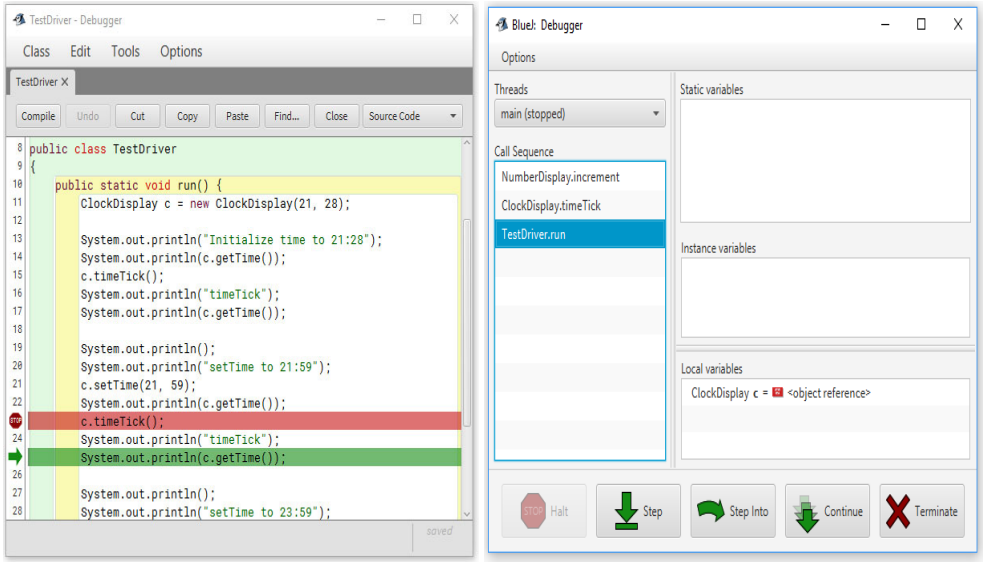
Forelæsning Uge 2 – Torsdag

- **Niveauer af programbeskrivelser**
  - Statiske / dynamiske beskrivelser
- **Klassevariabler og klassemetoder**
  - Variabler og metoder der et 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 (inplementation)

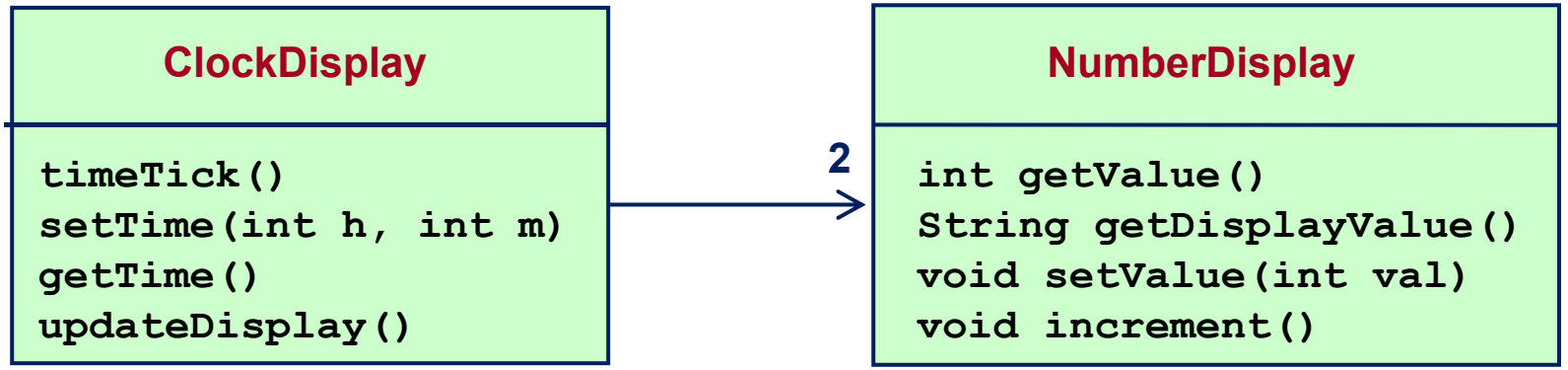
Statisk (struktur)
- rum for hvad der generelt kan ske

- **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)

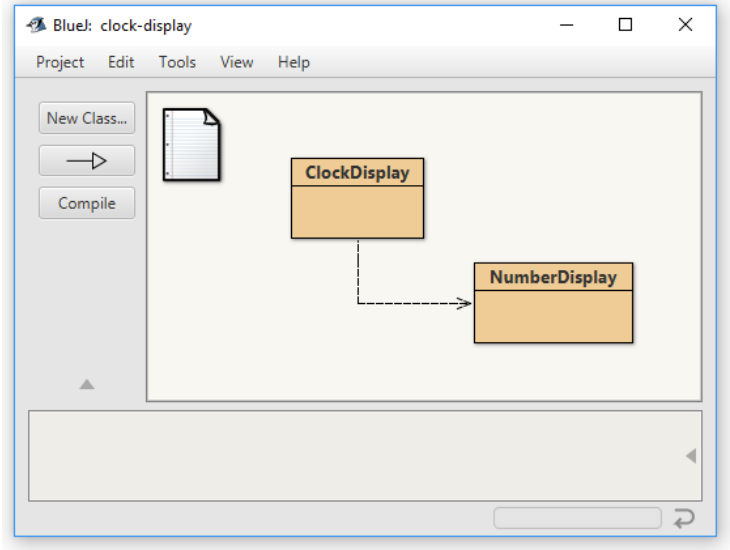
UML



UML klassediagrammer viser sommetider 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

