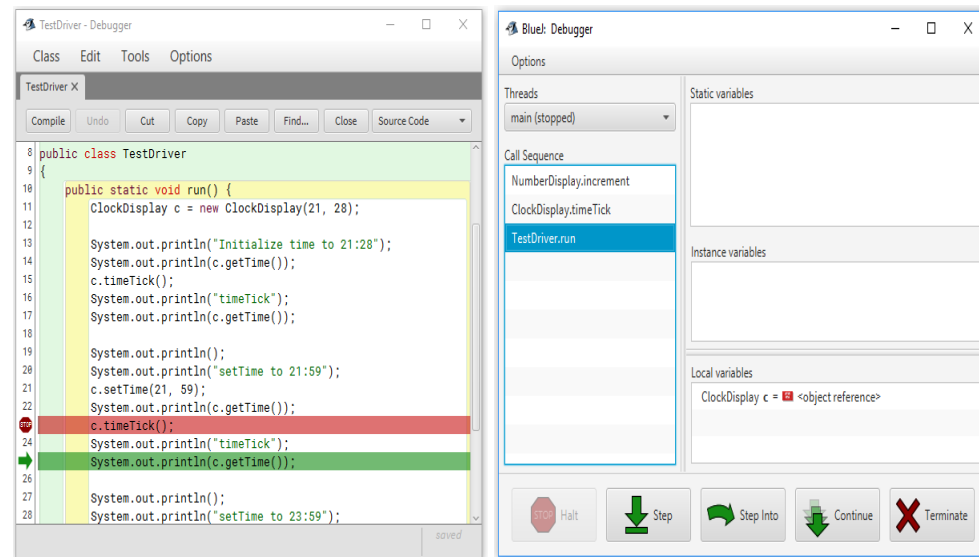


Forelæsning Uge 2 – Torsdag

- **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)**

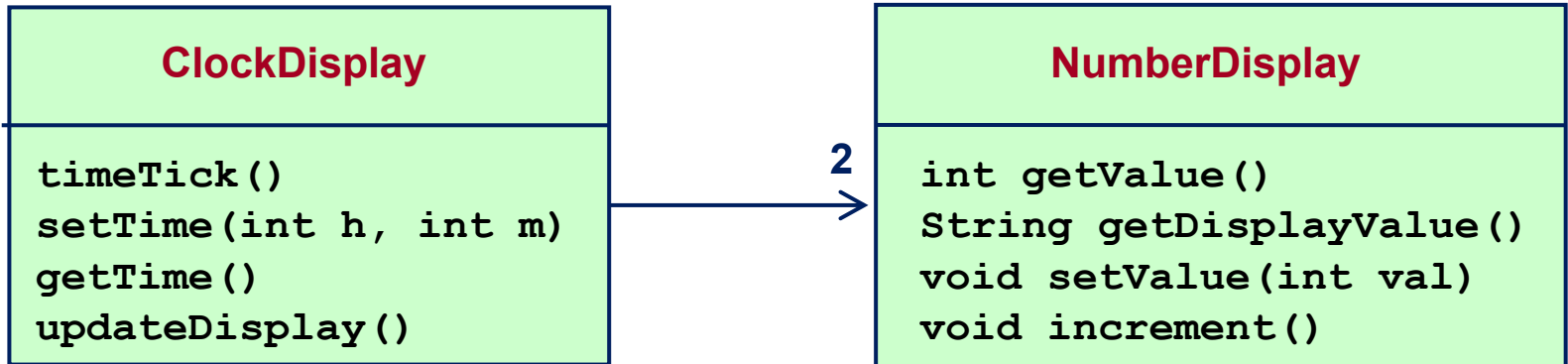
- Interaktion mellem objekter (metodekald)

Dynamisk (udførelse)

- scenarie for hvad der sker i en konkret situation

Klassediagram (statisk, oversigt)

