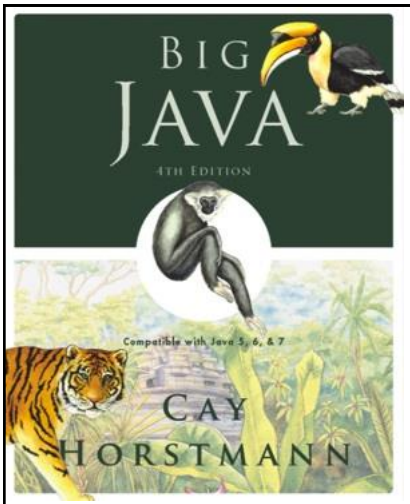


Kapitulli 15 – Hyrje ne strukture te dhenash

Pergatiti: Alda Kika

Qellimet e kapitullit

- Te mesohet se si te perdoren listat e lidhura te siguruara nga libraria standarte
- Te qenurit te afte te perdoren iteratoret per te kaluar listat e lidhura
- Te kuptohet implementimi i listave te lidhura
- Te dallohet ndermjet tipeve abstrakte dhe konkrete te tipeve te te dhenave
- Te njihet eficenca e operacioneve themelore te listave dhe tabelave
- Te qenurit familjare me stiven dhe rradhen

Perdorimi i listave te lidhura

- Nje liste e lidhur permban nje numer nyjesh, secila prej tyre ka nje reference tek nyja pasardhese
- Shtimi dhe heqja e elementeve ne mes te nje liste te lidhur eshte eficente
- Vizitimi i elementeve te nje liste te lidhur ne rendin sekuencial eshte eficent
- Aksesi random nuk eshte eficent

Shtimi i nje elementi ne nje liste te lidhur

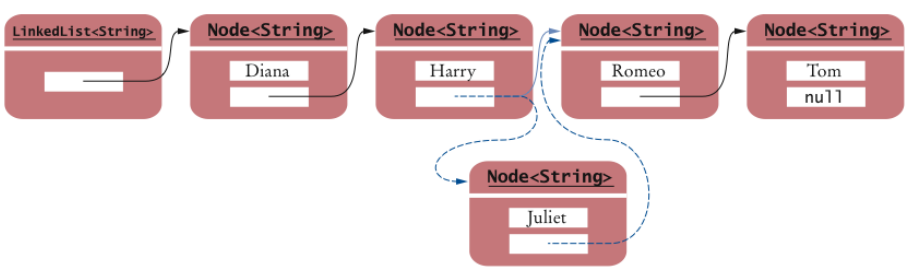


Figure 1 Inserting an Element into a Linked List

Klasa e Javes `LinkedList`

- Klase gjenerike
  - *Specifikoni tipin e elementeve brenda kllapave <>*
- Paketa: `java.util`

Table 1 `LinkedList` Methods

`LinkedList<String> lst = new LinkedList<String>();`	An empty list.
`lst.addLast("Harry")`	Adds an element to the end of the list. Same as `add`.
`lst.addFirst("Sally")`	Adds an element to the beginning of the list. `lst` is now `[Sally, Harry]`.
`lst.getFirst()`	Gets the element stored at the beginning of the list; here `"Sally"`.
`lst.getLast()`	Gets the element stored at the end of the list; here `"Harry"`.
`String removed = lst.removeFirst();`	Removes the first element of the list and returns it. `removed` is `"Sally"` and `lst` is `[Harry]`. Use `removeLast` to remove the last element.
`ListIterator<String> iter = lst.listIterator()`	Provides an iterator for visiting all list elements (see Table 2 on page 634).

List Iterator

- **`Tipi`** `ListIterator`
- Jep akses tek elementet brenda nje liste te lidhur
- Enkapsulon nje pozicion ne cdo vend brenda listes se lidhur
- Ruan listen e lidhur ndersa jep akses