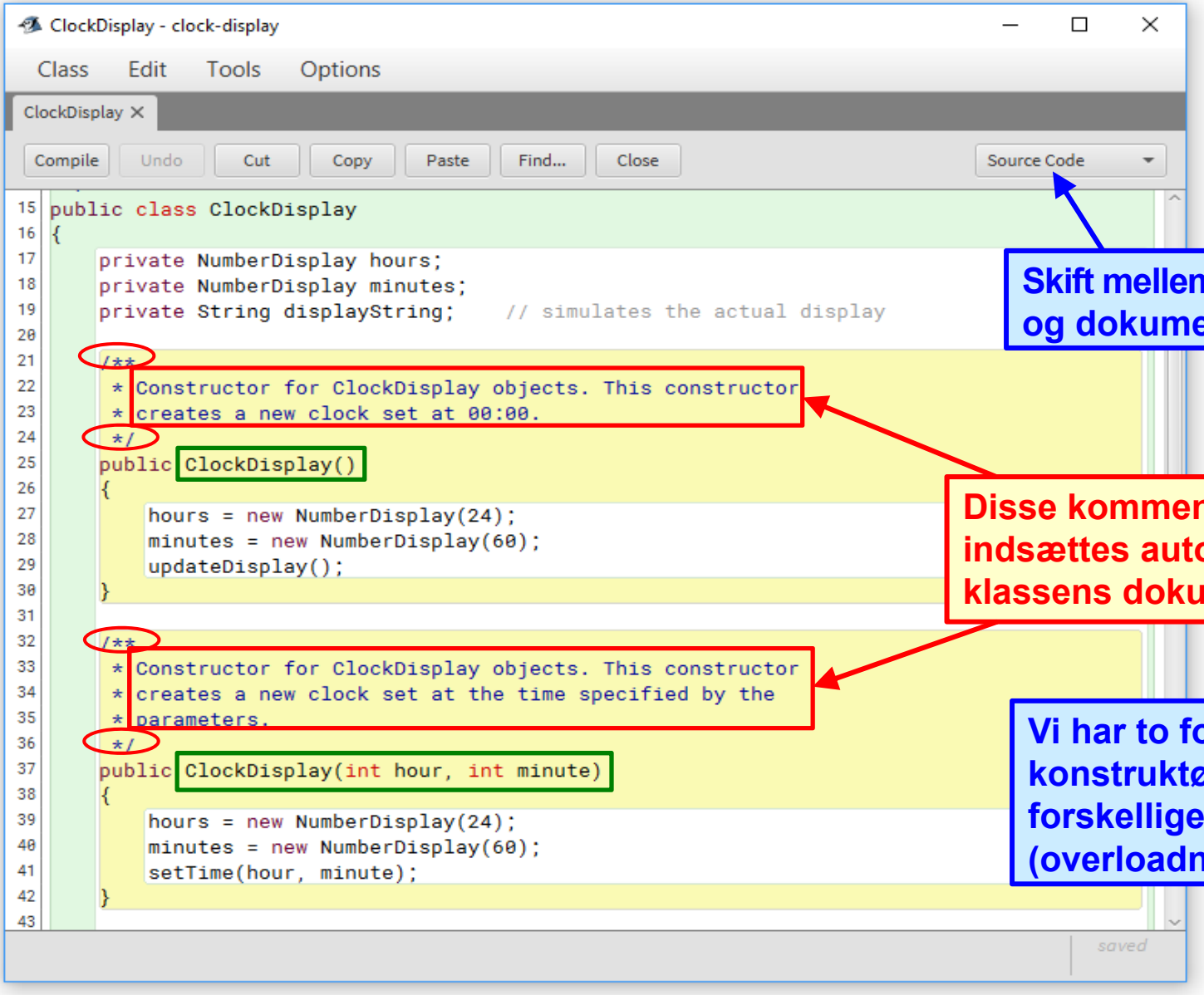
BlueJ



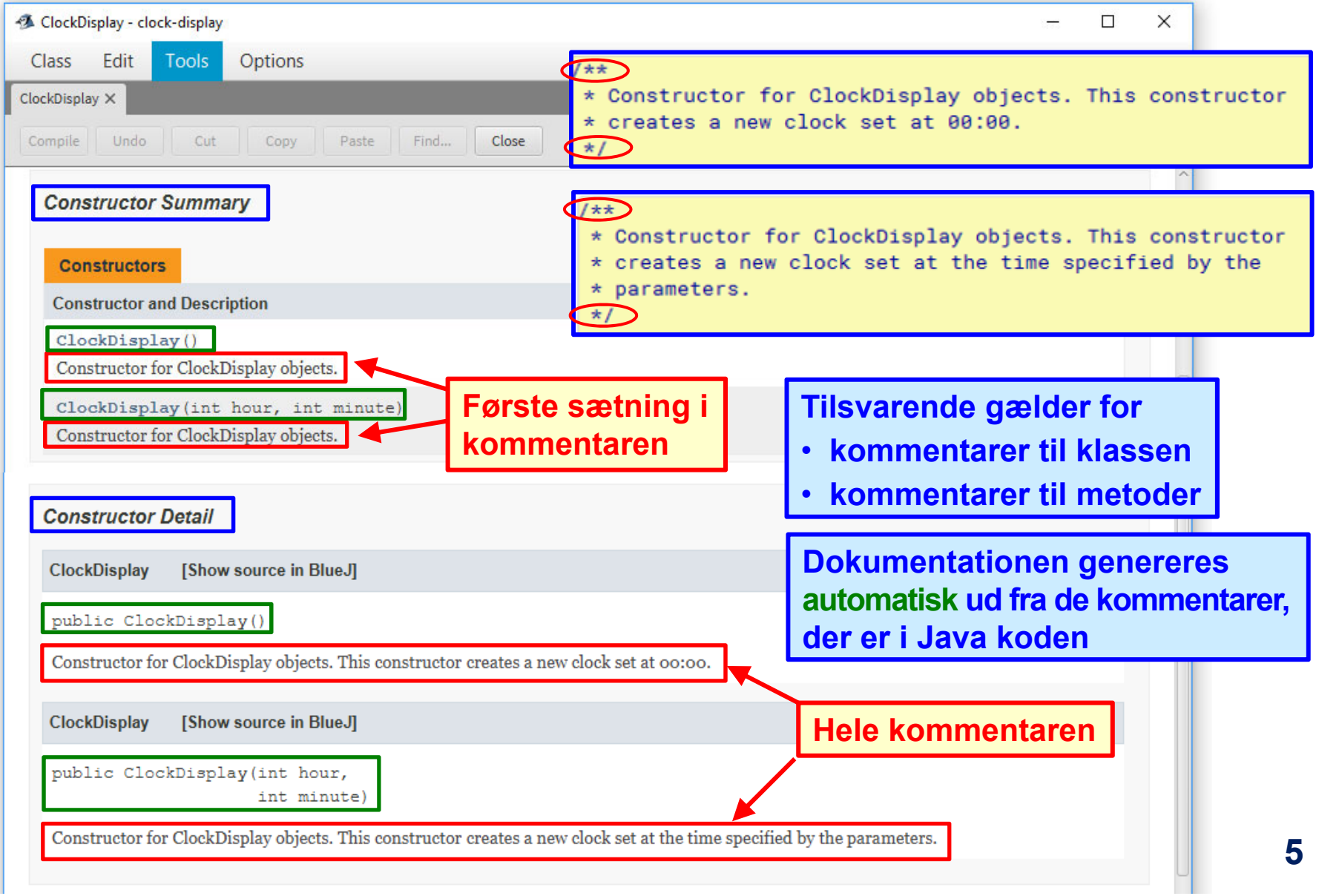
Her kan man kun se klassernes navne og pilene imellem dem, men ved at skabe et objekt og

