15. Шаблон Итератор (Iterator)

ЛЕКЦИОНЕН КУРС: ШАБЛОНИ ЗА ПРОЕКТИРАНЕ

ГЛ. АС. Д-Р ЕМИЛ ДОЙЧЕВ

Общи сведения

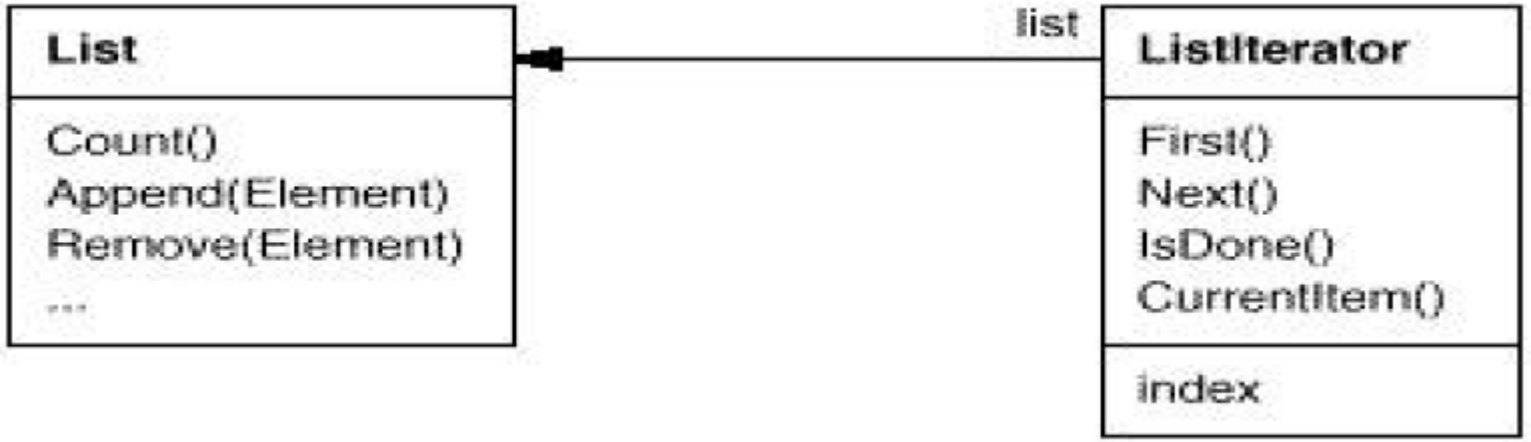
- ✓ **Вид:** Поведенчески за обекти
- ✓ **Цел:** Предоставя начин за последователен достъп до елементи на съставен обект, без да се разкрива същинското му представяне.
  - Под съставен обект се разбира такъв, който съдържа други обекти с цел да ги групира в общ компонент.
  - Нарича се още контейнер или колекция.
  - Примери: хеш-таблица или свързан списък.
- ✓ **Известен и като:** Курсор (Cursor)

Мотивация

- ✓ Сложен обект, като списък, който съдържа други обекти, трябва да позволява елементите му да бъдат обхождани без да разкрива вътрешната си структура.
- ✓ Освен това трябва да позволява:
  - обхождане с различни методи;
  - паралелно обхождане от няколко нишки;
- ✓ Не трябва да се добавят тези методи към интерфейса на съставния обект (колекцията)!

Пример 1

- ✓ Списък с итератор



Пример 1 (продължение)