List Iterator

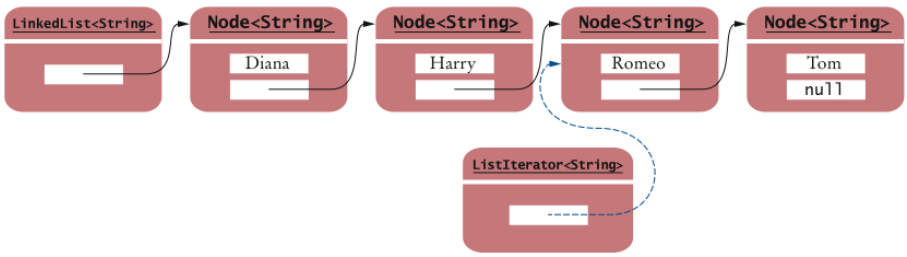


Figure 2 A List Iterator

Pamje konceptuale e List Iterator

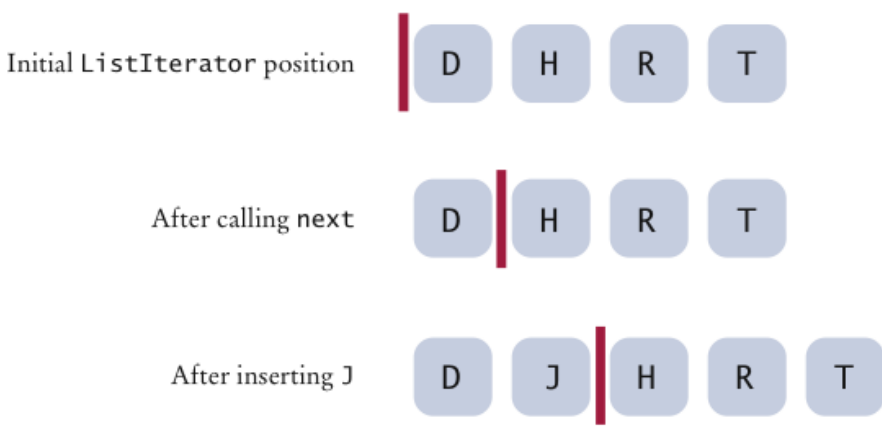


Figure 3 A Conceptual View of the List Iterator

List Iterator

- Mund te mendohet per nje iterator si shenjues ndermjet dy elementeve
  - *Analogji: Si kursori ne nje word procesor qe shenon ndermjet dy karaktereve*
- Metoda `listIterator` e klases `LinkedList` kthen nje list iterator

```
LinkedList<String> employeeNames = ...;
ListIterator<String> iterator =
   employeeNames.listIterator();
```

List Iterator

- Ne fillim, iteratori shenon perpara elementit te pare
- Metoda `next` leviz iteratorin:

```
iterator.next();
```

- `next` hedh nje `NoSuchElementException` nese jeni tashme pas fundit te listes
- `hasNext` kthen true nese eshte nje element next :

```
if (iterator.hasNext())
   iterator.next();
```

List Iterator

- Metoda `next` kthen elementin qe iteratori kalon:

```
while iterator.hasNext()
{
   String name = iterator.next();
   Bej dicka me name
}
```

```
for (String name : employeeNames)
{
   Bej dicka me name
}
```

Prapa skenes, lupa `for` perdor nje iterator per te vizituar te gjithe elementet e listes.

List Iterator

- `LinkedList` eshte nje liste dydrejtimore
  - *Klasa ruan dy lidhje:*
    - *Nje tek elementi pasardhes dhe*
    - *Nje tek elementi paraardhes*
- Per te levizur pozicionin e listes prapa, perdoret:
  - *hasPrevious*
  - *previous*

Shtimi dhe heqja nga `LinkedList`

- Metoda `add`:
  - *Shton nje objekt pas iterator-it*
  - *Leviz pozicionin e iterator-it pas pozicionit te elementit te ri:*

```
iterator.add("Juliet");
```

Shtimi dhe heqja nga nje `LinkedList`

- Metoda `remove`
  - *Heq dhe*
  - *Kthen objektin qe u kthye nga therritja e fundit nga* `next` *ose* `previous`

```
//Remove all names that fulfill a certain condition
while (iterator.hasNext())
{
    String name = iterator.next();
    if (name ploteson kushtin)
       iterator.remove();

}
```

Shtimi dhe heqja nga `LinkedList`

- Duhet patur kujdes kur therritet `remove`:
  - *Mund te therritet vetem nje here pas therritjes se `next` ose `previous`:*

```
iterator.next();
iterator.next();
iterator.remove();
iterator.remove();
// Error: You cannot call remove twice.
```

  - *Nuk mund ta therrisni ate menjehere pas therritjes se `add`:*

```
iter.add("Fred");
iter.remove(); // Error: Can only call remove after
               // calling next or previous
```

  - *Nese ju e therrisni ne menyre te papershtatshme, ai hedh nje `IllegalStateException`*

Metodat e nderfaqjes `ListIterator`

Table 2 Methods of the `ListIterator` Interface

`String s = iter.next();`	Assume that `iter` points to the beginning of the list `[Sally]` before calling `next`. After the call, `s` is `"Sally"` and the iterator points to the end.
`iter.hasNext()`	Returns `false` because the iterator is at the end of the collection.
`if (iter.hasPrevious()) { s = iter.previous(); }`	`hasPrevious` returns `true` because the iterator is not at the beginning of the list.
`iter.add("Diana");`	Adds an element before the iterator position. The list is now `[Diana, Sally]`.
`iter.next(); iter.remove();`	`remove` removes the last element returned by `next` or `previous`. The list is again `[Diana]`.

Programi

- `ListTester` eshte nje program qe
  - *Fut stringje ne nje liste*
  - *Leviz brenda listes, duke shtuar dhe hequr elemente*
  - *Printo listen*

ListTester.java

```
 1  import java.util.LinkedList;
 2  import java.util.ListIterator;
 3
 4  /**
 5     A program that tests the LinkedList class
 6  */
 7  public class ListTester
 8  {
 9     public static void main(String[] args)
10     {
11        LinkedList<String> staff = new LinkedList<String>();
12        staff.addLast("Diana");
13        staff.addLast("Harry");
14        staff.addLast("Romeo");
15        staff.addLast("Tom");
16
17        // | in the comments indicates the iterator position
18
19        ListIterator<String> iterator = staff.listIterator(); // |DHRT
20        iterator.next(); // D|HRT
21        iterator.next(); // DH|RT
22
```

Vazhdim

ListTester.java

```
23          // Add more elements after second element
24
25          iterator.add("Juliet"); // DHJ|RT
26          iterator.add("Nina"); // DHJN|RT
27
28          iterator.next(); // DHJNR|T
29
30          // Remove last traversed element
31
32          iterator.remove(); // DHJN|T
33
34          // Print all elements
35
36          for (String name : staff)
37             System.out.print(name + " ");
38          System.out.println();
39          System.out.println("Expected: Diana Harry Juliet Nina Tom");
40       }
41    }
```

Vazhdim

ListTester.java

Ekzekutimi i Programit:

```
Diana Harry Juliet Nina Tom
Expected: Diana Harry Juliet Nina Tom
```

Pyetje

A marrin me teper hapesire listat e lidhura se sa tabelelat me te njejten madhesi?

