



Programmering för Språkteknologer II

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



Innehåll

- Vektorer som listor
- Stackar
- Parametrisering
 - "Generics"
- Iteratorer
- Undantag



Vektorer som listor

- Exempel på tavlan



Stackar

- En datastruktur som fungerar enligt principen Last In, First Out: LIFO
 - Tänk lagen om anställningsskydd
- Har tre operationer
 - Push: lägger ett värde på stacken
 - Peek: tittar på översta värdet
 - Pop: tar bort översta värdet



Vad behöver vi veta för att klara av operationerna?

- Push
 - Vilket värde ska vi lägga på stacken?
- Peek
 - Vad är överst på stacken?
- Pop
 - Vad är under det översta värdet?



Vad vet vi när vi utför en operation?

- Push
 - Vad är överst på stacken innan vi lägger på det nya värdet!
- Peek
 - Vad är överst på stacken!
- Pop
 - Vad är överst på stacken!



Om vi lägger ihop

- Push
 - Vilket värde ska vi lägga på stacken?
 - Vad är överst på stacken innan vi lägger på det nya värdet!
- Peek
 - Vad är överst på stacken?
 - Vad är överst på stacken!
- Pop
 - Vad är under det översta värdet?
 - Vad är överst på stacken!

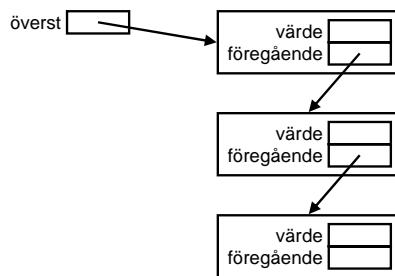


Lösning

- Varje värde kommer ihåg vad som ligger under
- Kallas länkning
 - Värdena är länkade till varandra



Grafiskt

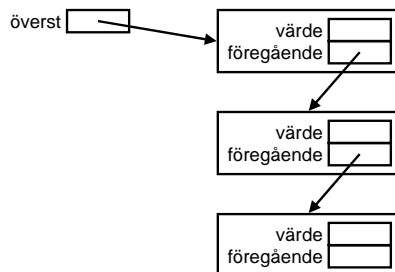


Pop

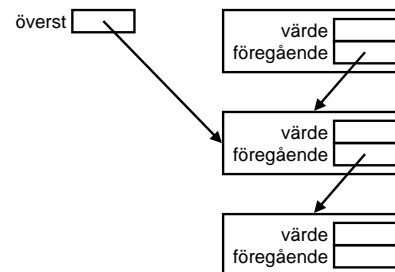
- Det översta värdet ska vara det nuvarande översta värdets föregångare
- `överst = överst.föregångare`