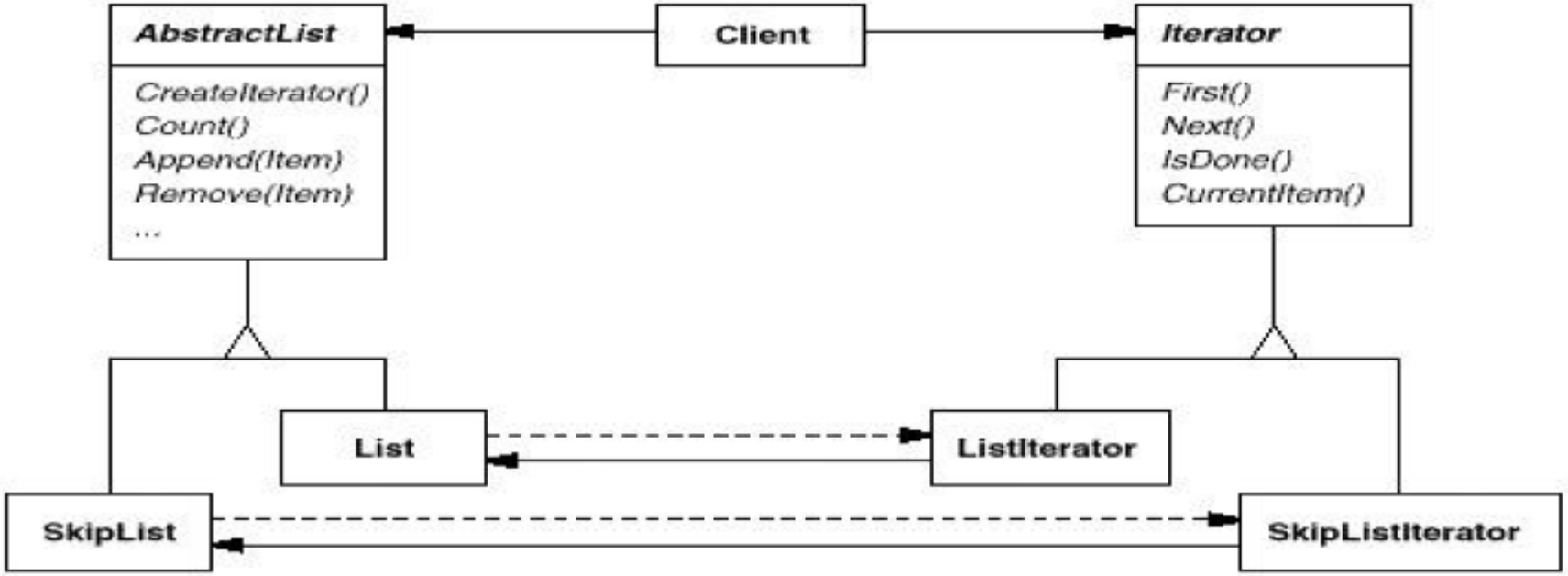
- ✓ Обичаен код в клиента

```
...
List list = new List();
...
ListIterator iterator = new ListIterator(list);

iterator.First();
while (!iterator.IsDone()) {
  Object item = iterator.CurrentItem();
  // Обработка на item
  iterator.Next();
}
...
```

Пример 2

- ✓ Полиморфен итератор



Пример 2 (продължение)