Pergjigje: Po. Keni nevoje te ruani referencat e nyjeve dhe cdo nyje eshte nje objekt me vete.

Pyetje

Pse nuk kemi nevoje per iteratore me tabelat?

Pergjigje: Nje indeks numer i plote mund te perdoret per te aksesuar cdo vendodhje tabele.

Implementimi i Listave te lidhura

- Seksioni i meparshem: Klasa ne Java `LinkedList`
- Duhet te implementojme nje version te thjeshtezuar te kesaj klase
- Ajo do te tregoje se si operacionet e listes manipulojne lidhjet ndersa lista modifikohet
- Per ta thjeshtezuar, do ta implementojme si nje liste nje drejtimore
  - *Klasa do te siguroje akses direkt vetem tek elementi i pare dhe jo tek i fundit*
- Lista nuk do te perdore nje tip parameter

Implementimi i Listave te lidhura

- `Node`: Ruan nje objekt dhe nje reference tek nyja pasardhese
- Metodat e klases se listes se lidhur dhe klases se iteratorit kane aksese frekuente tek variablat e instances `Node`
- Per ta bere me te thjeshte perdorimin:
  - *Nuk do ti bejme variablat e instances private*
  - *Ne bejme `Node` nje klase private te brendshme te `LinkedList`*
  - *Eshte e sigurt te lihet variablat e instances public*
    - *Asnje nga metodat e listes kthen nje objekt `Node`*

Implementimi i Listave te lidhura

```
public class LinkedList
{
   ...
   private class Node
   {
      public Object data;
      public Node next;
   }
}
```

Implementimi i Listave te lidhura

- Klasa `LinkedList`
  - *Mban nje reference `first` tek nyja e pare*
  - *Ka nje metode per te marre elementin e pare*

Implementimi i Listave te lidhura

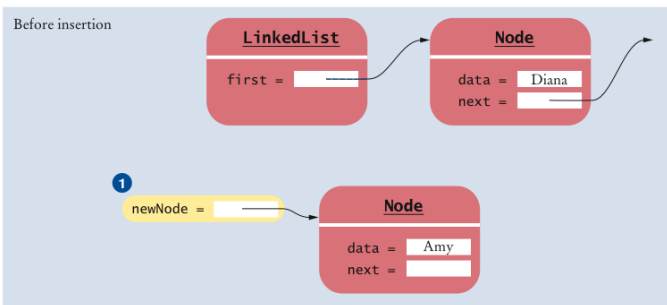
```
public class LinkedList
{
   private Node first;
   ...
   public LinkedList()
   {
      first = null;
   }
   public Object getFirst()
   {
      if (first == null)
         throw new NoSuchElementException();
      return first.data;
   }
}
```

Shtimi i elementit te pare

- Kur nje nyje e re shtohet ne liste
  - *Behet koka e listes*
  - *Koka e vjeter e listes behet nyja e tij pasardhese*

Shtimi i elementit te pare

```
public void addFirst(Object obj)
{
   Node newNode = new Node(); 1
   newNode.data = obj;
   newNode.next = first;
   first = newNode;
}
```



Shtimi i elementit te pare

```
public void addFirst(Object obj)
{
   Node newNode = new Node();
   newNode.data = obj;
   newNode.next = first; 2
   first = newNode; 3
}
```

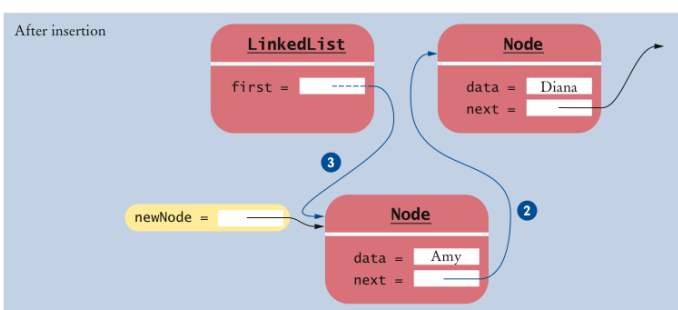


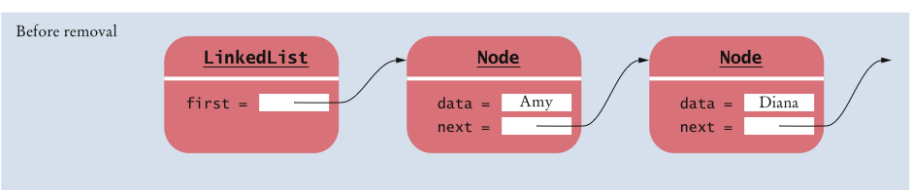
Figure 4 Adding a Node to the Head of a Linked List

Heqja e elementit te pare

- Kur elementi i pare hiqet
  - *E dhena e nyjes se pare ruhet dhe kthehet si rezultati i metodes*
  - *Pasardhesi i nyjes se pare behet nyja e pare e listes me te shkurter*
  - *Nyja e vjeter do te mblidhet nga objekti Garbage kur te mos kete me referenca ne te*

