F12 - Collections

ID1004 Objektorienterad programmering

Fredrik Kilander fki@kth.se

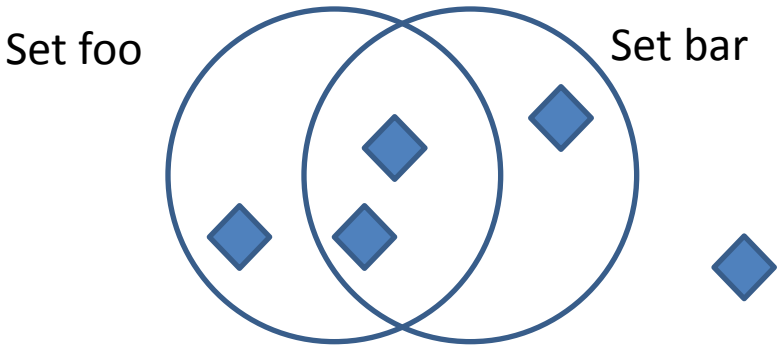
Collections (samlingar)

- En collection är ett objekt som fungerar som en samling av andra objekt

- En collection erbjuder vanligen
  - Lägg till objekt (add, put)
  - Ta bort objekt (remove, clear)
  - Hitta objekt (contains, indexOf, get)

Collections (samlingar)

- En instans ur en collection-klass har en egen datamängd av *referenser*
- Add, remove, m.m. hanterar endast referenser
- Olika instanser av collection kan därför samtidigt hantera samma objekt



Collections (samlingar)

- En collection är en samling data och operationer på dessa data
- Tillsammans utgör de en *abstrakt datatyp*

- Hur samlingen hanterar sitt innehåll är dolt (inkapslat)
- Samlingen har ett väldefinierat *gränssnitt*

Interface java.util.Collection

- add(E e)
- addAll(Collection)
- clear()
- contains(Object o)
- containsAll(Collection)
- equals(Object o)
- hashCode()
- isEmpty()
- iterator()
- remove(Object o)
- removeAll(Collection)
- retainAll(Collection)
- size()
- toArray()
- toArray(T[] a)

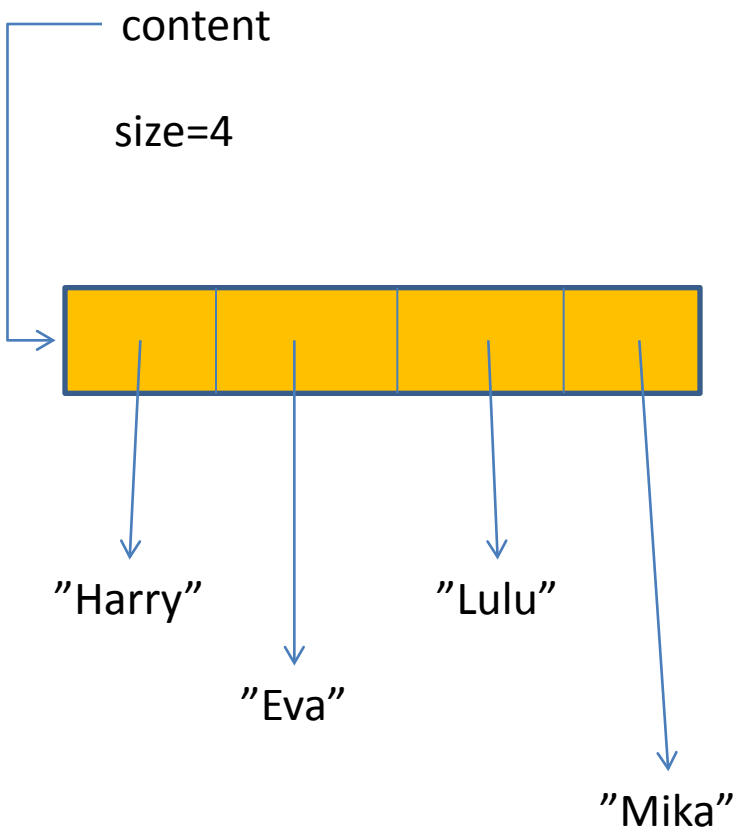
Interface och implementationer (java.util)

- Interface Collection och dess sub-interface
  - BlockingDeque
  - BlockingQueue
  - Deque
  - List
  - NavigableSet
  - Queue
  - Set
  - SortedSet

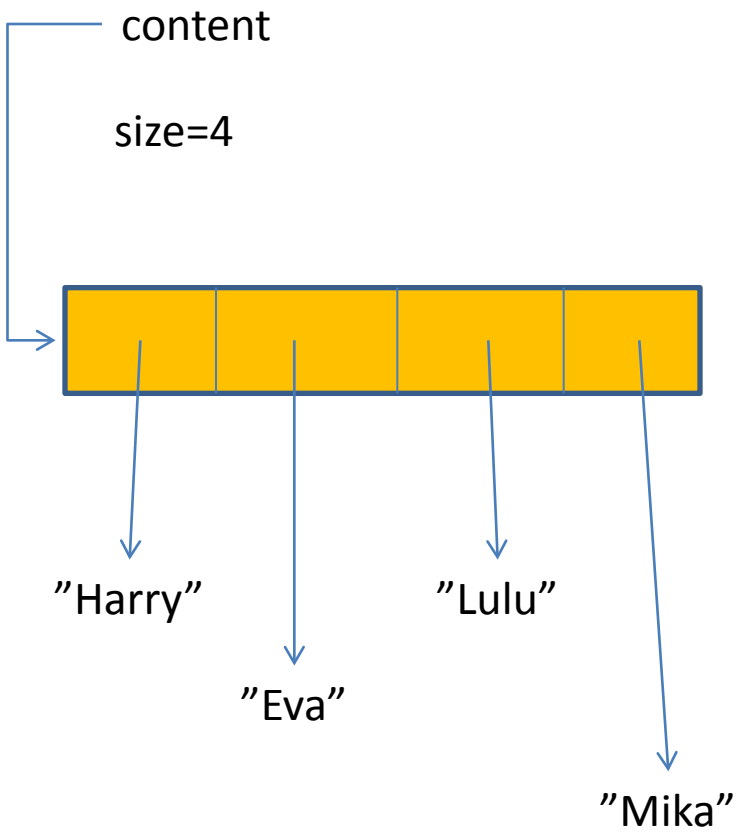
ArrayList

- ArrayList är en samling baserad på en intern array
- Objekt som adderas placeras i arrayen
- Om arrayen tar slut skapas en ny, längre array

