

Objekter der samarbejder

Debugger i BlueJ

Grundlæggende Programmering med Projekt

Dan Witzner Hansen

Diverse

- Ønsker til forelæsningsne?
- Sidste gang?

I dag

- Objekter der samarbejder
 - abstraktion & modularisering
 - objektdiagram og klassediagram
 - primitive typer og objekttyper
 - objekter der skaber andre objekter
 - metodekald – interne og eksterne
- Debugger
 - statisk syn på programmet: kildeteksten
 - dynamisk syn på programmet: udførelse
 - debugger: inspicer programmets udførelse

Digitalt ur

11:03

Abstaktion & Modularisering

- Abstaktion
 - ignorerer detaljer og fokuser på de højere niveauer af et problem
 - eksempel – design af en cykel
- Modularisering
 - opdel et større problem i mindre dele
 - undersøg og løs delproblemerne separat
 - saml delløsningerne til en samlet løsning
 - “del og hersk” (Divide-and-Conquer)
 - eksempel – konstruktion af en cykel: stel, pedaler, kæde, gear, hjul, bremser, styr, sadel, bagagebærere, ... – de enkelte dele er udskiftelige

Modularisering af det digitale ur

11:03

Et display med fire tal?

Eller to displays
med to tal?

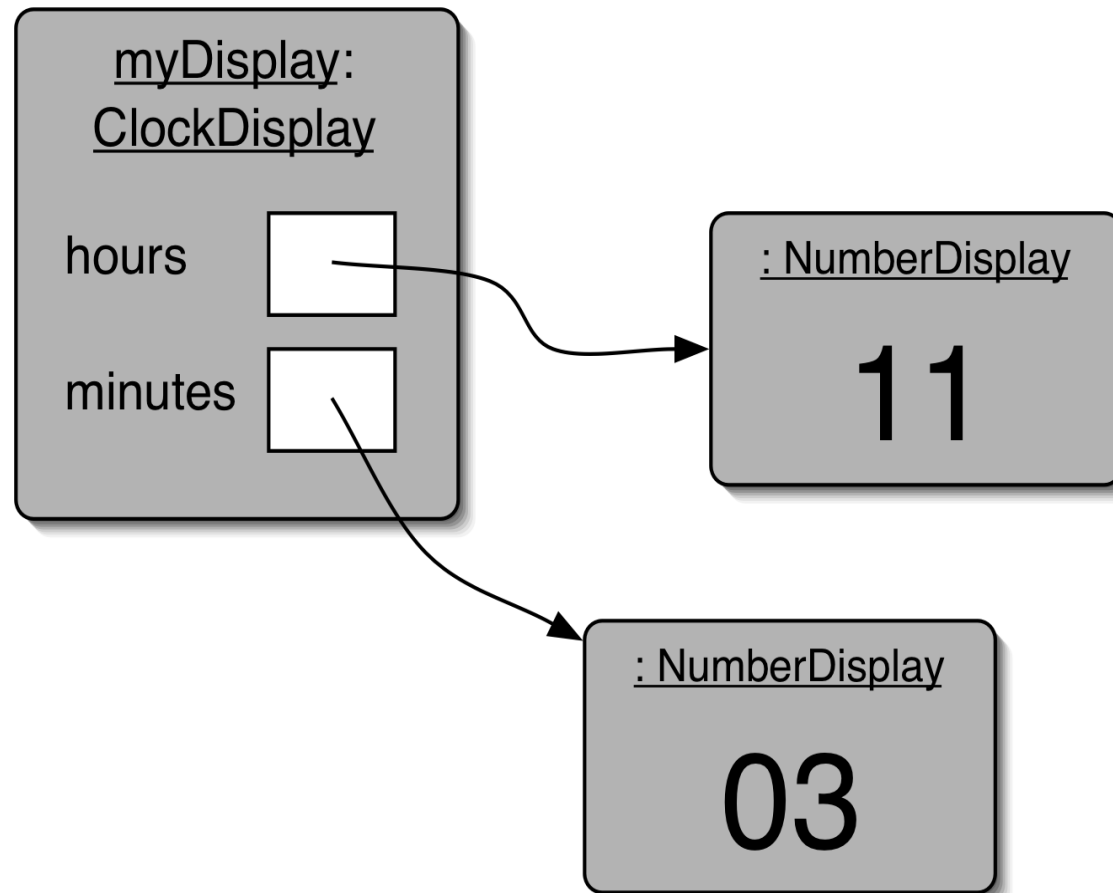
11 03

NumberDisplay og ClockDisplay

```
public class ClockDisplay {  
    private NumberDisplay hours;  
    private NumberDisplay minutes;  
  
    ...  
}
```

