



## Programmering för Språkteknologer II

Markus Saers  
markus.saers@lingfil.uu.se  
Rum 9-2040  
stp.lingfil.uu.se/~markuss/ht09/pst2



## Innehåll

- Undantag
- Paket
- Dubbellänkade listor



## I Java

```
import java.util.Iterator;
public class Stack<T> implements Iterable<T> {
    private class StackIterator implements Iterator<T> {
        private Node at;
        public StackIterator(Node top) {
            at = top;
        }
        public T next() {
            T value = at.value;
            at = at.prev;
            return value;
        }
        public boolean hasNext() {
            return at != null;
        }
        public void remove()
            throws UnsupportedOperationException {
            throw new UnsupportedOperationException();
        }
    }
    public Iterator<T> iterator() {
        return new StackIterator(top);
    }
}
```



## Undantag / Exceptionella händelser

- Exception på engelska
- Ett sätt att hantera fel utan att programmet måste krascha
- När något märkligt händer kan man "kasta ett undantag" som någon annan får ta hand om



## Att kasta undantag

- Undantag kastas med hjälp av det reserverade ordet **throw**, följt av ett undantagsobjekt
- En metod som innehåller ett **throw**-uttryck måste deklarera att den kan kasta det aktuella undantaget
- Detta görs med det reserverade ordet **throws** följt av en lista med undantagsklasser

```
public void remove() throws UnsupportedOperationException {
    throw new UnsupportedOperationException();
}
```

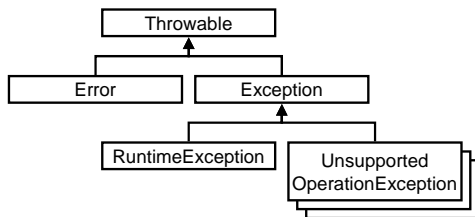


## Undantagsklasser/-objekt

- Klasser som ärver från **Exception**
- Instanser av sådana klasser

## Undantagsklasser/-objekt

- Klasser som ärver från **Exception**
- Instanser av sådana klasser



## Olika feltyper

- **Error**
  - Helt utanför programmerarens kontroll
  - Typ minnet har tagit slut
- **RuntimeException**
  - Uppstår på grund av dålig programmering
  - NullPointerException
  - ArrayIndexOutOfBoundsException
- Övriga **Exception**
  - Något svårförutsett inträffade

## Felhantering

- **Error**
  - Ohanterliga
- **RuntimeException**
  - Korrigera programmet
- Övriga **Exception**
  - Hantera felet!

## Hantera felet

- Aktiv hanteringar
  - Upptäcka att det inträffade
  - Vidtag adekvat åtgärd
- Passiv hantering
  - Kasta vidare felet (het potatis)

## Aktiv hantering?

- Försök utföra de satser där felet kan uppstå
- Var beredd på att fånga upp eventuella fel som kastas

## Upptäcka felet?

- Försök utföra de satser där felet kan uppstå
- Var beredd på att fånga upp eventuella fel som kastas

```

try {
    // ...
} catch (Exception e) {
    // ...
}
  
```



## Adekvat åtgärd?

- Utför något i **catch**-satsen
  - Skriv ut ett felmeddelande
  - Initialisera en variabel annorlunda
  - Kasta ett nytt undantag!



## Passiv hantering?

- Deklarera att metoden kastar vidare undantaget

```
public void removeAll(Iterator<String> i)
throws UnsupportedOperationException {
    while (i.hasNext()) {
        String s = i.next();
        i.remove();
        System.out.println("Removed string " + s);
    }
}
```



## Passiv / Aktiv hantering

```
public void removeAll(Iterator<String> i)
throws UnsupportedOperationException {
    while (i.hasNext()) {
        String s = i.next();
        i.remove();
        System.out.println("Removed string " + s);
    }
}

public void removeAll(Iterator<String> i) {
    while (i.hasNext()) {
        String s = i.next();
        try {
            i.remove();
        } catch (UnsupportedOperationException e) {
            System.out.println("Remove operator " +
                "not supported by supplied iterator");
        }
    }
}
```



## Vilka fel måste hanteras?

- Alla metoder kastar implicit **Error** och **RuntimeException**
  - Men de går att fånga om man vill...
- Alla andra undantag måste hanteras!



## Nya undantagstyper

- Mitt program genererar fel som inte passar in på någon befintlig undantagsklass
- Skapa en egen undantagsklass!



## **Exception** = bra basklass

- Klassen **Exception** är konstruerad som en bra basklass
- Ärv från den och skriv konstruktörer
  - Behöver inte skapa alla, bara de som man tänker använda...