ArrayList

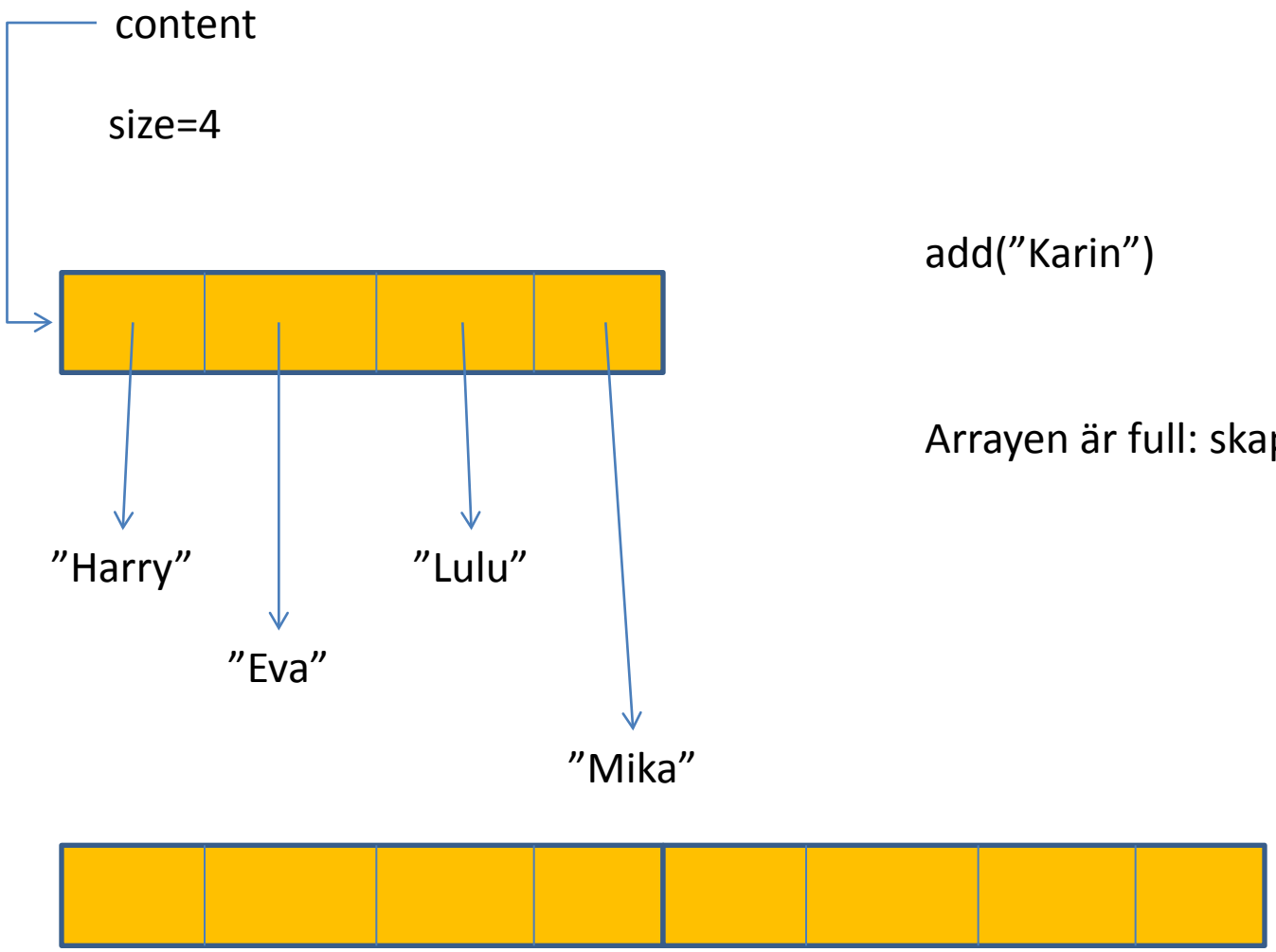


ArrayList



add(”Karin”)

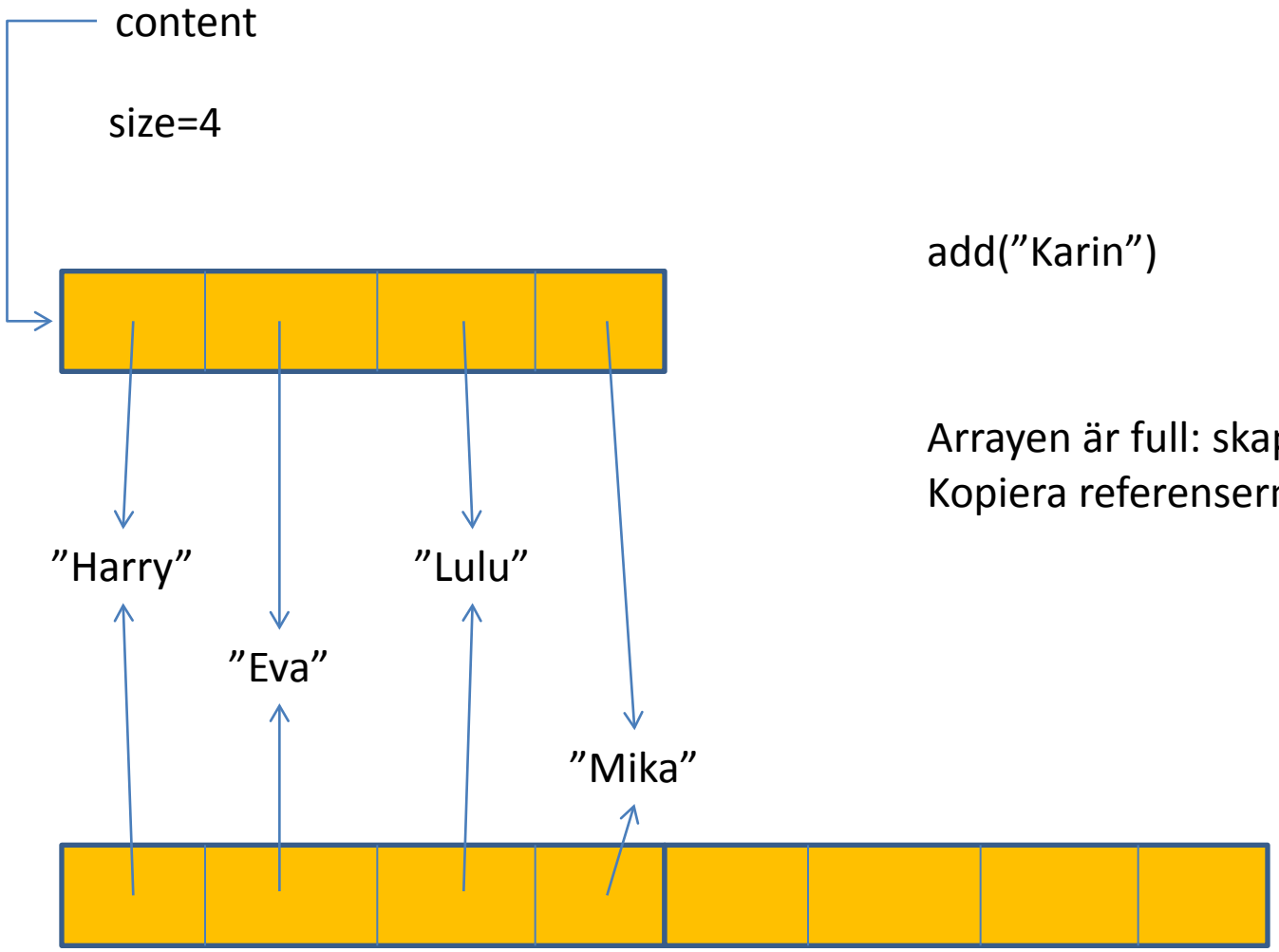
ArrayList



add(”Karin”)