- højre-klikke på det, kan man se, hvilke public metoder klassen har
- åbne objektets inspector, kan man se klassens feltvariabler

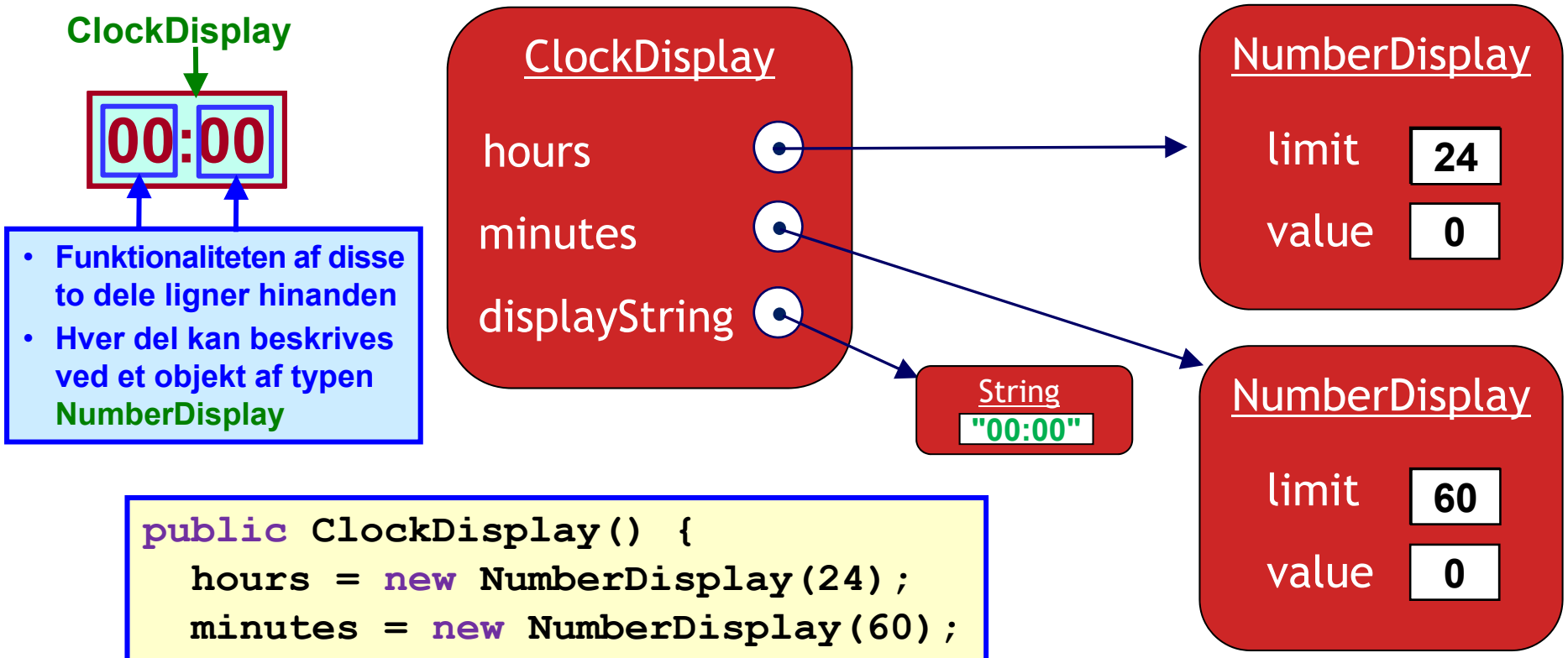
Java-kode (statisk, detaljeret)