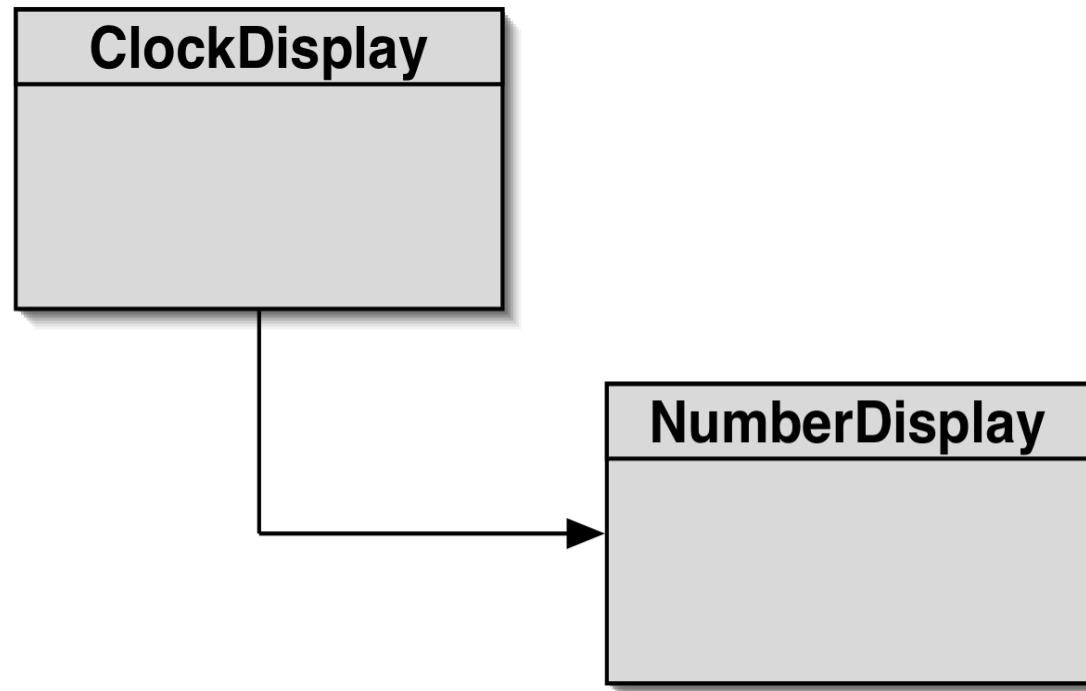
```
public class NumberDisplay {  
    private int limit;  
    private int value;  
  
    ...  
}
```

Objektdiagram



- Objekt `myDisplay` af klasse `ClockDisplay` har referencer til to objekter af klasse `NumberDisplay`

Klassediagram



- Klasse **ClockDisplay** bruger klasse **NumberDisplay**

Kildetekst: NumberDisplay del 1

- Optælling begynder forfra ved 24 eller 60:

```
public class NumberDisplay {  
    private int limit;  
    private int value;  
  
    public NumberDisplay(int rollOverLimit) {  
        limit = rollOverLimit;  
        value = 0;  
    }  
  
    public void increment() {  
        value = (value + 1) % limit;  
    }  
    ...  
}
```