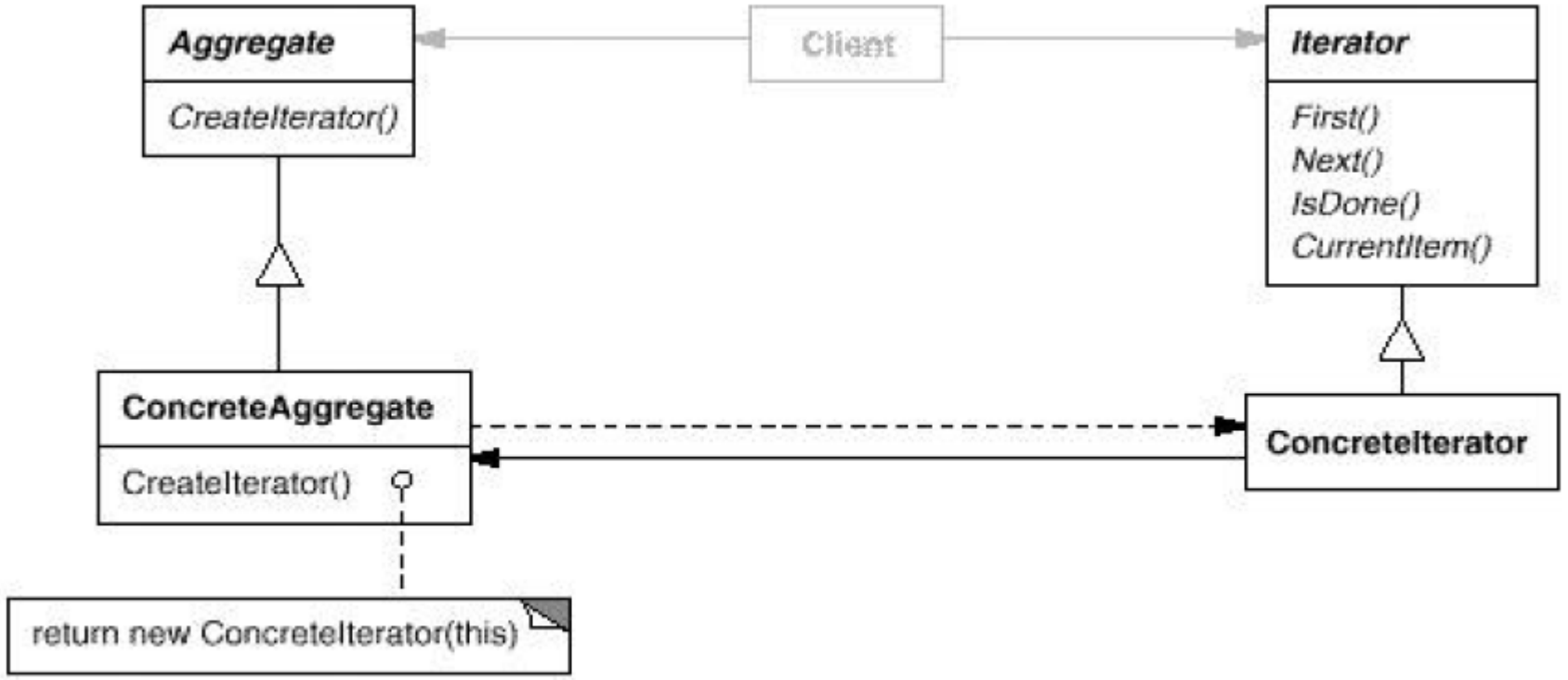
- ✓ Обичаен код в клиента

```
List list = new List();
SkipList skipList = new SkipList();
Iterator listIterator = list.CreateIterator();
Iterator skipListIterator = skipList.CreateIterator();
handleList(listIterator);
handleList(skipListIterator);
...
public void handleList(Iterator iterator) {
  iterator.First();
  while (!iterator.IsDone()) {
    Object item = iterator.CurrentItem();
    // Обработка на item
    iterator.Next();
  }
}
```

Приложимост

- ✓ За предоставяне на възможност за обхождане на елементите на съставен обект без да се разкрива вътрешната му структура.
- ✓ За поддържане на обхождане от множество паралелни процеси.
- ✓ За предоставяне на общ интерфейс за обхождане на различни съставни структури – т.е. полиморфични итерации.

Структура


Aggregate
CreateIterator()
Client
Iterator
First()
Next()
IsDone()
CurrentItem()
ConcreteAggregate
CreateIterator()
ConcreteIterator
return new ConcreteIterator(this)

Участници

- ✓ **Iterator** – Дефинира интерфейс за обхождане и достъп до елементи.
- ✓ **ConcreteIterator**
  - Имплементира интерфейса Iterator.
  - Запазва текущата позиция в итерираната структура.
- ✓ **Aggregate** – Дефинира интерфейс за създаване на обект Iterator (метод фабрика!).
- ✓ **ConcreteAggregate** – имплементира интерфейса за създаване на Iterator и връща инстанция на правилния ConcreteIterator

Следствия

- ✓ **Предимства**
  - Опростява интерфейса на съставния обект (aggregate) като не го обременява с методи за обхождане.
  - Поддържа множество паралелни обхождания.
  - Дава възможност за промяна на алгоритъма на обхождане, чрез смяна на инстанцията на итератора.
- ✓ **Недостатъци**
  - Няма

Имплементация

- ✓ Кой контролира итерацията?
  - Клиента => повече гъвкавост, нарича се външен итератор;
  - Самият итератор – нарича се вътрешен итератор.
- ✓ Кой дефинира алгоритъма за обхождане?
  - Итератора – най-често срещания случай; възможност за поддържане на различни варианти на обхождане;
  - Съставния обект – итератора само пази информация за текущото състояние на итерацията.
- ✓ Може ли съставния обект да бъде модифициран докато е активно обхождане на елементите му?
  - Итератор, който позволява добавяне и премахване на елементи без да се засегне процеса на обхождане и без да се прави копие на съставния обект, се нарича *стабилен (robust) итератор.*
- ✓ Трябва ли да разширим интерфейса на итератора чрез добавяне на допълнителни операции (напр. previous)?