JavaDOC (statisk, mellem-niveau)



Objektdiagram (dynamisk, oversigt)

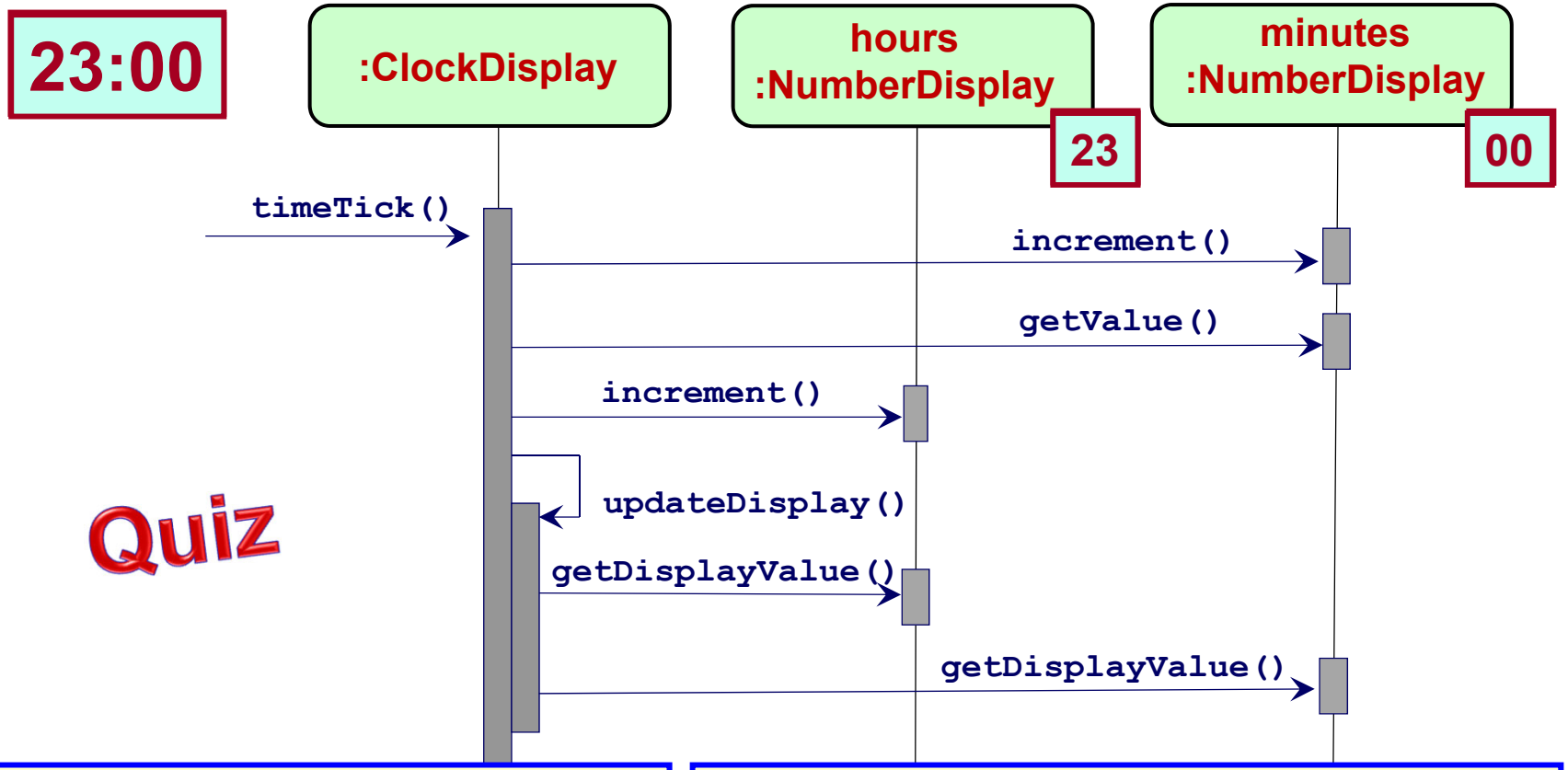



```
public ClockDisplay() {
   hours = new NumberDisplay(24);
   minutes = new NumberDisplay(60);
   updateDisplay();
}
```

Konstruktører

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

Sekvensdiagram for timeTick (dynamisk, detaljeret)