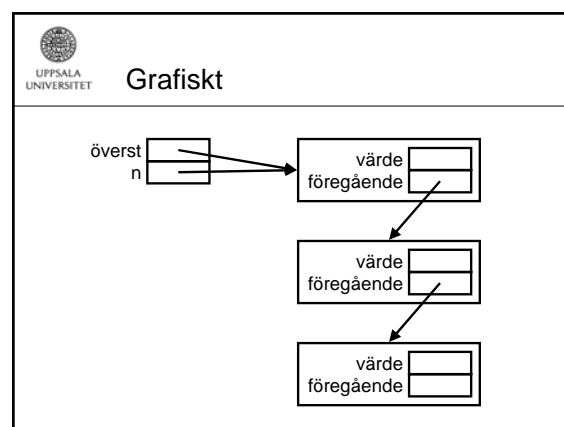
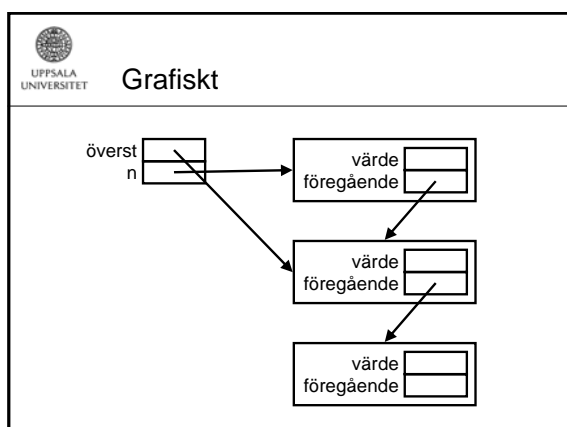
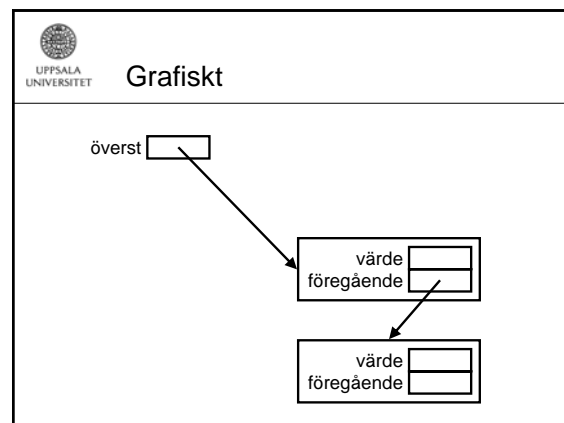
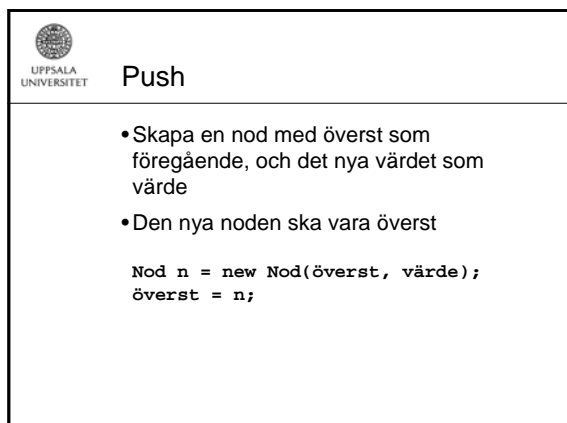
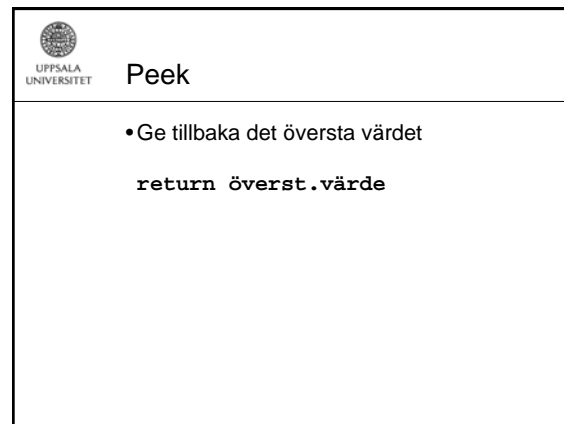
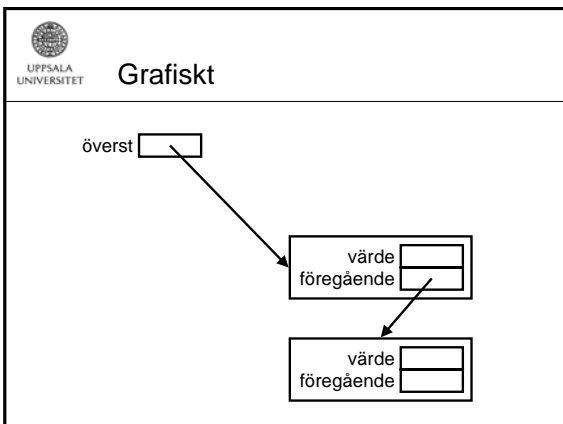