Heqja e elementit te pare

```
public Object removeFirst()
{
   if (first == null)
      throw new NoSuchElementException();
   Object obj = first.data;
   first = first.next;
   return obj;
}
```



Heqja e elementit te pare

```
public Object removeFirst()
{
   if (first == null)
      throw new NoSuchElementException();
   Object obj = first.data;
   first = first.next; 1
   return obj;
}
```

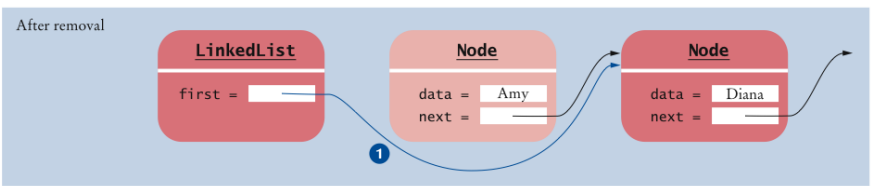


Figure 5 Removing the First Node from a Linked List

Linked List Iterator

Percaktojme `LinkedListIterator`: klase private e brendshme e `LinkedList`

- Implementoni nje nderfaqje te thjeshte `ListIterator`
- Ka akses ne fushen `first` dhe ne klasen private `Node`
- Klientat e `LinkedList` nuk e dine emrin e klases iterator
  - *Ato dine se eshte nje klase qe implementon nderfaqjen* `ListIterator`

LinkedListIterator

- Klasa `LinkListIterator` :

```
public class LinkedList
{
   ...
   public ListIterator listIterator()
   {
      return new LinkedListIterator();
   }

   private class LinkedListIterator implements
      ListIterator
   {
      private Node position;
      private Node previous;

       ...
```

Vazhdim

LinkedListIterator

```
      public LinkedListIterator()
      {
         position = null;
         previous = null;
      }
   }
   ...
}
```

Metoda `next` e Linked List Iterator

- `position`: referenca tek nyja e fundit e vizituar
- Gjithashtu, ruan nje reference tek referenca e fundit perpara asaj tek `previous`
- `next`: referenca `position` avancohet tek `position.next`
- pozicioni i vjeter ruhet tek `previous`
- Nese shenjuesi referon perpara elementit te pare te listes, atehere pozicioni i vjeter eshte `null` dhe `position` duhet te vendoset tek `first`

Metoda `next` e Linked List Iterator

```
public Object next()
{
   if (!hasNext())
      throw new NoSuchElementException();
   previous = position; // Remember for remove
   if (position == null)
      position = first;
   else position = position.next;
   return position.data;
}
```

Metoda `hasNext` e Linked List Iterator

- Metoda `next` duhet te therritet vetem kur iteratori nuk eshte ne fund te listes ose kur lista eshte bosh
- Metoda duhet te ktheje false
  - *Nese lista eshte bosh* `(first == null)`
  - *Nese nuk ka elemente pas pozicionit korrent* `(position.next == null)`

Metoda `hasNext` e Linked List Iterator

```
public boolean hasNext()
{
   if (position == null)
      return first != null;
   else
      return position.next != null;
}
```

Metoda `remove` e Linked List Iterator

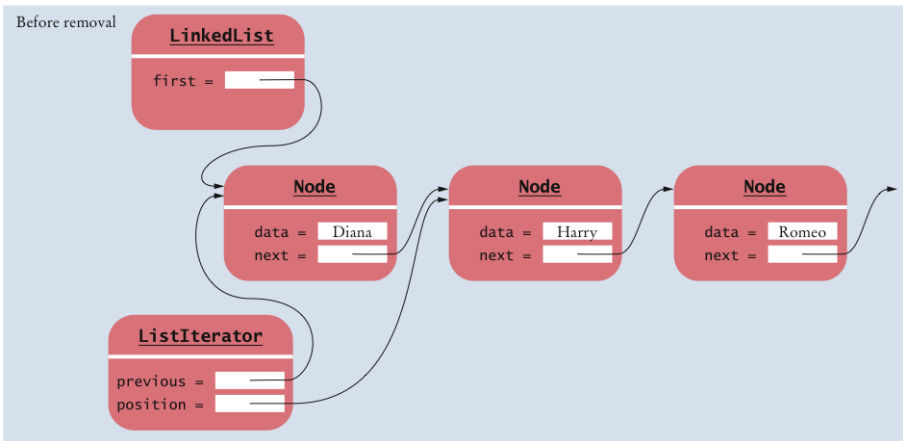
- Nese elementi qe do te hiqet eshte elementi i pare, atehere therritet `removeFirst`
- Perndryshe, nyja qe eshte perpara elementit per tu hequr ka nevoje te kete referencen e saj `next` te ndryshuar qe te kaloje elementin e hequr
- Nese referenca `previous` eshte e barabarte me `position`:
  - *Kjo thirrje nuk ndjek menjehere nje thirrje tek `next`*
  - *Hidhet nje `IllegalArgumentException`*
- Eshte jo legale te thirret `remove` dy here rresht
  - *remove vendos referencen previous tek position*

Metoda `remove` e Linked List Iterator

```
public void remove()
{
   if (previous == position)
      throw new IllegalStateException();
   if (position == first)
   {
      removeFirst();
   }
   else
   {
      previous.next = position.next;
   }
   position = previous;
}
```

Vazhdim

Metoda `remove` e Linked List Iterator



Vazhdim

Metoda `remove` e Linked List Iterator

```
public void remove()
{
   If (previous == position)
      throw new IllegalStateException();
   if (position == first)
   {
      removeFirst();
   }
   else
   {
      previous.next = position.next; (1)
   }
   position = previous; (2)
}
```