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

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

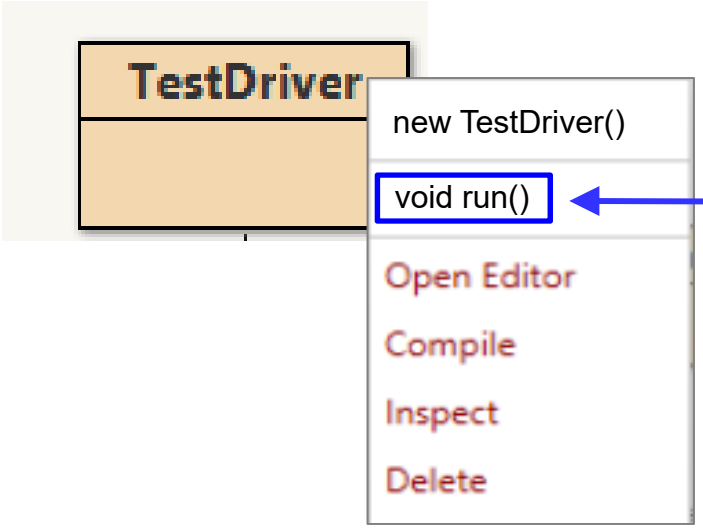
Klassevariabler og klassemetoder

- **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 reserverede ord **static** (dårligt ordvalg – levn fra gamle programmeringssprog)

Har I set dem før?

- I har allerede mødt klassemetoder?



Klassemetode

Kan kaldes via klassen uden først at lave et objekt af typen TestDriver

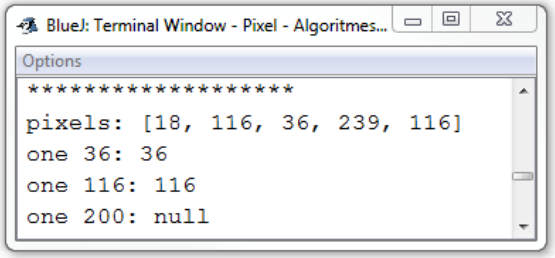
- I har også mødt klassevariabler?

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

Klasse i Java's standard bibliotek

Klassevariabel af (type PrintStream)

Metode i PrintStream



Udskriver på BlueJ's terminal

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) == a^b
  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);
```

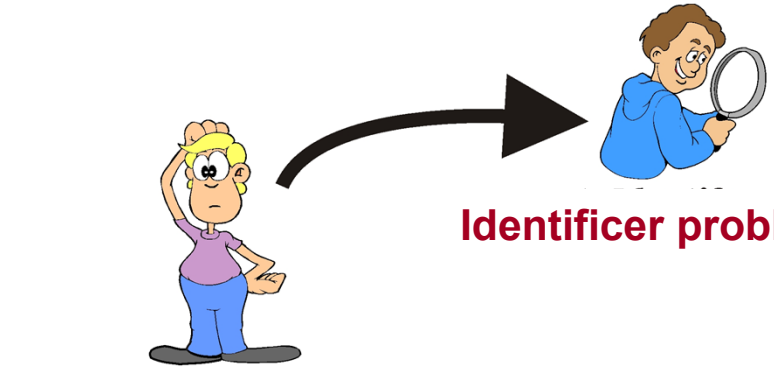
- De kan også tilgås via et objekt, men det er dårlig programmeringsstil og kan være forvirrende

```
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



Identificer problemet

Analyse
- Mindst lige så vigtig
- Kan spare masser af tid i programmeringsfasen

Analyser løsningsmuligheder

Vælg den bedste idé

Implementer idéen via et program (f.eks. i Java)

