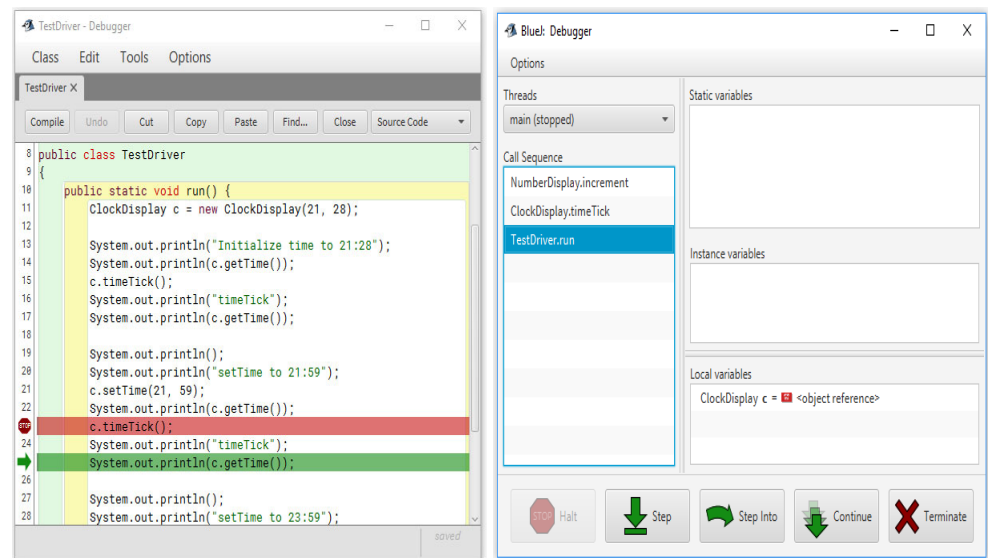


# Forelæsning Uge 2 – Torsdag

- **Niveauer af programbeskrivelser**
  - Statiske / dynamiske beskrivelser
- **Klassevariabler og klassemetoder**
  - Variabler og metoder der er tilknyttet klassen (i stedet for at være tilknyttet objekter)
- **Problemløsning / Analyse**
- **BlueJ's Code Pad**
  - Nyttig til små eksperimenter
- **BlueJ's Debugger**
  - Nyttig til at finde fejl i kode



# ● Niveauer af programbeskrivelser

---

- **Klassediagram (oversigt)**

- Hvad (specifikation)

- **JavaDoc (mellem-niveau)**

- Hvad (dokumentation)

- **Java-kode (detaljeret)**

- Hvordan (implementation)

**Statisk (struktur)**

- rum for hvad der generelt kan ske

---

- **Objektdiagram (oversigt)**

- Relationer mellem objekter (referencer)

- **Sekvensdiagram (detaljeret)**

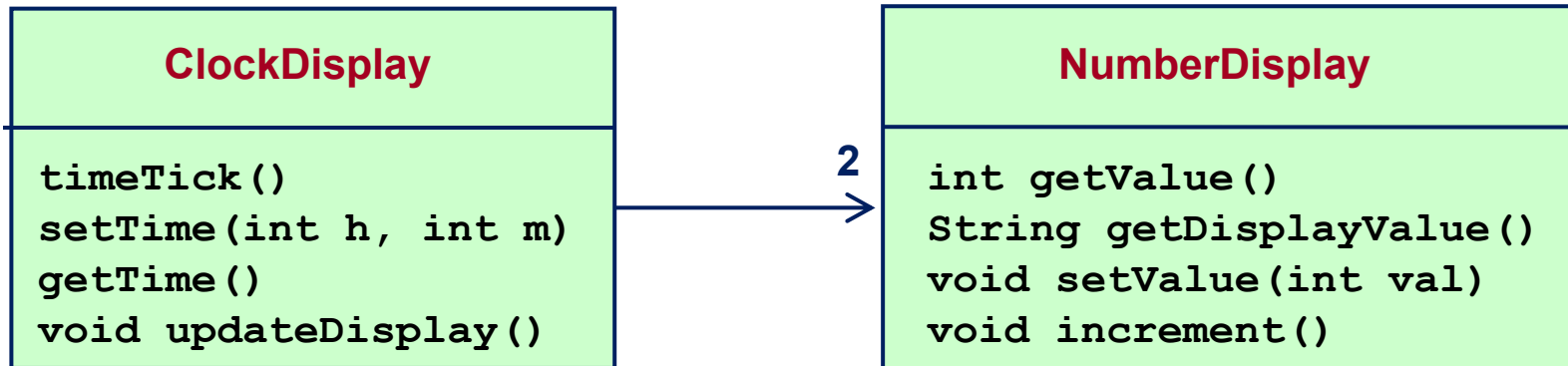
- Interaktion mellem objekter (metodekald)