Vazhdim

Metoda `remove` e Linked List Iterator

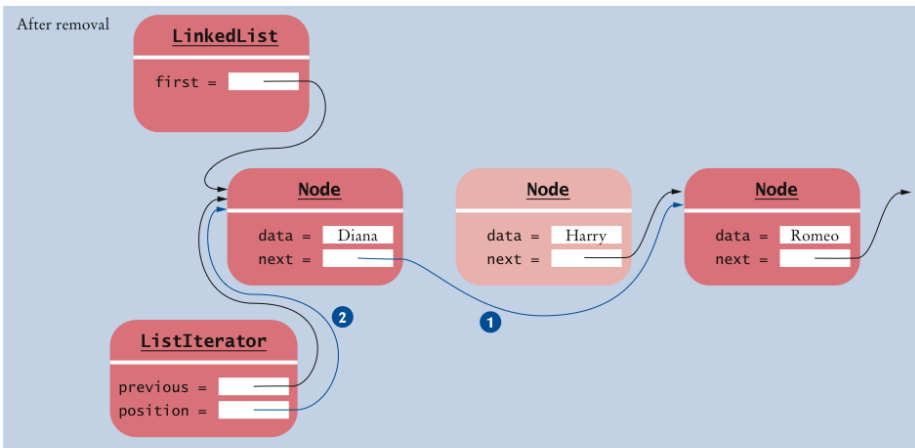


Figure 6 Removing a Node from the Middle of a Linked List

Metoda `set`e Linked List Iterator

- Ndryshon te dhenen qe ruhet ne nje element te vizituar me pare
- Metoda `set`

```
public void set(Object obj)
{
   if (position == null)
      throw new NoSuchElementException();
    position.data = obj;
}
```

Metoda `add` e Linked List Iterator

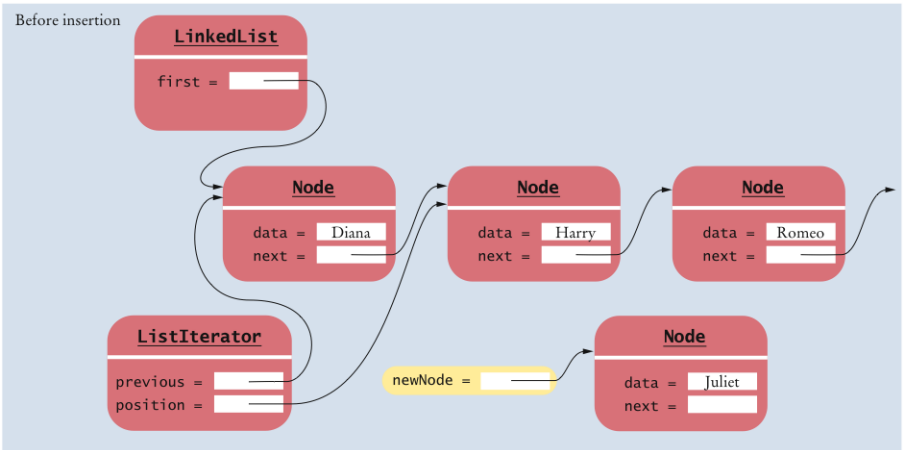
- Operacioni i shtimit te nje nyje
- `add` shton nje nyje te re pas pozicionit korrent
- Vendos pasardhesen e nyjes se re tek pasardhesja e pozicionit korrent

Metoda `add` e Linked List Iterator

```
public void add(Object obj)
{
   if (position == null)
   {
      addFirst(obj);
      position = first;
   }
   else
   {
      Node newNode = new Node();
      newNode.data = obj;
      newNode.next = position.next;
      position.next = newNode;
      position = newNode;
   }
   previous = position;
}
```

Vazhdim

Metoda `add` e Linked List Iterator



Vazhdim

Metoda `add` e Linked List Iterator

```
public void add(Object obj)
{
   if (position == null)
   {
      addFirst(obj);
      position = first;
   }
   else
   {
      Node newNode = new Node();
      newNode.data = obj;
      newNode.next = position.next;  (1)
      position.next = newNode;  (2)
      position = newNode;  (3)
   }
   previous = position;  (4)
}
```

Vazhdim

Metoda `add` e Linked List Iterator

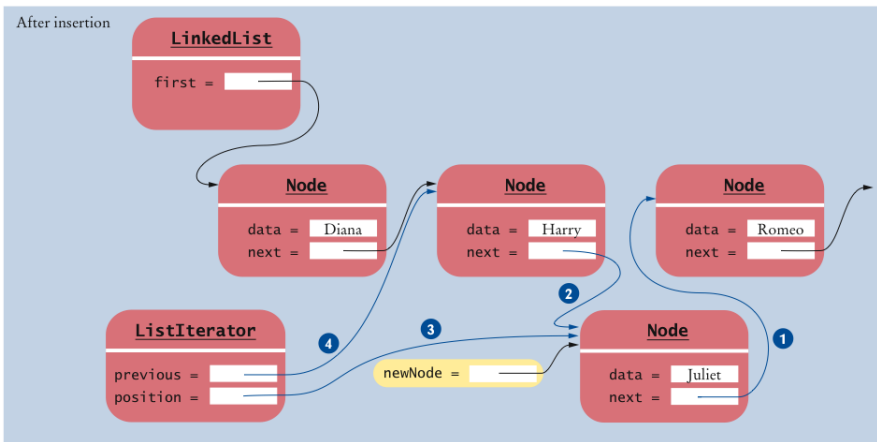


Figure 7 Adding a Node to the Middle of a Linked List

LinkedList.java

```
1  import java.util.NoSuchElementException;
2
3  /**
4     A linked list is a sequence of nodes with efficient
5     element insertion and removal. This class
6     contains a subset of the methods of the standard
7     java.util.LinkedList class.
8  */
9  public class LinkedList
10 {
11    private Node first;
12
13    /**
14       Constructs an empty linked list.
15    */
16    public LinkedList()
17    {
18       first = null;
19    }
20
```