Programmering

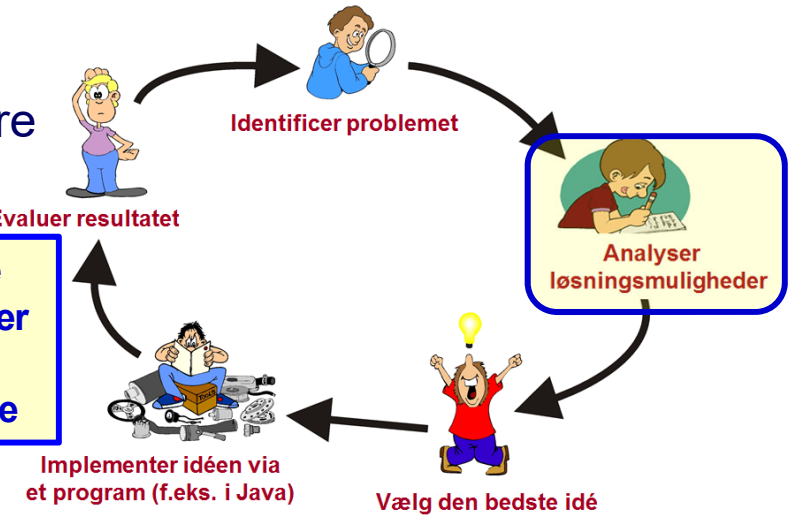
Evaluer resultatet

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

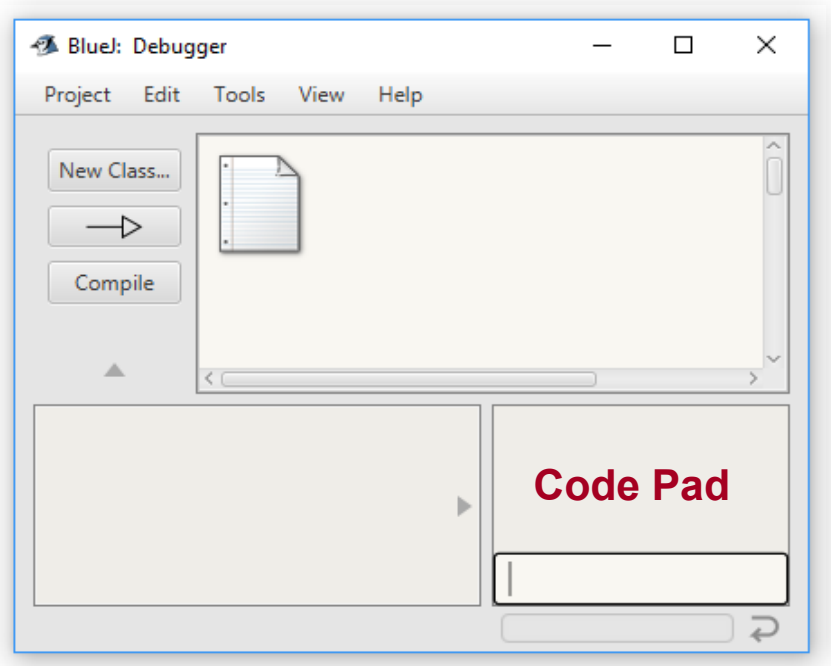
- **Løsning for X = 29**
  - 13 sekstendedels finaler + 3 oversiddere
  - 8 ottendedels finaler
  - 4 kvartfinaler
  - 2 seminfinaler
  - 1 finale
  - I alt 28 kampe