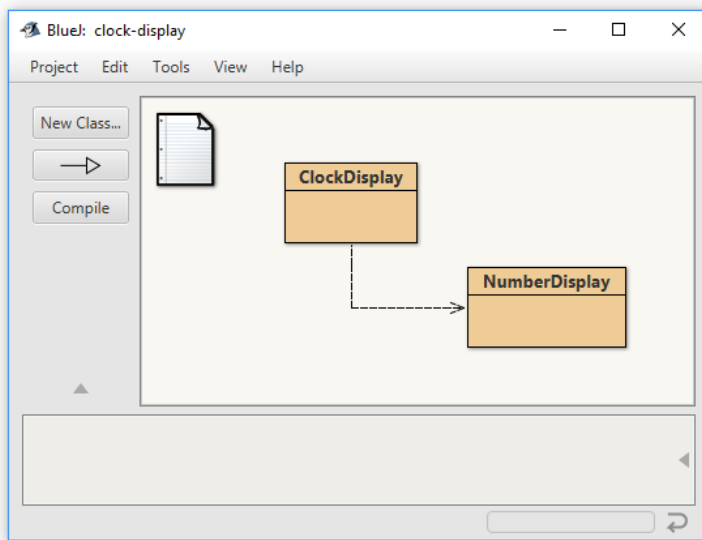
UML



UML klassediagrammer viser
sometider også nogle af
feltvariablerne

Man udelader sommetider nogle klasser, metoder,
feltvariabler, parametre eller returtyper, f.eks. er
der her kun returtyper med i den ene af klasserne

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 klassen har
- åbne objektets inspector, kan man se klassens feltvariabler

Java-kode (statisk, detaljeret)

The screenshot shows an 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 located on the right side of the toolbar. The code editor 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 in the image include:

- A blue arrow pointing to the "Source Code" dropdown menu with the text: "Skift mellem Java kode og dokumentation".
- A red box around the first Javadoc comment (lines 21-24) with a red arrow pointing to it from a larger red box containing the text: "Disse kommentarer indsættes automatisk i klassens dokumentation".
- A red box around the second Javadoc comment (lines 32-35) with a red arrow pointing to it from the same larger red box.
- A blue box with the text: "Vi har to forskellige konstruktører med forskellige parametre (overloading)".

The status bar at the bottom right shows the word "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 active. The "Constructor Summary" section is visible, showing two constructors: `ClockDisplay()` and `ClockDisplay(int hour, int minute)`. The "Constructor Detail" section shows the source code for these constructors. Annotations include red boxes around the first sentence of each constructor's Javadoc comment, a red box around the entire Javadoc comment, and blue boxes explaining the Javadoc format and generation process.

Constructor Summary

Constructors

Constructor and Description

`ClockDisplay()`
Constructor for ClockDisplay objects.

`ClockDisplay(int hour, int minute)`
Constructor for ClockDisplay objects.

Constructor Detail

ClockDisplay [Show source in Blue.J]

`public ClockDisplay()`
Constructor for ClockDisplay objects. This constructor creates a new clock set at 00:00.

ClockDisplay [Show source in Blue.J]

`public ClockDisplay(int hour, int minute)`
Constructor for ClockDisplay objects. This constructor creates a new clock set at the time specified by the parameters.

Annotations:

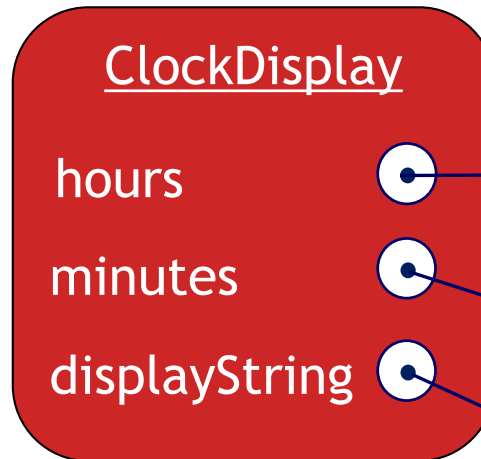
- Første sætning i kommentaren** (First sentence in the comment) - points to the first sentence of the constructor comments.
- Hele kommentaren** (The whole comment) - points to the entire Javadoc comment for the `ClockDisplay(int hour, int minute)` constructor.
- Tilsvarende gælder for** (The same applies to) - points to the list of items:
 - kommentarer til klassen (comments for the class)
 - kommentarer til metoder (comments for methods)
- Dokumentationen genereres automatisk ud fra de kommentarer, der er i Java koden** (The documentation is generated automatically from the comments that are in the Java code)

Objektdiagram (dynamisk, oversigt)

ClockDisplay



- Funktionaliteten af disse to dele ligner hinanden
- Hver del kan beskrives ved et objekt af typen **NumberDisplay**



NumberDisplay

limit 24

value 0

NumberDisplay

limit 60

value 0

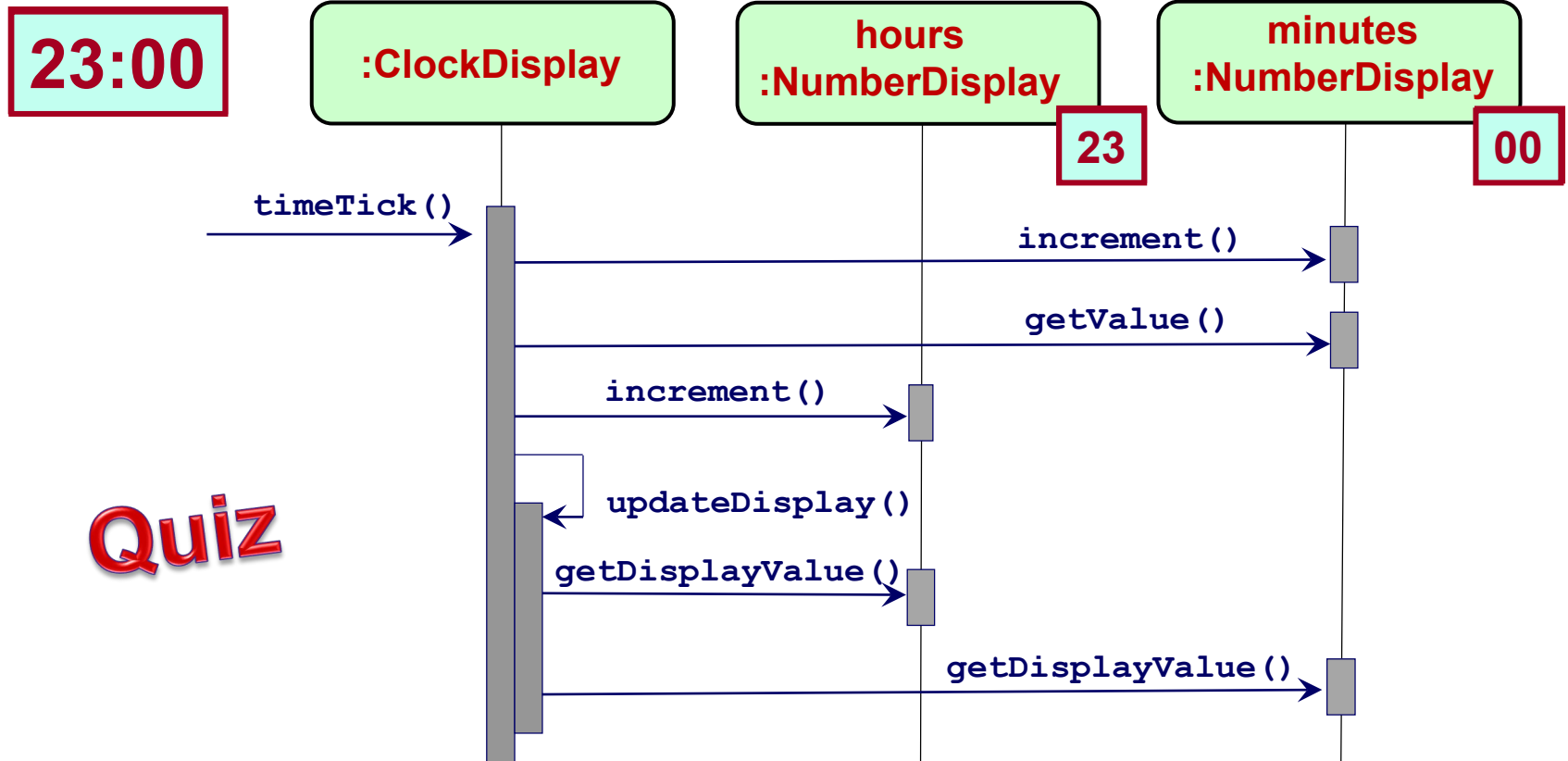
String
"00:00"

```
public ClockDisplay() {  
    hours = new NumberDisplay(24);  
    minutes = new NumberDisplay(60);  
    updateDisplay();  
}
```