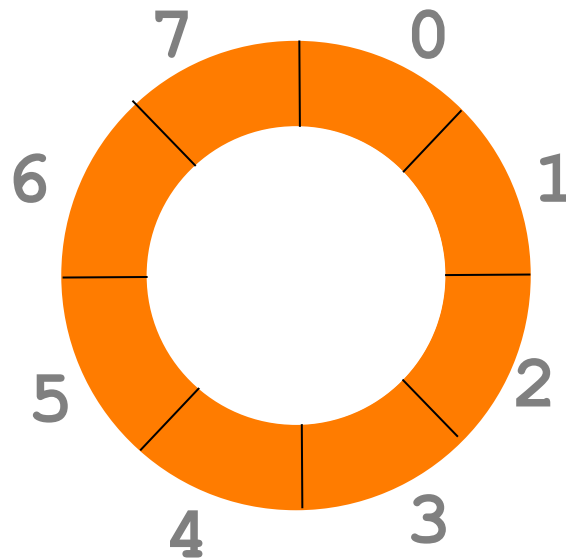
Modulo operatoren

`mod` operatoren (%) bruges til at finde rest ved division

– $8\%5 = 3$, $1\%5 = 1$, $2\%5 = 2$, $5\%5 = 0$

– Tælle eksempel

`(value + 1) % limit`



Kildetekst: NumberDisplay del 2

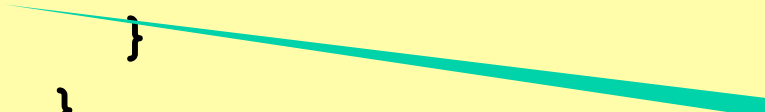
- Værdien 4 skal vises som "04":

```
public class NumberDisplay {  
    ...  
    public String getDisplayValue() {  
        if (value < 10) {  
            return "0" + value;  
        } else {  
            return "" + value;  
        }  
    }  
    ...  
}
```

Logiske operatorer

Java	Matematik	Betydning
<code>!x</code>	$\neg x$	ikke x
<code>x && y</code>	$x \wedge y$	x og y
<code>x y</code>	$x \vee y$	x eller y

```
public class NumberDisplay {  
    ...  
    public void setValue(int replacementValue) {  
        if ((replacementValue >= 0) &&  
            (replacementValue < limit))  
        {  
            value = replacementValue;  
        }  
    }  
}
```



else kan udelades

Miniøvelser uden pc

- Opgaveseddel 2.1:
 - B&K 4+5 ed: 3.6, 3.7, 3.8: tjekopgaver om `setValue`
 - B&K 3.13: tjekopgave om `getDisplayValue`
- Vi starter igen kl **om ca 25 mins**

Objekter der skaber objekter

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

- Lav to objekter af klasse NumberDisplay
- Felterne `hours` og `minutes` peger på dem

Primitive typer og objekt-typer

Type	Beskrivelse
<code>int</code>	32 bit heltal
<code>long</code>	64 bit heltal
<code>short</code>	16 bit heltal
<code>byte</code>	8 bit heltal
<code>char</code>	tegn 'a', 'A', ...
<code>double</code>	64 bit kommmatal
<code>float</code>	32 bit kommmatal
<code>boolean</code>	<code>false</code> eller <code>true</code>
<code>String</code>	tegnstreng (fra Java)
<code>Circle</code>	cirkel (fra B&K)
<code>Book</code>	bog (fra B&K)
...	... alle klasser ...

primitive
typer

objekt-
typer

Variable og felter:

Primitive typer versus objekt-typer

En variabel af primitiv type *indeholder* en værdi:

```
int i;
```

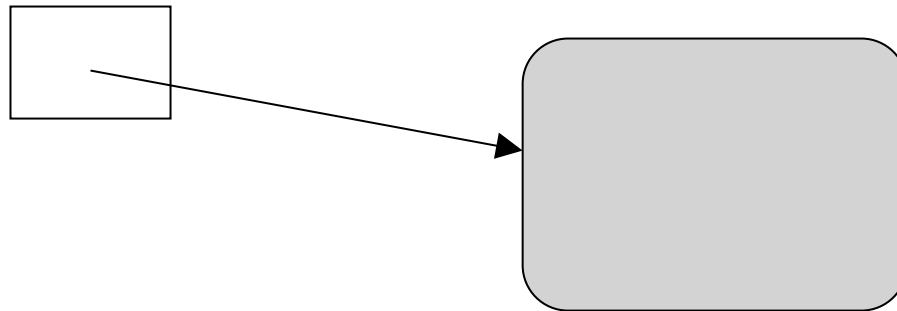


A small square box containing the number 32.

primitiv
type

En variabel af objekt-type *peger* på et objekt:

```
Book obj;
```



objekt-
type

Tildeling af primitiv type *kopierer værdien*

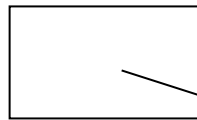
`int a;` 32

Udfør `b = a;`

`int b;` 32

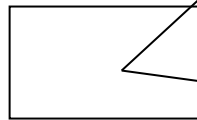
Tildeling af objekt-type kopierer *ikke* værdien