Arrayen är full: skapa ny array

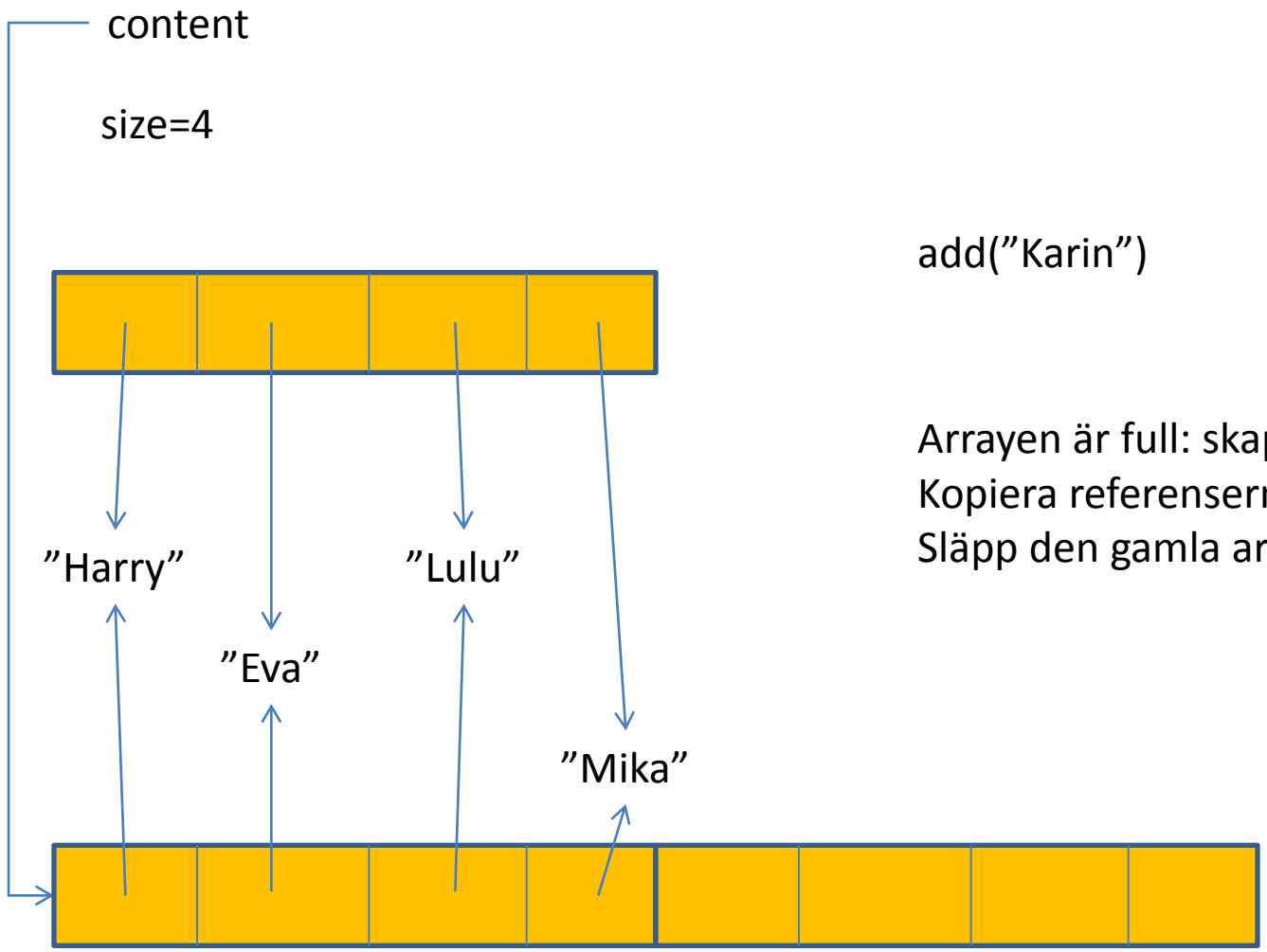
ArrayList



add(”Karin”)

Arrayen är full: skapa ny array
Kopiera referenserna

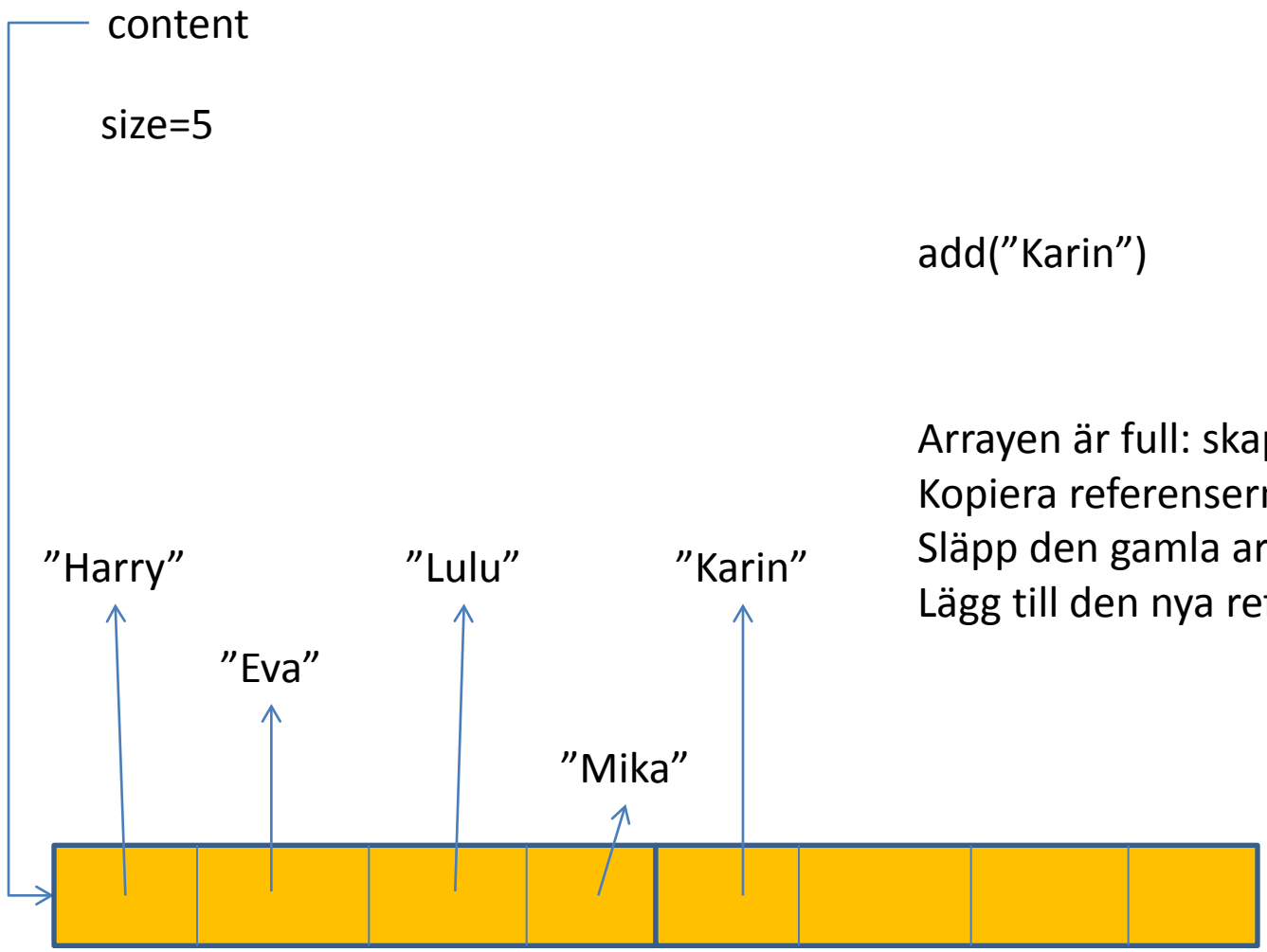
ArrayList



add(”Karin”)