**Dynamisk (udførelse)**

- scenarie for hvad der sker i en konkret situation

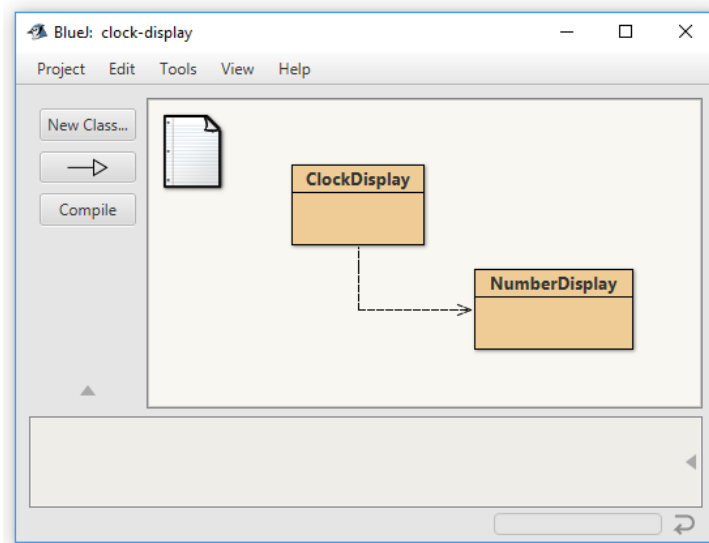
# Klassediagram (statisk, oversigt)

UML



UML klassediagrammer viser  
sometider også nogle af feltvariablerne

BlueJ



Her kan man kun se klassernes navne og  
pilene imellem dem, men ved at

- skabe et objekt og højre-klikke på det, kan man se, hvilke public metoder en klasse har
- åbne objektets inspector, kan man se feltvariablerne

# Java-kode (statisk, detaljeret)

The screenshot shows a Java IDE window titled "ClockDisplay - clock-display". The menu bar includes "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", and "Close". A "Source Code" dropdown menu is also present. The main editor area displays the following Java code:

```
15 public class ClockDisplay
16 {
17     private NumberDisplay hours;
18     private NumberDisplay minutes;
19     private String displayString;    // simulates the actual display
20
21     /**
22      * Constructor for ClockDisplay objects. This constructor
23      * creates a new clock set at 00:00.
24      */
25     public ClockDisplay()
26     {
27         hours = new NumberDisplay(24);
28         minutes = new NumberDisplay(60);
29         updateDisplay();
30     }
31
32     /**
33      * Constructor for ClockDisplay objects. This constructor
34      * creates a new clock set at the time specified by the
35      * parameters.
36      */
37     public ClockDisplay(int hour, int minute)
38     {
39         hours = new NumberDisplay(24);
40         minutes = new NumberDisplay(60);
41         setTime(hour, minute);
42     }
43 }
```

Annotations and callouts:

- A blue arrow points to the "Source Code" dropdown menu with the text: "Skift mellem Java kode og dokumentation".
- Red boxes highlight the Javadoc comments for the two constructors, with a red arrow pointing to them from the text: "Disse kommentarer indsættes i klassens dokumentation".
- A blue box points to the two constructor methods with the text: "Vi har to forskellige konstruktører med forskellige parametre (overloadning)".

The status bar at the bottom right shows "saved".

# JavaDOC (statisk, mellem-niveau)

The screenshot shows a Java IDE window titled "ClockDisplay - clock-display". The "Tools" menu is open, and the "ClockDisplay X" tab is selected. The IDE displays the JavaDoc for the `ClockDisplay` class, which is organized into sections: "Constructor Summary" and "Constructor Detail".

**Constructor Summary:** This section lists the constructors for the `ClockDisplay` class. It includes two constructors: `ClockDisplay()` and `ClockDisplay(int hour, int minute)`. The first constructor is described as "Constructor for ClockDisplay objects." and the second as "Constructor for ClockDisplay objects.".

**Constructor Detail:** This section provides the full JavaDoc for each constructor. It includes the source code for each constructor and its corresponding JavaDoc comment. The first constructor is shown as `public ClockDisplay()` with the comment "Constructor for ClockDisplay objects. This constructor creates a new clock set at 00:00.".