Vazhdim

LinkedList.java

```
21      /**
22         Returns the first element in the linked list.
23         @return the first element in the linked list
24      */
25      public Object getFirst()
26      {
27         if (first == null)
28            throw new NoSuchElementException();
29         return first.data;
30      }
31
32      /**
33         Removes the first element in the linked list.
34         @return the removed element
35      */
36      public Object removeFirst()
37      {
38         if (first == null)
39            throw new NoSuchElementException();
40         Object element = first.data;
41         first = first.next;
42         return element;
43      }
44
```

Vazhdim

LinkedList.java

```
45      /**
46         Adds an element to the front of the linked list.
47         @param element the element to add
48      */
49      public void addFirst(Object element)
50      {
51         Node newNode = new Node();
52         newNode.data = element;
53         newNode.next = first;
54         first = newNode;
55      }
56
57      /**
58         Returns an iterator for iterating through this list.
59         @return an iterator for iterating through this list
60      */
61      public ListIterator listIterator()
62      {
63         return new LinkedListIterator();
64      }
65
```

Vazhdim

LinkedList.java

```
66      class Node
67      {
68         public Object data;
69         public Node next;
70      }
71
72      class LinkedListIterator implements ListIterator
73      {
74         private Node position;
75         private Node previous;
76
77         /**
78            Constructs an iterator that points to the front
79            of the linked list.
80         */
81         public LinkedListIterator()
82         {
83            position = null;
84            previous = null;
85         }
86
```

Vazhdim

LinkedList.java

```
87         /**
88            Moves the iterator past the next element.
89            @return the traversed element
90         */
91         public Object next()
92         {
93            if (!hasNext())
94               throw new NoSuchElementException();
95            previous = position; // Remember for remove
96
97            if (position == null)
98               position = first;
99            else
100              position = position.next;
101
102           return position.data;
103        }
104
```

Vazhdim

LinkedList.java

```
105         /**
106            Tests if there is an element after the iterator position.
107            @return true if there is an element after the iterator position
108         */
109         public boolean hasNext()
110         {
111            if (position == null)
112               return first != null;
113            else
114               return position.next != null;
115         }
116
```

Vazhdim

LinkedList.java

```
117         /**
118            Adds an element before the iterator position
119            and moves the iterator past the inserted element.
120            @param element the element to add
121         */
122         public void add(Object element)
123         {
124            if (position == null)
125            {
126               addFirst(element);
127               position = first;
128            }
129            else
130            {
131               Node newNode = new Node();
132               newNode.data = element;
133               newNode.next = position.next;
134               position.next = newNode;
135               position = newNode;
136            }
137            previous = position;
138         }
139
```

Vazhdim

LinkedList.java

```
140         /**
141            Removes the last traversed element. This method may
142            only be called after a call to the next() method.
143         */
144         public void remove()
145         {
146            if (previous == position)
147               throw new IllegalStateException();
148
149            if (position == first)
150            {
151               removeFirst();
152            }
153            else
154            {
155               previous.next = position.next;
156            }
157            position = previous;
158         }
159
```

Vazhdim

LinkedList.java

```
160         /**
161            Sets the last traversed element to a different value.
162            @param element the element to set
163         */
164         public void set(Object element)
165         {
166            if (position == null)
167               throw new NoSuchElementException();
168            position.data = element;
169         }
170      }
171   }
```

ListIterator.java

```
1  /**
2     A list iterator allows access of a position in a linked list.
3     This interface contains a subset of the methods of the
4     standard java.util.ListIterator interface. The methods for
5     backward traversal are not included.
6  */
7  public interface ListIterator
8  {
9     /**
10       Moves the iterator past the next element.
11       @return the traversed element
12    */
13    Object next();
14
15    /**
16       Tests if there is an element after the iterator position.
17       @return true if there is an element after the iterator position
18    */
19    boolean hasNext();
20
```

Vazhdim

ListIterator.java

```
21    /**
22       Adds an element before the iterator position
23       and moves the iterator past the inserted element.
24       @param element the element to add
25    */
26    void add(Object element);
27
28    /**
29       Removes the last traversed element. This method may
30       only be called after a call to the next() method.
31    */
32    void remove();
33
34    /**
35       Sets the last traversed element to a different value.
36       @param element the element to set
37    */
38    void set(Object element);
39 }
```

Pyetje

Gjurmoni nepermjet metodes `addFirst` kur shtohet elementi ne nje liste te lidhur bosh.

Pergjigje: Kur lista eshte bosh, `first` eshte `null`. Nje nyje e re `Node` alokohet. Variabli i instances `data` vendoset ne nje objekt te ri te futur. Variabli I instances `next` vendoset ne `null` sepse `first` eshte `null`. Instanca e variablit `first` vendoset ne nyjen e re. Rezultati eshte nje liste e lidhur me gjatesi 1.