Arrayen är full: skapa ny array
Kopiera referenserna
Släpp den gamla arrayen

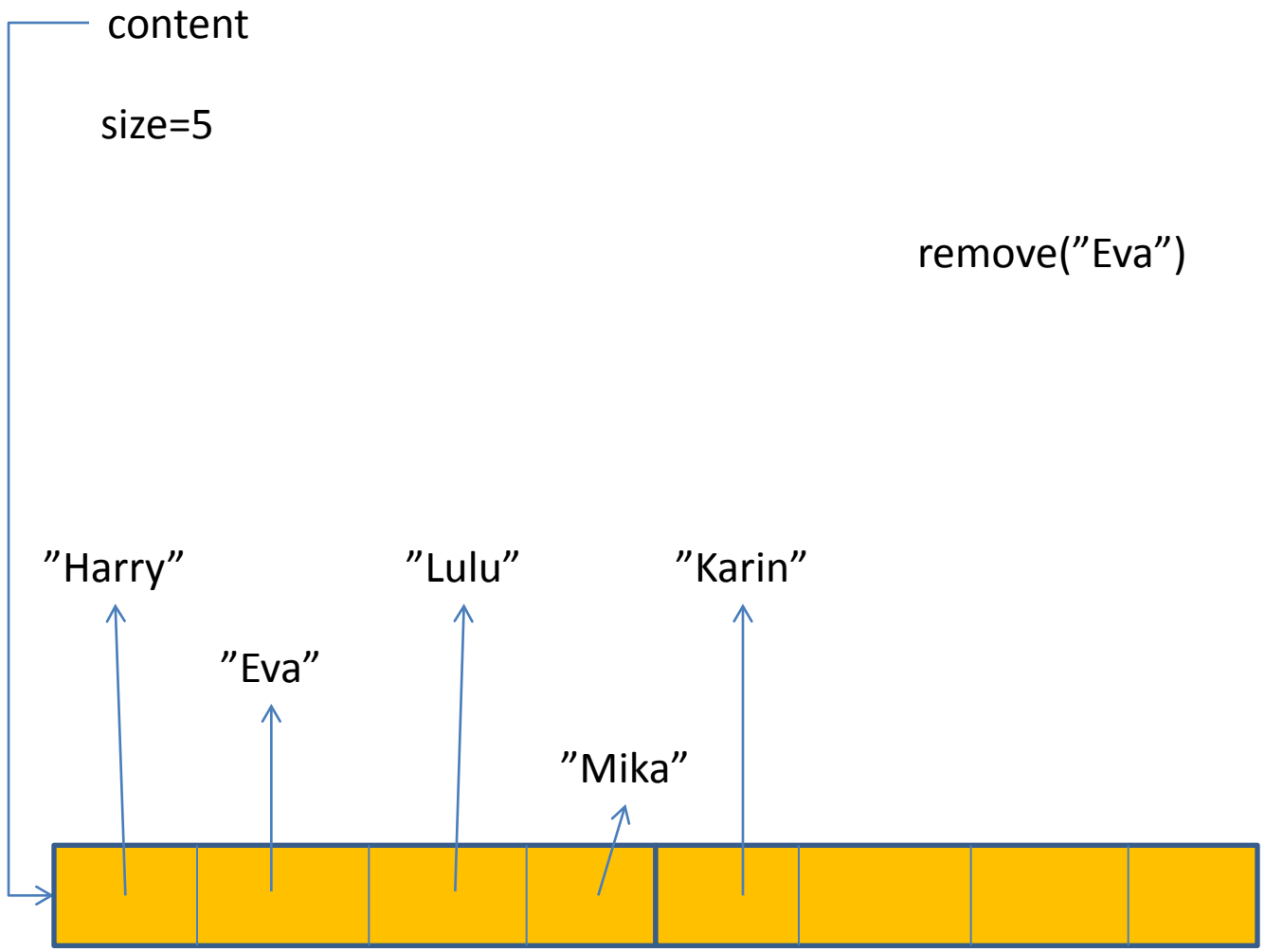
ArrayList



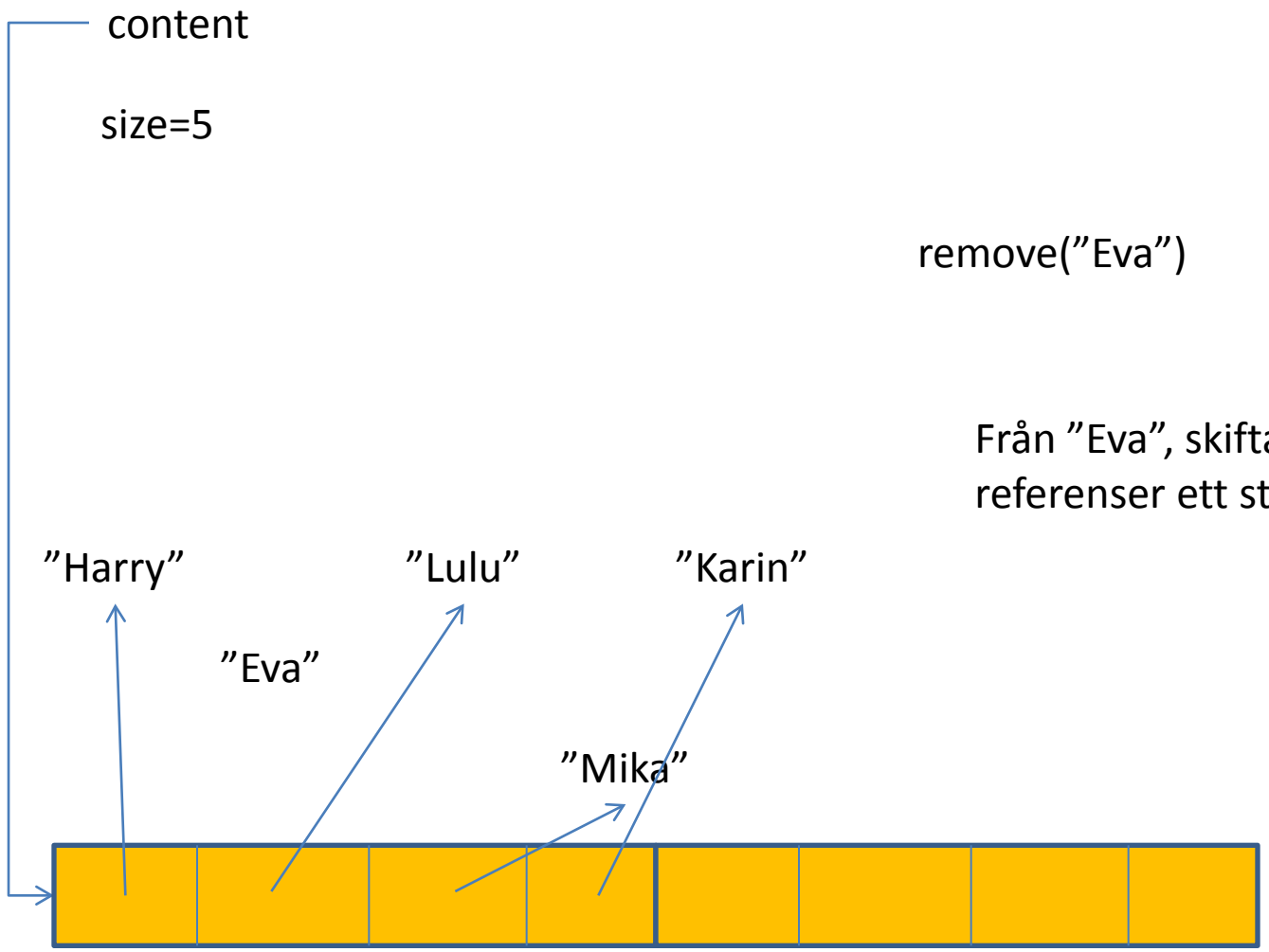
add(”Karin”)

Arrayen är full: skapa ny array
Kopiera referenserna
Släpp den gamla arrayen
Lägg till den nya referensen