**Annotations:** Several annotations are present on the slide, highlighting specific parts of the JavaDoc:

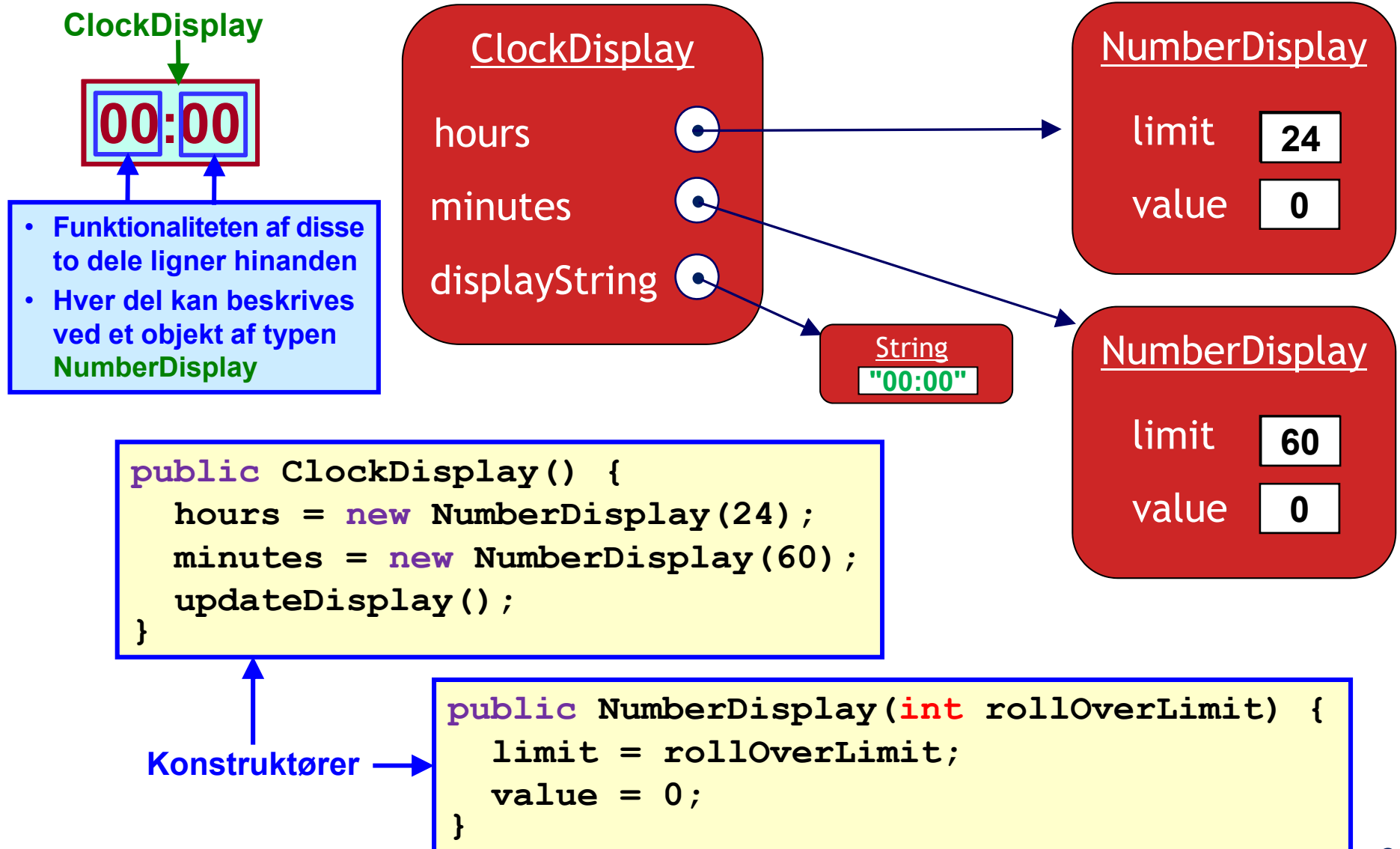
- Første sætning i kommentaren** (First sentence in the comment): This annotation points to the first sentence of the JavaDoc comment for the `ClockDisplay()` constructor, "Constructor for ClockDisplay objects.".
- Tilsvarende gælder for** (Corresponds to): This annotation points to the first sentence of the JavaDoc comment for the `ClockDisplay(int hour, int minute)` constructor, "Constructor for ClockDisplay objects.".
- Dokumentationen genereres automatisk (ud fra de kommentarer, der er i Java koden)** (Documentation is generated automatically (from the comments that are in the Java code)): This annotation points to the JavaDoc comment for the `ClockDisplay()` constructor.
- Hele kommentaren** (The whole comment): This annotation points to the entire JavaDoc comment for the `ClockDisplay(int hour, int minute)` constructor.