Book a;



Udfør **b = a;**

Book b;



Overloading: flere metoder med samme navn

```
public class ClockDisplay {  
    ...  
    public ClockDisplay() {  
        hours = new NumberDisplay(24);  
        minutes = new NumberDisplay(60);  
        updateDisplay();  
    }  
  
    public ClockDisplay(int hour, int minute) {  
        hours = new NumberDisplay(24);  
        minutes = new NumberDisplay(60);  
        setTime(hour, minute);  
    }  
}
```

Skelnes på
parametrene

Eksterne metodekald

- Et andet objekts metoder kan kaldes ved at skrive `objekt.metode()`

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

Kald
`increment`
på `minutes`

Kald
`increment`
på `hours`

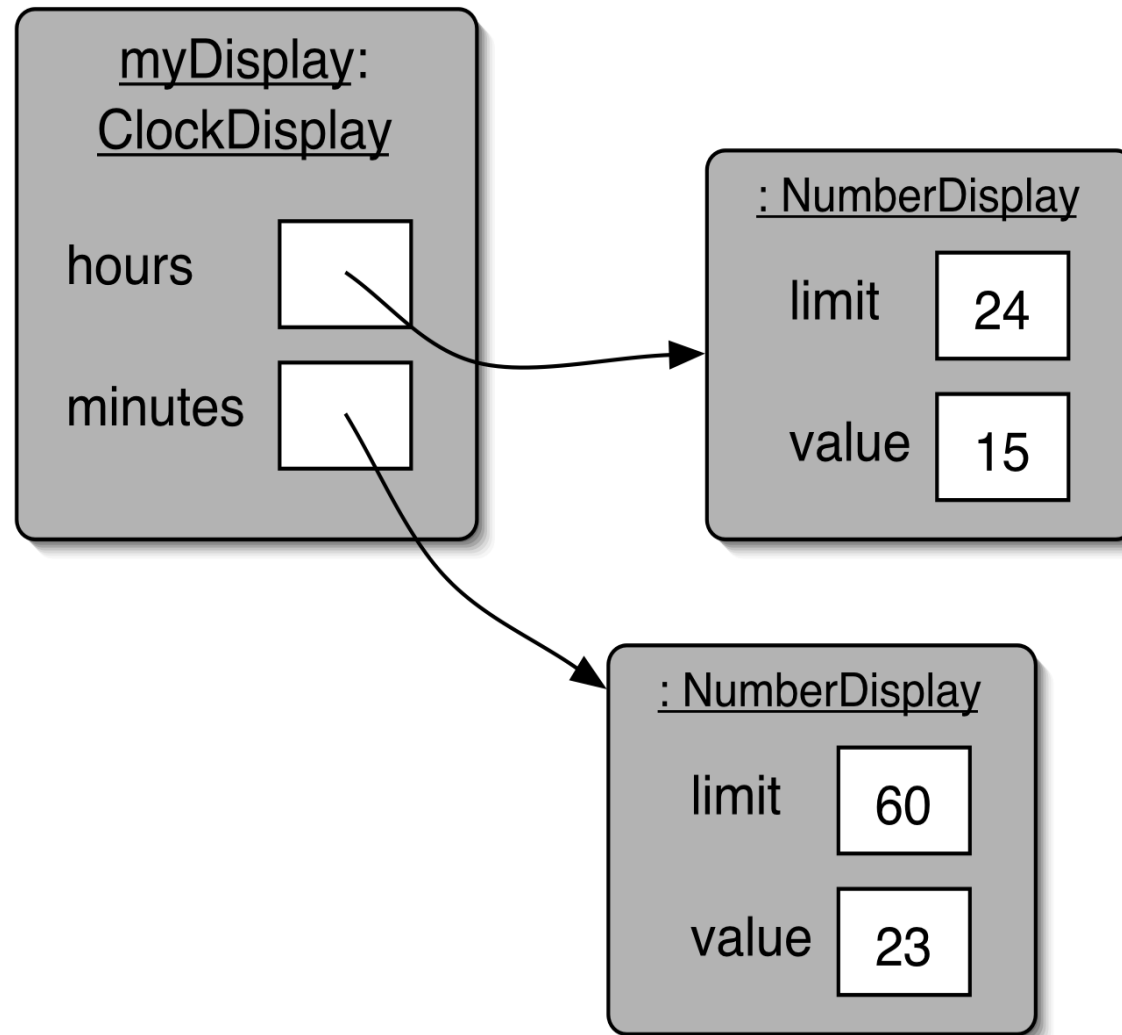
Interne metoder

- Interne hjælpemetoder er gerne **private**
- Det er ikke meningen de skal kaldes udefra

Intern
hjælpemetode

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

ClockDisplay objektdiagram



Objekter der skaber andre objekter

```
public class NumberDisplay {  
    private int limit;  
    ...  
    public NumberDisplay(int rolloverLimit) {  
        limit = rolloverLimit;  
        ...  
    }  
}
```

Formel
parameter

```
public class ClockDisplay {  
    private NumberDisplay hours;  
    ...  
    public ClockDisplay() {  
        hours = new NumberDisplay(24);  
        ...  
    }  
}
```

Aktuel
parameter

Generel form for metodekald:

objekt . metode(parameterliste)

```
public class NumberDisplay {  
    public void increment()  
    { ... }  
    ...  
}
```

Eksternt

```
public class ClockDisplay {  
    public void timeTick() {  
        minutes.increment();  
        ...  
        updateDisplay();  
    }  
    private void updateDisplay()  
    { ... }  
}
```