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




```
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

- **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, idet det så er nemmere at gentage eksperimentet

Quiz

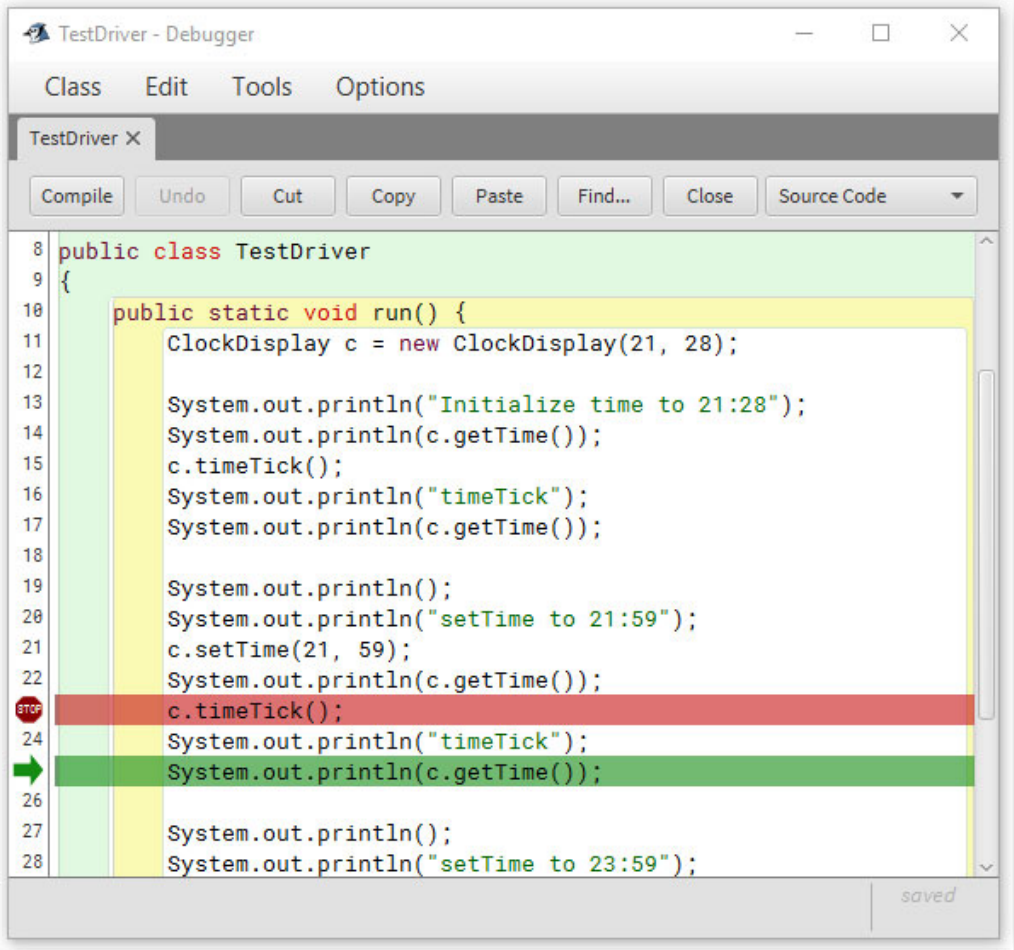
BlueJ's debugger (afluser = fejlfinder)

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)



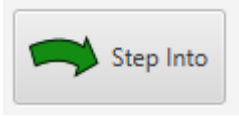
Herefter kan man "steppe" gennem koden sætning for sætning

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

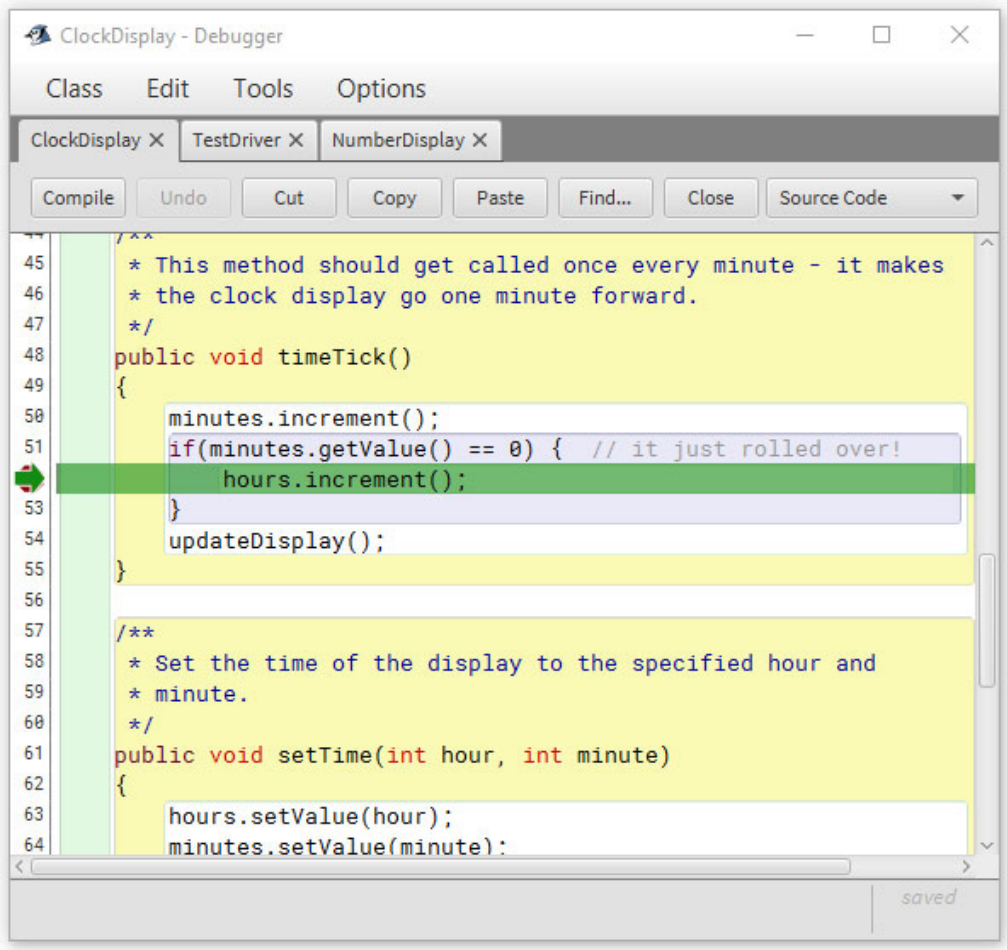
Metodekald

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

Udfører hele metodekaldet uden at man ser detaljerne

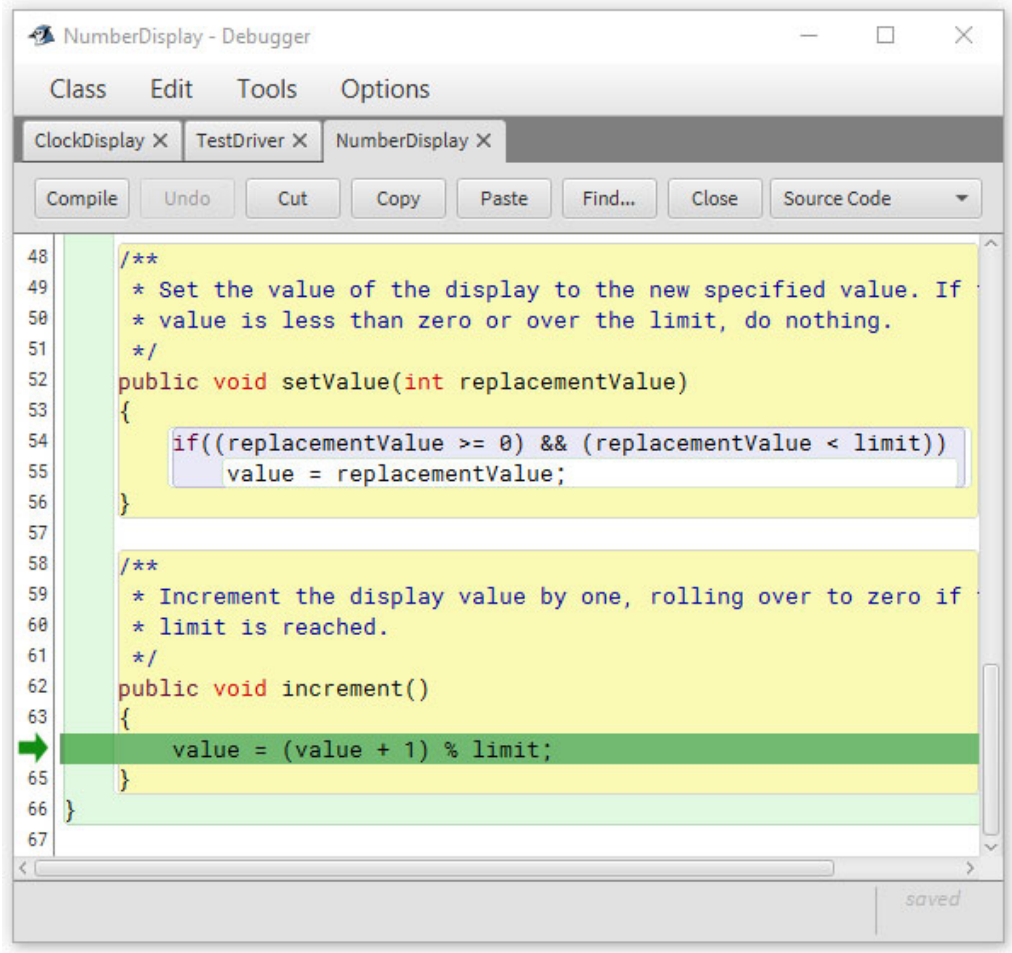


Starter metodekaldet, men stopper inden første sætning i kroppen af den kaldte metode



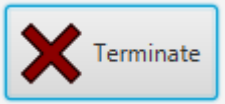