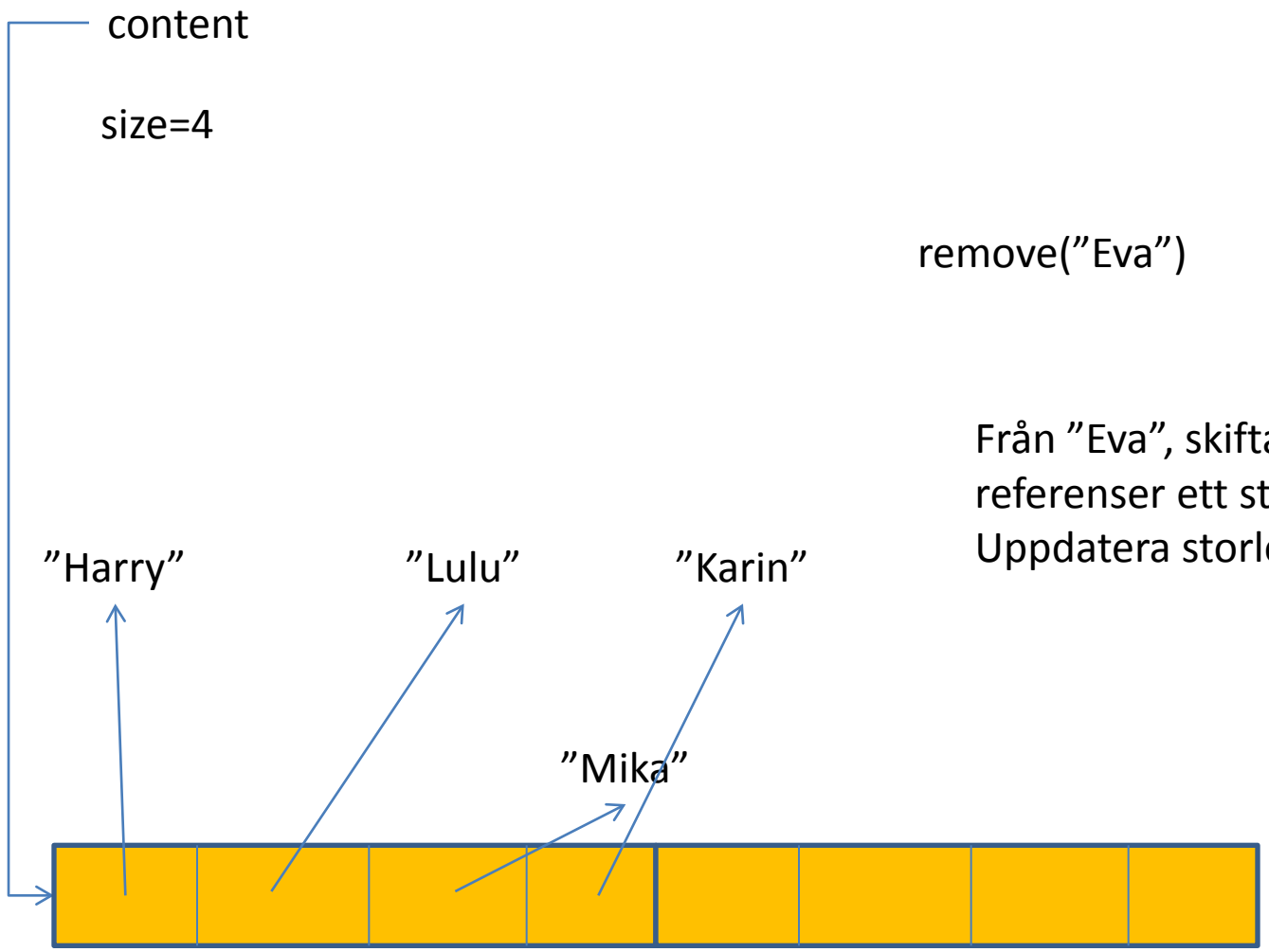
ArrayList



ArrayList


content
size=5
remove(”Eva”)
Från ”Eva”, skifta alla referenser ett steg
”Harry”
”Lulu”
”Karin”
”Eva”
”Mika”

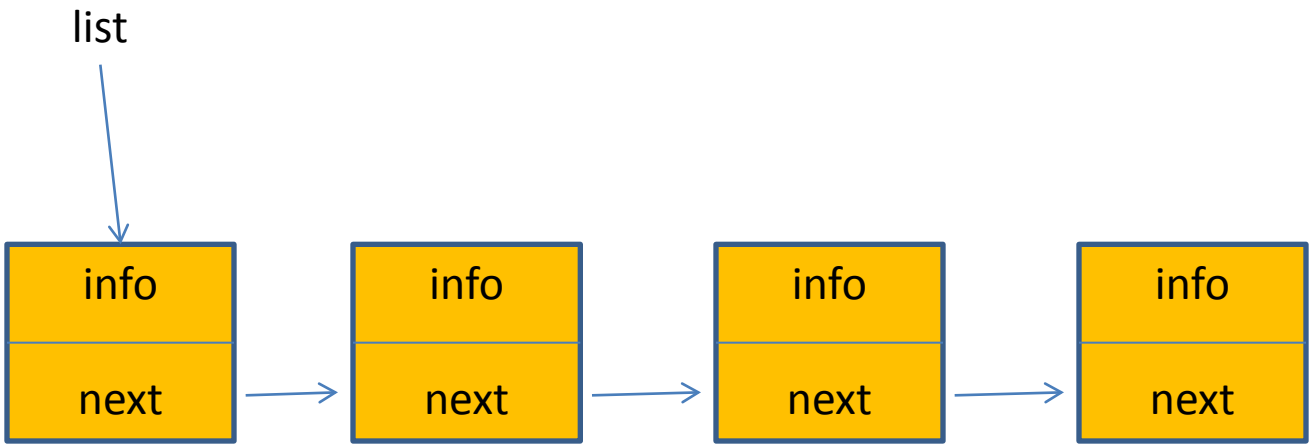
ArrayList


content
size=4
remove(”Eva”)
Från ”Eva”, skifta alla
referenser ett steg
Uppdatera storleken
”Harry”
”Lulu”
”Karin”
”Mika”