Pyetje

Pse metoda `add` ka dy raste te ndara?

Pergjigje: Nese pozicioni eshte `null`, duhet te jemi ne krye te listes, dhe futja e nje elementi kerkon ndryshimin e references `first`. Nese jemi ne mes te listes, referenca `first` nuk duhet ndryshuar.

Tipet abstrakte te te dhenave

- Ka dy menyra te parit tek nje liste e lidhur
  - *Te menduarit e nje implementimi konkret te nje liste te tille*
    - Sekuenca e objekteve nyje me lidhje ndermjet tyre
  - *Mendoni per konceptin abstrakt te nje liste te lidhur*
    - Sekuenca e renditur e elementeve te te dhenave qe mund te bridhet nepermjet nje iteratori

Tipet abstrakte te te dhenave

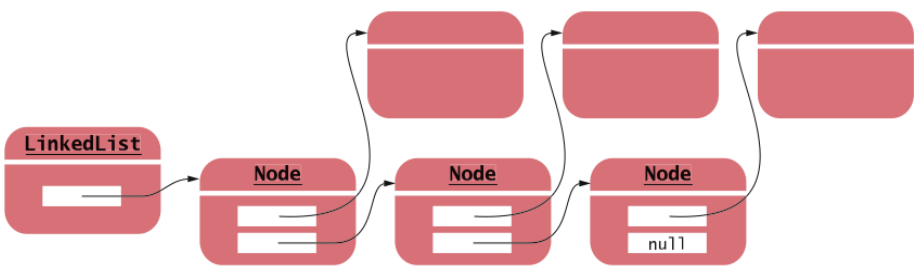


Figure 8 A Concrete View of a Linked List

Figure 9 An Abstract View of a List

Tipet abstrakte te te dhenave

- Percaktoni operacionet themelore te te dhenave
- Mos specifikoni nje implementim

Tipet konkrete dhe abstrakte te tabelave

- Si liste e lidhur jane dy menyra per te pare nje tabele
- Implementim konkret: nje tabele e mbushur pjeserisht me referenca objektesh
- Ne zakonisht nuk mendojme rreth implementimit konkret kur perdoret nje array list
  - *Marrim kendveshtrimin abstrakt*
- Kendveshtrimi abstrakt: Sekuence e renditur e elementeve te te dhenave, secila prej tyre mund te aksesohet nepermjet nje indeksi numer i plote

Tipi abstrakt dhe konkret i tipeve te te dhenave

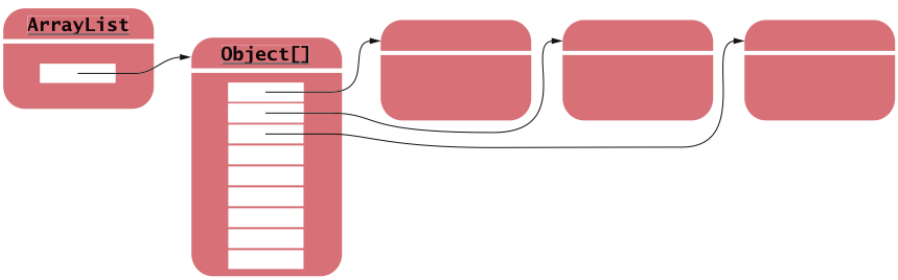


Figure 10 A Concrete View of an Array List

Figure 11 An Abstract View of an Array

Tipi abstrakt dhe konkret i tipeve te te dhenave

- Implementimi konkret e nje liste te lidhur dhe nje liste tabele jane te ndryshme
- Abstraksioni duket sikur eshte i ngjashem fillimisht
- Per te pare diferencen, konsideroni nderfaqjen publike me minimumin e asaj qe eshte e nevojshme

Operacionet themelore ne nje Array List

Nje array list lejon akses *random per te gjitha* elementet:

```
public class ArrayList
{
   public Object get(int index) {...}
   public void set(int index, Object value) {...)
   ...
}
```

Operacionet themelore ne nje Array List

Nje liste e lidhur lejon aksesin sekuencial drejt elementeve te tij:

```
public class LinkedList
{
   public ListIterator listIterator() {...}
   ...
}

public interface ListIterator
{
   Object next();
   boolean hasNext();
   void add(Object value);
   void remove();
   void set(Object value);
   ...
}
```

Tipet Abstrakte te te dhenave

- `ArrayList`: Kombinon nderfaqjet e nje tabele dhe nje liste
- Te dyja `ArrayList` dhe `LinkedList` implementojne nje nderfaqje te quajtur `List`
  - `List` *percakton operacionet per akses random dhe akses sekuencial*
- Terminologjia nuk eshte ne perdorim te pergjithshem jashte librarise Java
- Terminologji traditicionale: *array* dhe *list*
- Libraria Java siguron implementim konkret te `ArrayList` dhe `LinkedList` per keto tipe abstrakte te te dhenave
- Tabelat Java jane nje tjeter implementim i tipit abstrakt te tabeles

Eficenca e Operacioneve per tabelat dhe listat

- Shtimi ose heqja e elementeve(lista)
  - *Nje numer fiks i referencave qe kane nevoje per tu modifikuar per te shtuar ose hequr nje nyje pavaresisht madhesise se listes*
  - *Ne nocionin big-Oh : O(1)*
- Shtimi ose heqja e elementeve(tabela)
  - *Mesatarisht n/2 elementa nevojiten te levizen*
  - *Ne nocionin big-Oh : O(n)*