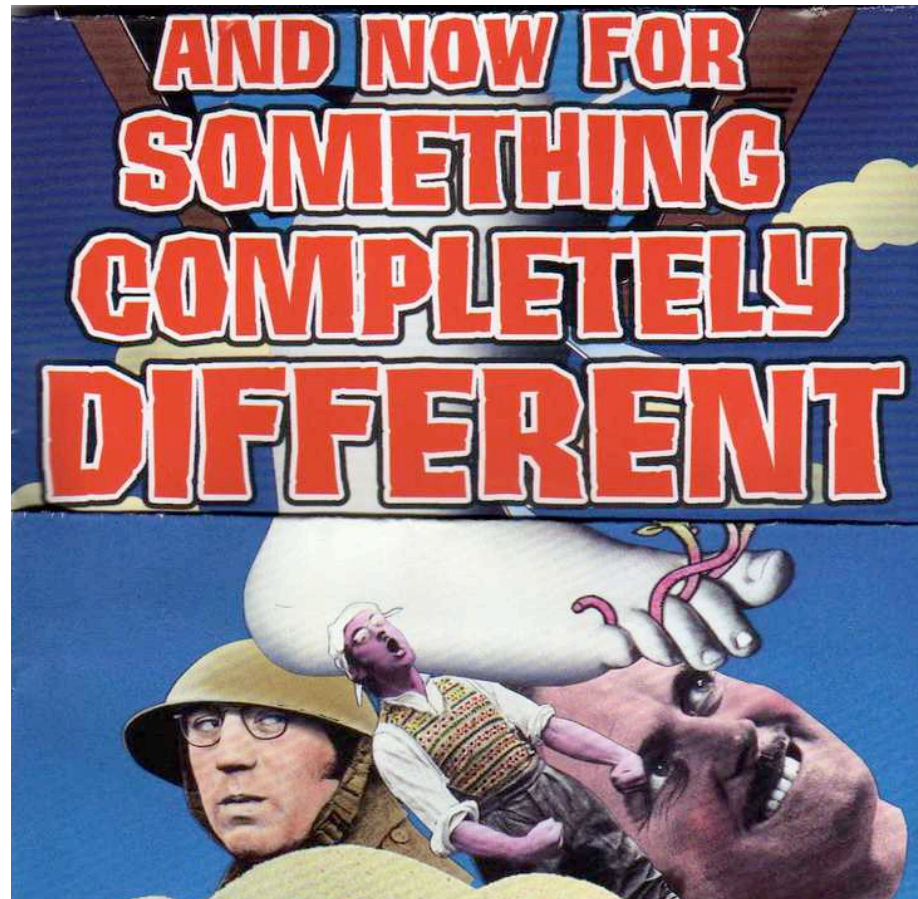
Internt

Samme som
`this.updateDisplay()`

Fælles øvelse

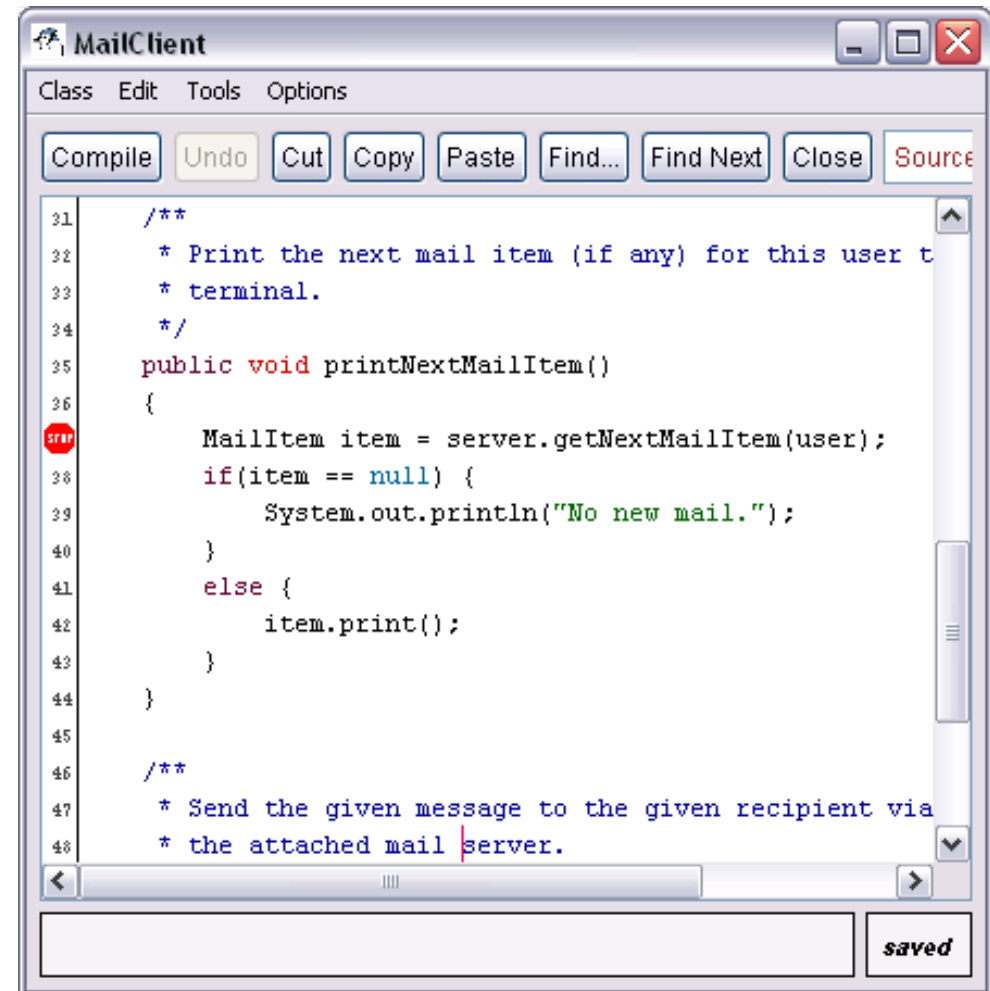
- Hvad skal laves om hvis vi lave et ur med timer, minutter, sekunder?
- Hvad viser det om objektorienteret programmering?





