## Kursus 02199: Programmering
## Tabeller (eng.: arrays): afsnit 6.1-6.4

**Anne Haxthausen, IMM, DTU**

1. Hvad er en tabel? (afsnit 6.1)
2. Tabel-typer. (afsnit 6.1)
3. Hvordan erklærer og opretter man en tabel? (afsnit 6.1)
4. Hvordan initialiserer man en tabel? (afsnit 6.1)
5. Operationer på tabeller (længde af, opslag i, opdatering). (afsnit 6.1)
6. Tabel aliasing. (afsnit 6.1)
7. Hvordan sammenligner man tabel objekter? (Lighed.)
8. Hvordan kopierer man indholdet fra en tabel til en anden?
9. Tabeller som argumenter til metoder. (afsnit 6.1)
10. Tabeller, hvor elementerne er objekter. (afsnit 6.2)
11. Flerdimensionelle tabeller. (afsnit 6.4)
12. Tabeller med variabel længde. (afsnit 6.2)
13. Anvendelse af tabeller til sortering. (afsnit 6.3)



## Hvad er en tabel?

**Generelt grundbegreb:**

En *tabel* er en samling elementer nummereret 0, 1, 2, . . . .
Alle elementerne har samme type.

**Eksempel:** grafisk illustration af en tabel med 12 elementer af heltals type:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 4 | 1 | 7 | 19 | 2 | 5 | 1 | 0 | 13 | 0 | 0 |

**I Java:** er tabeller en speciel slags objekter, hvor elementerne svarer til felter (variable).

Resten af forelæsningen handler om tabeller i Java.



## Tabel-typer

Hvis en tabels elementer har type `T`, så har tabellen selv type `T[]`.

I eksemplet ovenfor var typen **int**`[]`.

## Opsummering: forskellige slags typer

- primitive typer:
  - heltals-typer: **int**, ...
  - kommatals-typer: **double**, ...
  - tegn-typen: **char**
  - sandhedsværdi-typen: **boolean**
- reference typer:
  - klasse-typer: `String`, `Time`, ...
  - interface-typer: `TimeInterface`, ...
  - tabel-typer **int**`[]`, `Time[]`, ...



## Hvordan erklærer og opretter man en tabel?

Erklæring af tabel-variabel (opretter ikke selve tabellen):

```
int[] days;
```

Oprettelse af tabel med 12 elementer, (alle initialiseret til 0):

```
days = new int[12];
```

days → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Samtidig erklæring og oprettelse af tabel:

```
int[] days = new int[12];
```

Tabellen selv er et objekt, og tabel-variablen `days` er blot en henvisning ('reference') til tabellen.

## Hvordan initialiserer man en tabel?

Erklæring, oprettelse og initialisering på en gang:

```
int[] days = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 28 | 31 | 30 | 31 | 30 | 31 | 31 | 30 | 31 | 30 | 31 |



## Operationer på tabeller

**Længden af en tabel**

Længden af (dvs. antal elementer i) en tabel beregnes med:

```
days.length
```

I eksemplet er `days.length` lig med 12.

**Opslag i tabellen**

Skaf indholdet af element 3:

```
days[3]
```

Udtrykket i `[...]` kaldes for *indekset*.

Lovlige indeksværdier er `0, 1, 2, ..., days.length-1`.

Bruges et ulovligt indeks afbrydes programmet med en fejlmelding:

```
java.lang.ArrayIndexOutOfBoundsException: ...
```



**Opdatering af et element i tabellen**

Sæt element nummer 1 til 29:

```
days[1] = 29;
```



## Eksempel1 på brug af tabeller

En metode, der tager et månedsnummer (1–12) som argument og returnerer antal dage i måneden:

```
static int monthlen(int mth) {
  int[] days = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
  return days[mth-1];
}
```

## Eksempel2 (opgave 17) på brug af tabeller

```
public class Dato {
  final static String[] danishMonths =
    {"januar", "februar", "marts", "april", "maj", "juni", "juli",
      "august", "september", "oktober", "november", "december"};

  private int year, month, day; //felterne
  ...
  public String danishText() {
    return day +". "+ danishMonths[month-1] +" "+ year;
} }

public class TestDato {
  public static void main(String[] args) {
    Dato today = new Dato(2001, 10, 23);
    System.out.println("Danish text: " + today.danishText());
}
```

`Testdato` udskriver: `Danish text: 23. oktober 2001`



## Tabel aliasing

To tabel-variable kan henvise til samme tabel objekt.

**Eksempel**

```
int[] days1 = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int[] days2 = days1
```

**Implikation ved tabel aliasing:**

Hvis indholdet af en tabel ændres, så betyder det en ændring for *alle* de referencer (dvs. tabel variable), der peger på den tabel.

**Eksempel**

```
days1[1] = 29;
```

nu er både `days1[1] == 29` og `days2[1] == 29`



## Hvordan sammenligner man indholdet af to tabeller?

**Eksempel**

Antag givet:

```
int[] days1 = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int[] days2 = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Hvordan kan vi undersøge om de tabeller, `days1` og `days2` refererer til, har samme indhold?



