# INF1000 - Forelesning 8: Oppramstyper, HashMap, innstikksortering, javadoc

8. mars 2010,
Christian M. Hansen
Institutt for informatikk, UiO

## Oppramstyper (`enum`) - motivasjon

- Java-program for å registrere møtedeltakelse
  - Vi trenger f.eks. klassene *Møte* og *Deltaker*
  - En deltaker kan enten
    - *Delta* på møtet
    - *Ikke* delta på møtet
    - *Kanskje* delta på møtet
  - Lagres som *deltakerstatus* for hver *deltaker*
- Hvordan representere i Java?
  - To `boolean`-variable: `delta`, `ikkeDelta`
    - Delta: `delta == true`, `ikkeDelta == false`
    - Ikke delta: `delta == false`, `ikkeDelta == true`
    - Kanskje: `delta == true`, `ikkeDelta == true`
- Tungvindt! Bruk heller `enum` i Java...



## `enum` – å lage egne oppramstyper

Brukes til å lage 'typer' som har et lite antall verdier

```
enum Status {
       DELTAR,
       DELTAR_IKKE,
       DELTAR_KANSKJE;
}
class EnumEks {
    public static void main(String[] args) {
        Status s = Status.DELTAR;
        System.out.println("Status s er " + s);
        for (Status ss : Status.values()) {
            System.out.println("Status ss er " + ss);
        }
    }
}
```

```
Status s er DELTAR
Status ss er DELTAR
Status ss er DELTAR_IKKE
Status ss er DELTAR_KANSKJE
```



'enum' kan ha metoder; en enum virker omtrent som en klasse-deklarasjon.

```
public enum Karakter {
    A(6),
    B(5),
    C(4),
    D(3),
    E(2),
    F(0);

    final int verdi;

    boolean erBedre(Karakter k){
        return this.verdi > k.verdi;
    }

    Karakter (int v){
        verdi = v;
    }
}// end enum Karakter
```

```
class EnumEks2 {
  public static void main(String[] args) {

    Karakter min = Karakter.A, din = Karakter.E;
    System.out.println("Karakteren min  er:" + min);

    if (min.erBedre(din))
      System.out.println("Min karakter:" + min
      + " er bedre enn din " + din);
    }
} // end class EnumEks2
```

```
Karakteren min  er:A
Min karakter:A er bedre enn din E
```

## Holde orden på objekter - HashMap

- Ofte har vi flere, mange objekter av en bestemt klasse - eks. :
  - elever på en skole
  - biler som har passert bomringen i Oslo
  - telefonsamtaler fra en bestemt person,.
- Vi har hittil lært arrayer (Elev [] elevene = new Elev[400], Bil [] bomringBiler = new Bil[10000];.....) og må da passe på at vi har
  - nok plass
  - for å finne et bestemt objekt må vi ofte lete gjennom 'hele' arrayen
- Vi skal nå lære en bedre måte å lagre objekter hvor det er viktig å raskt og enkelt finne igjen ett av objektene (som da har ett bestemt kjennetegn som: Navnet til eleven, registreringsnummeret til en bil, ...)
- Et slikt kjennetegn som skiller ett objekt fra alle andre objekter, kaller vi en nøkkel (key)
- **HashMap** er svaret

5

## **HashMap** = lagre objekter med en søkenøkkel

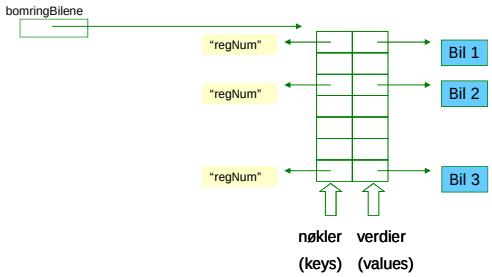
- Brukes til å holde orden på en samling objekter
- Alternativ til arrayer
- Med arrayer kan man:
  - I en array legger vi inn objekter i en bestemt posisjon, og vi må gå tilbake til denne posisjonen/indeksen når vi senere skal se på objektet. Indeksen er et heltall mellom 0 og length-1.
- To viktige forskjeller mellom arrayer og HashMap:
  - I en HashMap oppgir vi en bestemt *nøkkel* når vi legger vi inn et nytt objekt (kalt *verdien*), og vi oppgir denne nøkkelen når vi senere skal se på objektet.
  - En HashMap har ingen gitt lengde når vi lager den. Den 'vokser' når legger inn nye elementer (inntil maskinens minne er oppbrukt…)
- En Hashmap er mer fleksibel måte å lagre flere/mange elementer i et program

6

## Hvordan vi tenker oss en HashMap

en HashMap er som en slags dobbelt-array (f.eks bomringBilene)

```
HashMap bomringBilene = new HashMap();
```




7

## Ulike versjoner i Java 1.4 (gammel) og Java 1.5/1.6 av HashMap

- Vi gjennomgår begge måtene, men anbefaler klart at 1.5-måten nyttes, da den hjelper deg mot visse feil (som ellers er lett å gjøre).
- 1.4 måten gjennomgås (bare) fordi mange gamle programmer inneholder slik kode.

8

## Eksempel på bruk av HashMap (1.5)

Importer pakken java.util

Opprett en HashMap og forteller hvilke klasser nøkkelen og verdier har.

```
import java.util.*;

class BrukAvHashMap {
   public static void main (String[] args) {
      HashMap<String,Person> h = new HashMap <String,Person>();

      String fnr1 = ”30128812344";
      Person per1 = new Person(fnr1, "Harald Olsen");
      h.put(fnr1, per1);

      String fnr2 = ”14109522547";
      Person per2 = new Person(fnr2, ”Lena Torsen");
      h.put(fnr2, per2);

      Person p = h.get(”30128812344”);
   }
}

class Person {
   String fnr;
   String navn;

   Person(String fnr, String navn) {
      this.fnr  = fnr;
      this.navn = navn;
   }
   String fåNavn() { return navn;}
}
```

Legg inn Person-objekt i HashMap’en

Legg inn Person-objekt i HashMap’en

Hent Person-objekt fra HashMap’en



## Opprette en HashMap, Java1.4 (gammel) og Java 1.5-1.6

- I starten av programmet:

  ```
  import java.util.*;
  ```

  Dette importerer pakken java.util hvor bl.a. klassen HashMap ligger.

- I klassen eller metoden som skal bruke HashMap'en **Java 1.4**:

  ```
  HashMap h = new HashMap();
  ```

- I klassen eller metoden som skal bruke HashMap'en **Java 1.5** og nyere (best):

  ```
  HashMap <String,Person> h = new HashMap <String,Person>();
  ```

I Java 1.5 forteller vi hvilke klasser nøkkel- og verdi-objektene kommer fra. Vi sier at vi da låser objektene til både nøkkelen og verdi-objektene til å være av disse typene.

**NB:** Hvis tabellen skal brukes av flere metoder i en klasse, deklareres variabelen ovenfor i starten av klassen (som en objektvariabel).

Hvis tabellen kun skal brukes av en enkelt metode, er det naturlig å deklarere HashMap variabelen ovenfor inni den aktuelle metoden.



## Legge inn verdi-objekt i HashMap (samme i 1.4 og 1.5)

- Et hvilket som helst objekt i Java kan legges inn som verdi i en HashMap, men i 1.5 må det være av den klassen vi har ‘lovet’ systemet.
- Når vi legger et verdi-objekt inn i HashMap'en, må vi samtidig oppgi et nøkkel-objekt av riktig type. Nøkkelen skal entydig identifisere verdi-objektet.
- Vi trenger denne nøkkelen dersom vi senere skal finne eller fjerne verdi-objektet i HashMap'en.
- Eksempel:

  ```
  String fnr = ”30126512345”;
  Person p = new Person(fnr, “Kari Olsen”);
  h.put(fnr, p);
  ```

  Her lager vi først et Person-objekt (med passende argumenter) og legger det deretter inn i tabellen med fødselsnummeret som nøkkel.



- Dersom vi legger inn flere objekter med samme nøkkel, er det bare det sist innlagte objektet som blir liggende i tabellen (de andre overskrives):

  ```
  Person p1 = new Person(...);
  Person p2 = new Person(...);
  Person p3 = new Person(...);
  String navn = "Jens";
  h.put(navn, p1); // p1 legges inn
  h.put(navn, p2); // p2 legges inn og p1 overskrives
  h.put(navn, p3); // p3 legges inn og p2 overskrives
  ```

- Noen ganger må vi konstruere en nøkkel ut fra flere variable for å få entydighet:

  ```
  String lengdegrad = "67.3";
  String breddegrad = "53.3";
  String posisjon = lengdegrad + ";" + breddegrad;
  Fjelltopp fjell = new Fjelltopp(posisjon, “Bjørnefjell”);
  h.put(posisjon, fjell);
  ```

## Hente objekt fra HashMap – Java 1.4 og 1.5

**Java 1.4:** For å hente et objekt med utgangspunkt i nøkkelen:

```
// 1.4: Vi vil finne en person ut fra fødselsnummeret:
Person p = (Person) h.get(fnr);
```

- Legg merke til at vi i 1.4 i starten må skrive i parentes navnet på klassen som objektet tilhører - i dette tilfellet klassen Person.
  - Årsaken er at i 1.4 HashMap'en ikke holder rede på hvilken klasse objektene som legges inn har - bare at det er objekter. Når objektene hentes ut må vi derfor "minne Java på" hvilken klasse objektet var av (dette er egentlig et møte med en avansert og svært nyttig mekanisme i objektorienterte språk som kalles *arv* og som blir tatt opp i INF1010).

**Java 1.5:** For å hente et objekt med utgangspunkt i nøkkelen, trenger vi ikke si hvilken klasse objektet har (det har vi jo sagt i deklarasjonen av HashMapen):

```
// 1.5: Vi vil finne en person ut fra fødselsnummeret:
Person p = h.get(fnr)
```

- **Merk**: å hente et objekt fra en HashMap slik som over medfører *ikke* at objektet fjernes fra HashMap'en (vi får bare en kopi av peker til objektet).

13

## Fjerne objekt fra HashMap

- For å fjerne et objekt med gitt fødselsnummer som nøkkel:

```
h.remove(fnr);
```

- Dersom det ligger et objekt i HashMap'en med den gitte nøkkelen, blir objektet fjernet og setningen ovenfor returnerer med en peker til objektet som fjernes.
- Dersom det ikke ligger et objekt i HashMap'en med den gitte nøkkelen, returnerer setningen ovenfor verdien `null`.

14

## Løp gjennom alle objekter i HashMap **Java 1.4**

- For å løpe gjennom alle objektene i en HashMap, lager vi en *oppramsing*:

```
Iterator it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
    Person p = (Person) it.next();
    System.out.println("Navn: " + p.fåNavn());
}
```

15

## Løp gjennom alle objekter i HashMap **Java 1.5**

- For å løpe gjennom alle objektene i en HashMap, lager vi en *oppramsing og låser samtidig det vi skal hente til en bestemt klasse*:

```
Iterator<Person> it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
    Person p = it.next();
    System.out.println("Navn: " + p.fåNavn());
}
```

- Vi kan også i 1.5 nytte ***den nye for-løkka*** som automatisk lager en iterator

```
for (Person p: h.values()) {
    System.out.println("Navn: " + p.fåNavn());
}
```

16

## To måter å løpe gjennom en HashMap – 1.5

- Løpe gjennom objektene (som på forrige foil):

```
Iterator<Person> it = h.values().iterator();

while (it.hasNext()) {
  Person p = it.next();
  <gjør noe med objektet p>
}
```

- Løpe gjennom nøklene:

```
Iterator <String> it = h.keySet().iterator();

while (it.hasNext()) {
  String nøkkel =  it.next();
  <gjør noe med nøkkelen>
}
```

17

## Metoder i HashMap

| Metode | Eksempel | Beskrivelse |
|---|---|---|
| put | `h.put(nøkkel, objekt);` | Legg inn objekt med gitt nøkkel |
| get -1.4<br>get -1.5 | `Person p = (Person) h.get(nøkkel);`<br>`Person p = h.get(nøkkel);` | Finn objekt |
| remove | `h.remove(nøkkel);` | Fjern objekt |
| containsKey | `if (h.containsKey(nøkkel)) {`<br>`  // gjør et eller annet`<br>`}` | Sjekk om nøkkel finnes i tabell |
| values | `Iterator it = h.values().iterator();` | Lag oppramsing av objektene |
| keySet | `Iterator it = h.keySet().iterator();` | Lag oppramsing av nøklene |

18

## Iterator (oppramsing)

| | Eksempel<br>`Iterator it1 = h.values().iterator();`<br>`Iterator it2 = h.keySet().iterator();` | Beskrivelse<br>deklararasjon |
|---|---|---|
| hasNext() | `while (it.hasNext()) {`<br>`  < les neste og gjør noe>;`<br>`}` | returnerer true hvis flere objekter igjen i oppramsingen |
| next() 1.5<br>next() 1.4 | `Person p = it.next();`<br>`Person p = (Person) it.next();` | Finn neste objekt |
| remove() | `Person p = it.next();`<br>`if (p.navn.equals("Arne"))`<br>`  it.remove();` | Fjern siste objekt som ble returnert med next() |
| | `while (it1.hasNext()) {`<br>`  Person p1 = it1.next();`<br>`}`<br>`for (Person p2 : h.values()){`<br>`  ....`<br>`}` | To måter å gå gjennom alle verdiene (objektene) i h |

19

```
import java.util.*;
import easyIO.*;

class Hasheksempel {
  public static void main(String[] argv) {
    In tastatur = new In();
    HashMap <String,Person> personregister =  new HashMap <String,Person>();
    System.out.print("Antall personer som registreres : ");
    int ant = tastatur.inInt();

    for (int i = 0; i < ant; i++) {
      System.out.println("Gi neste person");
      Person p = new Person(tastatur);
      personregister.put(p.telefonnr, p);
    }
    // Skriv ut alle personobjektene
    System.out.println("Viser alle personer" +
    "(ukjent rekkefølge):");

    for (Person p: personregister.values()){
      p.skrivData();
    }
  }
}
```

```
class Person {
  String navn, adresse, telefonnr;

  Person (In tastatur) {
    System.out.print("Oppgi navn : ");
    navn = tastatur.inLine();
    System.out.print("Oppgi adresse : ");
    adresse = tastatur.inLine();
    System.out.print("Oppgi telefonnummer : ");
    telefonnr = tastatur.inLine();
  }
  void skrivData() {
    System.out.println("Navn : " + navn);
    System.out.println("Adresse : " + adresse);
    System.out.println("Telefonnummer : "
      +telefonnr);
  }
  String fåNavn() { return navn;}
}
```

Eksempel fra boka s.186

## Sortering

- Lære å løse et vanskelig problem
  - Sortering – mange metoder, her *innstikksortering*
  - Sortere hva:
    - Heltall
    - Tekster
- Lære abstraksjon
  - Når vi har løst ett problem, kan lignende problemer løses tilsvarende
- Lære å lage 'proff' programvare ved å lage en generell klasse (en vektøyboks) for sortering
  - Hvordan deklarere en slik klasse
  - Javadoc – lage dokumentasjon
  - Testing
  - Hvordan utvikle programmet



## Sortering

- Mange datatyper kan sorteres
  - Tall
  - Tekster (leksikografisk = i samme rekkefølge de ville stått i et leksikon)
  - Tabeller av tekster eller tall
- Vi må ha en algoritme (fremgangsmåte) for sortering
  - Det finns mange metoder for sortering
  - Dere skal lær den som er raskest når vi skal sortere få elementer, si < 50 elementer



## Hvorfor sorterer vi

- For å få noen tall i sortert rekkefølge
  - eks: lotto-tallene
- Sortere tekster (navnelister)
- Sortere noen opplysninger som hører sammen. Sorterer da på en av opplysningene.
  - Eks. Telefonkatalogen: navn, adresse, telefonnummer sortert på navn



## Vi skal først lære å sortere heltall

- Dette skal vi så med minimale endringer bruke til å sortere:
  - String-arrayer (tekster)

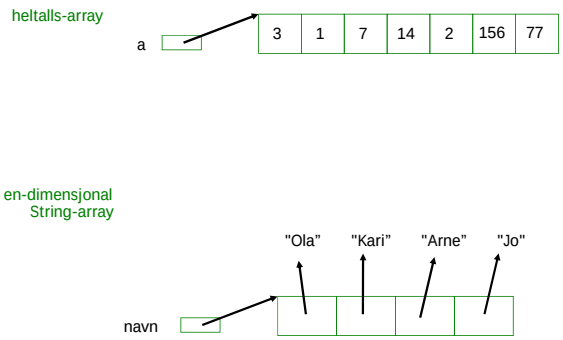
## Vi ønsker en klasse med to varianter av sortering: Heltall og tekster

```
public class ISort {

  public static void sorter(int [] a) {

  }
  public static void sorter(String [] a) {

  }
} // end class ISort
```



**Test-program** for sortering

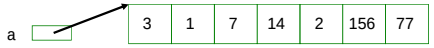
```
class TestInnstikkSortering
{

  public static void  main ( String[] args)  {

     int [] a = {3,1,7,14,2,156,77};
     String [] navn = {"Ola", "Kari", "Arne", "Jo"};

       // sorter heltall - skriv ut
      ISort.sorter(a);
      for (int i = 0; i < a.length; i++)
        System.out.println("a[" + i +"]= " + a[i]);

        System.out.println("\n Test tekst-sortering:");

      // sorter Stringer - skriv ut
      ISort.sorter(navn);
      for (int i = 0; i < navn.length; i++)
        System.out.println("navn[" + i +"]= " + navn[i]);

}}
```




Test av test-programmet med **tomme** sortering-metoder

```
>java InnstikkSortering
a[0]= 3
a[1]= 1
a[2]= 7
a[3]= 14
a[4]= 2
a[5]= 156
a[6]= 77

 Test tekst-sortering:
navn[0]= Ola
navn[1]= Kari
navn[2]= Arne
navn[3]= Jo
```

## En algoritme for å sortere heltall – innstikksmetoden

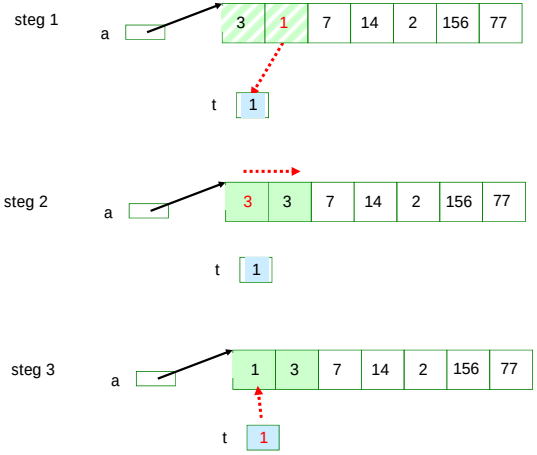



- Se på arrayen *ett for ett* element fra venstre mot høyre
- Sorterer det vi hittil har sett på, ved :
  - Hvis det nye elementet vi ser på **ikke** er sortert i forhold til de vi allerede har sett på:
    - Ta ut dette elementet (gjem verdien i en variabel **`t`**)
    - Skyv på de andre elementene vi her sett på en-etter-en, ett hakk høyreover til elemetet i **`t`** kan settes ned på sortert plass.
    - Da er den delen vi har sortert ett element lenger (fra venstre)
  - Når vi har sett på alle elementene, er hele arrayen sortert
  - Observasjon : Det første elementet er sortert i forhold til seg selv

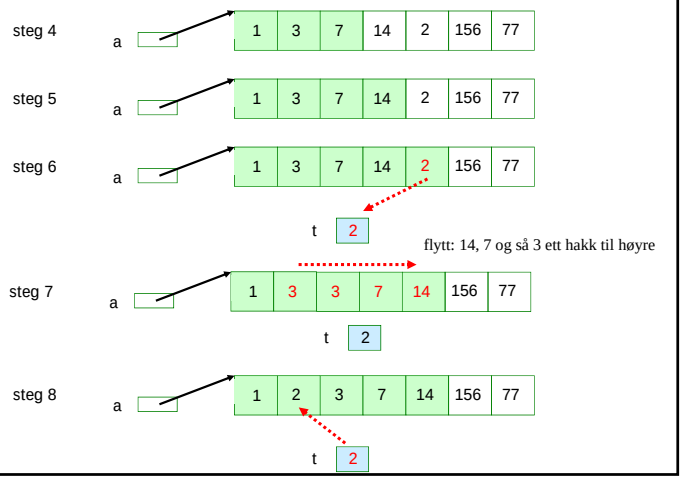
a



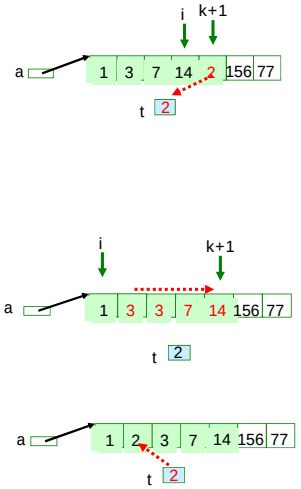
Sorter 1 på plass i forhold til 3




7 og 14 står riktig, Sorter 2 på plass i forhold til : 1,3,7,14




Kode for å flytte **ett** element på plass :



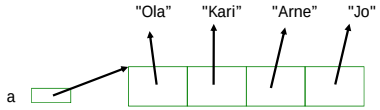

```
// a[k +1] står på
// feil plass, ta den ut
int t = a[k + 1], i = k;

// skyv a[i] mot høyre ett hakk til
// vi finner riktig plass til t
while (i >= 0 && a[i] > t) {
   a[i + 1] = a[i];
   i--;
}

// sett t inn på riktig plass
  a[i + 1] = t;
```

```
public class ISort {

  public static void sorter(int [] a) {
       for (int k = 0 ; k < a.length-1; k++) {
         if (a[k] > a[k+1]) {
              // a[k +1 ] står på feil plass, ta den ut
              int t = a[k + 1], i = k;
              // skyv a[i] mot høyre ett hakk til
              // vi finner riktig plass til t
              while (i >= 0 && a[i] > t) {
                     a[i + 1] = a[i];
                     i--;
              }
               // sett t inn på riktig plass
              a[i + 1] = t;
          }
       }
  } // end heltall-sortering
```

```
>java InnstikkSortering
a[0]= 1
a[1]= 2
a[2]= 3
a[3]= 7
a[4]= 14
a[5]= 77
a[6]= 156

 Test tekst-sortering:
navn[0]= Ola
navn[1]= Kari
navn[2]= Arne
navn[3]= Jo
```

Resultat av sortering med heltalls-metoden kodet, den andre uten kode

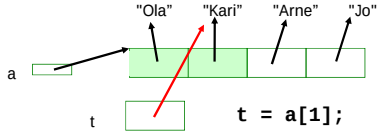
## Sortering av tekster (String)

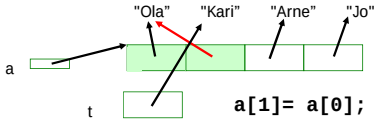


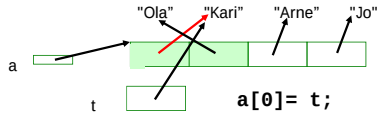

- Vi skal sortere denne ved å bytte om på pekerne (la a[0] peker på "Arne",..osv) med innstikkmetoden



Sortere de to første elementene ved å bytte om pekere



```
public static void sorter(int [] a) {
       // Sorterer heltallsarrayaen 'a'.
       for (int k = 0 ; k < a.length-1; k++) {
          if (a[k] > a[k+1]) {
               int t = a[k + 1];
               int i = k;
               while (i >= 0 && a[i] > t) {
                      a[i + 1] = a[i];
                      i--;
               }
               a[i + 1] = t;
       }  }
  } // end heltall-sortering

  public static void sorter(String [] a) {
        // Sorterer String-arrayen 'a'.
       for (int k = 0 ; k < a.length-1; k++) {
           if( a[k].compareTo(a[k+1]) > 0 ){
               String t = a[k + 1];
               int i = k;
               while (i >= 0 &&  ( a[i].compareTo(t) > 0) ){
                      a[i + 1] = a[i];
                      i--;
               }
               a[i + 1] = t;
       }  }
  } // end String-sortering
```

```
String s = "…";
String t = "…";
s.compareTo(t)
```

- returverdi < 0 hvis s er leksikografisk mindre enn t
- returverdi = 0 hvis s og t er tekstlig like
- returverdi > 0 hvis s er leksikografisk større enn t

```
>java InnstikkSortering
a[0]= 1
a[1]= 2
a[2]= 3
a[3]= 7
a[4]= 14
a[5]= 77
a[6]= 156

 Test tekst-sortering:
navn[0]= Arne
navn[1]= Jo
navn[2]= Kari
navn[3]= Ola
```

Test med heltall og enkel String-sortering kodet

## Javadoc – proff dokumentasjon av klassene

- Legg inn spesielle kommentarer i programmet ditt (over hver metode og klasse)
- I disse kommentarene kan man legge HTML-kommandoer (som <br> for å få linjeskift)
- Kjør programmet 'javadoc', og automatisk har du en fin dokumentasjon
- Største fordel: Kode og dukumentasjon vedlikeholdes på samme fil.