**JavaDoc Comments:** The JavaDoc comments for the constructors are as follows:

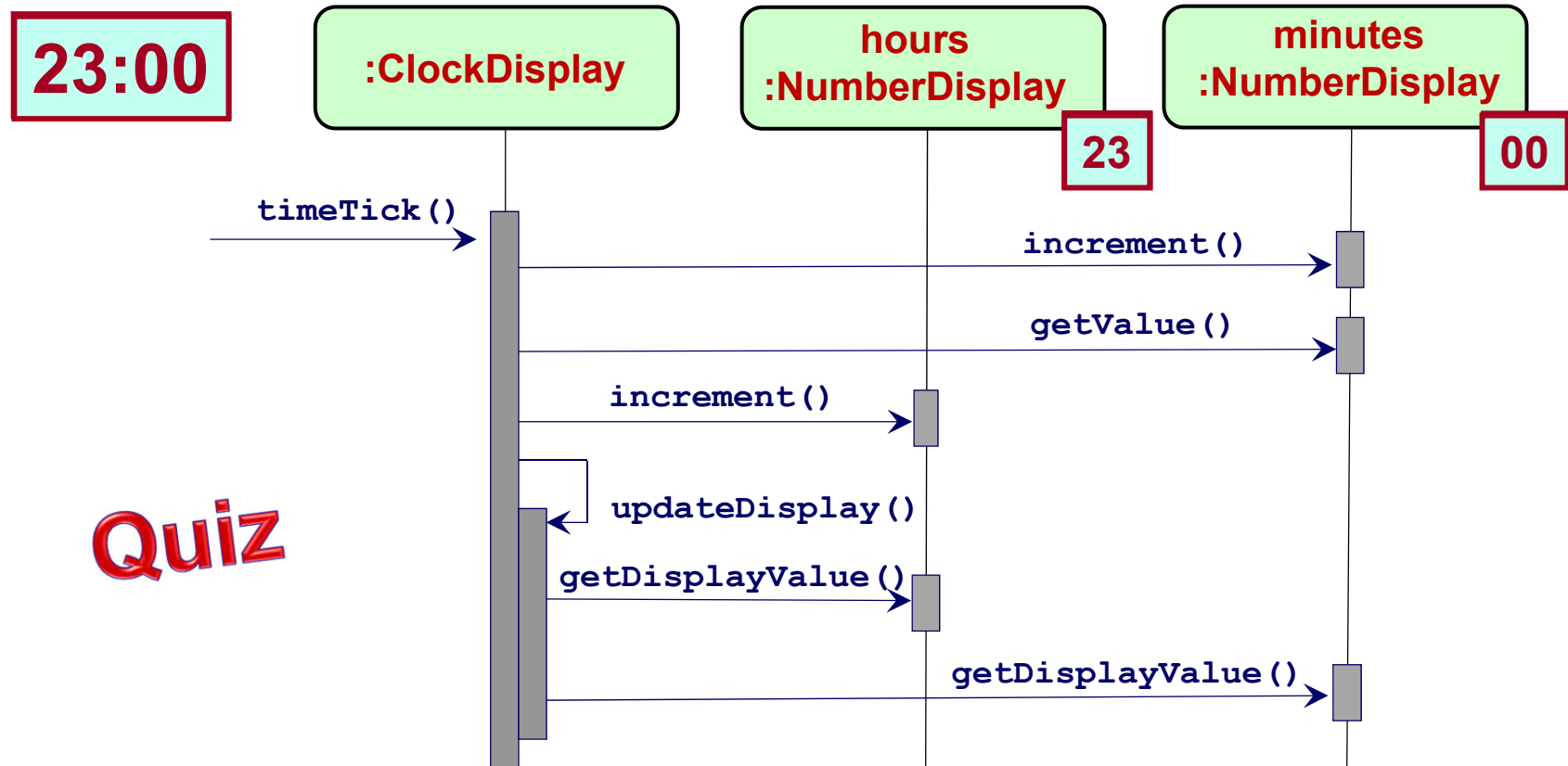
```
/**
 * Constructor for ClockDisplay objects. This constructor
 * creates a new clock set at 00:00.
 */

/**
 * Constructor for ClockDisplay objects. This constructor
 * creates a new clock set at the time specified by the
 * parameters.
 */
```

# Objektdiagram (dynamisk, oversigt)



# Sekvensdiagram for timeTick (dynamisk, detaljeret)



**Quiz**

```
public void timeTick() {
    minutes.increment();
    if(minutes.getValue() == 0) {
        hours.increment();
    }
    updateDisplay();
}
```

```
private void updateDisplay() {
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

# ● Klassevariabler og klassemetoder

---

- **Indtil nu har alle feltvariabler og metoder være tilknyttet objekter**
  - Hvert objekt har sine egne værdier for feltvariabler (instansvariabler)
  - Metoder kaldes ved at bede objekter om at udføre dem (instansmetoder)
- **Det er imidlertid også muligt at erklære variabler og metoder som i stedet er tilknyttet klassen**
  - **Klassevariabler** bruges til at modellere egenskaber for klassen, f.eks. ting der er fælles for alle objekter i klassen (så som myndighedsalder for alle personer og fælles rentesats for alle konti)
  - **Klassemetoder** bruges til at modellere operationer, der er uafhængige af objekters tilstande (så som ændring af myndighedsalderen og ændring af den fælles rentesats)
  - Klassevariabler og klassemetoder erklæres med det reservede ord **static** (dårligt ordvalg – levn fra gamle programmeringssprog)

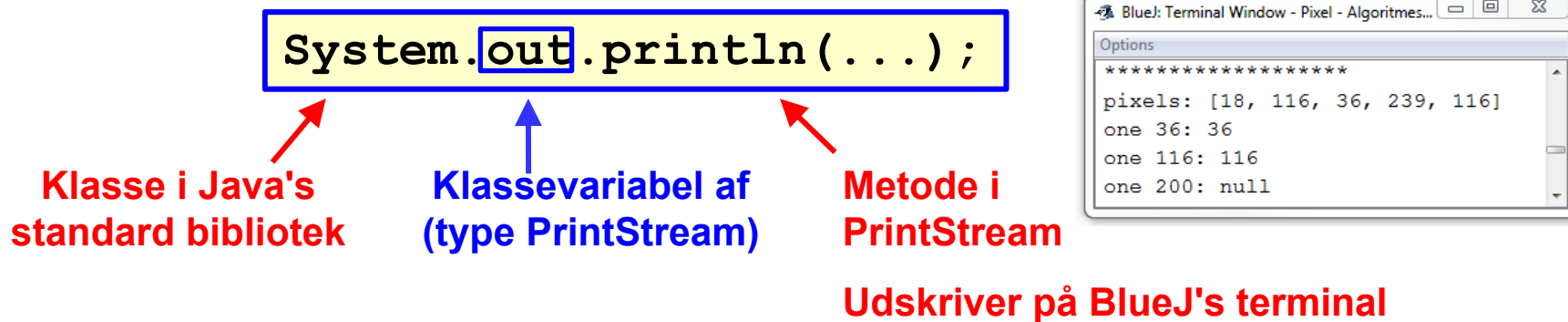


# Har I set dem før?

- I har allerede mødt klassemetoder?



- I har også mødt klassevariabler?



# Eksempler fra java.lang.Math

Kan bruges uden for klassen

Konstant (kan ikke ændres)