Konstruktører

```
public NumberDisplay(int rolloverLimit) {  
    limit = rolloverLimit;  
    value = 0;  
}
```

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?

`System.out.println(...);`

**Klasse i Java's
standard bibliotek**

**Klassevariabel af
(type `PrintStream`)**

**Metode i
`PrintStream`**

Udskriver på BlueJ's terminal

A screenshot of a terminal window titled "BlueJ: Terminal Window - Pixel - Algorithms...". The terminal shows the following output:

```
*****
pixels: [18, 116, 36, 239, 116]
one 36: 36
one 116: 116
one 200: null
```

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

- **Klassevariabler 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**

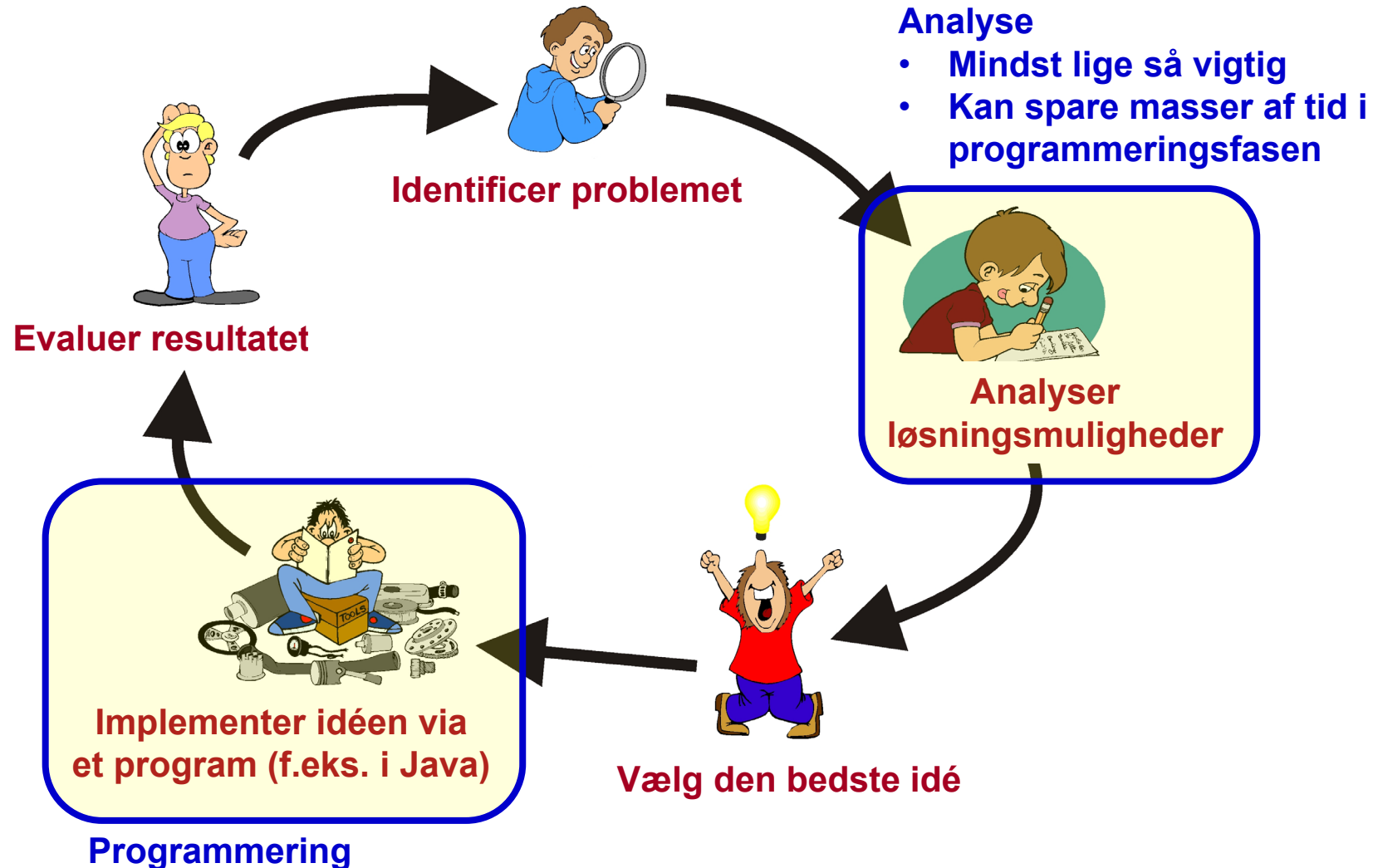
```
Account myAccount = new Account(...);  
myAccount.setInterestRate(1.035);
```