Effcenca e Operacioneve per tabelat dhe listat

Operacioni	Tabela	Lista
Akses Random	*O*(1)	*O*(*n*)
Hapi i bredhjes Lineare	*O*(1)	*O*(1)
Shtimi/heqja e nje elementi	*O*(*n*)	*O*(1)

Tipet abstrakte te te dhenave

- Lista Abstrakte
  - *Sekuence e renditur e elementeve qe mund te bridhet ne menyre sekuenciale*
  - *Lejon futjen dhe heqjen e elementeve ne cdo pozicion*
- Tabela abstrakte
  - *Sekuence e renditur e elementeve me akses random nepermjet nje indeksi numer i plote*

Pyetje

Cili eshte avantazhi i te parit te tipit abstrakt?

Pergjigje: Mund te fokusoheni ne karakteristikat esenciale te te dhenave pa u shqetesuar per detajet e implementimit.

Pyetje

Sa me i ngadalshem eshte algoritmi i kerkimit binar per nje liste te lidhur duke e krahasuar me algoritmin e kerkimit linear?

Pergjigje: Per te percaktuar elementin e mesit duhen n / 2 hapa. Per te percaktuar mesin e nenintervalit majtas apo djathtas duhen dhe n / 4 hapa. Shikimi tjeter merr n / 8 hapa. Pra presim , pothuaj n hapa per te percaktuar elementin. Ne kete pike, me mire mund te behet kerkimi linear qe kerkon mesatarisht n / 2 hapa.

Stivat dhe rradhet

- Stive: koleksion me elemente me terheqje “last in, first out”
- Rradhe: koleksion me elemente me terheqje : “first in, first out”

Stive

- Lejon futjen dhe terheqjen e elementeve vetem ne nje fund
  - *Tradicionalisht quhet maja e stives*
- Elementet e rinj shtohet ne maje te stives
- Elementet hiqen nga koka e stives
- E quajtur *last in, first out* ose rendi LIFO
- Tradicionalisht, operacionet e mbledhjes dhe heqjes quhen `push` dhe `pop`

Stiva

- Mund te mendohet si nje stive librash

Figure 12
A Stack of Books

Rradha

- Shtohen elementet ne nje fund te rradhes
- Hiqen elementet nga koka
- Rradha ruan elementet si nje *first in, first out* ose FIFO
- Elementet hiqen me te njejten rradhe sic jane shtuar
- Mund te merret shembulli i njerezve ne rradhe:
  - *Njerezit i bashkengjiten bishtit te rradhes dhe presin derisa ato te kene arritur fillimin (koken) e rradhes.*

Rradha

Figure 13 A Queue

Stivat dhe rradhat: Perdorimet ne shkencat kompjuterike

- Rradha
  - *Rradha e ngjarjeve, e mbajtur nga Java GUI system*
  - *Rradha e puneve te printerit*
- Stiva
  - *Stive e kohes se ekzekutimit qe nje procesor apo makine virtuale mban per te organizuar variabla e metodave te nderfutura*

Stiva dhe rradha ne librarine Java

- Klasa `Stack` implementon tipin abstrakt te te dhenes dhe operacionet `push` dhe `pop`
- Metodat e nderfaqjes `Queue` ne librarine standarte Java perfshijne:
  - `add` *per te shtuar nje element ne bisht te rradhes*
  - `remove` *per te hequr koken e rradhes*
  - `peek` *per te marre elementin qe ndodhet ne koke pa e hequr ate*
- Implementimet e `Queue` ne librarine standarte dizenjohen per tu perdorur me programet multithreaded
- Klasa `LinkedList` gjithashtu implementon nderfaqjen `Queue` dhe ju mund ta perdorni ate kur kerkohet nje rradhe:

```
Queue<String> q = new LinkedList<String>();
```

Puna me rradhet dhe stivat

Table 4 Working with Queues and Stacks

`Queue<Integer> q = new LinkedList<Integer>();`	The `LinkedList` class implements the `Queue` interface.
`q.add(1); q.add(2); q.add(3);`	Adds to the tail of the queue; `q` is now `[1, 2, 3]`.
`int head = q.remove();`	Removes the head of the queue; `head` is set to 1 and `q` is `[2, 3]`.
`head = q.peek();`	Gets the head of the queue without removing it; `head` is set to 2.
`Stack<Integer> s = new Stack<Integer>();`	Constructs an empty stack.
`s.push(1); s.push(2); s.push(3);`	Adds to the top of the stack; `s` is now `[1, 2, 3]`.
`int top = s.pop();`	Removes the top of the stack; `top` is set to 3 and `s` is now `[1, 2]`.
`head = s.peek();`	Gets the top of the stack without removing it; `head` is set to 2.

Pyetje

Vizatoni nje skice te nje tipi abstrakt rradhe, e ngjashme me ate te Figures 9 dhe 11.

Pergjigje:

Pyetje

Pse nuk do te donit te perdorni nje stive per te menaxhuar punet e printerit?

Pergjigje: Stiva perdor nje disipline te tipit “last in, first out. Nese ju jeni i pari qe vendosni nje pune print dhe shume njerez shtojne detyra te tipit print perpara se printeri te merret me detyren qe ju keni vendosur, ato marrin printimet e tyre perpara dhe ju duhet te prisni derisa te gjitha detyrat te jene perfunduar.