ArrayList

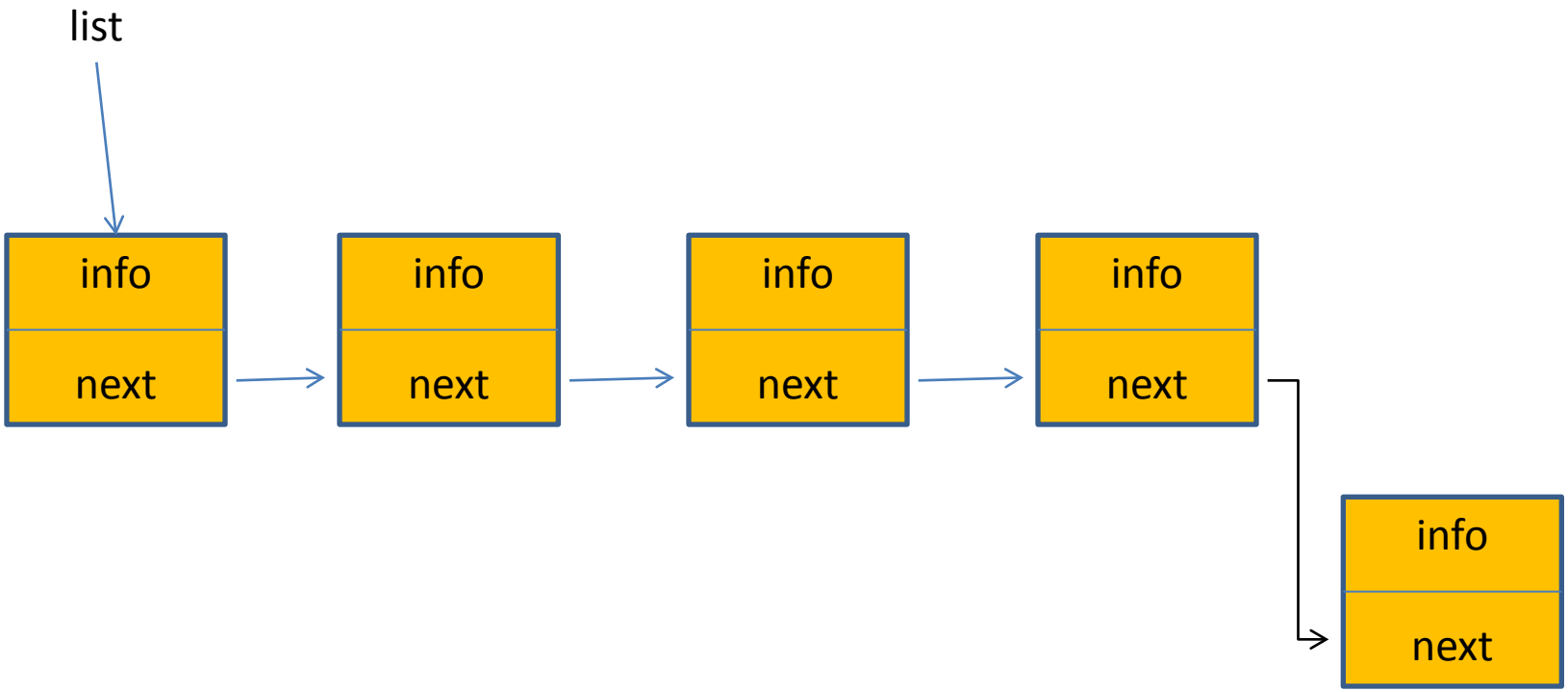
- Snabbt tillägg till slutet av listan
- Arrayen behöver inte växa så ofta
- Borttag kan vara kostsamma
  - om listan är lång
  - om det sökta elementet är i början av listan

Länkad lista

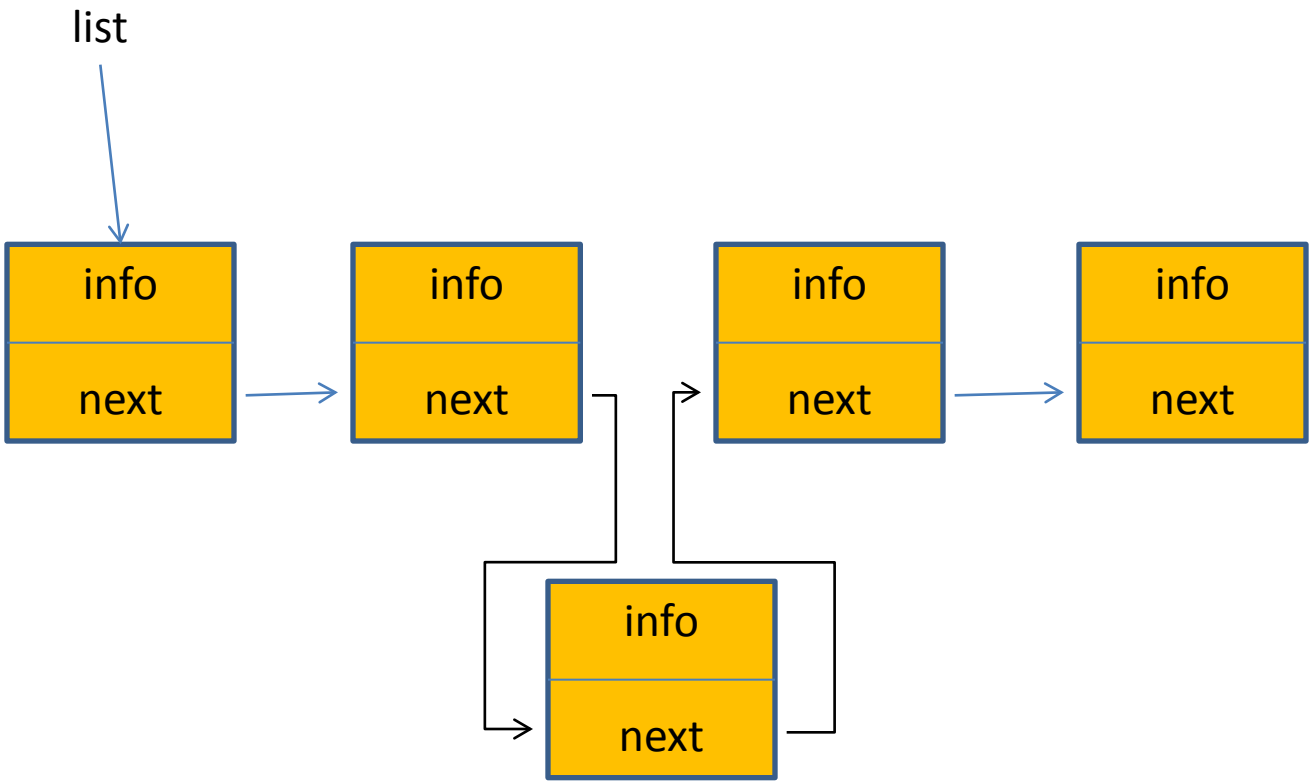



```
public class Node {
  Object info;
  Node next;
}
```

Länkad lista – add (i slutet)



Länkad lista – add (i mitten)



Länkad lista - add

```
public class Node {
  Object info;
  Node next;
}

public class LinkedList {
  private Node head;

  protected void addLast(Node nd) {
    for (Node search = head;
         search.next != null; search = search.next);
    nd.next = search.next;
    search.next = nd;
  }
}
```

Sök upp det sista elementet (next == null)

Länkad lista - add

```
public class Node {
  Object info;
  Node next;
}
public class LinkedList {
  private Node head;

  protected void addBefore(Node nd, Node before) {
    for (Node search = head;
         search.next != before; search = search.next);
    nd.next = search.next;
    search.next = nd;
  }

  protected void addLast(Node nd) {
    addBefore(nd, null);
  }
}
```

Sök upp ett givet element (referens), sätt in före det

Länkad lista - add

```
public class Node {
  Object info;
  Node next;
}
public class LinkedList {
  private Node head;

  protected void addAfter(Node nd, Node after) {
    for (Node search = head;
         search != after; search = search.next);
    nd.next = search.next;
    search.next = nd;
  }

}
```