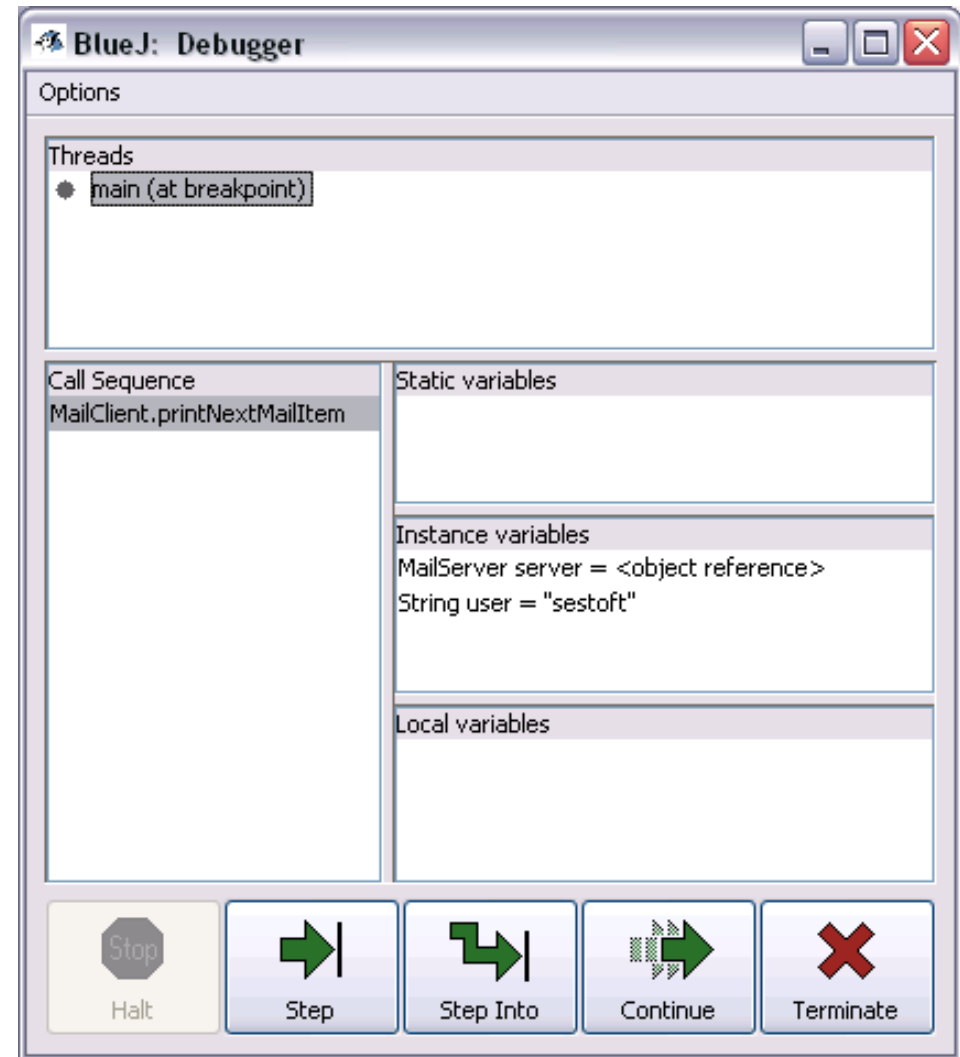
Debugger

- Breakpoint: sted hvor programmet standser
- Sæt breakpoint: dobbeltklikke i margen ud for programlinje
- Når programkørslen når til breakpointet
 - standser programmet
 - debugger åbner



Debuggervinduet

- Viser at vi er i metode `printNextMailItem`
- At feltet `server` peger på et objekt
- At feltet `user` har værdien "sestoft"
- Vi kan nu
 - gå ét skridt videre
 - gå ind i en metode (hvis der er et kald)
 - køre færdig, eller til næste breakpoint
 - afslutte programmet



Eksempel: Følg et kald til `timeTick()`

- Lav et `ClockDisplay(4,59)`-objekt
- Sæt breakpoint først i metode `timeTick()`
- Kald `timeTick()` fra BlueJ
- Se på debuggervinduet
- "Step into" på `minutes.increment()`
- Single step videre

Praktisk brug af debugger

- Kode
 - Statisk billede, “compile-time”
- Debugging
 - Dynamisk billede, “run-time”
- Fremgangsmåde ved debugging
 - Læs koden!
 - Sæt breakpoints i koden
 - Lav enkeltskridt (“Step”) med debuggeren
 - Hold øje med variable og objekters tilstand
 - Følg metodekald (“Stepping into methods”)
 - Eller spring metodekald over (“Stepping over”)

Torsdag og næste uge

- Til Torsdag
 - Læs B&K kapitel 3
 - Start på B&K kapitel 4: arrays og løkker
- Til tirsdag
 - Ingen forelæsning– RUStur

Ekstraopgaver (seddel02ekstra.pdf)

- Klasser Punkt og Person
- Objekt som parameter til metode:

