Uke 10 - UML: (Objekt- og) Klasse-diagrammer, litt javadoc Hashmap og Innstikksortering

25 oktober 2005,
Arne Maus
Inst. for informatikk, UiO

UML-diagrammer av programmene våre

- Hvorfor tegne diagrammer over programmene
  - Oversikt
  - Samarbeid med andre programmererer / systemutviklere
  - Arkitekter, ingeniører tegner først, så bygger de !
  - Enklere å endre en tegning enn programmet
- Objektdiagrammer
- Klassediagrammer
- UML – diagrammene er litt annerledes enn det vi har tegnet hittil (men mye av det samme)

  (i UML er det ca 10 andre diagramtyper vi ikke skal lære)

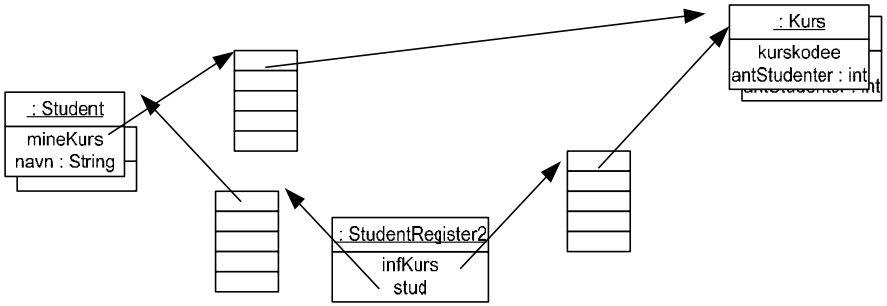
Objekt-diagrammer

- Vi tegner en typisk situasjon av objekter i systemet vårt, når vi har fått datastrukturen på plass.
- Vi tegner og navngir bare de mest sentrale dataene som:
  - pekere
  - peker-arrayer
  - noen sentrale variable i objektene

Et helt enkelt studentregister med kurs, studenter og registeret

- Vi har Studenter på Ifi som første semester tar tre kurs, samtidig som vi har behov for å registrere kurs og *hvor mange* studenter som tar hvert kurs (men ikke *hvilke* studenter).
- Vi tegner først en tenkt datastruktur – et UML objektdiagram
- så skriver vi programmet

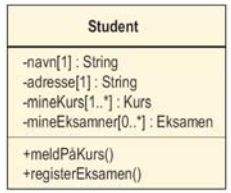
Objektdiagrammet er en forenkling av programmet. Det tar bare med den essensielle datastrukturen (mest pekere og peker-arrayer) som holder datastrukturen sammen



Klassediagrammer

- En mer kompakt måte enn objektdiagrammer å tegne sammenhengen i programmet
- Skiller seg fra objektdiagrammer ved at vi ikke direkte tegner datastrukturen (pekere og pekerarrayer), men bare forhold (assosiasjoner, forbindelser) mellom klassene.
- I klassediagrammer dokumenterer vi også sentrale metoder.
- Forholdene er linjer med et logisk navn og antall objekter i hver ende
- Anta at vi har laget en class Konto med tre objektvariable: saldo, eier og adresse og en metode: settinn().

Tre (fire) mulig felter i tegning av en klasse



- Navnefeltet (alltid)
  - klassenavnet

Kan utelates:

- Variabelfeltet (attributtene)
  - variabelnavn evt. med type
- Metode-feltet
  - Evt med parametere og returverdi
- (Unntaks-feltet)

Symboler for synlighet
(fra resten av programmet)

\+ public

\- private

\# protected

~ package

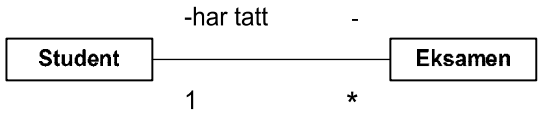
UML Klassediagrammet kan nyttes til

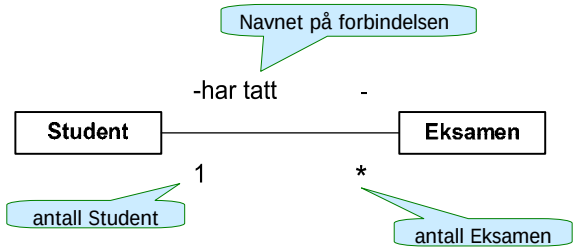
- Modell av problemområdet (domenemodell)
- Modell av klassene i programmet

  (+ modell av databasen,....)
- Men siden vi skal modellere virkeligheten en-til-en i programmet vårt, så blir de like i utgangspunktet

Forhold mellom klasser

- **En student har null eller flere eksamener**
- Vi tegner et forhold mellom to klasser som har med hverandre å gjøre logisk sett, og:
- hvor vi i programmet vil kunne følge pekere for å få adgang til variable eller metoder
- Vi skriver hvor mange objekter det maksimalt på ett tidspunkt kan være på hver side av et slikt forhold
- Siden vi med: Eksamen mener en avlagt enkelt-eksamen vil en Eksamen bare være tilknyttet en bestemt student



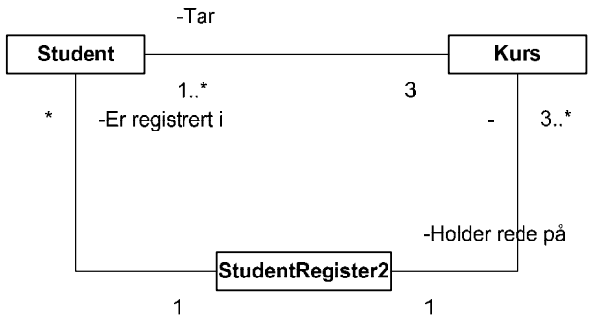


- Forbindelsen leses fra venstre:
  En student har tatt null, en eller flere Eksamener"
- Antallet objekter angis slik:

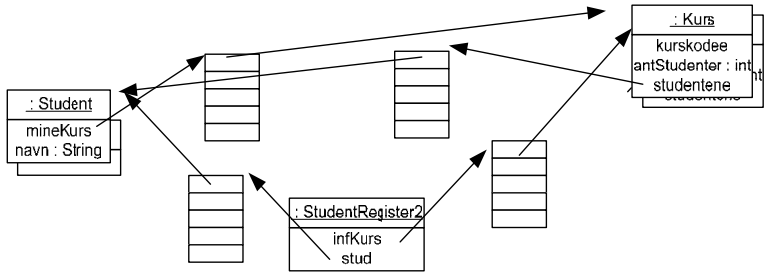
Skrivemåte	Betydning
1	en
*	null, en eller flere
1..*	minst en
3..*	minst tre
3,4,5	tre, fire eller fem

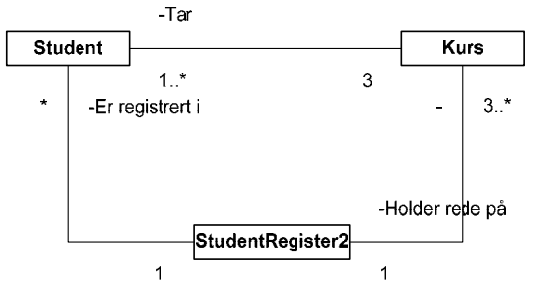
Studentregister2 – med tillegg: klassen Kurs vet *hvilke* Studenter som tar kurset

- Et studentregister holder orden på studentene og kursene, og en student tar 3 kurs hvert semester



Sammenligning: Objektdiagram og Klassediagram





Regler for å plassere riktige antall på et forhold

1. Anta at du står i **ett** objekt av en klasse og ser over til (langs en forbindelse) til en annen klasse:
2. Hvor mange objekter ser du da maksimalt *på et gitt tidspunkt* av den andre klassen
3. Det antallet noteres (jfr. tabellen) på den andre siden
4. Du går så over forbindelsen til den andre klassen og antar at du nå står i **ett** objekt av denne klassen og gjenntar pkt. 1-3

Hvilke forhold skal vi ha med i klassediagrammet

- Slike forhold hvor ett objekt av den ene klassen:
  - inneholder
  - består av
  - eier,..

  en eller flere objekter av den andre klassen
- Det vi i programmet vil følge en peker for å få tak i verdien på visse variable i den andre klassen eller kalle en metode.

  Det er da ikke 'naturgitt' hvilke forhold vi har i et klassediagram, det avhenger av hvilke spørsmål vi vil være interessert i å svare på.

HashMap = lagre objekter med en søkenøkkel

- Brukes til å holde orden på en samling objekter
- Alternativ til arrayer
- Akkurat som for 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.
- Viktig forskjell mellom arrayer og HashMap:
  - I en HashMap oppgir vi en bestemt *nøkkel* (vanligvis en tekststreng) når vi legger vi inn et nytt objekt (kalt *verdien*), og vi oppgir denne nøkkelen når vi senere skal se på objektet. Dvs. indeksen er en tekststreng.

Ulike versjoner i Java 1.4 (gammel) og Java 1.5

- 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.

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;
}
}
```

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

- 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** (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 variabelen ovenfor inni den aktuelle metoden.

Legge inn objekt i HashMap (samme i 1.4 og 1.5)

- Et hvilket som helst objekt i Java kan legges inn i en HashMap
- Når vi legger et objekt inn i HashMap'en, må vi samtidig oppgi en nøkkel, dvs en tekststreng som entydig identifiserer objektet.
- Vi trenger denne nøkkelen dersom vi senere skal finne eller fjerne 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 vårens INF1010).

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

```
// 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).

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**.

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());
}
```


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());
 }
```

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>
}
```

- Løpe gjennom nøklene:

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

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


Metoder i HashMap

Metode	Eksempel	Beskrivelse
put	`h.put(nøkkel, objekt);`	Legg inn objekt med gitt nøkkel
get -1.4 get -1.5	`Person p = (Person) h.get(nøkkel);` `Person p = h.get(nøkkel);`	Finn objekt
remove	`h.remove(nøkkel);`	Fjern objekt

containsKey	`if (h.containsKey(nøkkel)) {`
	`  // gjør et eller annet`
	`}`
		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

Metoder i Iterator (oppramsing)

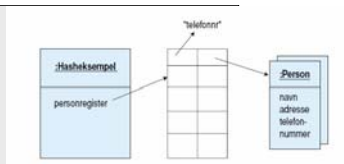
Metode	Eksempel	Beskrivelse

hasNext()	`while (it.hasNext()) {`
	`  < les neste og gjør noe>;`
	`}`
		returnerer true hvis flere objekter i oppramsingen

next() 1.5 next() 1.4	`Person p = it.next();` `Person p = (Person) it.next();`	Finn neste objekt

remove()	`Person p = it.next();`
	`if (p.navn.equals("Arne"))`
	`  it.remove();`
		Fjern siste objekt som ble returnet med next()


```
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("Antall gjenværende personer " +(ant - i));
         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);
   }}
```

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 titalls (hundretalls) 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)

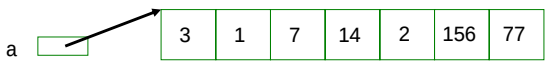
Vi ønsker en klasse med to varianter av sortering: Heltall og tekster

```
public class ISort {

  public static void sorter(int [] a) {

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

  }
} // end class ISort
```

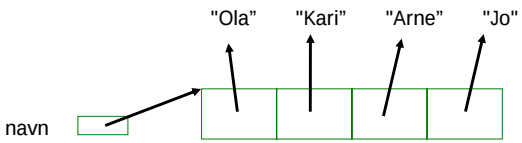

Test-program for sortering

```
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]);

         System.out.println("\n Test 2dim tekst-sortering:");


}}
```

heltalls-array



en-dimensjonal String-array



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

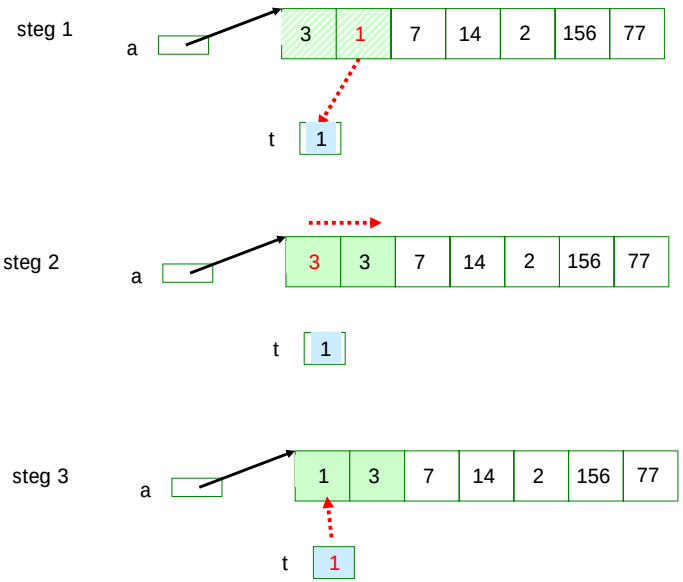
```
>java InnstikkSortering
a[0]= 3
a[1]= 1
a[2]= 7
a[3]= 14
a[4]= 2
a[5]= 156
a[6]= 77