Известни употреби

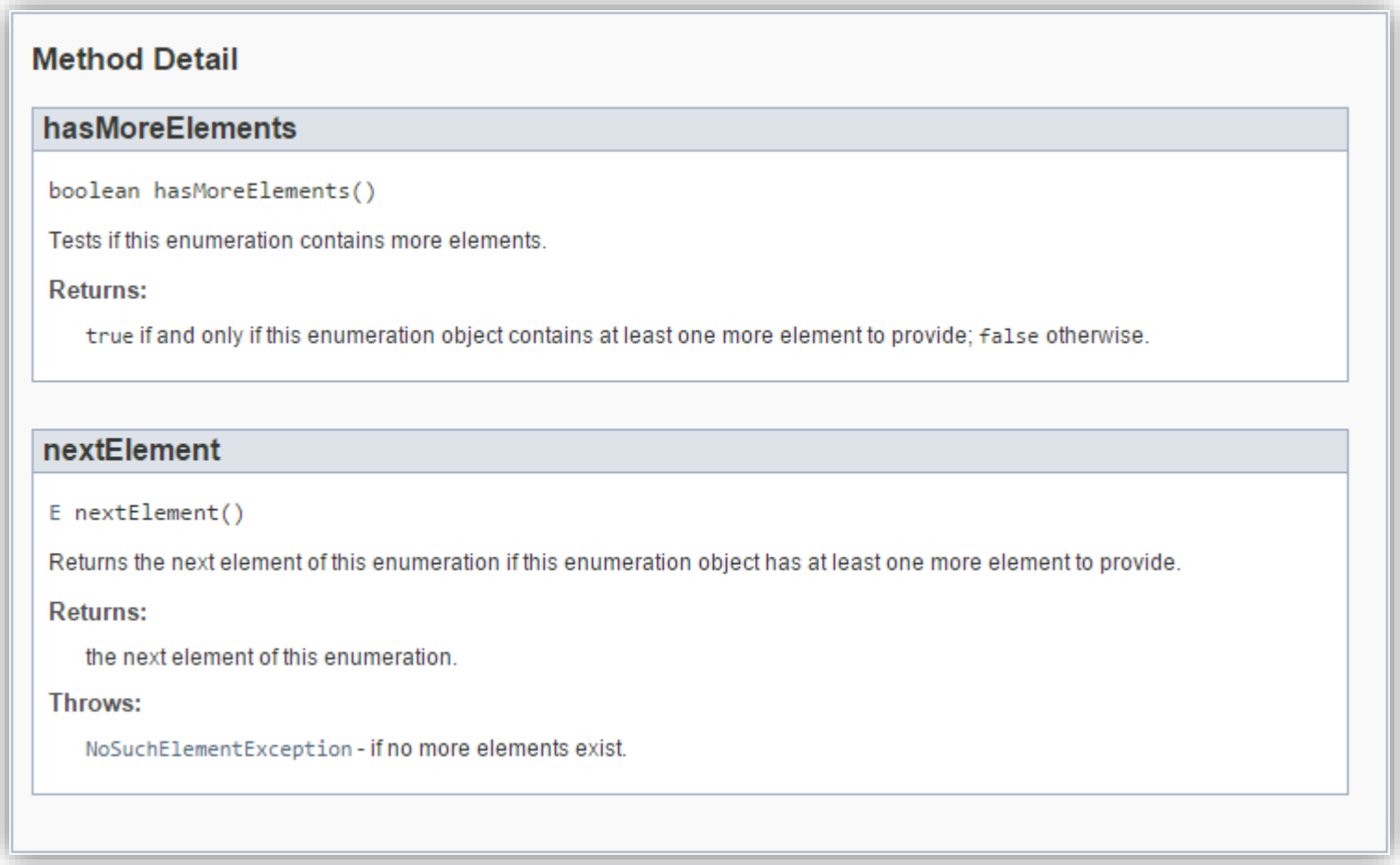
- ✓ Известни употреби
  - java.util.Enumeration interface
  - Java 2 Collections Framework Iterator interface
- ✓ Свързани шаблони
  - Метод фабрика – полиморфните итератори използват метод фабрика за инстанциране на правилния подклас на итератор.
  - Композиция – итераторите се използват често за рекурсивно обхождане на композиции.

