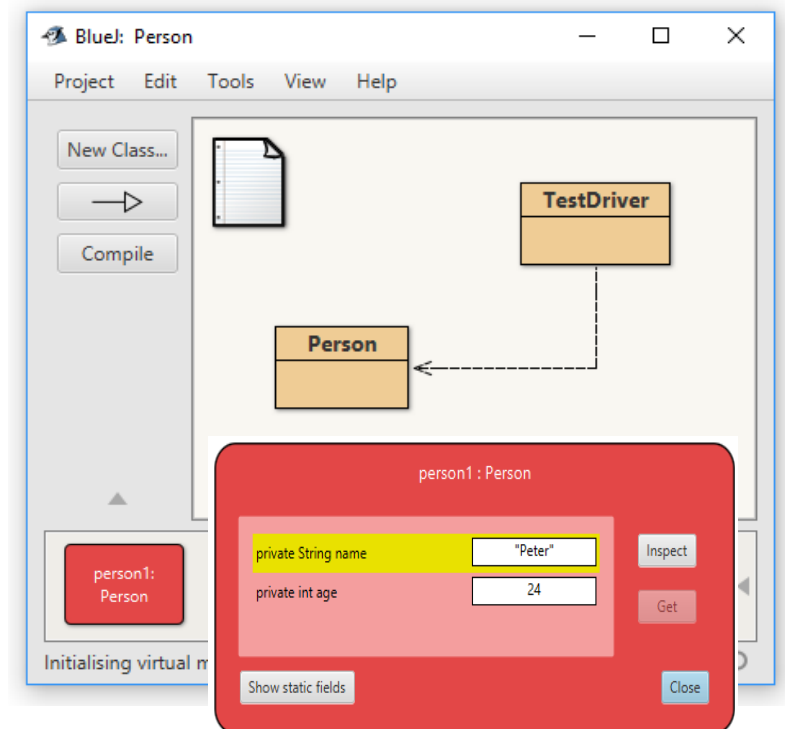


Forelæsning Uge 1 – Torsdag

- **Objekters tilstand og opførsel**
 - Java og BlueJ
- **Skabelse af objekter (via new-operatoren)**
 - Objektdiagrammer
- **Iteration (gentagelser), selektering (valg) og parametrisering**
 - Java's for løkke
 - Java's if sætning
 - Parametre i metoder
- **Forskellige slags variabler**
 - Feltvariabler
 - Lokale variabler

Studerende med merit for kurset

- er velkomne til at følge dele af forelæsninger og øvelser, hvis de ønsker det
- kan få genoprettet deres adgang til kursets webboard ved at sende mig en mail herom



● Objekters tilstand og opførsel i Java

- **Tilstand**

- Et objekts **tilstand** er defineret ved et sæt af feltvariabler (fields)
- Feltvariablerne er fastlagt i klassens erklæring
- **Alle objekter** (af en given klasse) har de **samme feltvariabler**
- Hvert objekt har sin **egen tilstand** (værdier af feltvariabler)

- **Opførsel**

- Et objekts **opførsel** er defineret ved et sæt konstruktører og metoder
- Konstruktører og metoder er fastlagt i klassens erklæring
- **Alle objekter** (af en given klasse) har de **samme konstruktører** og **metoder**

Objekters tilstand i Java

Erklæring/beskrivelse af en **klasse**, der hedder **Person** og kan bruges af **alle**

```
public class Person {
```

```
    private String name;  
    private int age;
```

```
    public Person(String n, int a) {  
        name = n;  
        age = a;  
    }
```

```
    public String getName() {  
        return name;  
    }
```

```
    public void setName(String n) {  
        name = n;  
    }
```

```
    public int getAge() {  
        return age;  
    }
```

```
    public void birthday() {  
        age = age + 1;  
        System.out.println("Happy birthday " + name + "!");  
    }  
}
```

Tilstand beskrives ved hjælp af

Feltvariabler

- **Access modifier**, der fortæller hvorfra feltvariablen kan anvendes / tilgås
- Access modifieren bør altid være **private**
- Det betyder, at feltvariablen kun kan anvendes / tilgås i objekter af den pågældende klasse
- **Type** der fortæller hvilke værdier feltvariablen kan antage
- **Navn**

Objekters opførsel i Java

```
public class Person {  
    private String name;  
    private int age;
```

```
    public Person(String n, int a) {  
        name = n;  
        age = a;  
    }
```

```
    public String getName() {  
        return name;  
    }
```

```
    public int getAge() {  
        return age;  
    }
```

```
    public void setName(String n) {  
        name = n;  
    }
```

```
    public void birthday() {  
        age = age + 1;  
        System.out.println("Happy birthday " + name + "!");  
    }
```

```
}
```

Opførsel beskrives ved hjælp af

Konstruktører

- Skaber og initialiserer et objekt, der tilhører den pågældende klasse

Accessor metoder

- Aflæser feltvariablers værdi og returnerer denne
- Hedder ofte **getXXX**, hvor XXX er den feltvariabel, der aflæses

Mutator metoder

- Ændrer feltvariablers værdi
- Hedder ofte **setXXX**, hvor XXX er den feltvariabel, der ændres
- Kan også have andre navne (fx birthday)

Hoved for konstruktører og metoder

```
public class Person {  
    private String name;  
    private int age;  
  
    public Person(String n, int a) {  
        name = n;  
        age = a;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setName(String n) {  
        name = n;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void birthday() {  
        age = age + 1;  
        System.out.println("Happy birthday " + name + "!");  
    }  
}
```

Access modifier

- Fortæller hvor metoden kan kaldes fra
- Er ofte public, men kan også være private

Returtype

- Fortæller hvilke slags værdier metoden returnerer
- Hvis der ikke returneres noget er returtypen void (tom)
- Konstruktører har ikke en returtype (de kan aldrig returnere noget)

Navn

- Konstruktører har altid samme navn som klassen

Parameterliste

- Angiver "input" til metoden
- Hvis der ikke er parametre, er parentes tom ()

Signatur = metodens navn + parametrene's typer

- Signaturen bestemmes af hovedet
- Returtypen indgår **ikke** i signaturen og det gør parametrene's navne heller ikke

Feltvariabler, konstruktører og metoder

```
public class Person {  
    private String name;  
    private int age;  
  
    public Person(String n, int a)  
    { ... }  
  
    public int getAge()  
    { ... }  
  
    public void birthday()  
    { ... }  
}
```

