за промяна на всички срещнати в списъка обекти val с елементите от списък newVals, реализирана чрез итераторен add:

```
public static void replace(List l, Object val, List newVals) {
    for (ListIterator i = l.listIterator(); i.hasNext(); ) {
        if (val==null ? i.next()==null : val.equals(i.next())) {
            i.remove();
            for (Iterator j= newVals.iterator(); j.hasNext(); )
                i.add(j.next());
        }
    }
}
```

Въпроси?