Dagens föreläsning

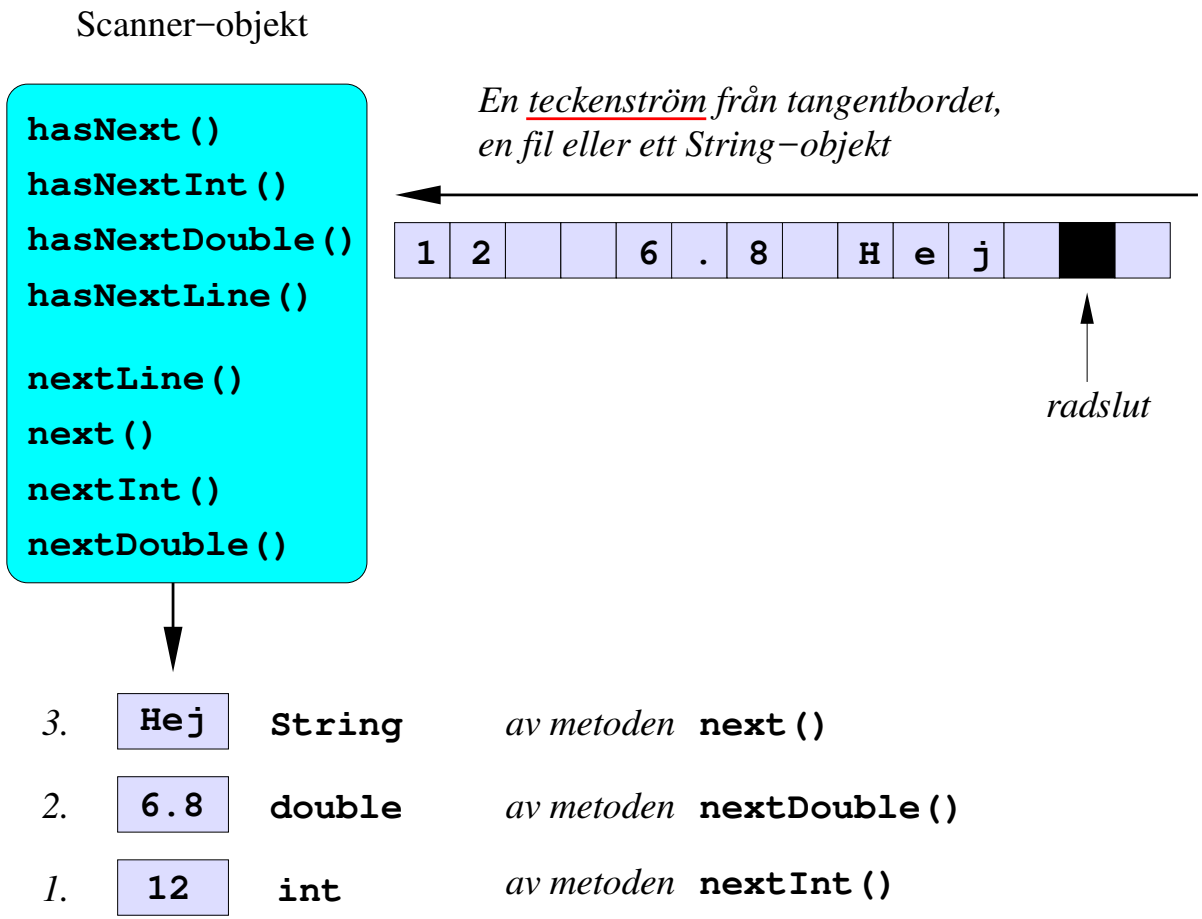
- `Scanner`-repetition
- `ArrayList`
- Omslagsklasser
- Metoderna `equals` och `compareTo`
- Läsa filer
- Programs hantering av fel.

Scanner-repetition

Ett `Scanner`-objekt ”klumpar ihop” tecken större enheter som heltal, flyttal och strängar:

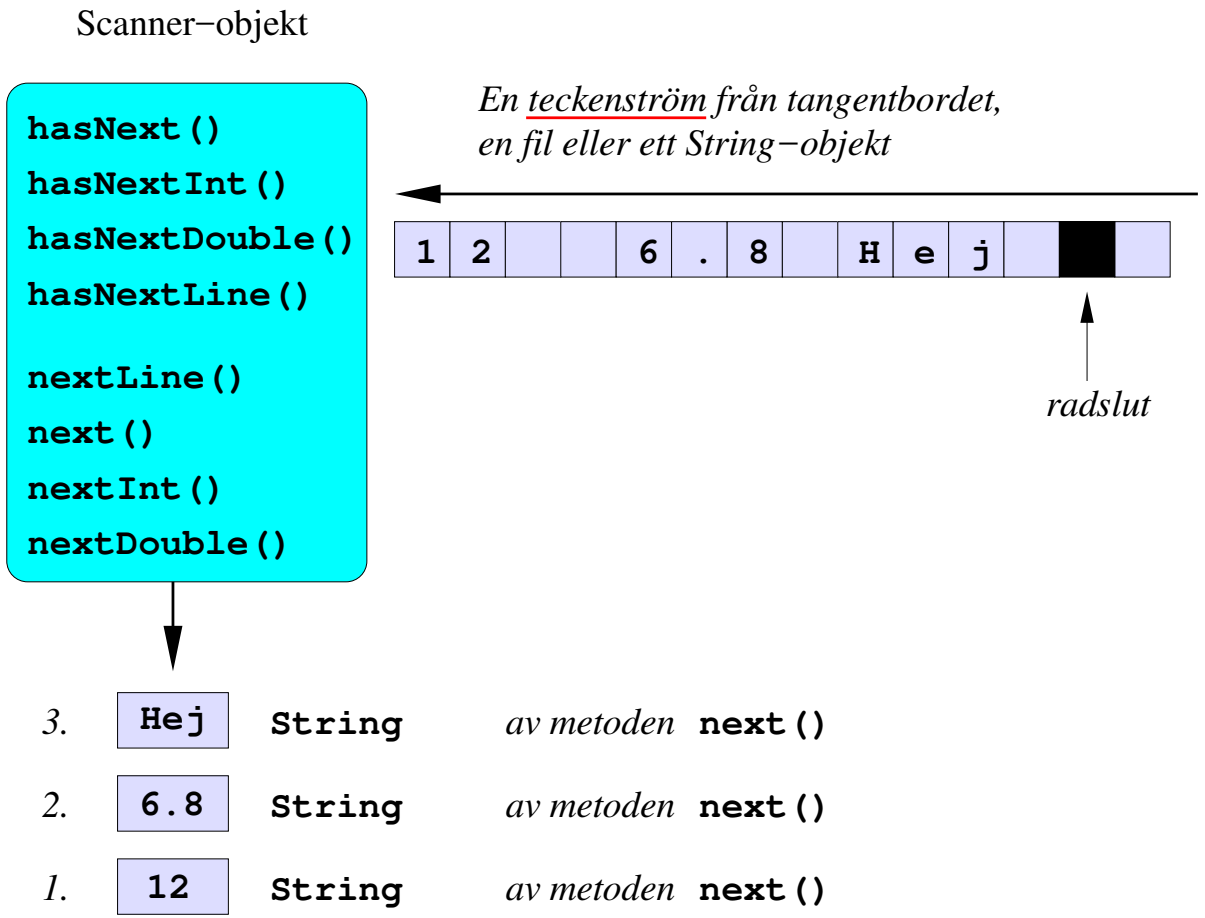
Scanner-repetition

Ett `Scanner`-objekt "klumpar ihop" tecken större enheter som heltal, flyttal och strängar:



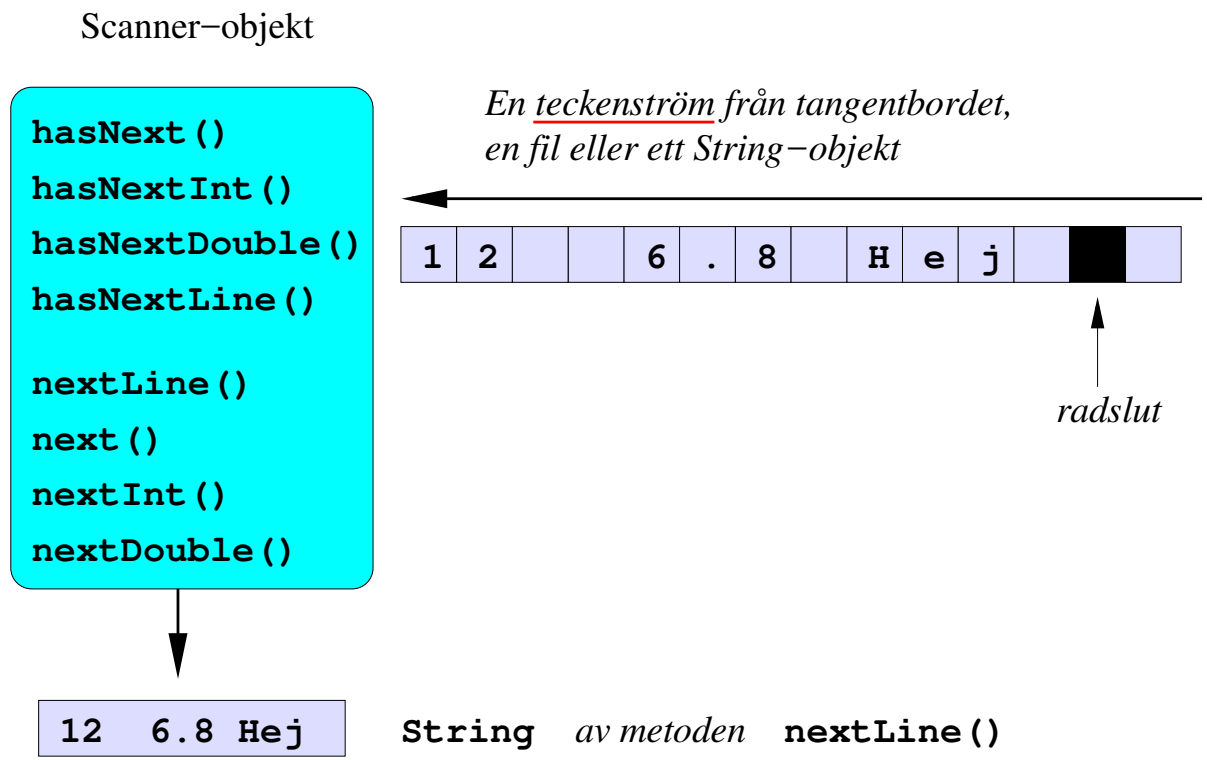
Scanner-illustration

Klumpar kan alltid tolkas som strängar:



Scanner-illustration

Man kan också hantera strömmen *radvis*:



Arrayers styrkor och svagheter

Arrayers styrkor och svagheter

+ Effektivt minnesutnyttjande

Arrayers styrkor och svagheter

+ Effektivt minnesutnyttjande
+ Snabb åtkomst till element med visst index

Arrayers styrkor och svagheter

+ Effektivt minnesutnyttjande
+ Snabb åtkomst till element med visst index
+ Snabb sökning om sorterat (med binär sökning eller hashteknik)

Arrayers styrkor och svagheter

+ Effektivt minnesutnyttjande
+ Snabb åtkomst till element med visst index
+ Snabb sökning om sorterat (med binär sökning eller hashteknik)
– Fix storlek

Arrayers styrkor och svagheter

+ Effektivt minnesutnyttjande
+ Snabb åtkomst till element med visst index
+ Snabb sökning om sorterat (med binär sökning eller hashteknik)
- Fix storlek
- Inlägg ”besvärligt” om sorterat

Ett alternativ till arrayer: ArrayList

En `ArrayList` är en ”arrayliknande” struktur med följande egenskaper:

Ett alternativ till arrayer: ArrayList

En `ArrayList` är en ”arrayliknande” struktur med följande egenskaper:

- Kan växa och krympa ”automatiskt”

Ett alternativ till arrayer: ArrayList

En `ArrayList` är en ”arrayliknande” struktur med följande egenskaper:

- Kan växa och krympa ”automatiskt”
- Kan ”skjuta in” nya element var som helst

Ett alternativ till arrayer: ArrayList

En `ArrayList` är en ”arrayliknande” struktur med följande egenskaper:

- Kan växa och krympa ”automatiskt”
- Kan ”skjuta in” nya element var som helst
- Kan ta bort element var som helst

Ett alternativ till arrayer: ArrayList

En `ArrayList` är en ”arrayliknande” struktur med följande egenskaper:

- Kan växa och krympa ”automatiskt”
- Kan ”skjuta in” nya element var som helst
- Kan ta bort element var som helst
- Kan använda index men en annan syntax än i arrayer

Ett alternativ till arrayer: ArrayList

En `ArrayList` är en ”arrayliknande” struktur med följande egenskaper:

- Kan växa och krympa ”automatiskt”
- Kan ”skjuta in” nya element var som helst
- Kan ta bort element var som helst
- Kan använda index men en annan syntax än i arrayer
- Kan bara lagra objekt — inte primitiva datatyper