Имплементация на итератор в Java

- ✓ Възможно е да се имплементира шаблона Итератор от самото начало.
- ✓ Както при шаблона Наблюдател Java предоставя вградена поддръжка на шаблона Итератор
  - Интерфейса java.util.Enumeration действа като интерфейс на итератор.
- ✓ Класовете, реализиращи съставни (aggregation) обекти, предоставят методи, които връщат референция към обект от тип Enumeration.
- ✓ Този обект имплементира Enumeration интерфейса, който позволява обхождане на съставния (aggregation) обект.
- ✓ Java 1.1 има ограничен брой класове, имплементиращи съставни обекти – Vector и Hashtable.
- ✓ Java 1.2 въвежда нов collections пакет с поддръжка на повече такива класове – set, list, map и интерфейса Iterator.

Интерфейса Enumeration

- ✓ Съответствия:
  - **`hasMoreElements`**`()` => **`IsDone`**`()`.
  - **`nextElement`**`()` => **`Next`**`()` последвано от **`CurrentItem`**`()`.
  - Няма **`First`**`()` – това се извършва автоматично когато се създаде `Enumeration` обекта.



Method Detail

hasMoreElements

```
boolean hasMoreElements()
```

Tests if this enumeration contains more elements.

Returns:

`true` if and only if this enumeration object contains at least one more element to provide; `false` otherwise.

nextElement

```
E nextElement()
```

Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Returns:

the next element of this enumeration.

Throws:

`NoSuchElementException` - if no more elements exist.

Пример с Enumeration

```
import java.util.*;

public class TestEnumeration {

  public static void main(String args[]) {
    // Create a Vector and add some items to it.
    Vector v = new Vector();
    v.addElement(new Integer(5));
    v.addElement(new Integer(9));
    v.addElement(new String("Hi, There!"));

    // Traverse the vector using an Enumeration.
    Enumeration ev = v.elements();
    System.out.println("\nVector values are:");
    traverse(ev);
```

Пример с Enumeration (продължение)

```
  // Now create a hash table and add some items to it.
  Hashtable h = new Hashtable();
  h.put("Bob", new Double(6.0));
  h.put("Joe", new Double(18.5));
  h.put("Fred", new Double(32.0));

  // Traverse the hash table keys using an Enumeration.
  Enumeration ekeys = h.keys();
  System.out.println("\nHash keys are:");
  traverse(ekeys);

  // Traverse the hash table values using an Enumeration.
  Enumeration evalues = h.elements();
  System.out.println("\nHash values are:");
  traverse(evalues);
}
```

Пример с Enumeration (продължение)

```
  private static void traverse(Enumeration e) {
    while (e.hasMoreElements()) {
      System.out.println(e.nextElement());
    }
  }

}
```

Пример с Enumeration (продължение)

- ✓ Изход на тестовата програма

```
Vector values are:
5
9
Hi, There!

Hash keys are:
Joe
Fred
Bob

Hash values are:
18.5
32.0
6.0
```

Vector Enumeration

- Поглед към **`Vector.java`**

```
public class Vector implements Cloneable, java.io.Serializable {
  ...
  protected Object elementData[];
  protected int elementCount;
  protected int capacityIncrement;

  public Vector(int initialCapacity, int capacityIncrement) {
    super();
    this.elementData = new Object[initialCapacity];
    this.capacityIncrement = capacityIncrement;
  }

  public Vector(int initialCapacity) {this(initialCapacity, 0);}
  public Vector() {this(10);}
```

Vector Enumeration (продължение)

- ✓ Поглед към **`Vector.java`**

```
  public final synchronized Enumeration elements() {
    return new VectorEnumerator(this);
  }

  public final synchronized void addElement(Object obj) {
    int newcount = elementCount + 1;
    if (newcount > elementData.length) {
      ensureCapacityHelper(newcount);
    }
    elementData[elementCount++] = obj;
  }
...
}
```

Vector Enumeration (продължение)

- ✓ Поглед към **Vector.java**

```
final class VectorEnumerator implements Enumeration {
  Vector vector;
  int count;

  VectorEnumerator(Vector v) {
    vector = v;
    count = 0;
  }

  public boolean hasMoreElements() {
    return count < vector.elementCount;
  }
```

Vector Enumeration (продължение)

