

# Iteratorer, arrays, for-løkker

Grundlæggende Programmering  
med Projekt

# Sidste gang

- Klasse repræsentanter??

# Denne gang

- Arrays
- For-løkker
- Iteratorer
- Null
- Avancerede løkker

# Arrays vs arraylist

- **Arrayliste:**
  - Et *variabelt* antal elementer af samme type
  - Eksempel-type: **ArrayList<String>**
  - Oprettelse: **new ArrayList<String>()**
- **Array:**
  - Et *fast* antal elementer af samme type
  - Eksempel-type: **String[]**
  - Oprettelse: **new String[12]**

0	1	2	3	4	5	6	7	8	9	10	11

# Erklæring af array-variable

- Variabel hvis type er array af heltal:  
`int[] hourCounts;`
- Variabel, array af flydende-kommatal:  
`double[] coefficients;`
- Variabel, array af tegnstreng:  
`String[] names;`
- Variabel, array af Forest-objekter:  
`Forest[] forests;`

# Oprettelse af array-objekter

- Opret array med plads til 24 heltal:  
`hourCounts = new int[24];`  
Hver plads er 0 fra start.
- Opret array med plads til n kommatal:  
`coefficients = new double[n];`  
Hver plads er 0.0 fra start.
- Opret array med plads til 42 strenge:  
`name = new String[42];`  
Hver plads er `null` fra start.

# Initialisering af array-objekter

- Opret array med de tre heltal -30, 9, 3:  
`int[] cs = new int[] { -30, 9, 3 };`
- Opret array med månedsnavnene:  
`String[] months = new String[] {  
 "Jan", "Feb", "Mar", ..., "Dec" };`

# Længde og indeksering

- Lad `cs` være et array
- Så er `cs.length` antallet af elementer
- Og `cs[i]` er element nummer `i`, regnet fra 0
- Der skal gælde  $0 \leq i < cs.length$



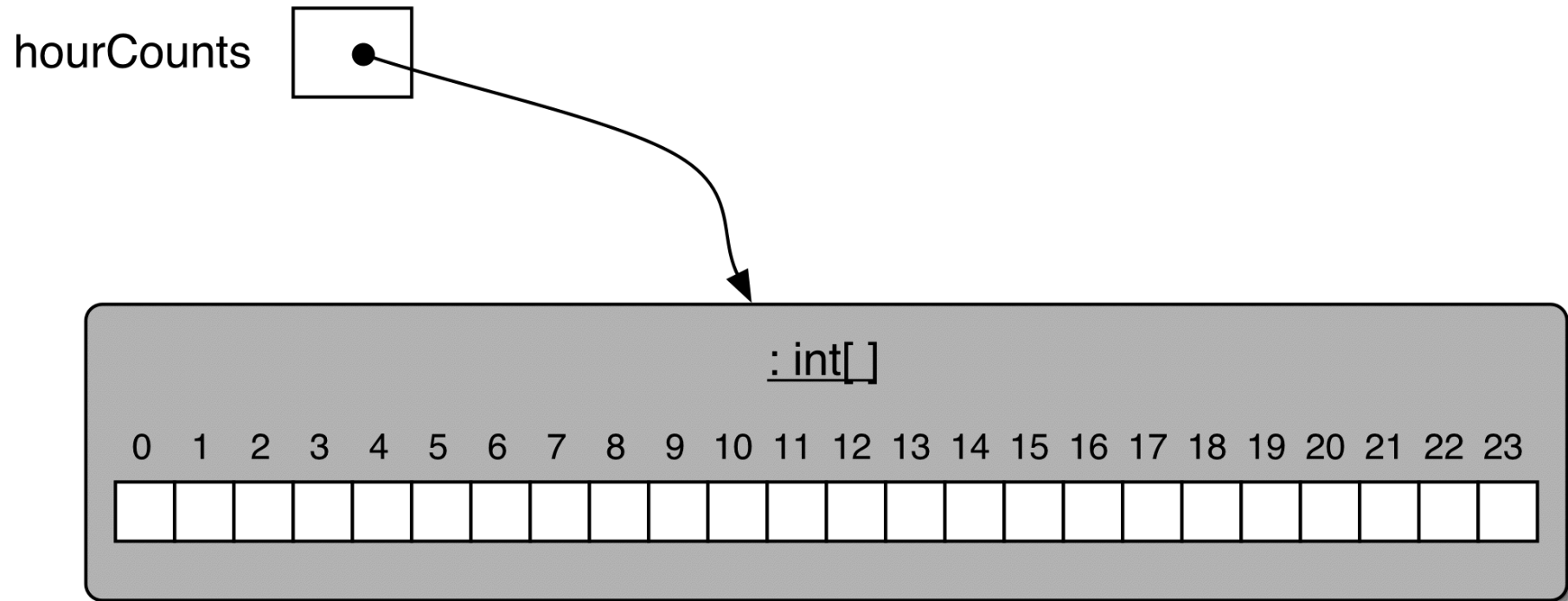
# Eksempel: Logfil-analysator (histogram)