Exempel `ArrayList`

Exempel `ArrayList`

```
import java.util.ArrayList;
```

Måste importeras

Exempel ArrayList

```
import java.util.ArrayList;
```

Måste importeras

```
ArrayList<String> names;
```

Typen måste anges i deklarationen

Exempel ArrayList

```
import java.util.ArrayList;
```

Måste importeras

```
ArrayList<String> names;
names = new ArrayList<String>();
```

Typen måste anges i deklarationen och i konstruktionen

Exempel ArrayList

```
import java.util.ArrayList;
```

Måste importeras

```
ArrayList<String> names;
names = new ArrayList<String>();
```

Typen måste anges i deklarationen och i konstruktionen

```
names.add("Olle");
names.add("Lisa");
names.add("Lasse");
```

`add` *lägger till sist*

Exempel ArrayList

```
import java.util.ArrayList;
```

Måste importeras

```
ArrayList<String> names;
names = new ArrayList<String>();
```

Typen måste anges i deklarationen och i konstruktionen

```
names.add("Olle");
names.add("Lisa");
names.add("Lasse");
```

`add` *lägger till sist*

```
System.out.println(names.get(1));
System.out.println(names);
System.out.println("Size: " +
                   names.size());
```

```
Lisa
[Olle, Lisa, Lasse]
Size: 3
```

Exempel `ArrayList`

Der går att ”skjuta in”, ändra och ta bort element.

Exempel `ArrayList`

Der går att "skjuta in", ändra och ta bort element.

Exempel:

```
names.add(1, "Anna");
System.out.println(names);          [Olle, Anna, Lisa, Lasse]
```

Exempel `ArrayList`

Der går att "skjuta in", ändra och ta bort element.

Exempel:

```
names.add(1, "Anna");
System.out.println(names);          [Olle, Anna, Lisa, Lasse]

names.set(2, "Britta");
System.out.println(names);          [Olle, Anna, Britta, Lasse]
```

Exempel ArrayList

Der går att "skjuta in", ändra och ta bort element.

Exempel:

```
names.add(1, "Anna");
System.out.println(names);          [Olle, Anna, Lisa, Lasse]

names.set(2, "Britta");
System.out.println(names);          [Olle, Anna, Britta, Lasse]


names.remove(0);
names.remove("Lasse");
System.out.println(names);          [Anna, Britta]
```

Exempel ArrayList

En ArrayList kan användas för att lagra godtyckliga objekt (av samma typ).

Exempel:

```
ArrayList<Turtle> turtles = new ArrayList<Turtle>();

ArrayList<Dice> dice = new ArrayList<Dice>();

ArrayList<int[]> intLists = new ArrayList<int[]>();
```

ArrayList: Summering av metoder

`int size()`	Returnerar aktuellt antal lagrade värden.
`add(E e)`	Lägger till sist.
`add(int i, E e)`	Lägger till på index `i`.
`boolean contains(E e)`	`true` om `e` finns i listan, annars `false`.
`E get(int i)`	Returnerar värde lagrat på index `i`.
`E set(int i, E e)`	Byter ut element på plats `i`
`E remove(int i)`	Tar bort element på index `i`. Returnerar det borttagna
`boolean remove(E e)`	Tar bort (första förekomst av) element `e`
`void clear()`	Tar bort alla element ur listan.

ArrayList

`ArrayList`-objekt är implementerade med *arrayer*.

Det innebär, t ex att operationen `add` och `remove` kan flytta på element.

ArrayList

`ArrayList`-objekt är implementerade med *arrayer*.

Det innebär, t ex att operationen `add` och `remove` kan flytta på element.

Exempel: Vad är problemet koden

```
for (int i=0; i<al.size(); i++) {
   if (al.get(i).isDead()) {
      al.remove(i);
   }
}
```

ArrayList

`ArrayList`-objekt är implementerade med *arrayer*.

Det innebär, t ex att operationen `add` och `remove` kan flytta på element.

Exempel: Vad är problemet koden

```
for (int i=0; i<al.size(); i++) {
   if (al.get(i).isDead()) {
      al.remove(i);
   }
}
```

Kommer att missa det som låg på plats $i + 1$

ArrayList

`ArrayList`-objekt är implementerade med *arrayer*.

Det innebär, t ex att operationen `add` och `remove` kan flytta på element.

Exempel: Vad är problemet koden

```
for (int i=0; i<al.size(); i++) {
   if (al.get(i).isDead()) {
      al.remove(i);
   }
}
```

Kommer att missa det som låg på plats $i + 1$

Gör så här i stället:

```
for (int i = al.size()-1; i >= 0; i--) {
   if (al.get(i).isDead()) {
      al.remove(i);
   }
}
```

Omslagsklasser

Hur gör man för att lagra t ex `int` eller `double`?

Omslagsklasser

Hur gör man för att lagra t ex `int` eller `double`?

Man använder ”omslagsklasserna” `Integer` eller `Double`.

Omslagsklasser

Hur gör man för att lagra t ex `int` eller `double`?

Man använder ”omslagsklasserna” `Integer` eller `Double`.

Exempel: Koden

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
for (int i = 0; i<=10; i++) {
   numbers.add(new Integer(i-5));
}
System.out.println(numbers);
```

skriver

```
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
```

Omslagsklasser: Hantering av objekten

```
Integer io = numbers.get(7);
System.out.println("Integer: " + io);          Integer: 2

int i = io.intValue();
System.out.println("int    : " + i );          int    : 2
```

Omslagsklasser: Hantering av objekten

```
Integer io = numbers.get(7);
System.out.println("Integer: " + io);          Integer: 2

int i = io.intValue();
System.out.println("int    : " + i );          int    : 2
```

Fast det är i själva verket enklare – Java gör det automatiskt!

Omslagsklasser: ”Autoboxing” och ”autounboxing”

```
numbers.clear();
for (int i = 0; i<10; i++) {
    numbers.add(10-i);
}
System.out.println(numbers);     [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Omslagsklasser: ”Autoboxing” och ”autounboxing”

```
numbers.clear();
for (int i = 0; i<10; i++) {
    numbers.add(10-i);
}
System.out.println(numbers);     [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

int s = 0;
for (int i = 0; i<numbers.size(); i++) {
   s = s + numbers.get(i);
}
System.out.println("Summan: " + s);                    Summan: 55
```

Omslagsklasser: ”Autoboxing” och ”autounboxing”

```
numbers.clear();
for (int i = 0; i<10; i++) {
    numbers.add(10-i);
}
System.out.println(numbers);     [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

int s = 0;
for (int i = 0; i<numbers.size(); i++) {
   s = s + numbers.get(i);
}
System.out.println("Summan: " + s);                    Summan: 55
```

eller, ännu enklare med ”for-each”-loop:

```
for (int x : numbers) {
   s += x;
}
System.out.println("Summan: " + s);
```

Att fundera på

Vad tas bort av satsen

```
numbers.remove(3);
```

Är det element på plats 3 eller elementet som innehåller 3?

Hur gör man för att få den variant som det inte är?

Att fundera på

Vad tas bort av satsen

```
numbers.remove(3);
```

Är det element på plats 3 eller elementet som innehåller 3?

Hur gör man för att få den variant som det inte är?

Det är parameterns *datatyp* som avgör dvs

`numbers.remove(3);`	tar bort element på plats 3 medan
`numbers.remove(new Integer(3));`	tar bort första element med värde 3.

Jämförelse av strängar

Låt `s` och `t` vara referenser till två `String`-objekt.

`s.equals(t)` returnerar

- `true` om strängarna är lika långa och innehåller samma tecken
- `false` i annat fall

Jämförelse av strängar

Låt `s` och `t` vara referenser till två `String`-objekt.

`s.equals(t)` returnerar

- `true` om strängarna är lika långa och innehåller samma tecken
- `false` i annat fall

Exempel på användning:

```
if (answer.equals("yes")) { ... }

while (!p.getName().equals(q.getName())) { ... }
```

Jämförelse av strängar

Låt `s` och `t` vara referenser till två `String`-objekt.

`s.equals(t)` returnerar

- `true` om strängarna är lika långa och innehåller samma tecken
- `false` i annat fall

Exempel på användning:

```
if (answer.equals("yes")) { ... }

while (!p.getName().equals(q.getName())) { ... }
```

Använd *aldrig* operatorn == för att testa om två strängar är lika!

Jämförelse av strängar

Anropet `s.compareTo(t)` returnerar

- ett negativt heltal om `s` kommer före `t` i alfabetisk ordning,
- noll om `s` och `t` innehåller samma tecken (dvs om `s.equals(t)` är `true`) och
- ett positivt heltal om `s` kommer efter `t` i alfabetisk ordning

Observera: ”alfabetisk ordning” betyder här ”i ordning efter ascii-koder” (egentligen unikoder)

Exempel

Klassen `StringList`:

- Ska läsa strängar (”ord”) från en datafil.
- Ska lagras i *bokstavsordning* i ett `ArrayList`-objekt.
- Ett ord ska bara lagras en gång.

Klassen kommer illustrera insticksortering i en arraylist.
Se minilektionen om algoritmer för mer detaljerad diskussion om hur insticksortering fungerar.

Klassen kommer också visa hur man kan läsa från en *fil*.

Exempelklassen `StringList`

Exempelklassen StringList

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

public class StringList {
   private ArrayList<String> theList;

   public StringList() {
      theList = new ArrayList<String>();
   }
```

Exempelklassen `StringList`: `add`

Lägger in ett ord i listan så att sorteringen bibehålls.

Exempelklassen `StringList`: `add`

Lägger in ett ord i listan så att sorteringen bibehålls.

Exempelklassen StringList: add

Lägger in ett ord i listan så att sorteringen bibehålls.

```
public void add(String str) {
  int i=0;
  while( i < theList.size() &&
         str.compareTo(theList.get(i)) > 0 ) {
    i++;
  }
  if ( i == theList.size() ||
       str.equals(!theList.get(i)) ) { //If not found
    theList.add(i,str);
  }
}
```

Exempelklassen StringList: add

Lägger in ett ord i listan så att sorteringen bibehålls.

```
public void add(String str) {
  int i=0;
  while( i < theList.size() &&
         str.compareTo(theList.get(i)) > 0 ) {
    i++;
  }
  if ( i == theList.size() ||
       str.equals(!theList.get(i)) ) { //If not found
    theList.add(i,str);
  }
}
```

Observera: Ordningen i såväl `while`- som `if`-villkoren!

Exempelklassen StringList

```
/**
 * Reads a file and add the words to the list
 */
public void load(String filename) throws IOException {
   FileReader file = new FileReader(filename);
   Scanner fscan = new Scanner(file);

   while(fscan.hasNext()) {
      add(fscan.next().toLowerCase());
   }
 }
```

Exempelklassen StringList

```
/**
 * Reads a file and add the words to the list
 */
public void load(String filename) throws IOException {
   FileReader file = new FileReader(filename);
   Scanner fscan = new Scanner(file);

   while(fscan.hasNext()) {
      add(fscan.next().toLowerCase());
   }
 }
```

Observera:

Exempelklassen StringList

```
/**
 * Reads a file and add the words to the list
 */
public void load(String filename) throws IOException {
   FileReader file = new FileReader(filename);
   Scanner fscan = new Scanner(file);

   while(fscan.hasNext()) {
      add(fscan.next().toLowerCase());
   }
}
```

Observera:

- `FileReader`

Exempelklassen StringList

```
/**
 * Reads a file and add the words to the list
 */
public void load(String filename) throws IOException {
   FileReader file = new FileReader(filename);
   Scanner fscan = new Scanner(file);

   while(fscan.hasNext()) {
      add(fscan.next().toLowerCase());
   }
}
```

Observera:

- `FileReader`
- `throws IOException`

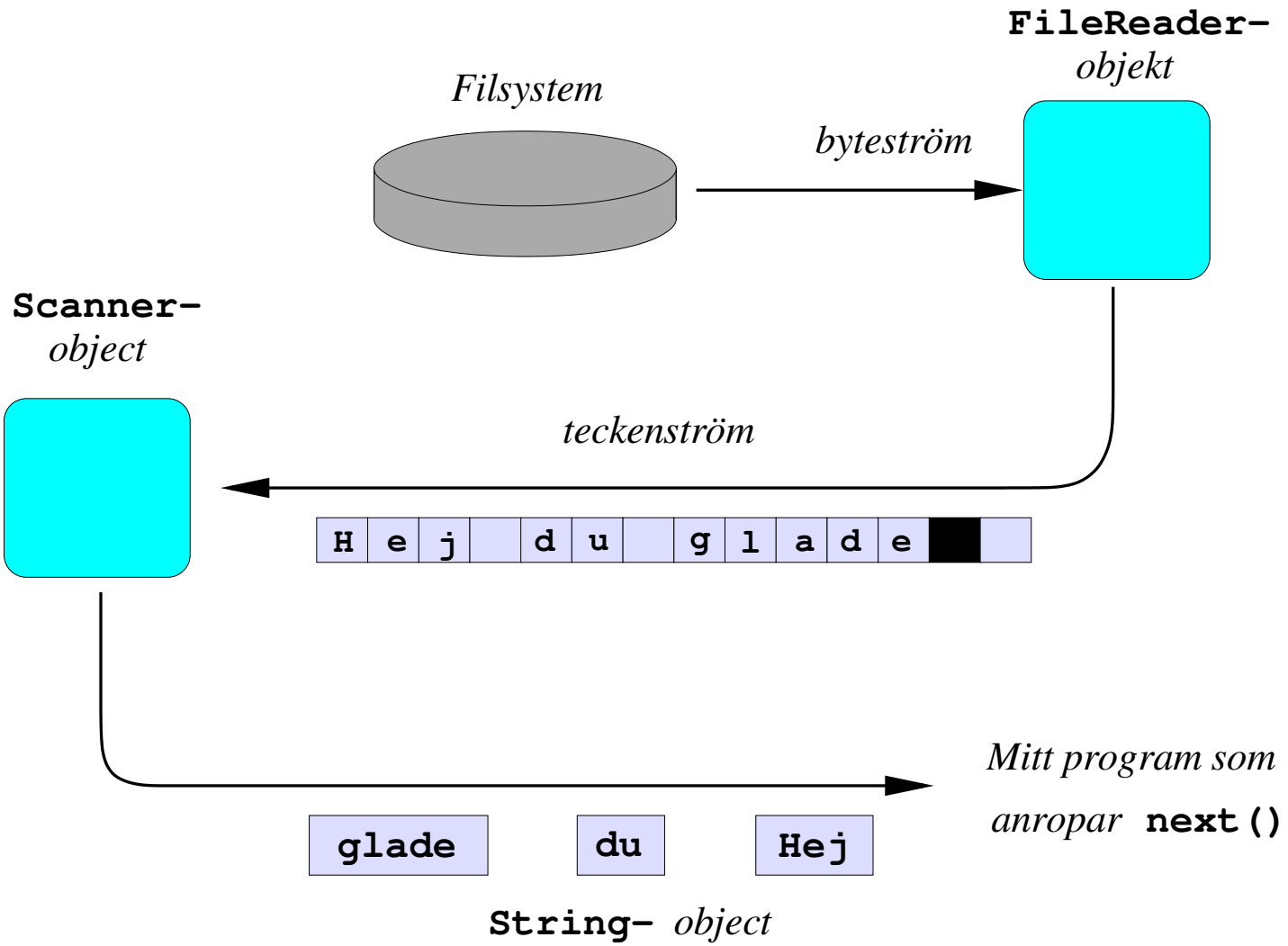
Exempelklassen StringList

```
/**
 * Reads a file and add the words to the list
 */
public void load(String filename) throws IOException {
   FileReader file = new FileReader(filename);
   Scanner fscan = new Scanner(file);

   while(fscan.hasNext()) {
      add(fscan.next().toLowerCase());
   }
}
```

Observera:

- `FileReader`
- `throws IOException`
- `toLowerCase()`

Läsa filer med `FileReader`



Exempelklassen StringList

```
public String toString() {
  return theList.toString();
}

public static void main(String[] a) throws IOException {
  StringList list = new StringList();
  list.load("indata.txt");
  System.out.println(list);
}
```

Exempelklassen StringList

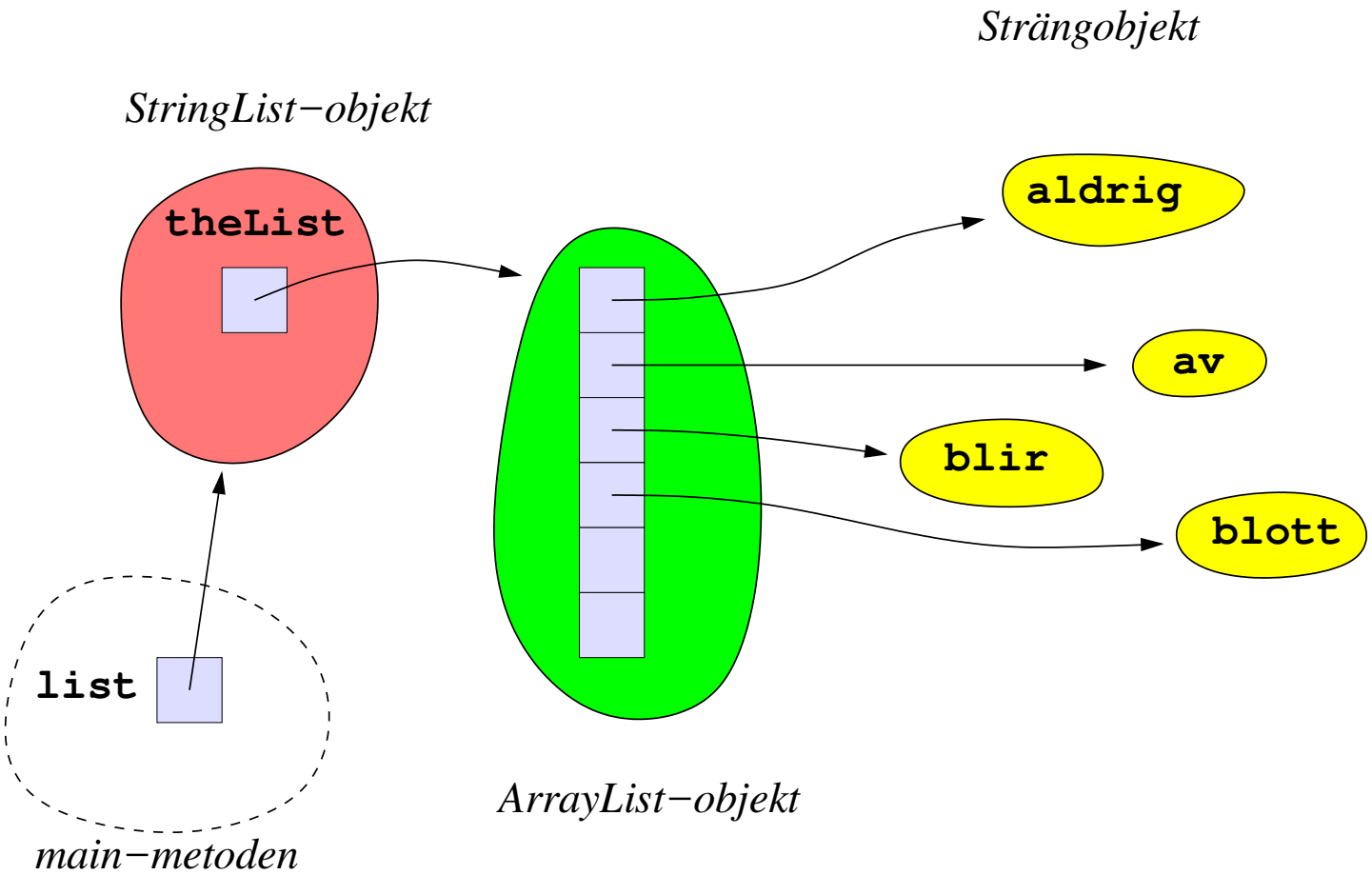
```
public String toString() {
  return theList.toString();
}

public static void main(String[] a) throws IOException {
  StringList list = new StringList();
  list.load("indata.txt");
  System.out.println(list);
}
```

Som ger utskriften

```
[aldrig, av, blir, blott, bryt, bryts, brödet, bästa, dag, dagen, den, det,
drömmen, där, elden, en, finns, full, färd, gryr, gång, hast, i, man, men,
mening, mätta, mål, målet, mödan, nattlång, nog, nya, och, oändligt, på, rast,
som, sover, stora, ställen, störst, sång, sömnen, trygg, tänds, törst, upp,
vägen, värd, vår, vårt, är, äventyr]
```

Länk till klassen StringList

Illustration av världen



Metoderna `equals` och `compareTo` i andra klasser

För att jämföra objekt ur en egen klass (säg klassen `Dice`) så skriver man metoderna

- `boolean equals(Dice d)` om man vill se om två objekt är lika
- `int compareTo(Dice d)` om man vill kunna ordna objekten i storleksordning

Alla inbyggda klasser i Java som behöver jämföra objekt använder just dessa namn.

Metoden `equals` finns alltid (definieras i basklassen `Objekt`) men den gör knappast vad man vill. (Hur skulle den kunna det?)

Exempelklasserna `Word` och `WordList`

Samma problem som i exemplet `StringList` *men* vi vill också veta hur många gånger varje ord förekommit.

Exempelklasserna `Word` och `WordList`

Samma problem som i exemplet `StringList` *men* vi vill också veta hur många gånger varje ord förekommit.

Varje ord måste förses men en frekvensräknare. Gör en till klass:

Exempelklasserna `Word` och `WordList`

Samma problem som i exemplet `StringList` *men* vi vill också veta hur många gånger varje ord förekommit.

Varje ord måste förses men en frekvensräknare. Gör en till klass:

```
public class Word {
   private String theWord;
   private int frequency;

   public Word(String theWord) {
     this.theWord = theWord;
     this.frequency = 1;
   }

  public String toString() {
    return theWord + " : " + frequency;
  }
```

Exempelklassen Word

```
public void increaseFrequency() {
  frequency++;
}
```

Exempelklassen Word

```
public void increaseFrequency() {
  frequency++;
}

public String getWord() {
  return theWord;
}
```

Exempelklassen Word

```
public void increaseFrequency() {
  frequency++;
}

public String getWord() {
  return theWord;
}

public int getFrequency() {
  return frequency;
}
```

Exempelklassen `Word`

```
public boolean equals(Word w) {
  return theWord.equals(w.theWord);
}
```

Exempelklassen Word

```
  public boolean equals(Word w) {
    return theWord.equals(w.theWord);
  }


  public int compareTo(Word w) {
    return theWord.compareTo(w.theWord);
  }

} // End of class Word
```

Exempelklassen WordList

Klassen WordList blir mycket lik klassen StringList:

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

public class WordList {
   private ArrayList<Word> theList;

   public WordList() {
      theList = new ArrayList<Word>();
   }
```

Exempelklassen WordList

```
public void add(Word word) {
  int i=0;
  while( i<theList.size() &&
         word.compareTo(theList.get(i))>0) {
    i++;
  }
  if (i==theList.size() ||
      word.compareTo(theList.get(i))!=0 ) {
    theList.add(i,word);
  } else {
    theList.get(i).increaseFrequency();
  }
}
```

Exempelklassen WordList

```
public void load(String filename) throws IOException {
  FileReader file = new FileReader(filename);
  Scanner fscan = new Scanner(file);
  while(fscan.hasNext()) {
    add(new Word(fscan.next().toLowerCase()));
  }
}
```

Exempelklassen WordList

```
public void load(String filename) throws IOException {
  FileReader file = new FileReader(filename);
  Scanner fscan = new Scanner(file);
  while(fscan.hasNext()) {
    add(new Word(fscan.next().toLowerCase()));
  }
}
```

```
public String toString() {
  return theList.toString();
}
```

Observera att arralylistens `toString` kommer att använda sig av `toString` i klassen `Word`.

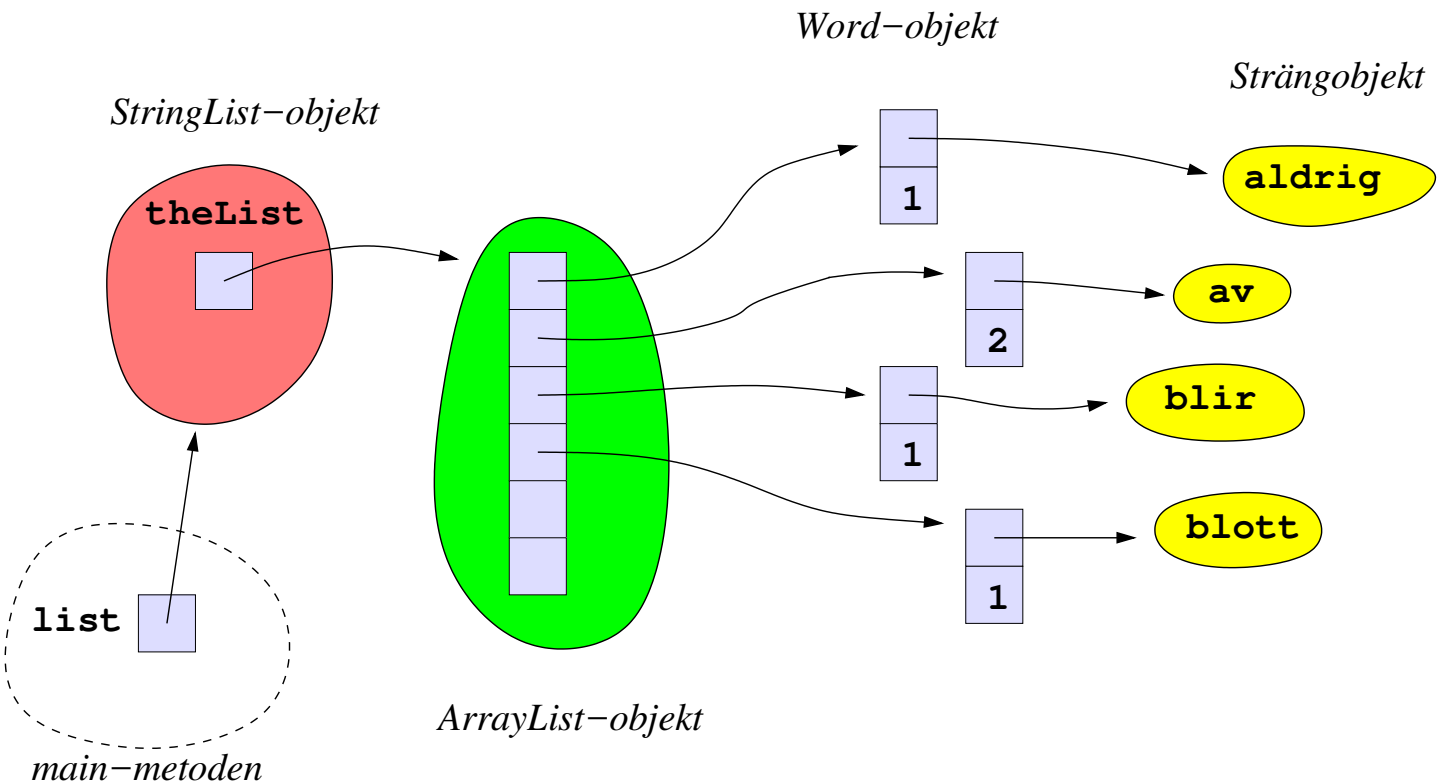
Exempelklassen WordList

```
public void print() {
   int i= 0;
   for (Word w : theList) {
       i++;
       System.out.format("%12s :%2d", w.getWord(), w.getFrequency());
       if (i % 5 == 0) {
          System.out.println();
       }
   }
   System.out.println();
}
```

Exempel på output:

```
      aldrig : 1         av : 2       blir : 1      blott : 1       bryt : 2
       bryts : 1     brödet : 1      bästa : 2        dag : 1      dagen : 3
         den : 4        det : 3    drömmen : 1        där : 2      elden : 1
          en : 3      finns : 1       full : 1       färd : 1       gryr : 1
        gång : 1       hast : 1          i : 2        man : 1        men : 1
     mening : 1       mätta : 1        mål : 1      målet : 1      mödan : 1
   nattlång : 1         nog : 1        nya : 1        och : 3   oändligt : 1
         på : 1        rast : 1        som : 1       sover : 1       stora : 1
    ställen : 1      störst : 1        sång : 1      sömnen : 1       tyngd : 1
```

Illustration av världen



Sortering av ordlistan i frekvensordning

```
public void printFrequencyOrdered() {

  // Construct a frequency ordered list

   ArrayList<Word> fsorted = new ArrayList<Word>();
   for (Word w : theList) {
        int i=0;
        while(i < fsorted.size() &&
              w.getFrequency() < fsorted.get(i).getFrequency()) {
           i++;
        }
        fsorted.add(i, w);
   }

  // and print it ...
```

Sortering av ordlistan i frekvensordning

```
public void printFrequencyOrdered() {

  // Construct a frequency ordered list

   ArrayList<Word> fsorted = new ArrayList<Word>();
   for (Word w : theList) {
        int i=0;
        while(i < fsorted.size() &&
              w.getFrequency() < fsorted.get(i).getFrequency()) {
           i++;
        }
        fsorted.add(i, w);
   }

  // and print it ...
```

(Se minilektionen om algoritmer för detaljer!)

Sortering av ordlistan i frekvensordning

```
    // and print it ...
    int i = 0;
    for (Word w : fsorted) {
        i++;
        System.out.format("%3d  %-11s",
                          w.getFrequency(),
                          w.getWord());
        if (i%5 == 0) {
           System.out.println();
        }
    }
    System.out.println();
}
```

Exempelklassen `WordList`

Exempel på utskrift:

6	är	4	den	3	och	3	en	3	det
3	dagen	2	upp	2	i	2	där	2	bästa
2	bryt	2	av	1	äventyr	1	vårt	1	vår
1	värd	1	vägen	1	törst	1	tänds	1	trygg
1	sömnen	1	sång	1	störst	1	ställen	1	stora
1	sover	1	som	1	rast	1	på	1	oändligt
1	nya	1	nog	1	nattlång	1	mödan	1	målet
1	mål	1	mätta	1	mening	1	men	1	man
1	hast	1	gång	1	gryr	1	färd	1	full
1	finns	1	elden	1	drömmen	1	dag	1	brödet
1	bryts	1	blott	1	blir	1	aldrig		

Länk till katalog med kod och data

Ytterligare något om strängar

- `String`-objekt är konstanta — kan ej förändras. Kan alltså riskfritt delas mellan objekt

Ytterligare något om strängar

- `String`-objekt är konstanta — kan ej förändras. Kan alltså riskfritt delas mellan objekt

- Klassen `StringBuffer` kan också användas för att representera stängar men dessa kan ändra både till längd och innehåll. Bra alternativ om långa strängar ska byggas upp i en loop (t ex i `toString`-metoder).

Ytterligare något om strängar

- `String`-objekt är konstanta — kan ej förändras. Kan alltså riskfritt delas mellan objekt
- Klassen `StringBuffer` kan också användas för att representera stängar men dessa kan ändra både till längd och innehåll. Bra alternativ om långa strängar ska byggas upp i en loop (t ex i `toString`-metoder).
- Det finns en stor mängd metoder för att hantera strängar. Se javadokumentationen!

Konverteringar från tal till `String`

Några olika sätt:

- Genom konkatenering med en sträng. Exempel:

```
int x = 12;
String s = "" + x;
```

Konverteringar från tal till `String`

Några olika sätt:

- Genom konkatenering med en sträng. Exempel:

```
int x = 12;
String s = "" + x;
```

- Med hjälp av metoden `String.valueOf(x)` där `x` är av någon primitiv datatyp.

Konverteringar från tal till `String`

Några olika sätt:

- Genom konkatenering med en sträng. Exempel:

```
int x = 12;
String s = "" + x;
```

- Med hjälp av metoden `String.valueOf(x)` där `x` är av någon primitiv datatyp.