## Att skapa egna undantagsklasser

```
public class MyException extends Exception {
    public MyException() {
        super();
    }
    public MyException(String msg) {
        super(msg);
    }
    public MyException(String msg, Throwable cause) {
        super(msg, cause);
    }
    public MyException(Throwable cause) {
        super(cause);
    }
}
```



## Frågor?



## Paket

- En samling klasser som har något gemensamt
  - java.util
  - java.util.zip
  - javax.sound.midi
  - javax.xml.xpath
  - org.w3c.dom
- Kan importeras och användas av andra javaprogram



## Går det att skapa egna paket?

- Ja!
- Paketnamnen bildar en filstruktur
  - java
    - util
      - zip
  - javax
    - sound
      - midi
    - xml
      - xpath
  - org
    - w3c
      - dom



## Går det att skapa egna paket?

- Skapa en egen filstruktur!
  - Byt ut alla "." mot "/"
- Placera in klasserna där
- Deklarera att klasserna är medlemmar i paketet
- Lämpliga paketnamn
  - Bör vara unika
  - Registrerade domännamn är unika!
    - Jämför org.w3c (www.w3c.org)
    - Vi borde ha se.uu.lingfil (www.lingfil.uu.se)
  - Följt av användarnamn



## Exempel

```
$ mkdir -p se/uu/lingfil/markuss
$ emacs se/uu/lingfil/markuss/Test.java &
package se.uu.lingfil.markuss;
public class Test {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
$ javac se/uu/lingfil/markuss/Test.java
$ java se.uu.lingfil.markuss.Test
Hello World!
$
```



## Fördelar med paket

- Saker som hör ihop klumpas ihop
- Lätt att flytta och distribuera
  - Läs på om Java Archive – JAR om du är intresserad av hur javaprogram distribueras



## Vad bör paketeras?

- Huvudsakligen "biblioteksklasser"
  - Klasser som tillhandahåller en allmän funktion som många kan vara intresserade av
    - java.util
    - java.io
  - Klasser som behandlar något specifikt problem eller dataformat
    - javax.sound.midi
    - javax.xml.xpath
    - org.w3c.dom



## Frågor?

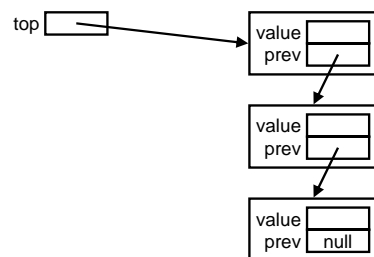


## Dubbellänkade listor

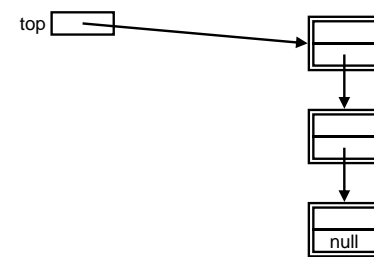
- Som enkellänkade listor (stackar) fast åt båda hållen

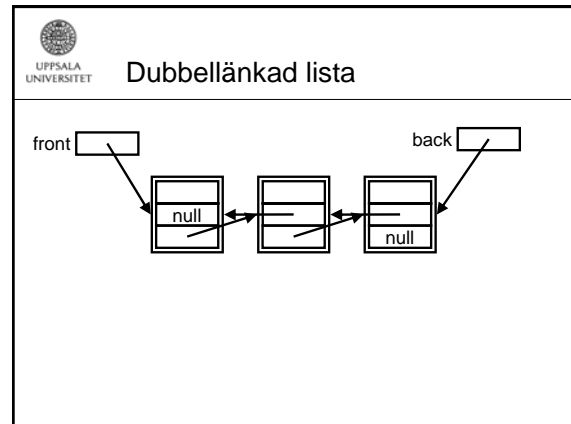
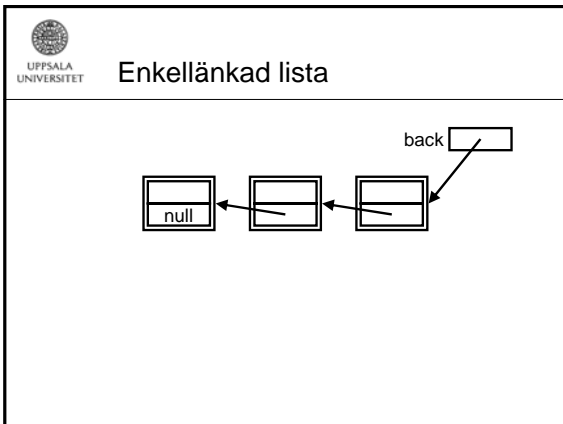