- ✓ Поглед към **`Vector.java`**

```
    public Object nextElement() {
      synchronized (vector) {
        if (count < vector.elementCount) {
          return vector.elementData[count++];
        }
      }
      throw new NoSuchElementException("VectorEnumerator");
    }
  }
...
}
```

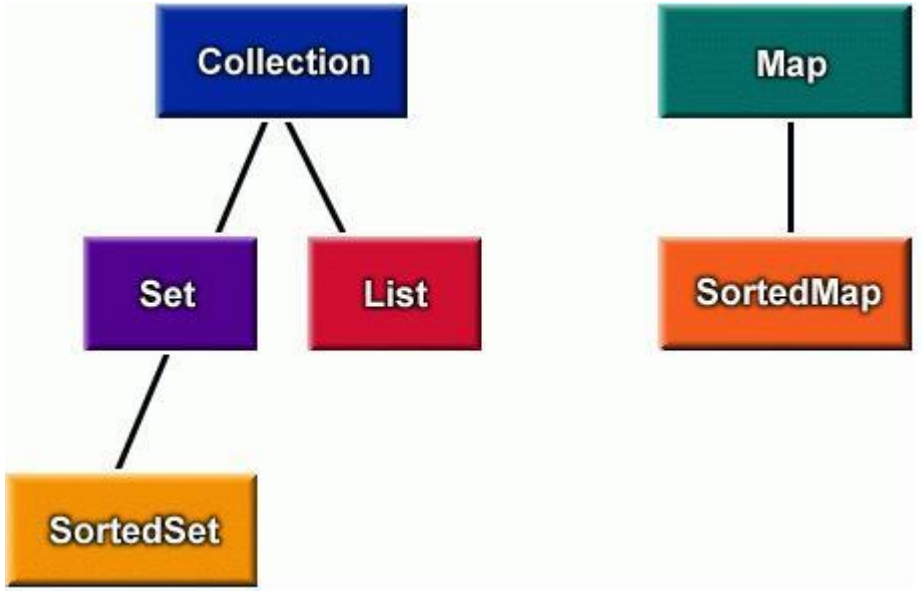
Vector Enumeration (заключение)

- ✓ Добавянето на нов елемент във Vector оказва ли влияние на Enumeration обекта?
  - Да – т.е. итератора, който се предоставя от Vector не е стабилен (robust).

Java Collections Framework

- ✓ JDK 1.2 въвежда нова рамка, която предоставя унифицирана архитектура за представяне и работа с колекции – Java Collections Framework.
- ✓ Рамката се състои от три неща:
  - Интрефейси: абстрактни типове от данни, които представят колекции.
  - Имплементации: конкретни имплементации на интерфейсите за колекции.
  - Алгоритми: методи, които имплементират полезни функции, като търсене и сортиране върху обекти, които имплементират интрефейсите на колекциите.

Интерфейси на Java Collections Framework



Интерфейса Collection

- ✓ Поглед към java.util.Collection

```
public interface Collection {
  int size();
  boolean isEmpty();
  boolean contains(Object element);
  boolean add(Object element);
  boolean remove(Object element);
  Iterator iterator();
  ...
}
```

Интерфейса Iterator

- ✓ Интерфейса Iterator е подобен на интерфейса Enumeration с две разлики:
  - Променени са имената на методите
    - hasNext() вместо hasMoreElements()
    - next() вместо nextElement()
  - Iterator е по-стабилен от enumeration, тъй като в определени случаи позволява премахването (и понякога добавянето) на елементи от/в колекцията по време на обхождане на структурата.
- ✓ Поглед към интерфейса java.util.Iterator

```
public interface Iterator {
  boolean hasNext();
  Object next();
  void remove();
}
```

Интерфейса List

- ✓ Поглед към java.util.List

```
public interface List extends Collection {
  Object get(int index);
  Object set(int index, Object element);
  void add(int index, Object element);
  Object remove(int index);
  boolean addAll(int index, Collection c);
  int indexOf(Object o);
  int lastIndexOf(Object);
  ListIterator listIterator();
  ListIterator listIterator(int index);
  List subList(int from, int to);
}
```

Интерфейса ListIterator

- ✓ Поглед към java.util.ListIterator

```
public interface ListIterator extends Iterator {
  boolean hasNext();
  Object next();
  boolean hasPrevious();
  Object previous();
  int nextIndex();
  int previousIndex();
  void remove();
  void set(Object o);
  void add(Object o);
}
```