- Med metoden `String.format`. Exempel:

```
String s = String.format("%5.2f \t %9.6f \t %12d",
                         Math.PI, Math.E, 42);
```

Konvertering från `String` till tal

- Kan (bl a) göras med hjälp av parsemetoderna i omslagsklasserna. Exempel:

```
double d = Double.parseDouble("2.5");
```

  Kan naturligtvis bli fel

Konvertering från `String` till tal

- Kan (bl a) göras med hjälp av parsemetoderna i omslagsklasserna. Exempel:

```
double d = Double.parseDouble("2.5");
```

  Kan naturligtvis bli fel

- Kan också göras med klassen `Scanner`. Exempel:

```
Scanner sc = new Scanner("13 3.5 12");
sc.useLocale(Locale.US);
System.out.println(sc.nextInt());           // Skriver 13
System.out.println(sc.nextDouble());        // Skriver 3.5
System.out.println(sc.nextInt());           // Skriver 12
```

Konvertering från `String` till tal

- Kan (bl a) göras med hjälp av parsemetoderna i omslagsklasserna. Exempel:

  ```
  double d = Double.parseDouble("2.5");
  ```

  Kan naturligtvis bli fel

- Kan också göras med klassen `Scanner`. Exempel:

```
Scanner sc = new Scanner("13 3.5 12");
sc.useLocale(Locale.US);
System.out.println(sc.nextInt());          // Skriver 13
System.out.println(sc.nextDouble());       // Skriver 3.5
System.out.println(sc.nextInt());          // Skriver 12
```

Finns stort antal andra klasser som kan användas för detta ändamål!

Programs hantering av fel

Vad ska koden göra om den får en ”omöjlig” uppgift?

Programs hantering av fel

Vad ska koden göra om den får en ”omöjlig” uppgift?

Exempel:

```
public class Measurements {
   private double[] storage;
   private int stored;

   public Measurements(int max) {
       theArray = new double(max);
    }
    ...
 }
```

Programs hantering av fel

Vad ska koden göra om den får en ”omöjlig” uppgift?

Exempel:

```
public class Measurements {
   private double[] storage;
   private int stored;

   public Measurements(int max) {
       theArray = new double(max);
    }
    ...
 }
```

Vad händer nu om vi gör

```
Measurements m = new Measurements(n);
```

där `n` råkar vara negativt eller 0?

DrJava

Programs hantering av fel

Vi får alltså ett avbrott med felutskrift vilket är bra.

Programmet skulle kunna fixa felet genom att inte bry sig om storleken om den är illegal och sätta någon legal storlek.

Programs hantering av fel

Vi får alltså ett avbrott med felutskrift vilket är bra.

Programmet skulle kunna fixa felet genom att inte bry sig om storleken om den är illegal och sätta någon legal storlek.

Till exempel:

```
public Measurements(int max) {
    if (max <= 0) {
        max = 10;
    }
    theArray = new double(max);
}
```

Programs hantering av fel

Vi får alltså ett avbrott med felutskrift vilket är bra.

Programmet skulle kunna fixa felet genom att inte bry sig om storleken om den är illegal och sätta någon legal storlek.

Till exempel:

```
public Measurements(int max) {
    if (max <= 0) {
        max = 10;
    }
    theArray = new double(max);
}
```

Är det bra att göra så?

Är det bra att låta konstruktorn fixa felet?

Är det bra att låta konstruktorn fixa felet?

Inte självklart!

Att skicka in ett omöjligt värde i konstruktorn tyder antingen på

- att man är helt ute och cyklar eller
- att man har en variabel som man *tror* innehåller något men som egentligen innehåller något annat.

Om man låter koden fixa felet bör man åtminstone ge en varningsutskrift!

Värre fel!

Fel som resulterar i felaktigt utan felmeddelande!

Värre fel!

Fel som resulterar i felaktigt utan felmeddelande!

Metoden `get:`

```
public double get(int index) {
  return storage[index];
}
```

Värre fel!

Fel som resulterar i felaktigt utan felmeddelande!

Metoden `get:`

```
public double get(int index) {
  return storage[index];
}
```

Om man anropar med ett negativt värde eller ett värde större än `storage.length` så får man ett avbrott *men* om man anropar met ett värde större än eller lika med `stored` och mindre än `storage.length` så får man inget felmeddelande utan bara ett felaktigt resultat (troligen 0).

Hur ska programmet slå larm i ett sådant fall?

Hur ska programmet slå larm i ett sådant fall?

- Felutskrift:

  `System.out.println("...");`

Hur ska programmet slå larm i ett sådant fall?

- Felutskrift:

  `System.out.println("...");`

- använda `assert`:

  `assert index < stored;`

Hur ska programmet slå larm i ett sådant fall?

- Felutskrift:

  ```
  System.out.println("...");
  ```

- använda `assert`:

  ```
  assert index < stored;
  ```

- kasta ett undantag

  ```
  if (index >= stored) {
    throw new RuntimeException("... ");
  }
  ```

Hur ska programmet slå larm i ett sådant fall?

- Felutskrift:
  ```
  System.out.println("...");
  ```
- använda `assert`:
  ```
  assert index < stored;
  ```
- kasta ett undantag
  ```
  if (index >= stored) {
    throw new RuntimeException("... ");
  }
  ```

Undantag är det mest generella sättet!

Enligt specifikationerna ska ni utnyttja detta i både lektion 9 och 10.

Läs lite mer om detta i lektion 8!

Tips för lektion 9

Tips för lektion 9

1. Sista exemplet i minilektionen om `Scanner`-klassen beskriver hur man läser raderna i en fil.

Tips för lektion 9

1. Sista exemplet i minilektionen om `Scanner`-klassen beskriver hur man läser raderna i en fil.

2. Metoderna `next`, `nextInt`, `nextDouble`, ... läser förbi s k *white space* (dvs blanktecken, tabtecken och radbyten) *framför* ordet (talet) men inte *efter* ordet (talet).
   Det innebär att om ett tal läses från en rad så ger nästa `nextLine` slutet på samma rad och inte raden efter det som talet stor på.

Tips för lektion 9

1. Sista exemplet i minilektionen om `Scanner`-klassen beskriver hur man läser raderna i en fil.

2. Metoderna `next`, `nextInt`, `nextDouble`, ... läser förbi s k *white space* (dvs blanktecken, tabtecken och radbyten) *framför* ordet (talet) men inte *efter* ordet (talet).
   Det innebär att om ett tal läses från en rad så ger nästa `nextLine` slutet på samma rad och inte raden efter det som talet stor på.

3. Följ instruktionerna (som säger *spara som*) när ni hämtar indatafilen `persons.xxx`.
   "Copy-paste" kan ge felaktiga radslut.

Tips för lektion 9

1. Sista exemplet i minilektionen om `Scanner`-klassen beskriver hur man läser raderna i en fil.

2. Metoderna `next`, `nextInt`, `nextDouble`, ... läser förbi s k *white space* (dvs blanktecken, tabtecken och radbyten) *framför* ordet (talet) men inte *efter* ordet (talet).
   Det innebär att om ett tal läses från en rad så ger nästa `nextLine` slutet på samma rad och inte raden efter det som talet stor på.

3. Följ instruktionerna (som säger *spara som*) när ni hämtar indatafilen `persons.xxx`.
   "Copy-paste" kan ge felaktiga radslut.

4. Indatafilen `persons` finns med olika teckenkodningar. Välj den som passar ditt system!