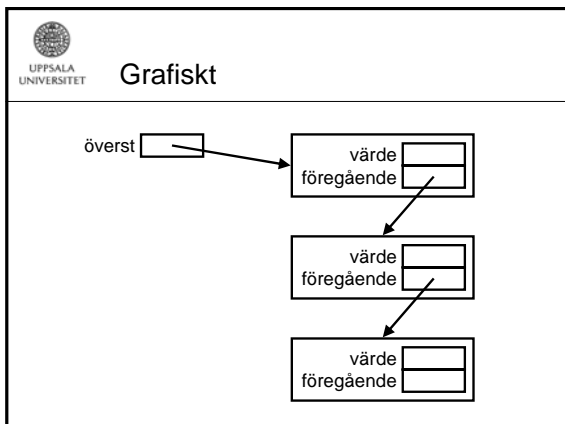
Grafiskt



Grafiskt







UPPSALA UNIVERSITET

Vad händer på slutet?

- Sista noden pekar inte på något!
- Har i själva verket värdet `null`
 - Ett av Javas reserverade ord
 - Betyder "Jag borde peka på något, men vet inte vad"
 - Värdet på objektsvariabler som inte tilldelats något riktigt värde

UPPSALA UNIVERSITET

null

```

public class Node {
    private String value;
    private Node prev;
    public Node(Node prev, String value) {
        this.prev = prev;
        this.value = value;
    }
    public static void main(String[] args) {
        Node top;
        for (String s : args) {
            top = new Node(top, s);
        }
    }
}
  
```

UPPSALA UNIVERSITET

null

```

public class Node {
    private String value;
    private Node prev;
    public Node(Node prev, String value) {
        this.prev = prev;
        this.value = value;
    }
    public static void main(String[] args) {
        Node top = null;
        for (String s : args) {
            top = new Node(top, s);
        }
    }
}
  
```

UPPSALA UNIVERSITET

Vi kör med args = {"a", "b", "c"}

top null

```

Node top = null;
for (String s : args) {
    top = new Node(top, s);
}
  
```

UPPSALA UNIVERSITET

Vi kör med args = {"a", "b", "c"}

top null

```

Node top = null;
for (String s : args) {
    top = new Node(top, s);
}
  
```

värde	a
prev	null

UPPSALA UNIVERSITET

Vi kör med args = {"a", "b", "c"}

```

Node top = null;
for (String s : args) {
    top = new Node(top, s);
}

```

UPPSALA UNIVERSITET

Vi kör med args = {"a", "b", "c"}

```

Node top = null;
for (String s : args) {
    top = new Node(top, s);
}

```

UPPSALA UNIVERSITET

Vi kör med args = {"a", "b", "c"}

```

Node top = null;
for (String s : args) {
    top = new Node(top, s);
}

```

UPPSALA UNIVERSITET

Interna klasser

- Man kan definiera en klass inuti en annan klass!
- Bra sätt att delegera exempelvis noder
 - Själva noden är ointressant utanför stacken

UPPSALA UNIVERSITET

Stack

```

public class Node {
    private String value;
    private Node prev;
    public Node(Node prev, String value) {
        this.prev = prev;
        this.value = value;
    }
}
public static void main(String[] args) {
    Node top;
    for (String s : args) {
        top = new Node(top, s);
    }
}

```

UPPSALA UNIVERSITET

Stack

```

public class Node {
    public String value;
    public Node prev;
    public Node(Node prev, String value) {
        this.prev = prev;
        this.value = value;
    }
}
public class Stack {
    public static void main(String[] args) {
        Node top;
        for (String s : args) {
            top = new Node(top, s);
        }
    }
}

```

Stack

```
public class Stack {
    private class Node {
        public String value;
        public Node prev;
        public Node(Node prev, String value) {
            this.prev = prev;
            this.value = value;
        }
    }
    public static void main(String[] args) {
        Node top;
        for (String s : args) {
            top = new Node(top, s);
        }
    }
}
```