- **Feltvariabler (attributter)**
 - bestemmer objektets tilstand
 - erklæres altid **private**
 - kan kun tilgås fra klassens egne konstruktører og metoder (vedkommer ikke andre)
- **Konstruktører og metoder**
 - bestemmer objektets opførsel
 - grænseflade til omverdenen
 - erklæres oftest **public**
 - kan kaldes fra objekter af alle klasser

Klasser og typer

- **Enhver klasse bestemmer en type**
 - Når vi erklærer en klasse erklærer vi samtidig en type (med samme navn)
 - F.eks. er String en klasse / type erklæret i Javas Standardbibliotek
- **En objekt type er en type, der er bestemt via en klasse**
 - De mulige værdier i typen er de objekter, der kan skabes (instansieres) af den pågældende klasse
 - Person og String klasserne er eksempler på objekt typer
 - Navne på objekt typer (klasser) skrives med stort begyndelsesbogstav (Person og String)
- **Primitive typer**
 - Har "simple" værdier, der ikke er objekter
 - Eksempler: heltal (int), reelle tal (double) og sandhedsværdier (boolean)
 - Navne på primitive typer skrives med lille begyndelsesbogstav (int, double og boolean)

● Objekters tilstand og opførsel i BlueJ

The screenshot shows the BlueJ IDE interface. In the background, the 'Person' class is visible with a 'new Person(String n, int a)' constructor call. In the foreground, the 'BlueJ: Create Object' dialog is open, showing the constructor signature and the instance name 'person1'. The instance is being created with the name 'Peter' and age 24. Below the dialog, the 'person1 : Person' object is shown in the workspace. The 'Inspector' window is open, displaying the object's state with fields 'private String name' and 'private int age'.

BlueJ: Create Object

```
// Construct a new person with the given name and age.  
// @param n name of the person  
// @param a age of the person  
Person(String n, int a)
```

Name of Instance: **person1**

new Person(**"Peter"** , **24**)

OK Cancel

Person

new Person(String n, int a)

person1 : Person

private String name	"Peter"
private int age	24

Inspect Get Show static fields Close

Initialising virtual machine... Do...

**n personens navn
a personens alder**

**n skal have typen String
a skal have typen int**

**Reference (pegepind)
til det nye objekt**

**Inspector, hvori vi
kan se værdierne af
feltvariablerne**

Lad os lave endnu et Person objekt

BlueJ: Person

Project Edit Tools View Help

New Class...

→

Compile

Person

new Person(String n, int a)

Open Editor

Compile

Inspect

Delete

Convert to Stride

Create Test Class

BlueJ: Create Object

```
// Construct a new person with the given name and age.  
// @param n name of the person  
// @param a age of the person  
Person(String n, int a)
```

Name of Instance: person2

new Person("Anna"
18

person2 : Person

private String name	"Anna"	Inspect
private int age	18	Get

Show static fields

Close

person1: Person

person2: Person

Initialising virtual machine... Do...

person

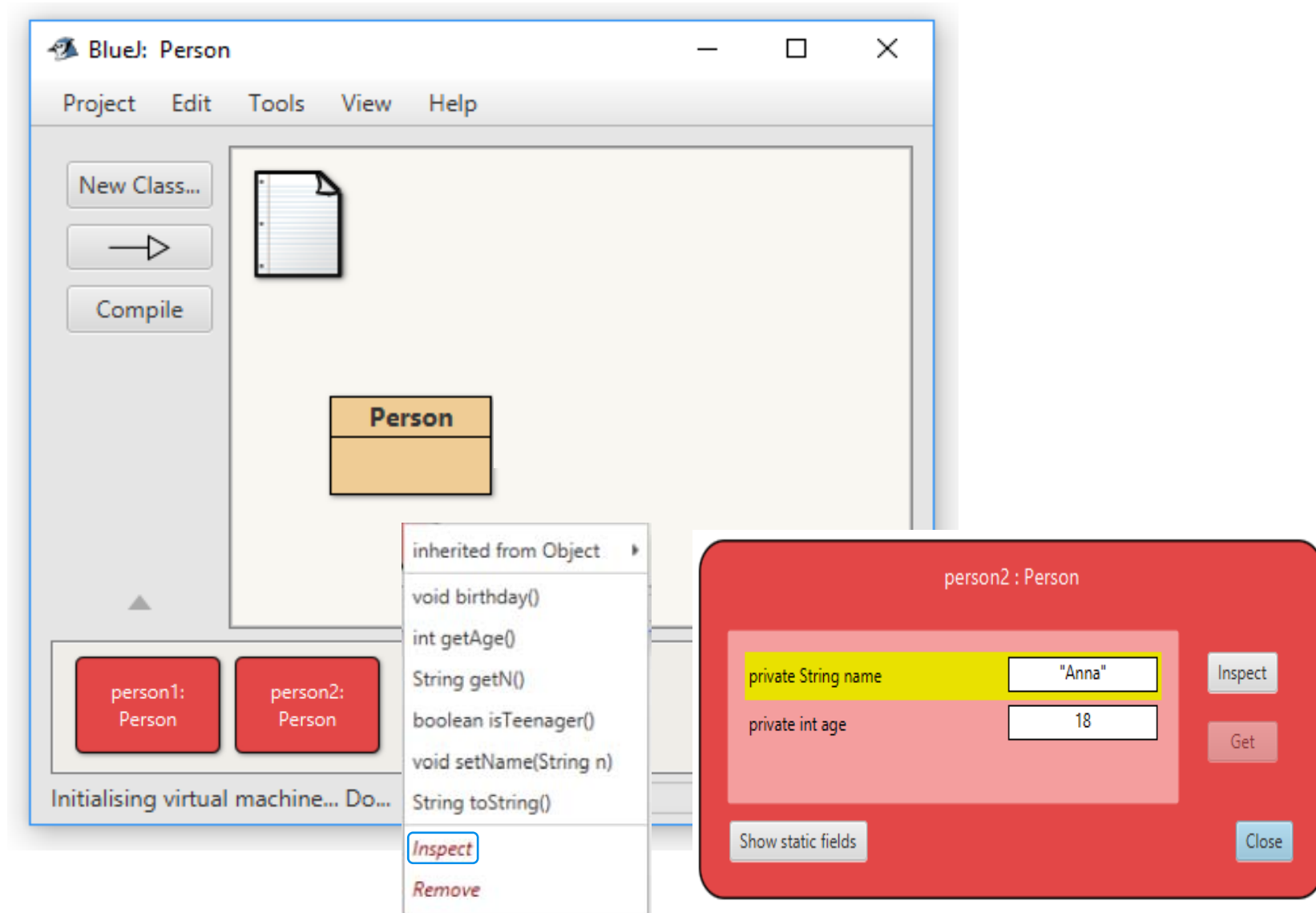
"Peter"	Inspect
24	Get

Show static fields

Close

De to objekter har de samme feltvariabler, men med forskellige værdier

Lad os højreklikke på person2



Kald af metoden getAge (accessor)