Sök upp ett givet element, stoppa in efter det

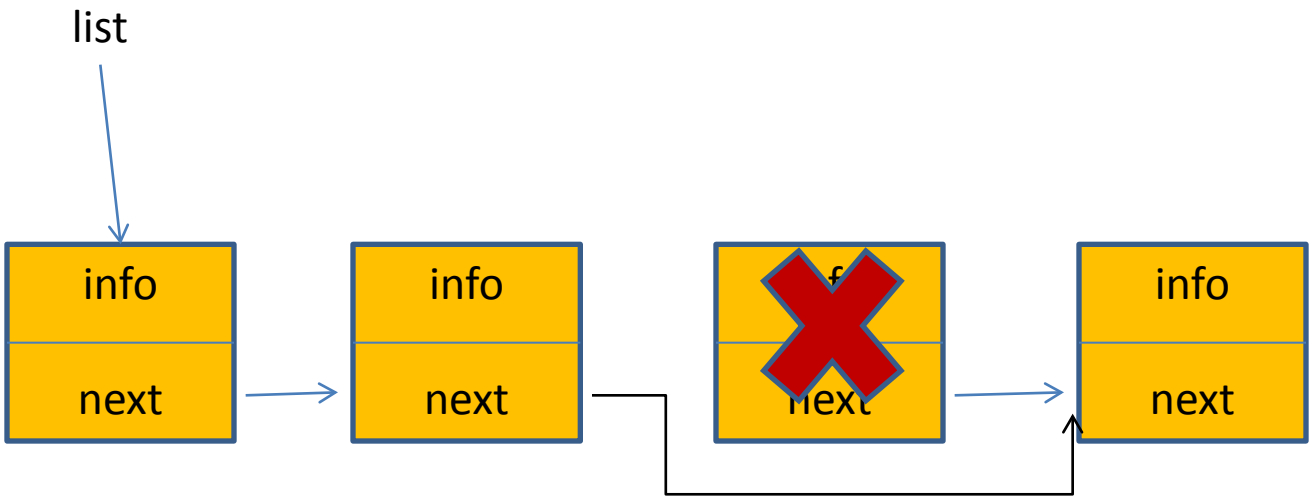
Länkad lista - add

```
public class Node {
  Object info;
  Node next;
  public Node (Object o) {info = o;}
}
public class LinkedList {
  private Node head;

  protected void add (Node nd) {
    if (head == null) {
      head = nd;
    } else {
      addAfter(nd, head);
    }
  }

  public void add(Object o) {
    add (new Node(o));
  }
}
```

Internt hanterar vi noder

Utåt hanterar vi objekt

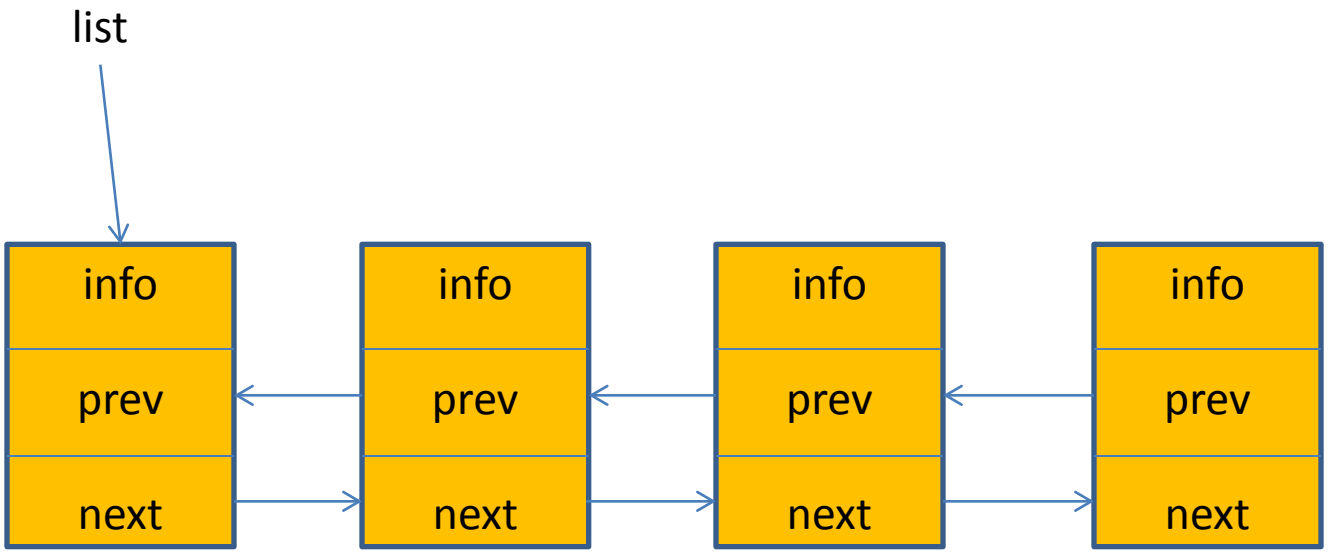
Länkad lista - remove



Länkad lista - remove

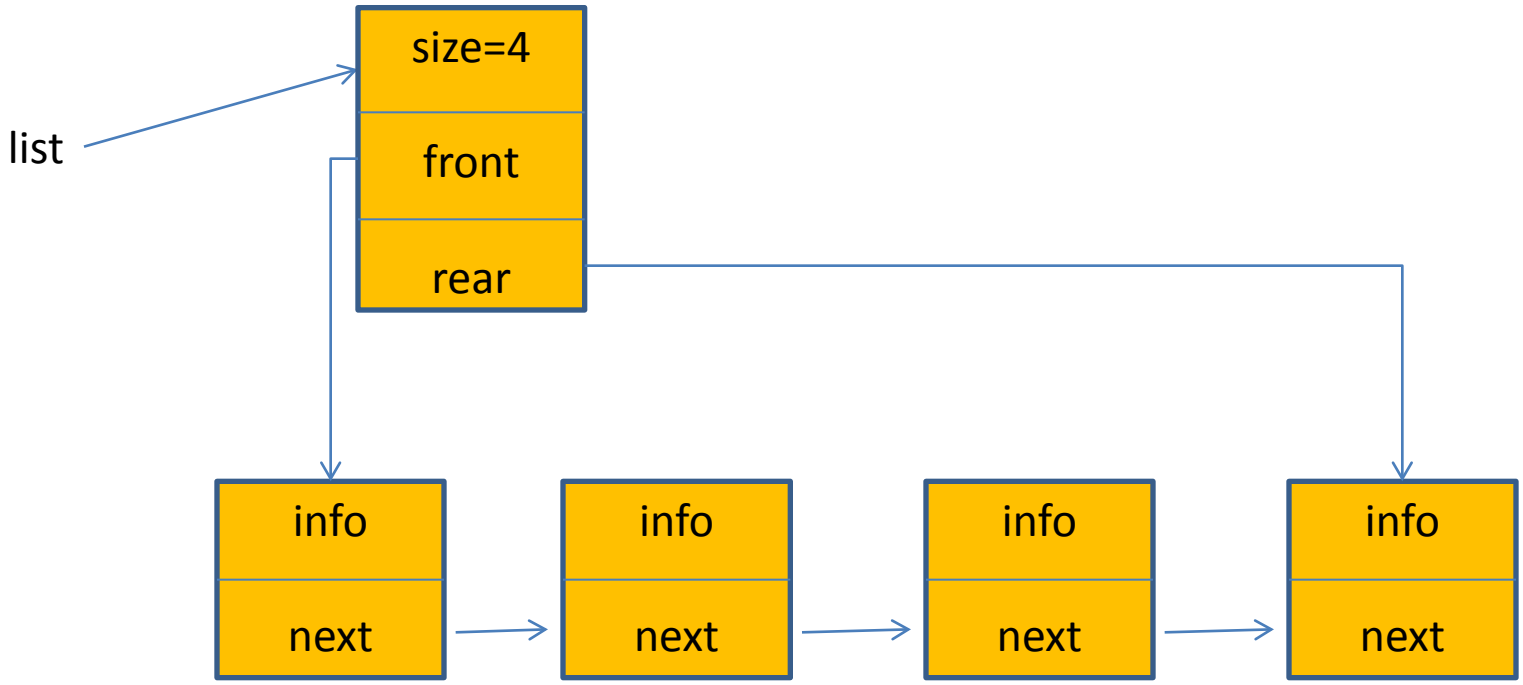
```
public class Node {
  Object info;
  Node next;
}
public class LinkedList {
  private Node head;

  protected void remove (Object o) {
    if (o.equals(head.info)) {
      head = head.next;
    } else {
      for(Node search = head; search != null;
          search = search.next) {
        if (o.equals(search.next.info)) {
          search.next = search.next.next;
          break;
        }
      }
    }
}}
```