Metodekald

Parat til at udføre første sætning i den kaldte metode

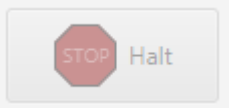


Andre knapper:

Fortsætter kørslen frem til næste breakpoint



Stopper kørslen

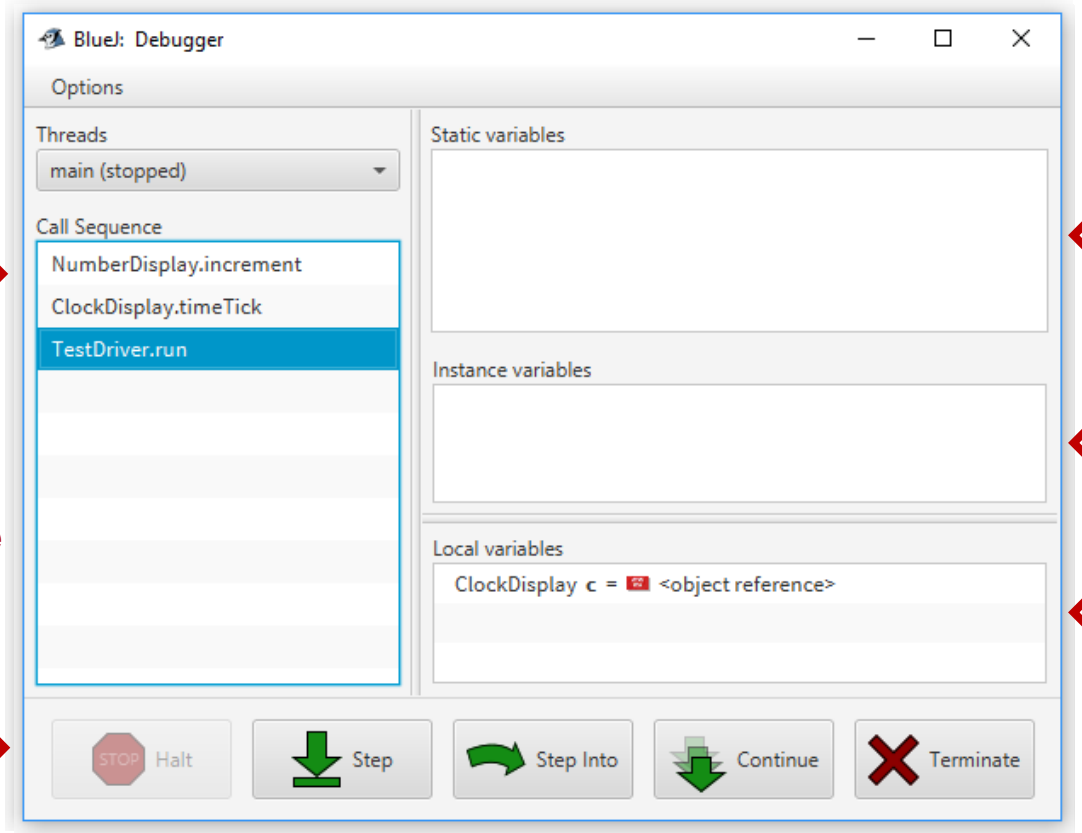


Nødstop (uendelig while løkke eller lignende)

Undervejs kan man inspicere tilstanden

Kaldsstak

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

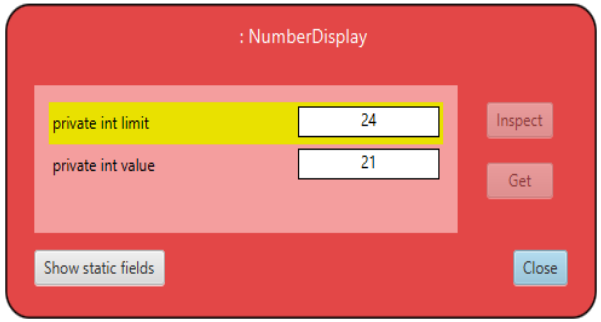


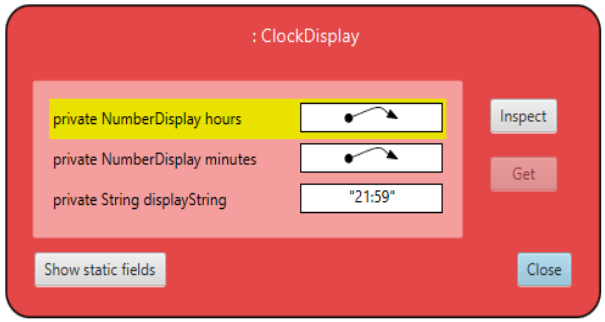
Knapper med de forskellige valgmuligheder

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

Vi vælger det objekt, som vi vil inspicere, via kaldstakken

Opsummering

- **Niveauer af programbeskrivelser**
  - Statiske / dynamiske beskrivelser
- **Klassevariabler og klassemetoder**
  - Variabler og metoder der et 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