The screenshot shows the BlueJ IDE interface. The main window is titled "BlueJ: Person" and contains a menu bar (Project, Edit, Tools, View, Help), buttons for "New Class...", a right arrow, and "Compile". A class diagram shows the "Person" class. Below it, two instances are listed: "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" instance, showing a list of methods. The method "int getAge()" is highlighted. A "BlueJ: Method Result" dialog box is open, showing the result of the "person2.getAge()" call: "int 18". A "person2 : Person" object monitor is also visible, showing the state of the object: "private String name" is "Anna" and "private int age" is 18.

BlueJ: Person

Project Edit Tools View Help

New Class...

→

Compile

Person

person1: Person

person2: Person

Initialising virtual machine... Do...

inherited from Object

- void birthday()
- int getAge()**
- String getN()
- boolean isTeenager()
- void setName(String n)
- String toString()

Inspect

Remove

BlueJ: Method Result

// Returns the person's age.
// @return person's age
int getAge()

person2.getAge() returned:

int 18

Inspect

Get

Close

person2 : Person

private String name "Anna"

private int age 18

Inspect

Get

Show static fields

Close

Kald af metoden setName (mutator)

The screenshot illustrates the process of calling the `setName` method on a `Person` object in the BlueJ IDE.

BlueJ: Person Window:

- Buttons: New Class..., Compile
- Class Diagram: `Person` class box
- Object List: `person1: Person`, `person2: Person`
- Status Bar: Initialising virtual machine... Do...

Method Call Dialog (BlueJ: Method Call):

```
// Changes the name of the person.  
// @param n new name  
void setName(String n)  
  
person2.setName( "Maria" )
```

Object Inspector (person2 : Person):

Field	Value
private String name	"Maria"
private int age	18

The `setName` method is highlighted in the object list, and the `name` field in the object inspector is circled, showing the value "Maria".

Kald af metoden birthday (mutator)

The screenshot shows the BlueJ IDE interface. The main window is titled "BlueJ: Person" and contains a menu bar (Project, Edit, Tools, View, Help), buttons for "New Class...", a right arrow, and "Compile". A class diagram shows the "Person" class. Below it, two instances are listed: "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" instance, showing the method "void birthday()" highlighted. The menu also lists other methods: "int getAge()", "String getN()", "boolean isTeenager()", "void setName(String n)", and "String toString()". At the bottom of the menu are "Inspect" and "Remove" options. A "Terminal Window - Person" is open, displaying "Happy birthday Maria!". A "person2 : Person" object monitor is shown, displaying fields "private String name" (value "Maria") and "private int age" (value 19, circled in blue). Buttons for "Inspect", "Get", "Show static fields", and "Close" are visible on the monitor.

BlueJ: Person

Project Edit Tools View Help

New Class...

→

Compile

Person

person1: Person

person2: Person

Initialising virtual machine... Do...

inherited from Object

void birthday()

int getAge()

String getN()

boolean isTeenager()

void setName(String n)

String toString()

Inspect

Remove

BlueJ: Terminal Window - Person

Options

Happy birthday Maria!

Can only enter input while your programmi

person2 : Person

private String name "Maria"

private int age 19

Inspect

Get

Show static fields

Close

Kald af metoden isTeenager (accessor)

The screenshot shows the BlueJ IDE interface. The main window is titled "BlueJ: Person" and displays the class hierarchy for the "Person" class. A context menu is open over the "Person" class, showing a list of methods. The method "boolean isTeenager()" is highlighted, and the "Remove" option is selected. A "Method Result" dialog box is open, showing the result of the "person2.isTeenager()" method call, which is "true". A "Person" object window is also visible, showing the object's state with "name" set to "Maria" and "age" set to "19".

BlueJ: Person

Project Edit Tools View Help

New Class...

→

Compile

Person

person1: Person

person2: Person

Initialising virtual machine... Do...

inherited from Object

- void birthday()
- int getAge()
- String getN()
- boolean isTeenager()**
- void setName(String n)
- String toString()

Inspect

Remove

BlueJ: Method Result

```
// Is the person a teenager?  
// @return true is the person is a teenager, otherwise false  
boolean isTeenager()
```

person2.isTeenager() returned:

boolean true

Inspect

Get

Close

person2 : Person

private String name "Maria"

private int age 19

Show static fields

Inspect

Get

Close

Lad os kalde metoden birthday igen

The screenshot shows the BlueJ IDE interface. The main window is titled "BlueJ: Person" and contains a menu bar (Project, Edit, Tools, View, Help) and a toolbar with buttons for "New Class...", a right arrow, and "Compile". The central workspace displays a class diagram for the "Person" class. Below the workspace, there are two instance monitors: "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" instance, showing a list of methods inherited from Object. The "void birthday()" method is highlighted. To the right, a "Terminal Window - Person" is open, displaying the output "Happy birthday Maria!" twice. At the bottom right, a detailed monitor for "person2 : Person" is shown, displaying the private fields "name" (with value "Maria") and "age" (with value 20, which is circled in blue). The monitor also includes buttons for "Inspect", "Get", "Show static fields", and "Close".

BlueJ: Person

Project Edit Tools View Help

New Class...

→

Compile

Person

person1: Person

person2: Person

Initialising virtual machine... Do...

inherited from Object

- void birthday()
- int getAge()
- String getN()
- boolean isTeenager()
- void setName(String n)
- String toString()

Inspect

Remove

BlueJ: Terminal Window - Person

Options

Happy birthday Maria!

Happy birthday Maria!

Can only enter input while your programmi

person2 : Person

private String name "Maria" Inspect

private int age 20 Get

Show static fields Close

Lad os kalde metoden isTeenager igen

The screenshot shows the BlueJ IDE interface. The main window is titled "BlueJ: Person" and contains a menu bar (Project, Edit, Tools, View, Help) and a toolbar with buttons for "New Class...", a right arrow, and "Compile". Below the toolbar is a workspace area with a "Person" class icon. At the bottom, there are two red buttons labeled "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" button, listing methods: "inherited from Object", "void birthday()", "int getAge()", "String getN()", "boolean isTeenager()", "void setName(String n)", and "String toString()". The "isTeenager()" method is highlighted. To the right, a "BlueJ: Method Result" window shows the code for the "isTeenager()" method and the result: "person2.isTeenager() returned: boolean false". Below this, a "person2 : Person" window shows the object's state: "private String name" with value "Maria" and "private int age" with value 20. Both the "isTeenager()" method result and the "person2 : Person" window have "Inspect" and "Get" buttons. The "person2 : Person" window also has a "Show static fields" button and a "Close" button.

BlueJ: Person

Project Edit Tools View Help

New Class...

→

Compile

Person

person1: Person

person2: Person

Initialising virtual machine... Do...

inherited from Object

void birthday()

int getAge()

String getN()

boolean isTeenager()

void setName(String n)

String toString()

Inspect

Remove

BlueJ: Method Result

```
// Is the person a teenager?  
// @return true is the person is a teenager, otherwise false  
boolean isTeenager()
```

person2.isTeenager() returned:

boolean false

Inspect

Get

Close

person2 : Person

private String name "Maria"

private int age 20

Show static fields

Inspect

Get

Close

Java code for Person klassen

The image displays the BlueJ IDE interface. On the left, the 'Project' pane shows a class named 'Person'. A context menu is open over the 'Person' class, listing actions: 'new Person(String n, int a)', 'Open Editor' (highlighted), 'Compile', 'Inspect', 'Delete', 'Convert to Stride', and 'Create Test Class'. Below the project pane, two red buttons labeled 'person1: Person' and 'person2: Person' are visible, along with the text 'Initialising virtual machine... Do...'. On the right, the 'Person - Person' window shows the source code of the 'Person' class. The code includes private fields for 'name' and 'age', a constructor, and methods for getting and setting name and age.

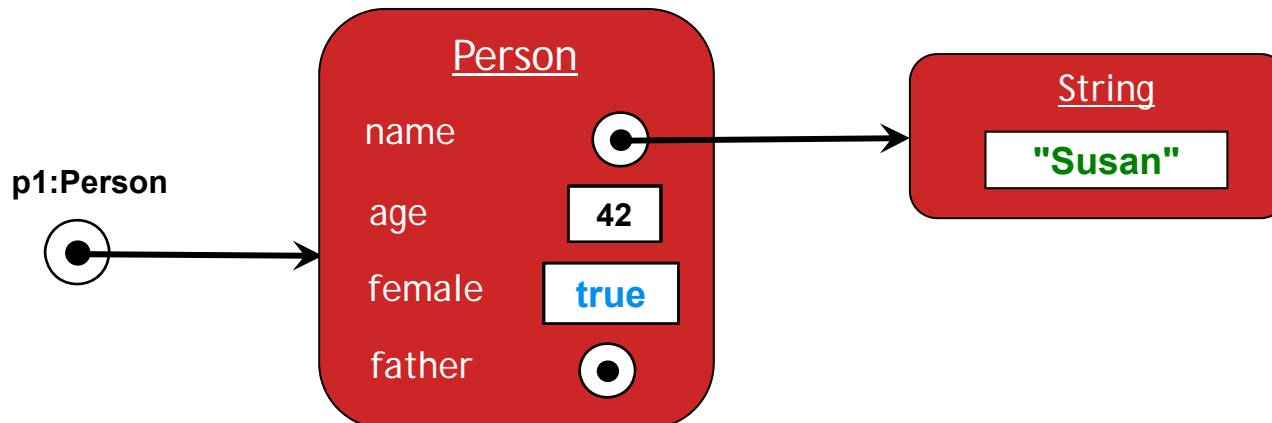
```
public class Person {  
    private String name;  
    private int age;  
  
    /**  
     * Construct a new person with the given name and age.  
     * @param n   name of the person  
     * @param a   age of the person  
     */  
    public Person(String n, int a) {  
        name = n;  
        age = a;  
    }  
  
    /**  
     * Returns the name of the person.  
     * @return    name of the person  
     */  
    public String getName() {  
        return name;  
    }  
  
    /**  
     * Changes the name of the person.  
     * @param n   new name  
     */  
    public void setName(String n) {  
        name = n;  
    }  
  
    /**  
     * Returns the person's age.  
     * @return    person's age  
     */  
    public int getAge() {  
        return age;  
    }  
}
```

● Skabelse af objekter (new operator)

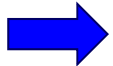
```
public class Person {  
    private String name;  
    private int age;  
    private boolean female;  
    private Person father;  
  
    public Person(String n, int a, boolean sex) {  
        name = n;  
        age = a;  
        female = sex;  
    }  
    ...  
}
```

Nu med 4 feltvariabler

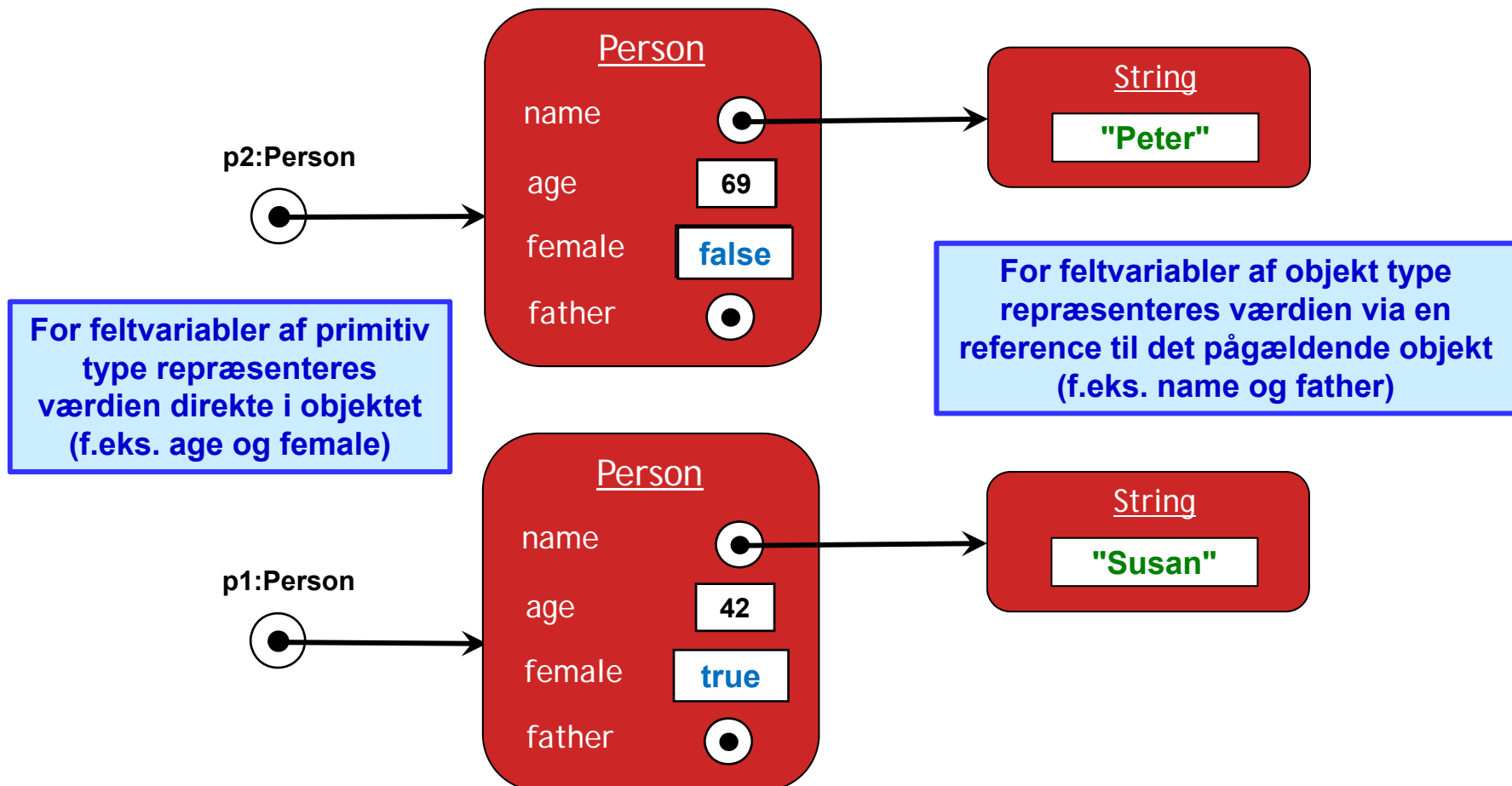
private Person p1;
p1 = new Person("Susan", 42, true);



Endnu et objekt



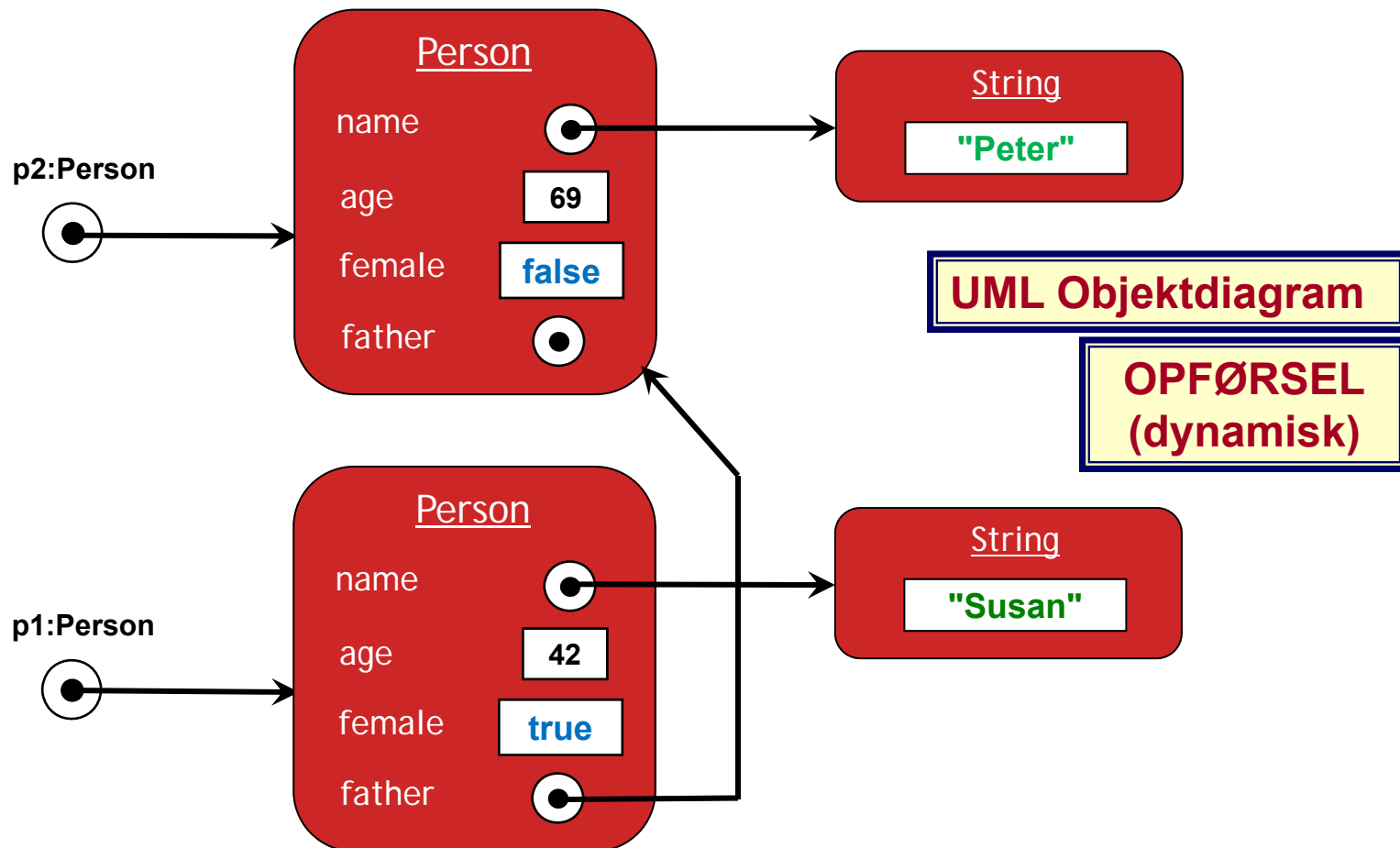
```
private Person p2;  
p2 = new Person("Peter", 69, false);
```



Metoden setFather (mutator metode)

```
p1.setFather(p2);
```

```
public void setFather(Person p) {  
    father = p;  
}
```



Metoden birthday (mutator metode)

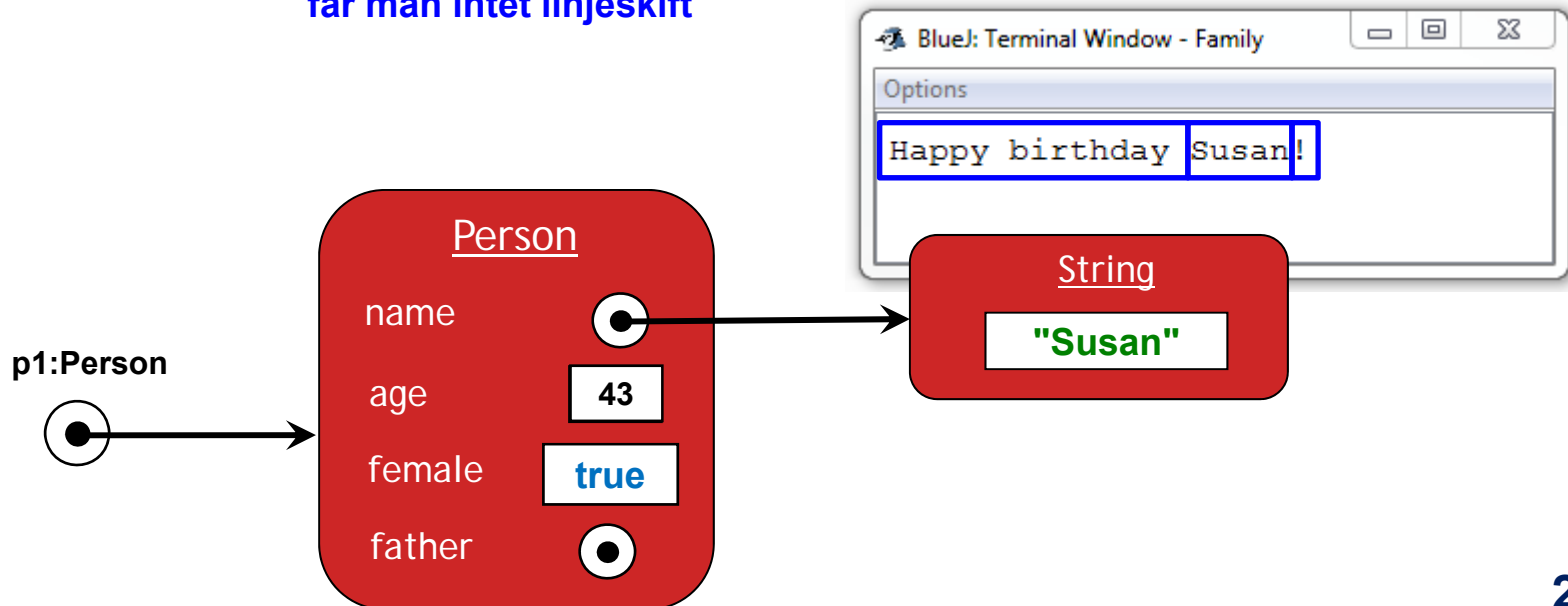
```
p1.birthday();
```

```
public void birthday() {  
    age = age + 1;  
    System.out.println("Happy birthday " + name + "!!");  
}
```

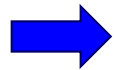
Kald af metode i Java's klassebibliotek
(udskriver linje på BueJ's terminal)

Hvis man kun skriver print,
får man intet linjeskift

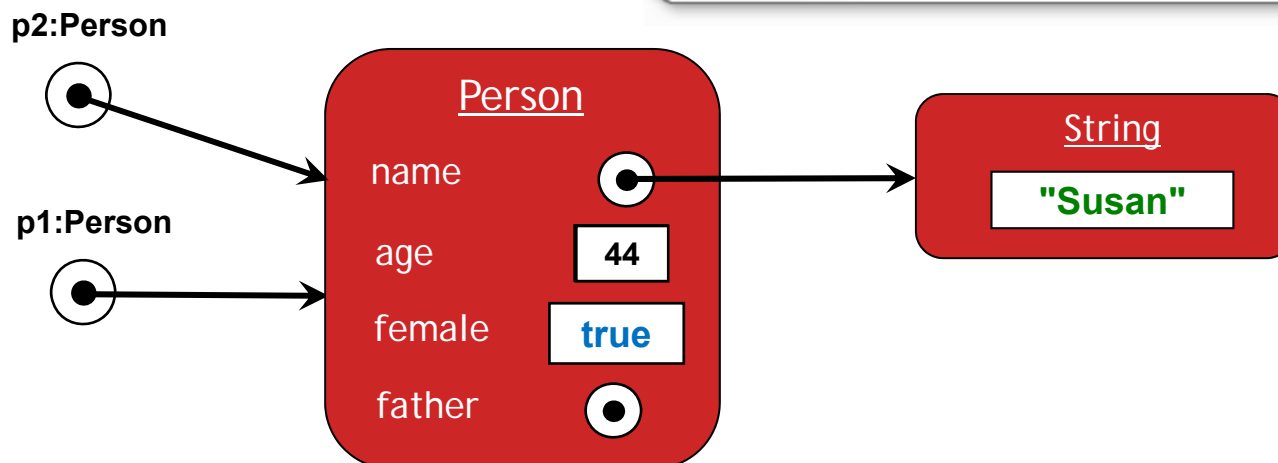
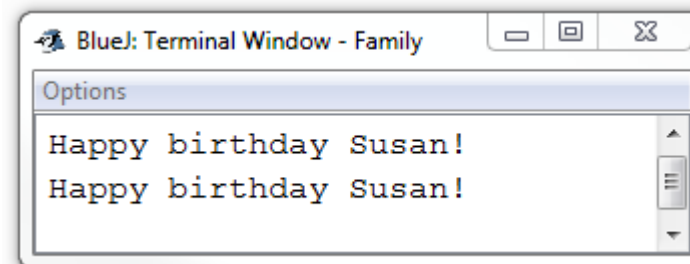
Sammensætning af
tekststreng
(konkatenering)



En person – to referencer



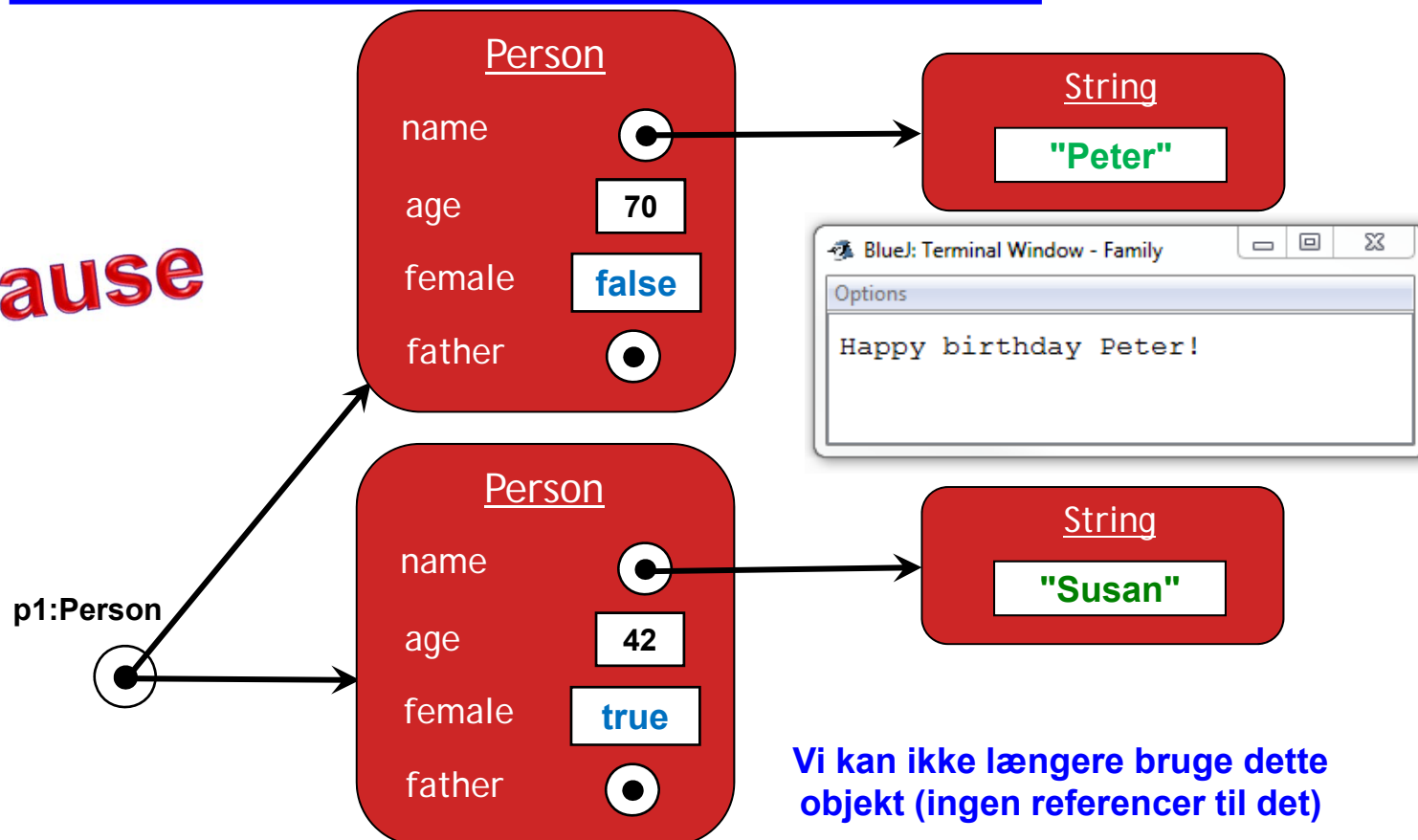
```
private Person p1, p2;  
p1 = new Person("Susan", 42, true);  
p2 = p1;  
p1.birthday();  
p2.birthday();
```



To personer – én reference

```
private Person p1;  
p1 = new Person("Susan", 42, true);  
p1 = new Person("Peter", 69, false);  
p1.birthday();
```

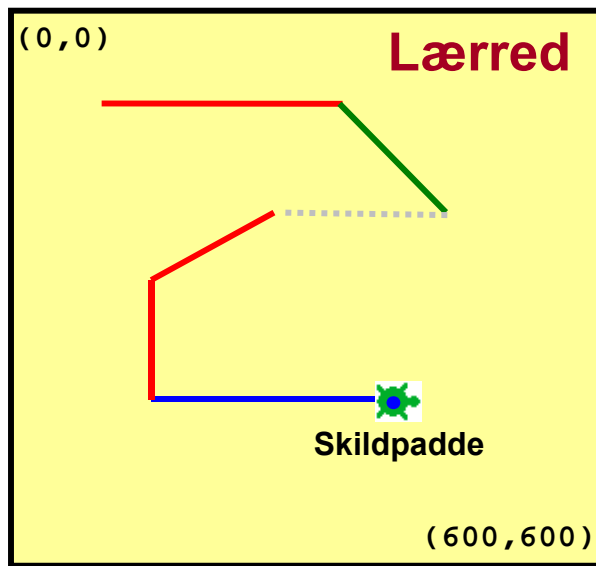
Pause



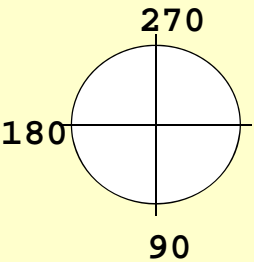
Vi kan ikke længere bruge dette objekt (ingen referencer til det)


● Iteration, selektion og parametrisering

- Skildpadden kan dirigeres rundt på et lærred
 - Den tegner en streg, hvor den kommer frem
 - Stregens farve kan skifte undervejs
 - Pennen kan trækkes op, så der ikke kommer en streg



Skildpaddens tilstand

- Position: (x,y)
- Vinkel:

A circular diagram with a vertical line and a horizontal line intersecting at the center. The angles are labeled: 0 (360) at the right, 90 at the bottom, 180 at the left, and 270 at the top.
- Farve: 

A horizontal bar divided into five colored segments: black, blue, green, white, and red. An ellipsis "..." is placed between the green and white segments.
- Pen status: up/down

Eksempel på tilstand

- $((450, 450), 0, \text{"blue"}, \text{down})$

Programmering af skildpadden

- **Vi antager, at Turtle klassen stiller en række simple metoder (tegneoperationer) til rådighed**
 - Flyt, drej, pen op/ned, ...
- **Dem vil vi supplere med nogle mere komplekse metoder**
 - Kvadrat, polygon, cirkel, ...