- Givet: Tidspunkter for opslag (access) til en webside
- Ønske: Find antal opslag for hver time i døgnet
- Idé: Brug et array med 24 pladser nummereret 0 til 23
  - Hver plads er 0 fra start
  - Hver gang vi ser klokkeslet h, øges plads nummer h med 1

2005	5	01	00	10
2005	5	01	00	10
2005	5	01	00	19
2005	5	01	01	27
2005	5	01	02	17
...				
2005	5	31	23	33
2005	5	31	23	47
2005	5	31	23	55
2005	5	31	23	55

31. maj 2005  
kl 23:55

# Arrayet hourcounts



# Hvilken skal vi bruge til at gennemløbe filen?

- Med foreach-løkke:

```
for (String file: files) {  
    System.out.println(file);  
}
```

- Med while-løkke:

```
int index = 0;  
while (index < files.size()) {  
    System.out.println(files.get(index));  
    index++;  
}
```

# Optælling i metode `analyzeHourlyData`

- Læs timetallet `hours`
- Brug det som indeks i array `hourCounts`
- Forøg værdien på den plads med 1
- (Brug debugger til at forstå løkken)

```
public void analyzeHourlyData() {  
    while (reader.hasMoreEntries()) {  
        LogEntry entry = reader.nextEntry();  
        int hour = entry.getHour();  
        hourCounts[hour]++;  
    }  
}
```

Samme som:

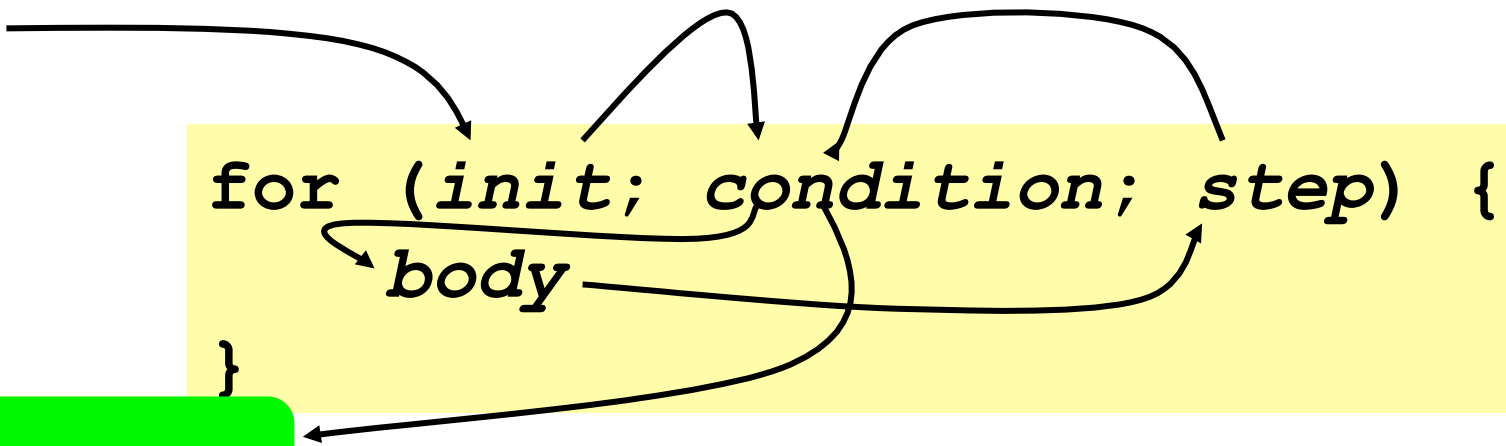
```
hourCounts[hour] = hourCounts[hour]+1;
```

# Generelle for-løkker

- Når opslag per time er talt sammen, skal de skrives ud
- Det kan gøres med en generel for-løkke

```
public void printHourlyCounts() {  
    System.out.println("Hr: Count");  
    for (int hour = 0; hour < hourCounts.length; hour++) {  
        System.out.println(hour + ": " + hourCounts[hour]);  
    }  
}
```

# Anatomi for generel for-løkke



```
for (init; condition; step) {  
    body  
}
```

The diagram illustrates the execution flow of a general for-loop. Arrows show the sequence: from the opening curly brace to the *body*, then to the *step* statement, then to the *condition*, and finally back to the *body*. An additional arrow points from the *init* statement to the *body*. A green box with the text 'Alles klar!' is positioned to the left of the loop structure.

Alles klar!

Udførelse:

1. Udfør erklæringen *init*
2. Udregn betingelsen *condition*; hvis false, stop løkken, ellers fortsæt med 3.
3. Udfør løkke kroppen *body*
4. Udfør sætningen *step*
5. Fortsæt med punkt 2

# Foreach kan også bruges på arrays

```
for (int hourCount : hourCounts) {  
    System.out.println(hourCount);  
}
```

- Men så skal vi selv tælle timetallene:

```
public void printHourlyCounts() {  
    System.out.println("Hr: Count");  
    int hour = 0;  
    for (int hourCount : hourCounts) {  
        System.out.println(hour + ": " + hourCount);  
        hour++;  
    }  
}
```

- Ikke kønnere end en generel for-løkke...

**Hvad kan man gøre med en for-løkke  
som man ikke kan med en while  
og omvendt?**



# For-løkker og while-løkker er ækvivalente, lige stærke

```
for (init; condition; step) {  
    body  
}
```

for

```
init  
while (condition) {  
    body  
    step  
}
```

while

# Flere for-løkker

- Udskriv tallene 1, 2, ..., 20:

```
for (int i=1; i<=20; i++) {  
    System.out.println(i);  
}
```

- Udskriv tallene 20, 19, ..., 2, 1:

```
for (int i=20; i>=1; i--) {  
    System.out.println(i);  
}
```

- Hvordan udskrives disse sekvenser?
  - Tallene 0, 2, 4, ..., 18, 20
  - Tallene 20, 18, ..., 4, 2, 0
  - Tallene 1, 4, 9, ..., 25, 36

# Løkker til at initialisere arrays

- Initialiser hver plads i xs til 42:

```
for (int i=0; i<xs.length; i++) {  
    xs[i] = 42;  
}
```

- Initialiser xs til 1, 2, 3, 4, ...

```
for (int i=0; i<xs.length; i++) {  
    xs[i] = i+1;  
}
```

- Initialiser xs til 1, 2, 3, 1, 2, 3, 1, ...

```
for (int i=0; i<xs.length; i++) {  
    xs[i] = (i % 3) + 1;  
}
```

# Arrays og for-løkker

- Antag `int[] xs` indeholder nogle tal
- Udskriv forlæns:

```
for (int i=0; i<xs.length; i++) {  
    System.out.println(xs[i]);  
}
```

- Udskriv baglæns:

```
for (int i=xs.length-1; i>=0; i--) {  
    System.out.println(xs[i]);  
}
```

# Spørgsmål

- Hvordan udskrives hvert andet element?
- Hvordan kan man få de fem første elementer udskrevet?

```
for (int i=0; i<xs.length; i++) {  
    System.out.println(xs[i]);  
}
```

# Repetition collection-gennemløb

- Med foreach-løkke:

```
for (String file: files) {  
    System.out.println(file);  
}
```

- Med while-løkke:

```
int index = 0;  
while (index < files.size()) {  
    System.out.println(files.get(index));  
    index++;  
}
```

## Tredje mulighed: iterator

Klasse

Metode

- Collection-gennemløb med iterator:

```
Iterator<String> iter = notes.iterator();  
while (iter.hasNext()) {  
    String note = iter.next();  
    System.out.println(note);  
}
```

Iterator-  
objekt

- En iterator peger ind i en collection
  - `iter.hasNext()` siger om der er flere elementer
  - `iter.next()` flytter til næste element og returnerer elementet

# Foreach er syntaktisk sukker for en iterator/while-løkke

- *Syntaktisk sukker* = pæn indpakning af noget der ellers er besværligt

```
for (String note : notes) {  
    System.out.println(note);  
}
```

Før foreach var man nødt til at skrive alt det her

```
Iterator<String> iter = notes.iterator();  
while (iter.hasNext()) {  
    String note = iter.next();  
    System.out.println(note);  
}
```



# Index versus iteratorer

- Fordele ved iteratorer
  - Kan stoppe tidligt (pga. while-løkke)
  - Virker ensartet for alle collections, ikke kun arraylister
  - (Man kan selv definere iteratorer på egne klasser, men det er ... Hmmm ...pokkers besværligt)
- Fordele ved indeksering
  - Fleksibelt; kan fx gennemløbe baglæns, hver andet element, ...

**OG NU TIL NOGET LIDT  
ANDET....**

# Referencetyper, primitive typer, og wrapping (boxing)

- Klasser er *objekttyper* (referencetyper):  
Book, Tree, Forest, String, ...
- Collections kan kun indeholde objekttyper, fx **`ArrayList<Tree>`**
- Hvad så med de *primitive typer*: **`int`**, **`double`**, **`boolean`**, ... ?
- En værdi af primitiv type kan pakkes ind ("wrappes") som objekt, fx **`ArrayList<Integer>`** hvor **`Integer`** er wrapper-klasse for typen **`int`**

# Wrapper-klasser for primitive typer

Type	Wrapper-klasse	Beskrivelse
<code>int</code>	Integer	32 bit heltal
<code>long</code>	Long	64 bit heltal
<code>short</code>	Short	16 bit heltal
<code>byte</code>	Byte	8 bit heltal
<code>char</code>	Character	tegn 'a', 'A', ...
<code>double</code>	Double	64 bit kommatal
<code>float</code>	Float	32 bit kommatal
<code>boolean</code>	Boolean	<code>false</code> eller <code>true</code>

# (Un)wrapping sker automatisk

- En `int` wrappes automatisk til en `Integer` når den kommer i en collection
- En `Integer` unwrappes automatisk til en `int` når den indgår i en beregning

```
ArrayList<Integer> udfald = new ArrayList<Integer>();  
udfald.add(4);  
udfald.add(1);  
udfald.add(6);  
udfald.add(2);  
int sum = 0;  
for (int x : udfald) {  
    sum += x;  
}
```

Wrapping af 4

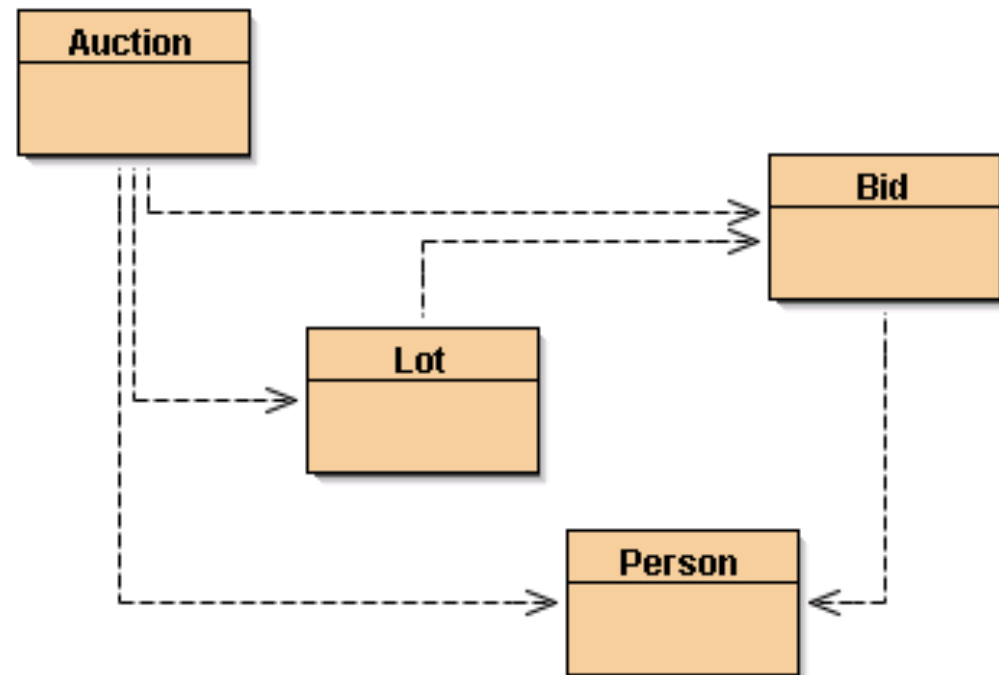
Unwrapping af x

# Arrays behøver ingen wrapping

- Kun collections behøver wrapping af `int`
- Med arrays kan man direkte have `int[], double[], boolean[], ...`
- Det tager meget længere tid at lave et wrapper-objekt (`Integer`) end en `int`-værdi
- Når elementerne er primitive types, så brug arrays i stedet for collections (hvis hastighed er vigtigt)

# Eksempel: Auktion

- Klasse Bid har en **Person** og en værdi (af type `long`, langt heltal)
- Klasse Lot har et højeste **Bid**
- Klasse Auction har en **ArrayList<Lot>**



- Projekt *auction*

# Kald fra metode til metode

- Metoden `bidFor` i Auction klassen
  - kalder metode `getLot` for at finde "lottet"
  - kalder metode `bidFor` i klasse *Lot*, på det valgte lot, for at se om buddet er bedre end de tidligere

```
public class Auction {  
    public void bidFor(int lotNumber, Person bidder, ...) {  
        Lot selectedLot = getLot(lotNumber);  
        if (selectedLot != null) {  
            boolean successful  
                = selectedLot.bidFor(new Bid(bidder, value));  
            if (successful) {  
                System.out.println("The bid ... successful.");  
            } else { ... }  
        }  
    }  
}
```





# Værdien null

- Feltet `highestBid` i `Lot` indeholder det hidtil bedste bud
- Q: Men hvilken værdi inden første bud?
- A: Felter af klassetype er `null` fra start
- Værdien `null` siger: "peger ikke på et objekt"
- Sådan ses om `bid` er det bedste hidtil:

```
public class Lot {  
    public boolean bidFor(Bid bid) {  
        if ((highestBid == null) ||  
            (bid.getValue() > highestBid.getValue())) {  
            // This bid is the best so far.  
            highestBid = bid;  
            return true;  
        } else { ... }  
    }  
}
```

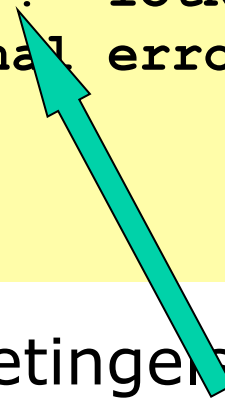
Første bud?

Eller bedste?

# Livrem og seler

- Metoden getLot i Auction tjekker om et lot har det nummer det burde have

```
public Lot getLot(int lotNumber) {  
    if ((lotNumber >= 1) && (lotNumber < nextLotNumber)) {  
        Lot selectedLot = lots.get(lotNumber - 1);  
        if (selectedLot.getNumber() != lotNumber) {  
            System.out.println("Internal error: ... ");  
        }  
        return selectedLot;  
    } else { ... } }  
}
```



- Hvis programmet virker er betingelsen falsk
- Men det er godt at opdage når det ikke virker
- Senere bruger vi **assert**-sætninger

# Fælles opgaver

- Opgave 1: Tilføj en **close** metode til **Auction** klassen. Metoden skal iterere over **lot samlingen** (*collection*) og *udskrive detaljerne om alle "lots"*. I kan enten bruge en for-each eller en while-løkke. Enhver genstand (lot) som har mindst et bud betragtes som værende solgt
- Opgave 2: Tilføj en metode **getUnsold** til **Auction** klassen med følgende signatur

***Public ArrayList<lot> getUnsold()***

Denne metode skal iterere over lots feltet og gemme de genstande som ikke er blevet solgt i en lokal variable af type ***ArrayList***.

Metoden skal returnere listen af ikke-solgte genstande