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**

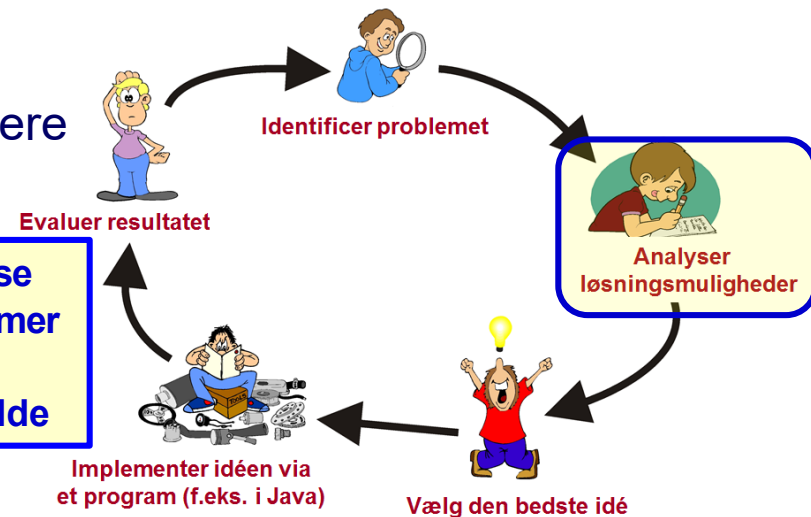
- Vinderen fortsætter til næste runde, mens taberen er slået ud af turneringen og ikke får flere kampe
- Vi vil gerne lave en algoritme, som beregner, hvor mange kampe, der skal til, hvis der er X spillere/hold i turneringen