Dubbellänkad lista




```
public class Node {
  Object info;
  Node prev, next;
}
```

Länkad lista med header



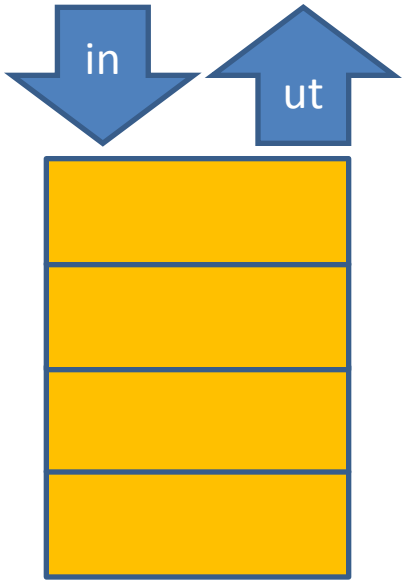

```
public class ListHeader{
  Node front, rear;
}
```

Fler linjära datastrukturer

- Kö – elementen läggs till sist och tas ut först
- FIFO – First-in first-out (kassakö)
  - enqueue
  - dequeue
  - empty

Fler linjära datastrukturer

- Stack – elementen läggs till först och tas ut först
- LIFO – Last-in first-out (tallriksstapel)
  - push
  - pop
  - peek
  - empty



En enkel stack

```
public class Stack {
  private Object[] stack = new Object[MAX_STK_SIZE];
  private int top = -1;

  public void push(Object o) {
    stack[++top] = o;
  }

  public Object pop() {
    return stack[top--];
  }

  public Object peek() {
    return stack[top];
  }

  public boolean isEmpty() {
    return top == -1;
  }
}
```

En enkel stack II

```
import java.util.Stack;

public static void main (String [] args) {
  Stack<String> stk = new Stack<String>();

  stk.push(”Harry”);
  stk.push(”Eva”);
  stk.push(”Kim”);

  System.out.println(stk.empty()); // false

  System.out.println(stk.peek()); // ”Kim”

  System.out.println(stk.pop()); // ”Kim”
  System.out.println(stk.pop()); // ”Eva”
  System.out.println(stk.pop()); // ”Harry”

  System.out.println(stk.empty()); // true
}
```

StackDemo

Collections implementationer

- Vector – lista
- ArrayList – lista
- LinkedList - lista
- Stack – stack
- PriorityQueue – sorterad kö (minst först)
- ArrayDeque - kö

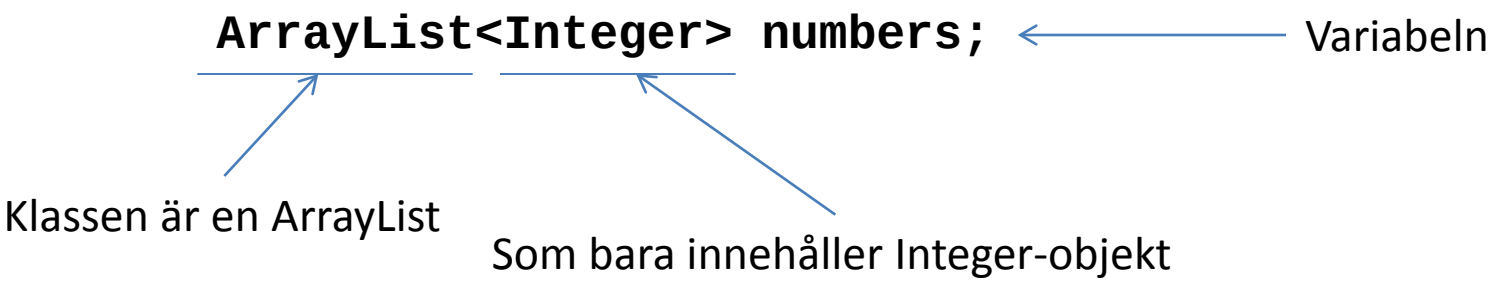
ListDemo

Collections implementationer

- Hashtable – lagrar par av nyckel och värde
- HashMap – lagrar par av nyckel och värde
- HashSet – mängd
- EnumMap – par av enum-nyckel och värde
- EnumSet – mängd för enums

Collections och *generics*

- Ofta är alla element i en collection av samma typ
- Kompilatorn kan hjälpa till med detta
- Då måste man berätta vilken typ man tänker sig på elementen.

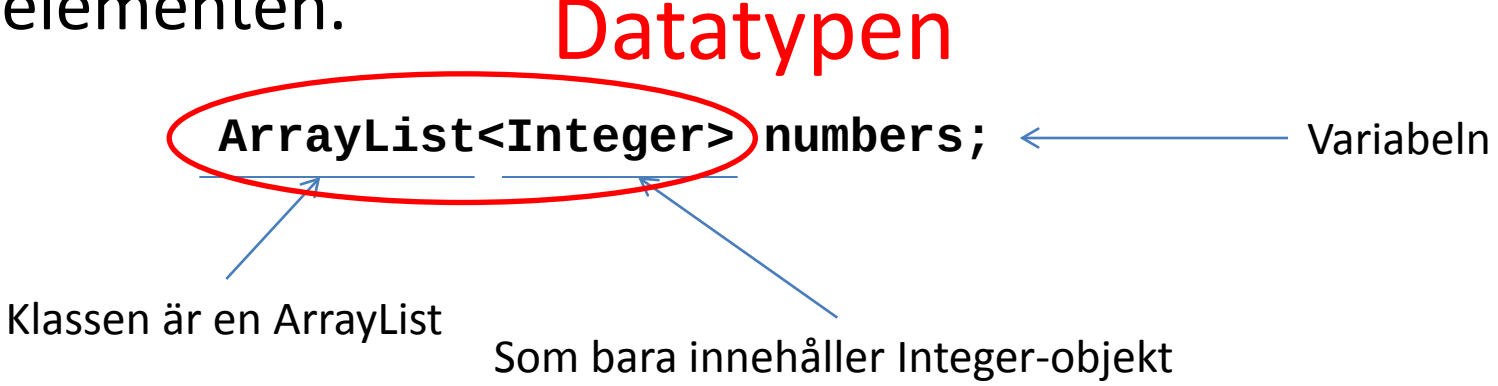



```
numbers = new ArrayList<Integer>();
```

Ovanstående gäller givetvis även för parametrar.

Collections och *generics*

- Ofta är alla element i en collection av samma typ
- Kompilatorn kan hjälpa till med detta
- Då måste man berätta vilken typ man tänker sig på elementen.




```
numbers = new ArrayList<Integer>();
```

Ovanstående gäller givetvis även för parametrar.

Utan *generics*

```
ArrayList numbers = new ArrayList();
…
for (int i = 0; i < 100; i++) {
  numbers.add(new Integer(i));
}
…
numbers.add(”Kraken”); // detta accepteras
numbers.add(new Vector()); // detta också
…
while (!numbers.isEmpty()) {
  Object obj = numbers.remove(0);
  if (obj instanceof Integer) {
    int n = ((Integer) obj).intValue();
  }
}
```

Med *generics* och *autoboxing*

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
…
for (int i = 0; i < 100; i++) {
  numbers.add(i); // autoboxing
}
…
numbers.add(”Kraken”); // kompileringsfel
numbers.add(new Vector()); // kompileringsfel
…
while (!numbers.isEmpty()) {
  int n = numbers.remove(0); // unboxing
}
```

(^_^’)

INGA FLER BILDER