```
public class Math {  
    public static final double PI = 3.141592653589793  
    ...  
  
    // 0.0 ≤ random() < 1.0  
    public static double random() {...}  
  
    // sqrt(a) == √a  
    public static double sqrt(double a) {...}  
  
    // pow(a,b) == ab  
    public static double pow(double a, double b) {...}  
  
    ...  
}
```

navne på konstanter skrives  
med store bogstaver og  
"underscores", fx MAX\_NO

- Math.PI
- Math.random()
- Math.sqrt(4.5)
- Math.pow(3.34, 2)

# Flere eksempler

```
public class Account {  
    private static double interestRate; // 1.035 ≈ 3.5%  
    private int balance;  
    private Person owner;  
    ...  
    public static void setInterestRate (double rate) {  
        interestRate = rate; // Opdaterer klassevariablen  
    }  
    public void addInterest() {  
        balance = (int) (balance * interestRate);  
    }  
    ...  
}
```

Type cast (ændrer typen fra double til int)  
Uden et type cast ville vi få en oversættelsesfejl

Almindelige metoder har adgang til **alle** variabler og **alle** metoder uanset om de er klassevariabler/klassemetoder eller ej

Klassemetoder har **kun** adgang til klassevariabler og klassemetoder

En klassemetode, kan dog godt oprette et eller flere nye objekter (fra egen eller andre klasser) og derefter tilgå feltvariabler og instansmetoder i disse objekter på normal vis

# Brug af klassevariabler og klassemetoder

---

- **Klassevariable og klassemetoder tilgås via klassen**