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

En algoritme for å sortere heltall – innstikksmetoden

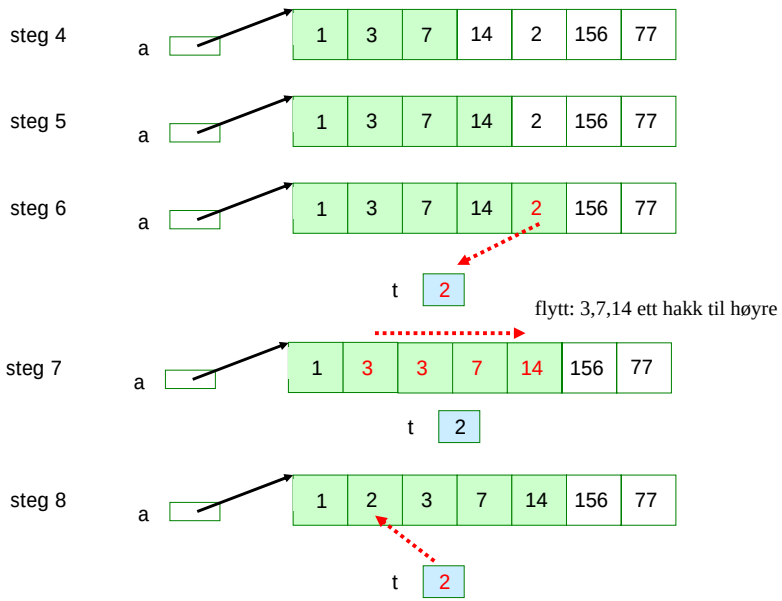
- Se på arrayen ett for ett element fra venstre
- Sorterer det vi hittil har sett på ved :
  - Hvis det nye elementet vi ser på **ikke** er sortert i forhold til de vi allerede har sett på:
    - Ta ut dette elementet (gjem verdien i en variabel **`t`**)
    - Skyv på de andre elementene vi her sett 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 til lenger (fra venstre)
  - Når vi har sett på alle elementene, er hele arrayen sortert
  - Observasjon : Det første elementet, er det sortert i forhold til seg selv

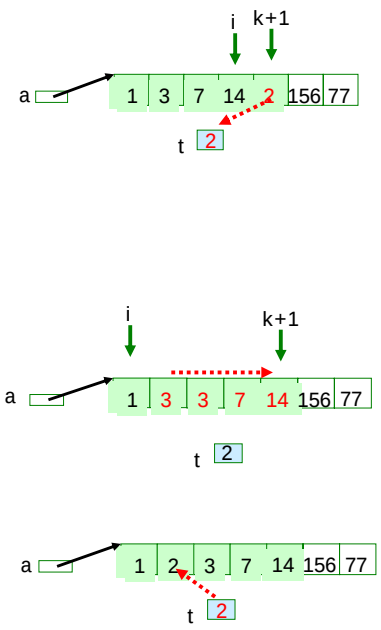
a

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 :

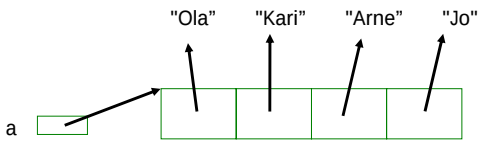
```
// a[k +1] står muligens på
// feil plass, ta den ut
int t = a[k + 1], i = k;

// 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++) {
             // a[k +1 ] står muligens 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
```

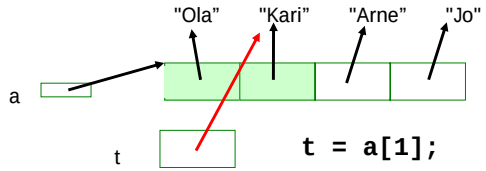
Resultat av sortering med heltalls-metoden kodet, den andre uten kode

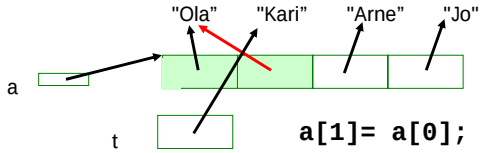
Sortering av tekster (String)

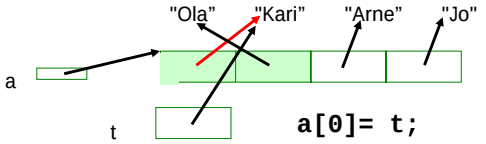


- Vi skal sortere denne ved å bytte om på pekerne (la a[0] peker på ”Arne”,..osv) med innstikkmetoden

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++) {
              int t = a[k + 1], 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++) {
              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
```

```
>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)
- Kjør programmet 'javadoc', og automatisk har du en fin dokumentasjon


```
/**
* 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, 2003
*************************************************************/
public class ISort {

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

  }

} // end class ISort
```

Dokumentasjon av klassen og metodene - javadoc

```
>javadoc 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...
```