## Hvordan sammenligner man indholdet af to tabeller, fortsat?

**Forkert:**

```
days1 == days2
```

dur *ikke* – den afgører, om `days1` og `days2` refererer til den samme tabel, d.v.s. er aliaser. (den vil altså give **false**.)

**Rigtigt:** Brug `java.util.Arrays.equals(days1, days2)` eller lav selv en metode, der undersøger det:

```
static boolean equals(int[] tabel1, int[] tabel2) {
  boolean same;
  same = (tabel1.length == tabel2.length);
  for (int i=0; same && i < tabel1.length; i = i+1)
     same = (tabel1[i] == tabel2[i]);
  return same;
}
```

`equals(days1, days2)` giver nu **true**,da indholdet af tabellerne, `days1` og `days2` peger på, er ens.

## Tabeller som argumenter til metoder.
## Eksempel: kopiering af indhold fra en tabel til en anden

Opgave: lav en metode, `copy`, der kopierer elementerne fra en heltalstabel til en anden heltalstabel med samme længde.

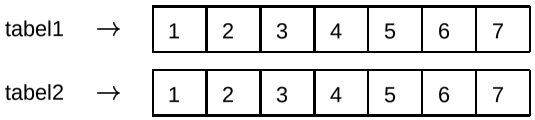
**Eksempel:** Antag givet

```
int[] tabel1 = { 1,  2,  3,  4,  5,  6,  7};
int[] tabel2 = { 8,  9, 10, 11, 12, 13, 14};
```

Da skal lageret efter udførelse af

```
copy(tabel1, tabel2);
```

se således ud:






## Tabeller som argumenter til metoder.
## Eksempel fortsat: kopiering af indhold fra en tabel til en anden

Hvilken af de to nedenstående metoder, gør det rigtige?

```
static void copy(int[] from, int[] to) { to = from; }
```

```
static void copy(int[] from, int[] to) {
  for (int i = 0; i < from.length; i = i+1)
    to[i] = from[i];
}
```

Hvad gør den forkerte?



## Hvad sker der med variable, der bruges som aktuelle parametre til metoder?

Java bruger *call-by-value*, når der overføres værdier til en metode — svarer til kopiering af den *aktuelle* parameters værdi til den *formelle* parameter i metoden. Efter et kald af metoden har den aktuelle parameter (her: variabel) den samme værdi som før kaldet af metoden.

**Betydning for tabeller som argumenter til metoder:**

- når variable af tabel typer er argumenter til en metode, kopieres *referencen* til tabellen, ikke tabellen selv
- hvis metode kroppen ændrer den formelle parameter reference (sætter den til at pege på en ny tabel), får det *ikke* betydning for argumentet
- hvis metode kroppen ændrer tilstanden (indholdet) af den tabel, den formelle parameter peger på, sker det samme for argumentet



## Parametrene `args` i `main` metoder

`args` i

```
public static void main (String[] args) { ... }
```

er en tabel af tegnstrenge, som man giver som parametre til programmet, når man kører det.

Inde i `main` metoden, kan man referere til dem med `args[0]`, `args[1]`, etcetera.

**Eksempel:**

```
class Pension {
  public static void main (String[] args) {
    double pension =  Integer.parseInt(args[0]) * 0.1;
    System.out.println("Din pensionsopsparing er: " + (int) pension);
  }
}
```

Eksempel på kørsel af dette program:

```
$ java Pension 300000
Din pensionsopsparing er: 30000
```

## Tabeller, hvor elementerne er (referencer til) objekter

En tabel kan indeholde alle slags data, også (referencer til) objekter.