```
Math.PI;  
Math.random();  
Account.setInterestRate(1.035);
```

- **De kan også tilgås via et objekt, men det er dårlig programmeringsstil og kan være forvirrende**

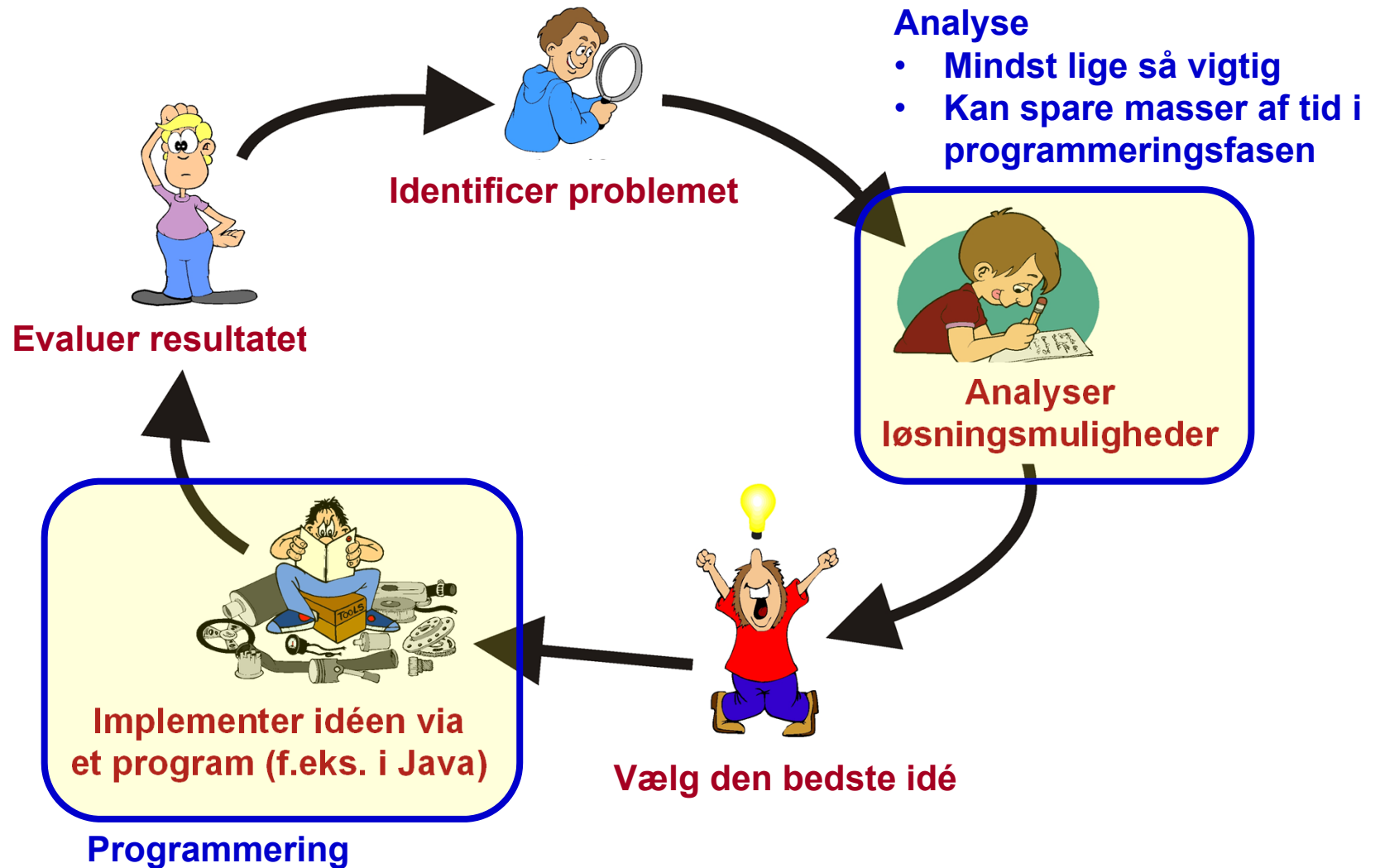
```
Account myAccount = new Account(...);  
myAccount.setInterestRate(1.035);
```



Metoden kaldes på en specifik bankkonto, men det er rentesatsen for alle konti, der ændres

**Pause**

# ● Problemløsning



# Ex: Cup turnering (fx tennis eller fodbold)

- **Spillerne/holdene mødes to og to**

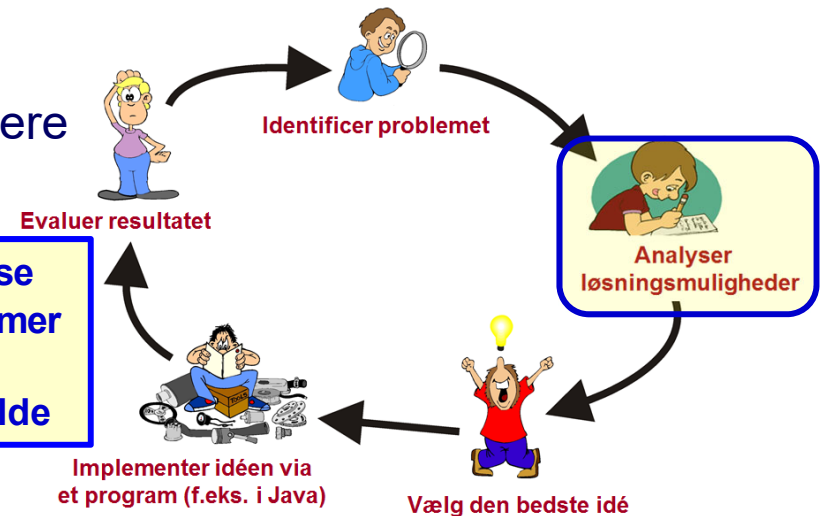
- Vinderen fortsætter til næste runde, mens taberen er slået ud af turneringen og ikke får flere kampe
- Vi vil gerne lave en algoritme, som beregner, hvor mange kampe, der skal til, hvis der er X spillere/hold i turneringen

- **Løsning for  $X = 29$**

- 13 sekstendedels finaler + 3 oversiddere
- 8 ottendedels finaler
- 4 kvartfinaler
- 2 semifinaler
- 1 finale
- I alt 28 kampe

**Husk at bruge tid på analyse**

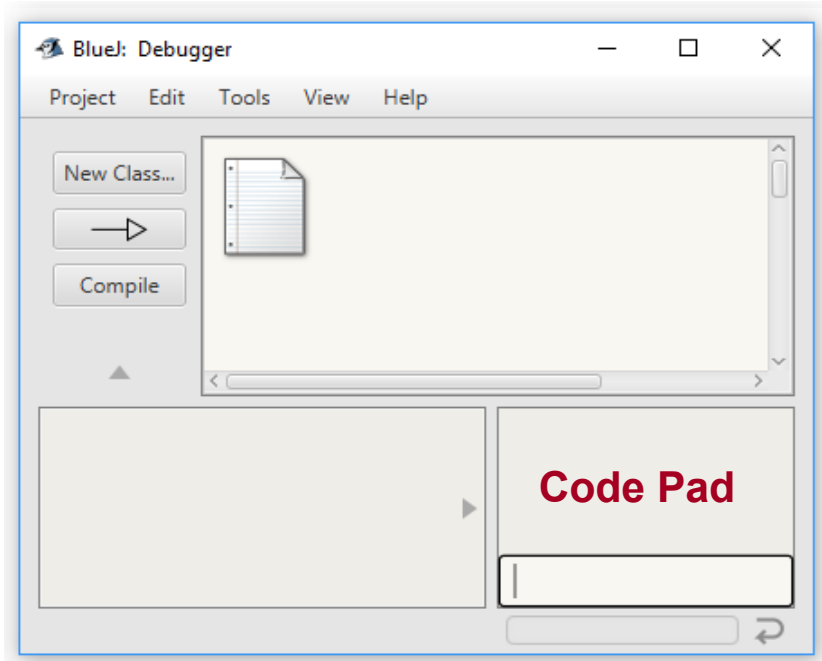
- Giver simplere programmer
- Hurtigere at skrive og nemmere at vedligeholde



- **Er der en lettere måde at løse opgaven på?**

- Der bliver slået ét hold ud i hver kamp
- Vi er færdige, når der kun er et hold tilbage (vinderen)
- Så vi skal bruge  $X-1$  kampe (hvor  $X$  er antallet af hold)

# ● Brug af BlueJ's Code Pad



- I Code Pad'en kan man indtaste erklæringer, sætninger og udtryk
  - Udtryk evalueres
  - Erklæringer og sætninger udføres
- Brug Code Pad'en til hurtige eksperimenter
  - Mere komplekse eksperimenter laves bedst via en testklasse

4 + 3 \* 5

19 (int)

int i = 7;

boolean female = false;

if(female = true) { i++; }

Hvad er værdien af female og i efter udførelsen af if sætningen?

female

true (boolean)

i

8 (int)

Quiz