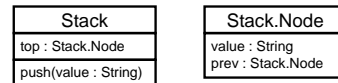
Stack

```
public class Stack {
    private class Node {
        public String value;
        public Node prev;
        public Node(Node prev, String value) {
            this.prev = prev;
            this.value = value;
        }
    }
    private Node top;
    public Stack() {
        top = null;
    }
    public static void main(String[] args) {
        Stack stack = new Stack();
        for (String s : args) {
            stack.top = new Node(top, s);
        }
    }
}
```

Stack

```
public class Stack {
    private class Node {
        public String value;
        public Node prev;
        public Node(Node prev, String value) {
            this.prev = prev;
            this.value = value;
        }
    }
    private Node top;
    public void push(String s) {
        top = new Node(top, s);
    }
    public Stack() {
        top = null;
    }
    public static void main(String[] args) {
        Stack stack = new Stack();
        for (String s : args) {
            stack.push(s);
        }
    }
}
```

UML



Frågor?

Parametrisering

- "Generics"
- Exempel
 - `List<String> list = new LinkedList<String>();`
 - `public class Scanner implements Iterator<String> ...`
 - `Map<String, Integer> freqDict;`
- Trevligt att använda
- Kan man skapa sådana själv?
 - Ja!
 - Kallas att parametrisera en klass

Idé

- En lista betar sig exakt likadant oavsett vad man lagrar i den
- Implementationen beror inte på innehållet
- Vi kan introducera en "platshållare" för innehållets typ
 - En parameter
- När vi skapar konkreta listor anger vi vad det ska vara för typ av innehåll
 - Samma implementation används oavsett innehåll

I Java (strängstack)

```

public class Stack {
    private class Node {
        public String value;
        public Node prev;
        public Node(Node prev, String value) {
            this.prev = prev;
            this.value = value;
        }
    }
    private Node top;
    public void push(String s) {
        top = new Node(top, s);
    }
    public Stack() {
        top = null;
    }
    public static void main(String[] args) {
        Stack stack = new Stack();
        for (String s : args) {
            stack.push(s);
        }
    }
}

```

I Java (parametriseradstack)

```

public class Stack<T> {
    private class Node {
        public T value;
        public Node prev;
        public Node(Node prev, T value) {
            this.prev = prev;
            this.value = value;
        }
    }
    private Node top;
    public void push(T s) {
        top = new Node(top, s);
    }
    public Stack() {
        top = null;
    }
    public static void main(String[] args) {
        Stack<String> stack = new Stack<String>();
        for (String s : args) {
            stack.push(s);
        }
    }
}

```

Komplikationer

- Vad händer om vi ger ett gränssnitt eller en abstrakt klass som parameter T?
 - Alla new T(...) blir ogiltiga

Lösningar

- Vad händer om vi ger ett gränssnitt eller en abstrakt klass som parameter T?
 - Alla new T(...) blir ogiltiga
- Lösningar
 - Tillåt bara konkreta klasser som parametrar
 - Tillåt inte skapandet av objekt av parametertyper i den parametriserade klassen

Lösningar

- Vad händer om vi ger ett gränssnitt eller en abstrakt klass som parameter T?
 - Alla new T(...) blir ogiltiga
- Lösningar
 - Tillåt bara konkreta klasser som parametrar
 - Tillåt inte skapandet av objekt av parametertyper i den parametriserade klassen



Konsekvenser

- Vi kan aldrig skapa objekt av typen T
- Vi kan aldrig skapa vektorer av objekt av typen T
 - Inte självklart, men så är det



Lösning

- Vi kan aldrig skapa objekt av typen T
 - Olösligt
- Vi kan aldrig skapa vektorer av objekt av typen T
 - Inte självklart, men så är det
 - Skapa en vektor av `Object`
 - Tvinga in den i en `T[]`-variabel

```
T[] array = new (T[]) Object[size];
```



Ger en varning

- **Note:** `MyClass.java` uses `unchecked` or `unsafe` operations
- Går att undertrycka genom att ange `@SuppressWarnings("unchecked")` före metoden där det inträffar
- Undertrycker alla varningar av den här typen för den aktuella metoden



Undertrycker varningen

- Se till att det finns en metod som bara utför det här steget

```
@SuppressWarnings("unchecked")
public T[] makeArray(int size) {
    return (T[]) new Object[size];
}
```

- Ger problemfri kompilering



Annotation

- `SuppressWarnings` är ett exempel på en annotation
- Metainformation om metoden som inte påverkar körning
- Ryms inte i kursen
 - Googla om ni är inträffade



Frågor?

Genomlöpfung av stack

- Går det att löpa igenom alla värden i en stack på ett smidigt sätt?

```
public static void main(String[] args) {
    Stack<String> stack = new Stack<String>();
    for (String s : args) {
        stack.push(s);
    }
    for (String s : stack) {
        System.out.println(s);
    }
}
```

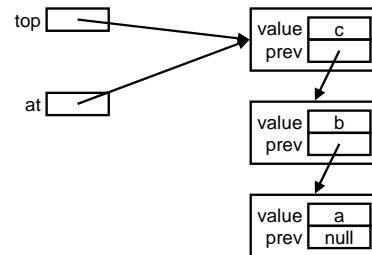
Krav

- `Stack<T>` måste implementera `Iterable<T>`
- Måste tillhandahålla metoden `Iterator<T> iterator()`
- Vi behöver en `Iterator<T>`

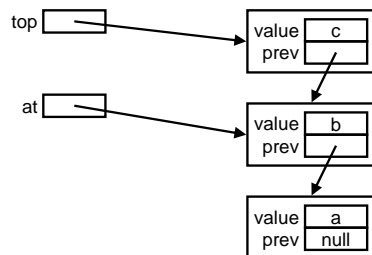
Stackiterator

- Metoder
 - boolean hasNext()
 - T next()
 - remove()
- Vad behöver vi för att uppfylla dem?
 - Börja på toppen
 - Hålla reda på var i stacken vi är
 - Märka när det inte finns fler element

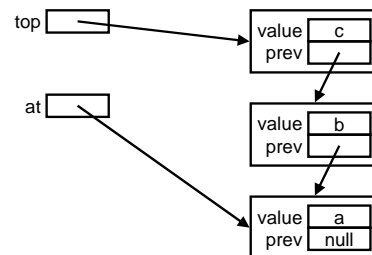
Grafiskt

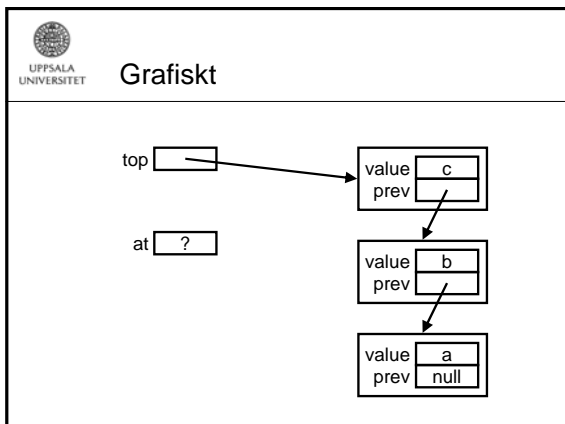


Grafiskt



Grafiskt





UPPSALA UNIVERSITET

Hur göra?

- Initialisering
 - Vi vill peka på det första elementet på stacken
`at = top;`
- next()
 - Ge tillbaka det värde vi titta på
 - Flytta till dess föregångare
`T value = at.value;`
`at = at.prev;`
`return value;`
- hasNext()
 - Kommer next () att ge ett giltigt värde?

UPPSALA UNIVERSITET

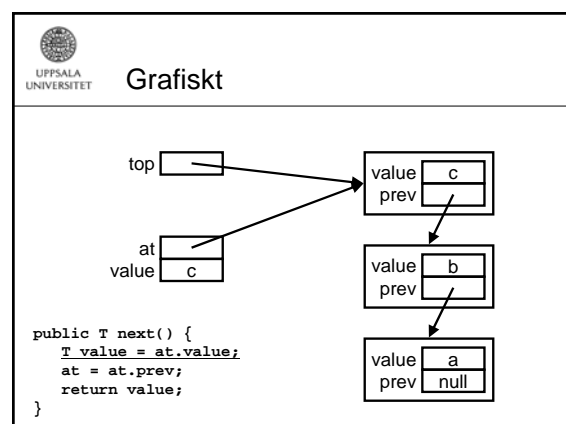
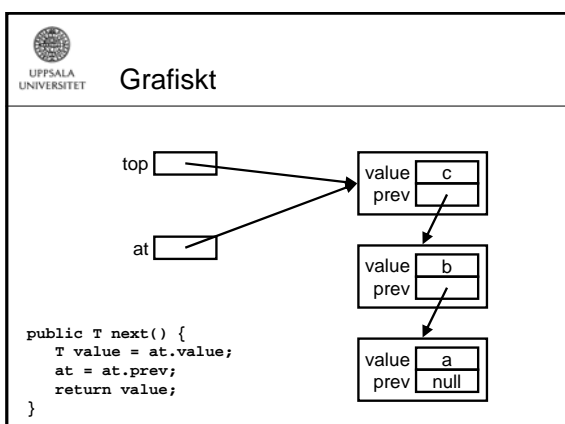
I Java

- Var behöver iteratoren finns för att ha tillgång till den information den behöver?
 - Node
 - Node.prev
- Stack!

UPPSALA UNIVERSITET

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() {
            // ...
        }
    }
    public Iterator<T> iterator() {
        return new StackIterator(top);
    }
}
```



UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Grafiskt

```

public T next() {
    T value = at.value;
    at = at.prev;
    return value;
}

```

UPPSALA UNIVERSITET

Avslutning

- När `at` inte längre är en giltig nod (`null`) finns inga fler värden

```
return at != null;
```

UPPSALA UNIVERSITET

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 Iterator<T> iterator() {
        return new StackIterator(top);
    }
}

```

UPPSALA UNIVERSITET

`remove()`

- Tråkigt att metoden `remove()` finns med i gränssnittet `Iterator<T>`, men vad gör man?
- Leder oss in på undantag!
 - Att ta bort något mitt i stacken stöds inte enligt vår specifikation
 - Försöker man kan det betraktas som en "exceptionell händelse"
 - Undantag är ett sätt att hantera det på

UPPSALA UNIVERSITET

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);
    }
}

```

UPPSALA UNIVERSITET

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();
}
```



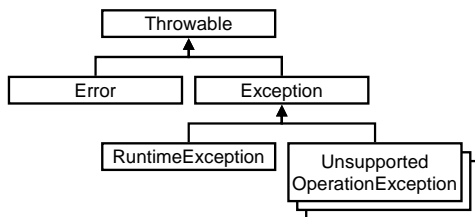
Undantagsklasser/-objekt

- 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**
 - Fånga det kastade felet
 - Hantera det!



Fånga felet?

- Försök utföra de satser där felet kan uppstå
- Fånga eventuella fel

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



Hantera felet!

- Javakompilatorn tvingar dig att hantera alla fel som inte ärver från **Error** eller **RuntimeException**
- Hantera
 - Vidtag lämplig åtgärd
 - Kasta vidare (het potatis)



Att skapa egna fel felklasser

```
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);  
    }  
}
```