**Eksempel:** tabeller af tidspunkter.

Erklæring af tabelvariabel med referencer til Time-objekter:

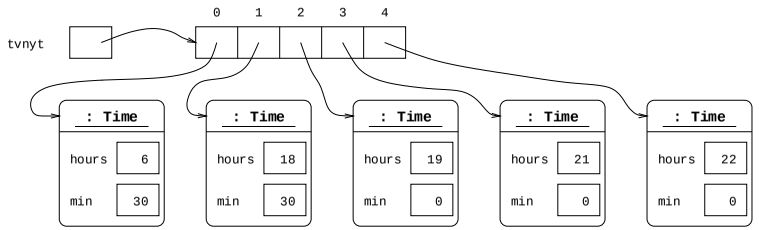
```
Time[] tvnyt;
```



## Tabeller, hvor elementerne er (referencer til) objekter

Erklæring og initialisering med henvisninger til Time-objekter (der oprettes samtidig):

```
Time[] tvnyt = { new Time(6, 30), new Time(18, 30),
                 new Time(19, 0), new Time(21, 0),
                 new Time(22, 0) };
```



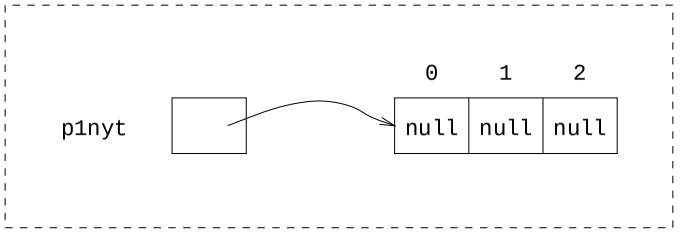



## Tabeller, hvor elementerne er (referencer til) objekter

Erklæring og oprettelse af tabel med plads til 3 henvisninger til Time-objekter:

```
Time[] p1nyt = new Time[3];
```

Herved oprettes ingen Time-objekter. Alle pladserne i tabellen `p1nyt` er initialiseret til **null**.



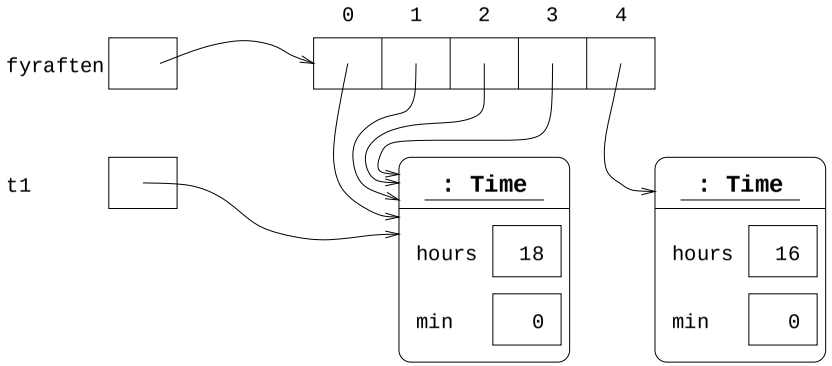

Time-objekterne skal oprettes udtrykkeligt, f.eks.:

```
p1nyt[0] = new Time(7, 0);
p1nyt[1] = new Time(8, 0);
p1nyt[2] = new Time(9, 0);
```



## Flere elementer i en tabel kan henvise til det samme objekt

```
Time t1 = new Time(18, 0);
Time[] fyraften = { t1, t1, t1, t1, new Time(16, 0) };
```



## Flerdimensionelle tabeller

Elementerne i en tabel kan igen være tabeller. Så har vi en flerdimensional tabel.

**Eksempel:** en biografsal kan modelleres som en tabel af stolerækker; en stolerække som en tabel af stole. For hver stol registreres det om den er optaget (**true**) eller (**false**).

Ergo: en biografsal er en to-dimensional tabel af **boolean**s.

Erklæring af en to-dimensionel tabel af **boolean**s:

```
boolean[][] bio;
```

Oprettelse af en "$5 \times 6$" biograf (alle elementerne er initialiseret til **false**):

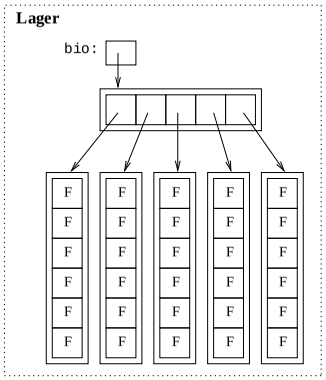
```
bio = new boolean[5][6];
```

Antallet af stolerækker: `bio.length`

Er række 3 sæde 1 ledigt? `bio[3][1]`

Reservation af række 2 sæde 5: `bio[2][5] = true;`



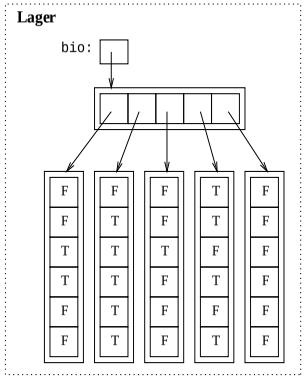





## Flerdimensionelle tabeller

Erklæring, oprettelse og initialisering på en gang:

```
boolean[][] bio =
{{false,false,true, true, false,false},
 {false,true, true, true, true, true },
 {false,false,true, false,false,false},
 {true, true, false,true, false,true },
 {false,false,false,false,false,false}};
```