- **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

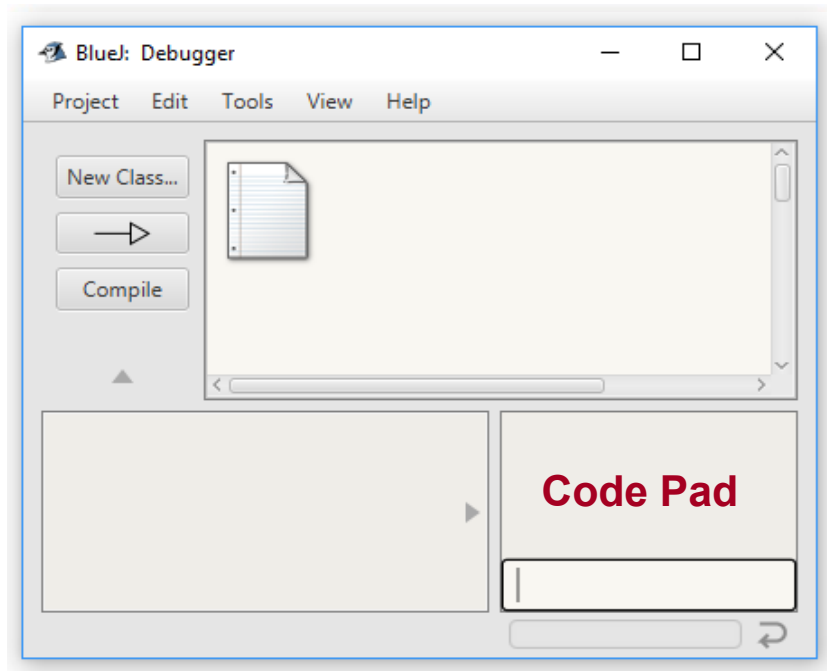
- Giver simplere programmer
- Hurtigere at skrive og nemmere at vedligeholde



- **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++; }

female

true (boolean)

i

8 (int)

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

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

Quiz

● BlueJ's debugger (afluser = fejlfinder)

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

Bruges i nogle af opgaverne i kursets anden halvdel

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



```
8 public class TestDriver
9 {
10     public static void run() {
11         ClockDisplay c = new ClockDisplay(21, 28);
12
13         System.out.println("Initialize time to 21:28");
14         System.out.println(c.getTime());
15         c.timeTick();
16         System.out.println("timeTick");
17         System.out.println(c.getTime());
18
19         System.out.println();
20         System.out.println("setTime to 21:59");
21         c.setTime(21, 59);
22         System.out.println(c.getTime());
23         c.timeTick();
24         System.out.println("timeTick");
25         System.out.println(c.getTime());
26
27         System.out.println();
28         System.out.println("setTime to 23:59");
29     }
30 }
```

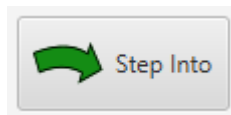
Mellem skridtene kan man inspicere systemets tilstand, dvs. værdierne af de forskellige slags variabler

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

```
45  /**  
46   * This method should get called once every minute - it makes  
47   * the clock display go one minute forward.  
48   */  
49  public void timeTick()  
50  {  
51      minutes.increment();  
52      if(minutes.getValue() == 0) { // it just rolled over!  
53          hours.increment();  
54      }  
55      updateDisplay();  
56  }  
57  
58  /**  
59   * Set the time of the display to the specified hour and  
60   * minute.  
61   */  
62  public void setTime(int hour, int minute)  
63  {  
64      hours.setValue(hour);  
65      minutes.setValue(minute);  
66  }
```

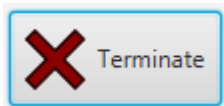
Metodekald

Parat til at udføre første sætning i den kaldte metode

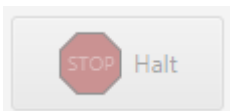
Andre knapper:



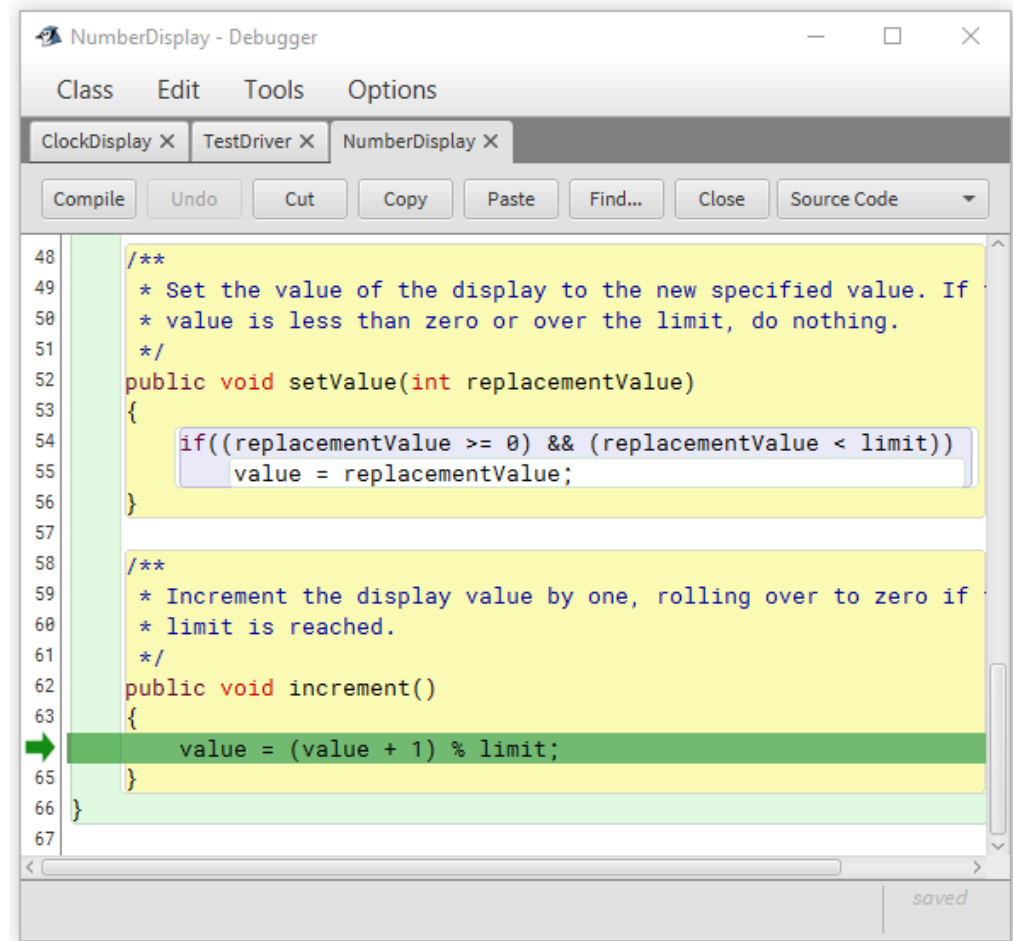
Fortsætter kørslen frem til næste breakpoint



Stopper kørslen



Nødstop (uendelig while løkke eller lignende)