Пример с LinkedList

- ✓ Класа java.util.LinkedList е конкретен клас, който имплементира List интерфейса
- ✓ Пример за използване на LinkedList

```
List list = new LinkedList();
...
ListIterator iterator = list.listIterator();
...
while (iterator.hasNext()) {
  Object item = iterator.next();
  // Обработка на item
}
```

Пример с LinkedList (продължение)

- ✓ Възможно е да се обхожда списъка и в обратна посока

```
List list = new LinkedList();
...
ListIterator iterator = list.listIterator(list.size());
...
while (iterator.hasPrevious()) {
  Object item = iterator.previous();
  // Обработка на item
}
```

Пример с LinkedList (продължение)

- ✓ Премахване на елементи по време на обхождането!

```
List list = new LinkedList();
...
ListIterator iterator = list.listIterator();
...
while (iterator.hasNext()) {
  Object item = iterator.next();
  // Ако елемента отговаря на определени
  // условия се премахва от колекцията, в
  // противен случай – се обработва.
  if (testOnItem(item))
    iterator.remove();
  else
    // Обработка на item
}
```

Пример с LinkedList (продължение)

- ✓ Методът remove() на итератора премахва последния елемент, който е бил върнат чрез извикване на next(). Методът remove() може да бъде извикан само веднъж за всяко извикване на next() и изхвърля изключение, ако това условие е нарушено.
- ✓ Единственият безопасен начин за модифициране на колекция по време на обхождане е чрез извикване на Iterator.remove(). Поведението не е дефинирано ако колекцията се модифицира по друг начин – всъщност се изхвърля `ConcurrentModificationException` в този случай.

Класа LinkedList

- ✓ Поглед върху LinkedList.java за да се онагледи как се постига стабилността.

```
/**
 * Linked list implementation of the List interface.
 * Note that this implementation is not synchronized. If multiple
 * threads access a list concurrently, and at least one of the
 * threads modifies the list structurally, it must be synchronized
 * externally. This is typically accomplished by synchronizing on
 * some object that naturally encapsulates the list or by wrapping
 * the list using the Collections.synchronizedList method:
 *   List list = Collections.synchronizedList(new LinkedList(...));
 */
public class LinkedList extends AbstractSequentialList
            implements List, Cloneable, java.io.Serializable {
  private transient Entry header = new Entry(null, null, null);
  private transient int size = 0;
  ...
```

Класа LinkedList (продължение)

```
/**
 * Returns a list-iterator of the elements in this list (in proper
 * sequence), starting at the specified position in the list.
 * The list-iterator is fail-fast: if the list is structurally
 * modified at any time after the Iterator is created, in any way
 * except through the list-iterator's own remove or add methods,
 * the list-iterator will throw a ConcurrentModificationException.
 * Thus, in the face of concurrent modification, the iterator
 * fails quickly and cleanly, rather than risking arbitrary,
 * non-deterministic behavior at an undetermined time in the
 * future.
 */
public ListIterator listIterator(int index) {
  return new ListItr(index);
}
```

Класа LinkedList (продължение)

```
/**
 * Private class that provides the implementation for the
 * iterator.
 */
private class ListItr implements ListIterator {
  private Entry lastReturned = header;
  private Entry next;
  private int nextIndex;
  private int expectedModCount = modCount;

  // Note: modCount is inherited from AbstractSequentialList.
  // It is incremented each time the list is modified.
  ...
  public boolean hasNext() {
    return nextIndex != size;
  }
```

Класа LinkedList (продължение)

```
public Object next() {
  checkForComodification();
  if (nextIndex == size)
    throw new NoSuchElementException();
  lastReturned = next;
  next = next.next;
  nextIndex++;
  return lastReturned.element;
}

final void checkForComodification() {
  if (modCount != expectedModCount)
    throw new ConcurrentModificationException();
}
```

Класа LinkedList (продължение)

```
    public void remove() {
      LinkedList.this.remove(lastReturned);
      if (next==lastReturned)
        next = lastReturned.next;
      else
        nextIndex--;
      lastReturned = header;
      expectedModCount++;
    }
  ...
  }
  ...
}
```

Край: Шаблон Итератор

ЛЕКЦИОНЕН КУРС: ШАБЛОНИ ЗА ПРОЕКТИРАНЕ