## Stack



## Stack





UPPSALA UNIVERSITET

### I Java

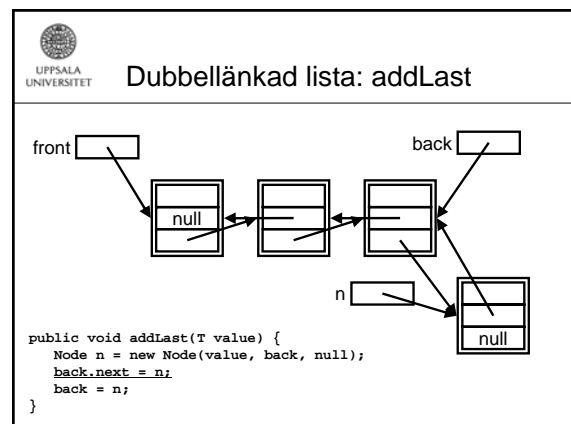
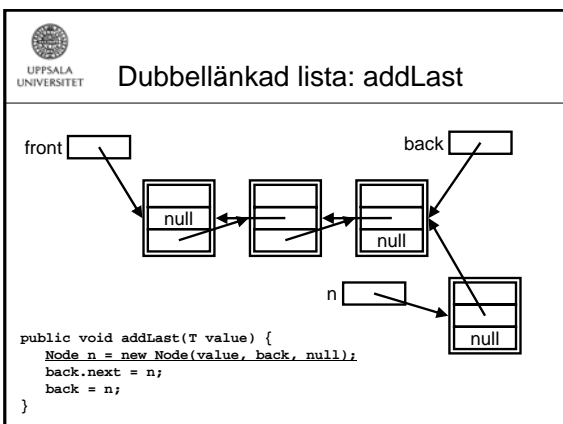
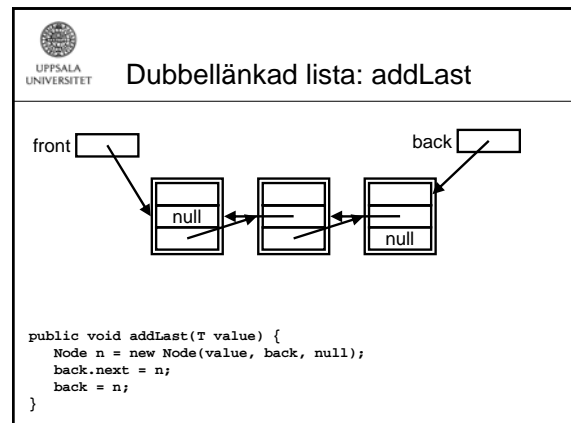
```

public LinkedList<T> {
    private Node front;
    private Node back;
    public T value;
    public Node prev;
    public Node next;
    public Node(T value, Node prev, Node next) {
        this.value = value;
        this.prev = prev;
        this.next = next;
    }
}

public void addFirst(T value) {
    Node n = new Node(value, null, front);
    front.prev = n;
    front = n;
}

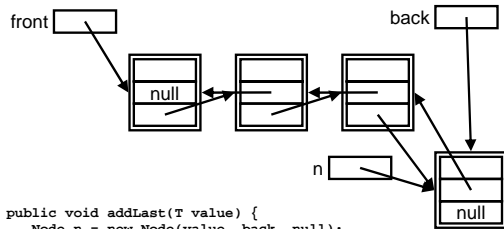
public void addLast(T value) {
    Node n = new Node(value, back, null);
    back.next = n;
    back = n;
}
}

```





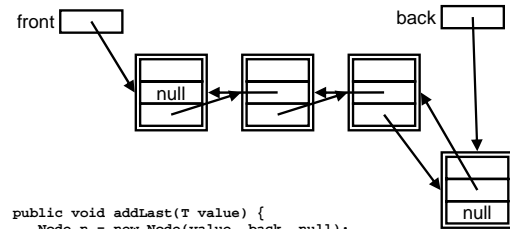
### Dubbellänkad lista: addLast



```
public void addLast(T value) {
    Node n = new Node(value, back, null);
    back.next = n;
    back = n;
}
```



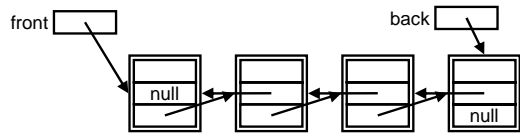
### Dubbellänkad lista: addLast



```
public void addLast(T value) {
    Node n = new Node(value, back, null);
    back.next = n;
    back = n;
}
```



### Dubbellänkad lista: addLast



```
public void addLast(T value) {
    Node n = new Node(value, back, null);
    back.next = n;
    back = n;
}
```



### Frågor?