## Flere fler-dimensionale tabeller

Rækkerne i en to-dimensional tabel *behøver ikke* have lige mange elementer:

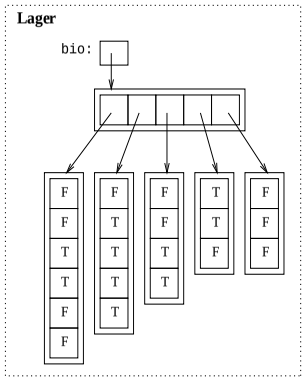
```
boolean[][] bio =
  {{false,false,true,true,false,false},
   {false,true, true,true,true},
   {false,false,true,true},
   {true, true, false},
   {false,false,false}};
```

Række 0 har 6 pladser.

Række 1 har 5 pladser.

Række 2 har 4 pladser.

Række 3 og 4 har 3 pladser.



## Kommandolinje og fler-dimensionale arrays 1/2

**Opgave:** lav et program som indlæser en række tal.

- Hvert tal angiver hvor mange sæder, der er på den givne række i en biograf.
- Antallet af tal angiver antallet af rækker i biografen.
- Slut af med at skrive et layout over biografen ud.

**Eksempel på kørsel:**

```
$ java CinemaDecl 6 5 4 3 3
O O O O O O
O O O O O
O O O O
O O O
O O O
```



## Kommandolinje og fler-dimensionale arrays 2/2

```
public static void main (String[] args) {
  boolean[][] bio = new boolean[args.length][];

  for (int i=0; i<args.length; i=i+1)
  {
    int seatsOnRow = Integer.parseInt(args[i]);
    bio[i] = new boolean[seatsOnRow];
  }

  for (int row=0; row<bio.length; row=row+1)
  {
    for (int seat=0; seat<bio[row].length; seat=seat+1)
      System.out.print("O␣");
    System.out.println();
  }
}
```



## Tabeller med variabel længde

I CD eksemplet i afsnit 6.2 i bogen er vist, hvordan man kan øge længden af en tabel dynamisk. (Se `increaseSize` metoden.)



## Anvendelse af tabeller til sortering

En kendt disciplin indenfor datalogi er *sortering* af tallene i en tabel, så mindre elementer kommer før større.

**Eksempel:** Sortering af tabellen

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 3 | 9 | 6 | 1 | 2 |

giver

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 |

**Formel definition:** en sorterings algoritme er en algoritme, der laver en permutation af elementerne i en tabel `t`, så `t[i] <= t[i+1]` for `0 <= i <= t.length-2`.

Der findes mange algoritmer til sortering. I bogen vises to af dem:
*selection sort* og *insertion sort*.

## Idéen i selection sort

Start fra en ende af. Find det mindste element af de elementer man endnu ikke har placeret rigtigt. Sæt det på den rigtige plads. Gentages indtil alle elementerne er blevet sat på plads.

Start ved 3. 1 er mindst. Byt 1 og 3.

| | | | | |
|---|---|---|---|---|
| 3 | 9 | 6 | 1 | 2 |

Start ved 9. 2 er mindst. Byt 2 og 9.

| | | | | |
|---|---|---|---|---|
| 1 | 9 | 6 | 3 | 2 |

Start ved 6. 3 er mindst. Byt 3 og 6.

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 6 | 3 | 9 |

Start ved 6. 6 er mindst. Byt 6 og 6.

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 |

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 6 | 9 |



## Selectionsort algoritmen

```
public static void selectionSort(int[] numbers) {
   int min, temp;

   for(int i=0; i<numbers.length-1 ; i=i+1)
   {  min = i;
      for(int scan=i+1; scan<numbers.length; scan=scan+1)
         if (numbers[scan] <numbers[min])
            min=scan;
      // Swap the values
      temp=numbers[min];
      numbers[min]=numbers[i];
      numbers[i] = temp;
   }
}
```