Hvis man skriver forkert og får en syntaksfejl, kan man trykke på ↑, hvilket kopierer det sidste man skrev

# ● BlueJ's debugger (afluser = fejlfinder)

Nyttig når man skal tjekke den detaljerede opførsel af kørende Java kode

Bruges i nogle af opgaverne efter efterårsferien

**Breakpoints** indsættes (og fjernes) ved at klikke i venstre margin af editoren

Under programudførelsen vil debuggeren stoppe, når et breakpoint nås, og vise positionen med en grøn pil (samt grøn farve)



Herefter kan man "steppe" gennem koden sætning for sætning



Mellem skridtene kan man inspicere systemets tilstand, dvs. værdierne af de forskellige slags variabler

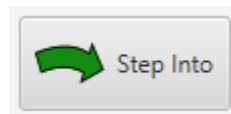


# Metodekald

Når næste sætning er et metodekald, har man to muligheder:



Udfører hele metodekaldet uden at man ser detaljerne



Starter metodekaldet, men stopper inden første sætning i kroppen af den kaldte metode

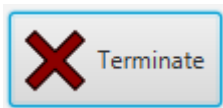
# Metodekald

Parat til at udføre første sætning i den kaldte metode

Andre knapper:



Fortsætter kørslen frem til næste breakpoint



Stopper kørslen



Nødstop (uendelig while løkke eller lignende)

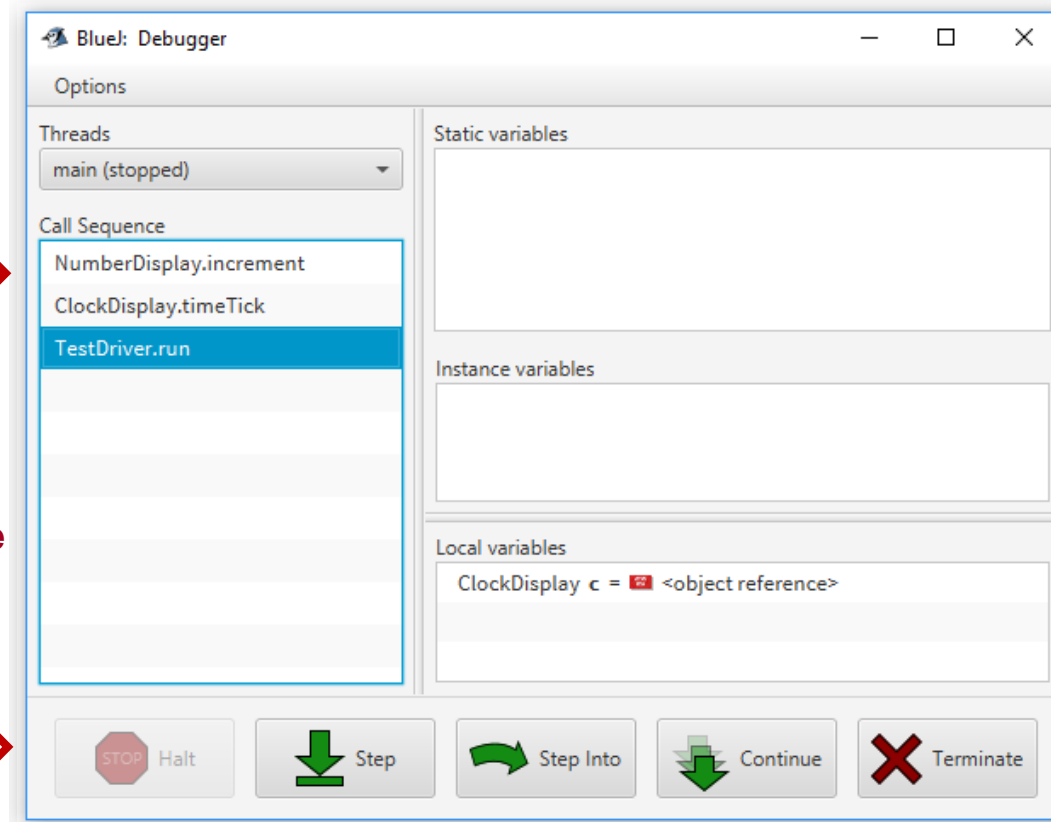
```
NumberDisplay - Debugger
Class Edit Tools Options
ClockDisplay X TestDriver X NumberDisplay X
Compile Undo Cut Copy Paste Find... Close Source Code
48 /**
49  * Set the value of the display to the new specified value. If
50  * value is less than zero or over the limit, do nothing.
51  */
52 public void setValue(int replacementValue)
53 {
54     if((replacementValue >= 0) && (replacementValue < limit))
55         value = replacementValue;
56 }
57
58 /**
59  * Increment the display value by one, rolling over to zero if
60  * limit is reached.
61  */
62 public void increment()
63 {
64     value = (value + 1) % limit;
65 }
66 }
67
saved
```

# Undervejs kan man inspicere tilstanden

## Kaldsstak

- Viser de igangværende metodekald
- Kan bruges til at vælge, hvilke variabler man vil se

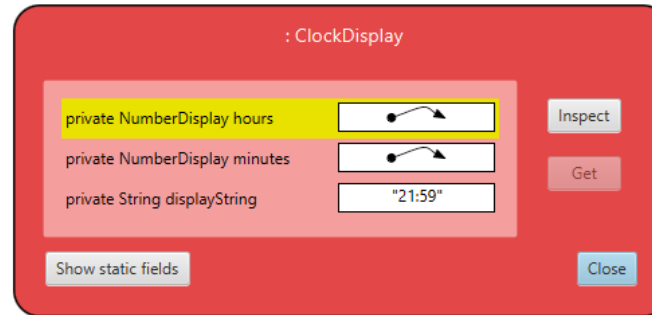
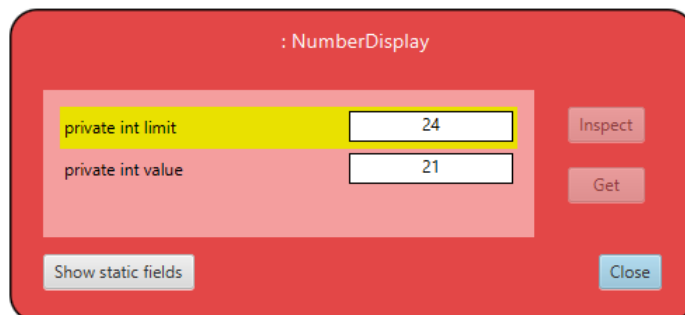
Knapper med de forskellige valgmuligheder



Værdier for klassevariabler

Værdier for feltvariabler

Værdier for lokale variabler (herunder parametre)



# ● Opsummering

---

- **Niveauer af programbeskrivelser**
  - Statiske / dynamiske beskrivelser
- **Klassevariabler og klassemetoder**
  - Variabler og metoder der er tilknyttet klassen (i stedet for at være tilknyttet objekter)
- **Problemløsning / analyse**
  - Husk at lave en grundig analyse
  - Det betaler sig i det lange løb
  - I store projekter bruger man ofte betydeligt mere tid på analyse end på programmering
- **BlueJ's Code Pad**
  - Nyttig til små eksperimenter
- **BlueJ's Debugger**
  - Nyttig til at finde fejl i kode
  - Bruges i projekter efter efterårsferien

# Så er vi klar til at forlade lokalet

---

- **Bliv siddende indtil I får besked på andet**
- **Alle går ud af døren fornedet til venstre for tavlerne**
  - Bliv siddende indtil jeg har fået den åbnet og sikret
  - De afsprittingsansvarlige bliver tilbage og hjælper med at afspritte alle borde og stole (dog ikke stofoverflader)
  - Vi starter med den side af auditoriet, der er nærmest døren
  - Rækkerne tømmes nede fra og op
- **Hvis der er nogen, som har spørgsmål til mig, bedes de vente hernede foran indtil lokalet er tømt, og jeg har fået pakket mit grej sammen**
- **Tak for i dag – Værsgo at begynde at gå ud**
  - Tag det stille og roligt og undgå at komme for tæt på andre
  - Vent på dem foran uden at mase på eller forsøge at overhale

**Det var alt for nu.....**

**... spørsmål**

---