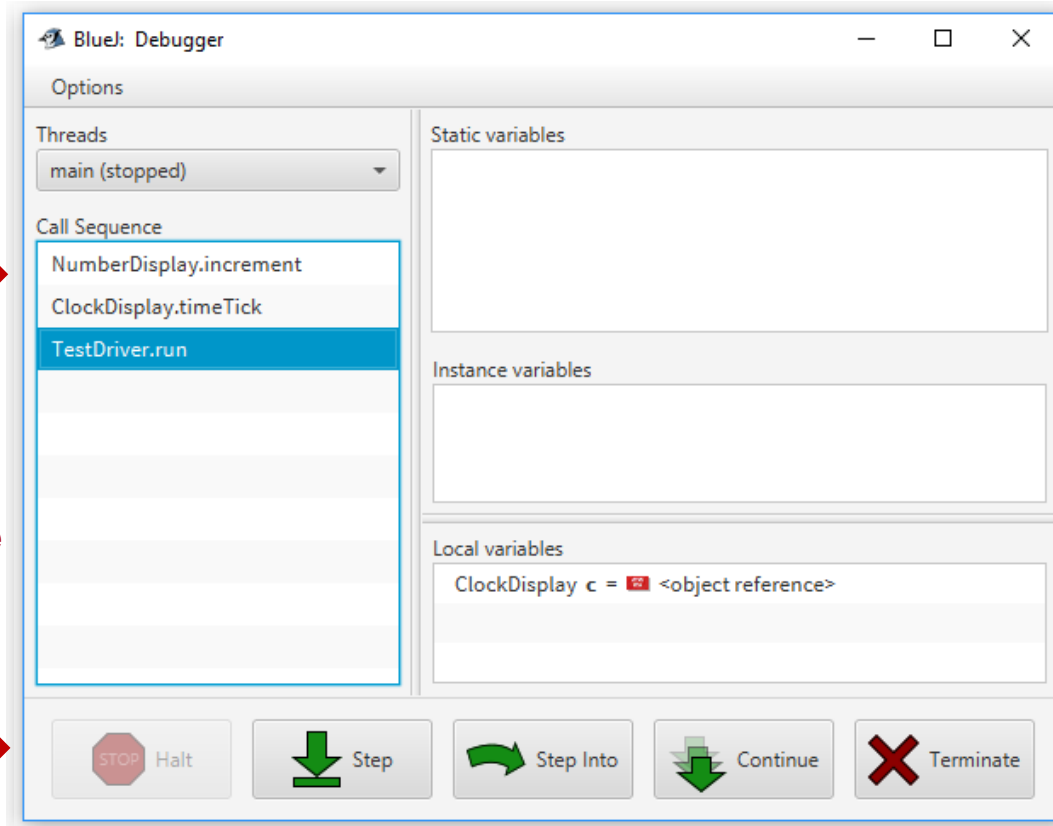
Undervejs kan man inspicere tilstanden

Kaldsstak

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



Knapper med de forskellige valgmuligheder



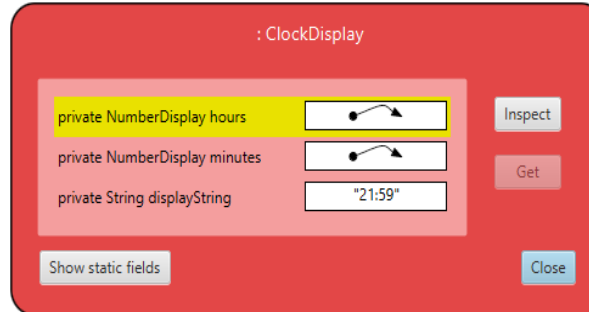
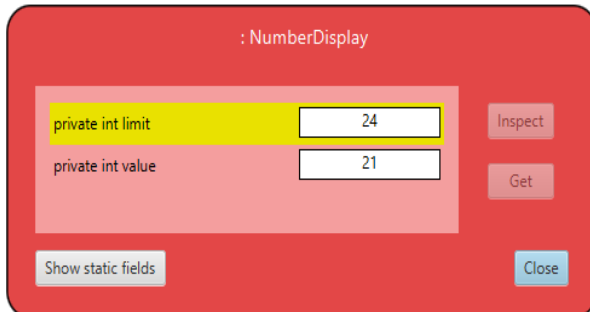
Værdier for klassevariabler



Værdier for feltvariabler



Værdier for lokale variabler (herunder parametre)



Når vi er stoppet ved et breakpoint (eller et statement nået via Step eller Step Into), kan vi inspicere alle variabler i de objekter, der er i gang med at udføre metodekald

● 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 i kursets anden halvdel

Det var alt for nu.....

... spørgsmål