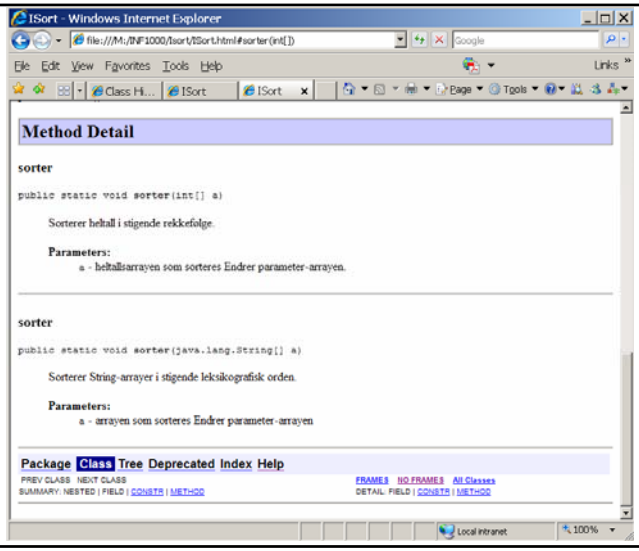
39

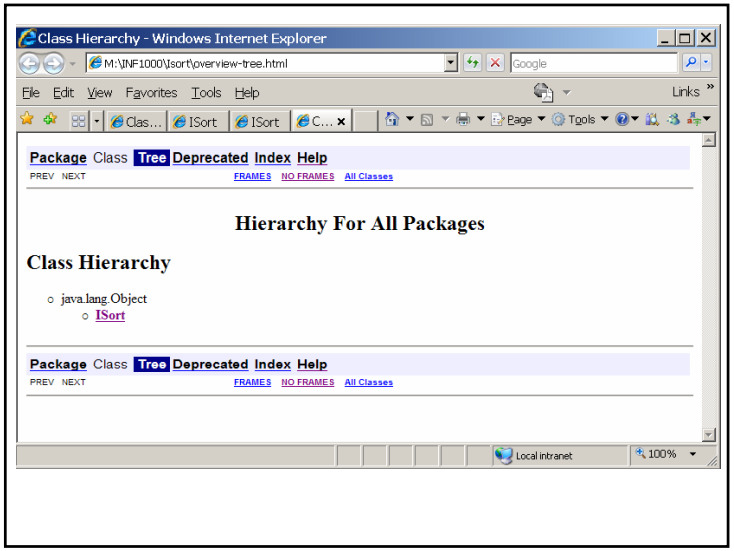
```
/**
* Klasse for sortering etter 'innstikk-metoden', se
* Rett på Java - kap.5.7.
* Sortering av heltallsarray, tekster og en to-dimensjonal
* tekst-array  sortert etter verdiene i første kolonne.<br>
*
* N.B. Bare velegnet for mindre enn 100 elementer.
*
* Copyright : A.Maus, Univ. i Oslo, 2008
*********************************************************/
public class ISort {

  /**
  * Sorterer heltall i stigende rekkefølge.
  * @param a heltallsarrayen som sorteres. <br>
  * Endrer parameter-arrayen.
  ****************************************/
  public static void sorter(int [] a) {
  }
  /**
  * Sorterer String-arrayer i stigende leksikografisk orden.
  * @param a arrayen som sorteres.<br>
  * Endrer parameter-arrayen
  ****************************************/
  public static void sorter(String [] a) {

  }

} // end class ISort
```

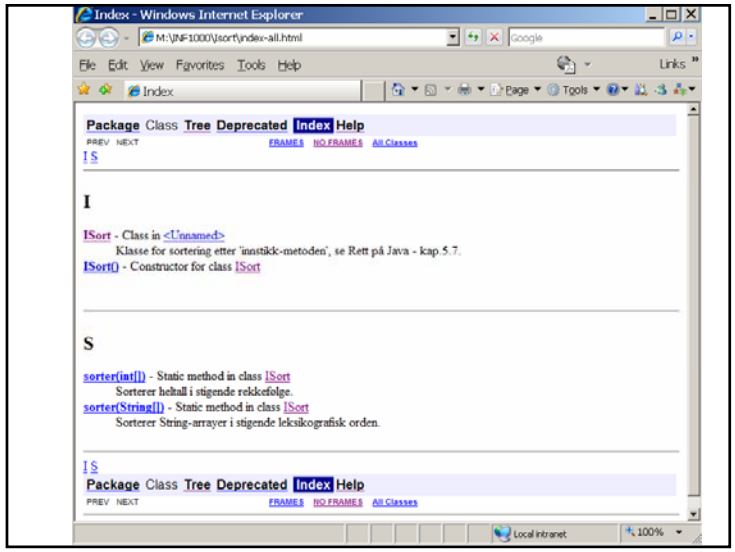
## Dokumentasjon av klassen og metodene - javadoc

```
M:\INF1000\Isort>javadoc –package  ISort.java
Loading source file ISort.java...
Constructing Javadoc information...
Standard Doclet version 1.5.0_02
Building tree for all the packages and classes...
Generating ISort.html...
Generating package-frame.html...
Generating package-summary.html...
Generating package-tree.html...
Generating constant-values.html...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating help-doc.html...
Generating stylesheet.css...

M:\INF1000\Isort>
```

41


Index - Windows Internet Explorer
M:\INF1000\Isort\index-all.html
Google
File Edit View Favorites Tools Help
Links
Index
Page Tools
Package Class Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES All Classes
I S
I
ISort - Class in <Unnamed>
Klasse for sortering etter 'innstikk-metoden', se Rett på Java - kap.5.7.
ISort() - Constructor for class ISort
S
sorter(int[]) - Static method in class ISort
Sorterer heltall i stigende rekkefølge.
sorter(String[]) - Static method in class ISort
Sorterer String-arrayer i stigende leksikografisk orden.
I S
Package Class Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES All Classes
Local intranet
100%