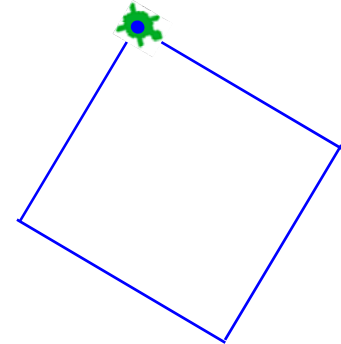
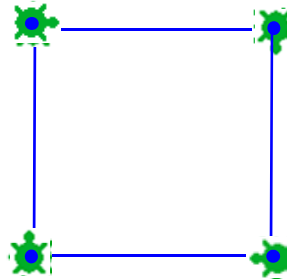
Turtle
<pre>move(double distance) turn(double degrees) penUp() penDown() ... square(double size) polygon(int n, double size) circle(double radius) ...</pre>

- **Typen double repræsenterer reelle tal**
 - Alle steder, hvor I skal bruge en double kan I i stedet bruge en int
 - Det omvendte gælder ikke
 - Hvis I vil indtaste et reelt tal indsættes et punktum
 - 360.0 er af typen double, mens 360 er af typen int

Kvadrat

- **Vi vil skrive noget kode, der kan tegne et kvadrat**
 - Efter udførelsen af koden skal skildpadden være tilbage i startposition og startvinkel
 - Koden skal virke for alle startpositioner og alle startvinkler

```
// Tegn kvadrat  
move(100); turn(90);  
move(100); turn(90);  
move(100); turn(90);  
move(100); turn(90);
```



- **Vi har lavet en algoritme, der beskriver, hvordan man tegner et kvadrat**
 - Algoritmen består af to operationer (move og turn) som hver gentages fire gange

Gentagelser af kode

```
// Tegn kvadrat  
move(100); turn(90);  
move(100); turn(90);  
move(100); turn(90);  
move(100); turn(90);
```

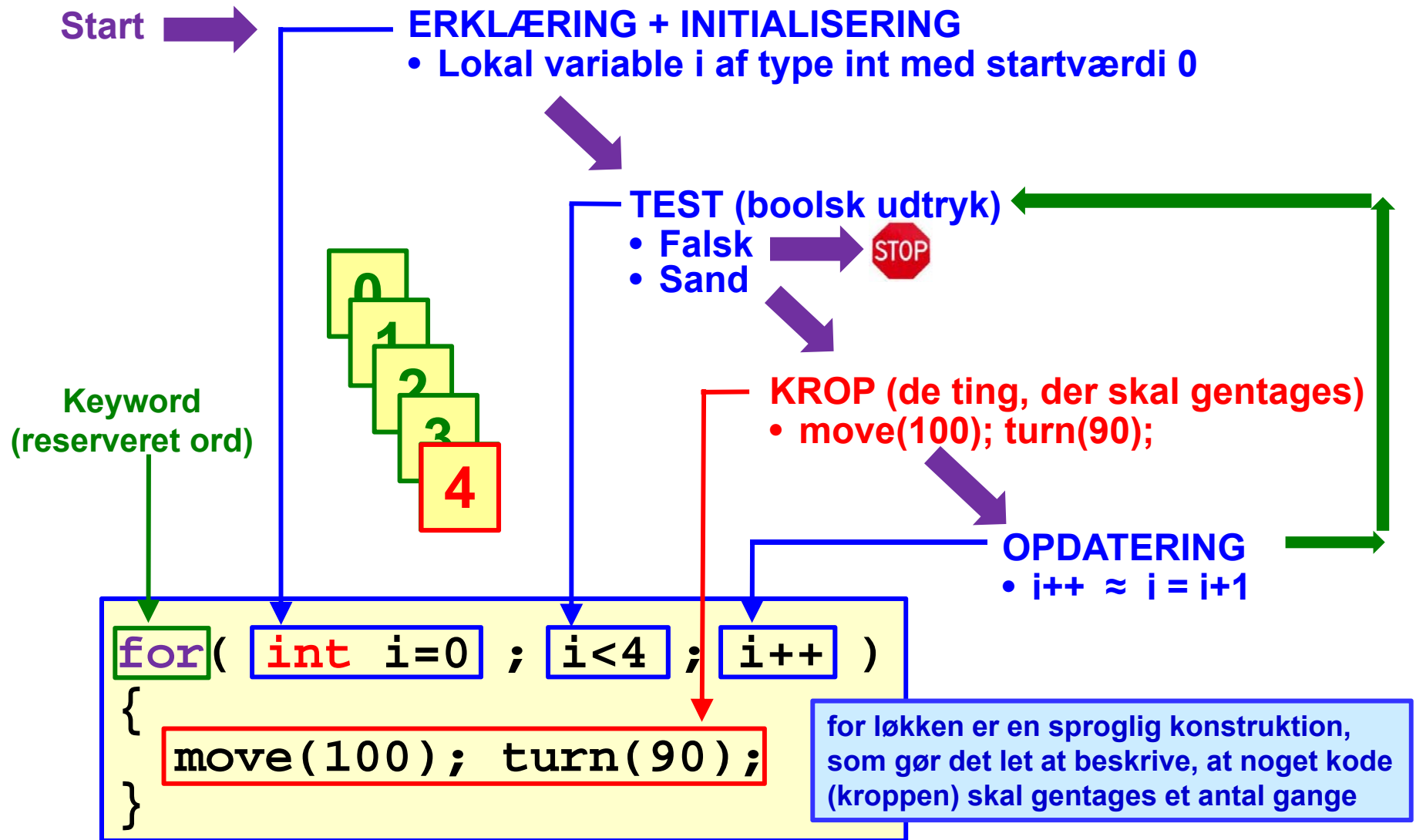
```
// Tegn kvadrat  
gentag 4 gange {  
    move(100);  
    turn(90);  
}
```

```
// Tegn tolvkant  
move(100); turn(30);  
move(100); turn(30);  
move(100); turn(30);  
move(100); turn(30);  
move(100); turn(30);  
move(100); turn(30);  
move(100); turn(30);  
move(100); turn(30);  
...  
move(100); turn(30);
```

```
// Tegn tolvkant  
gentag 12 gange {  
    move(100);  
    turn(30);  
}
```

- Hurtigere at skrive
- Nemmere at læse og forstå
- Lettere at vedligeholde (rette i)

for løkke i Java



Metode: kvadrat med længde 100

```
public class Turtle {  
    ...  
    // Tegn kvadrat med sidelængde 100  
    public void square100() {  
        for( int i = 0; i < 4; i++ ) {  
            move( 100 );  
            turn( 90 );  
        }  
    }  
    ...  
}
```

Længden 100 indsat direkte i metoden

I stedet kunne vi angive længden ved hjælp af en parameter

Det ville være smartere at lave en metode, der kan tegne kvadrater af vilkårlig størrelse

Metode: kvadrat med vilkårlig størrelse

```
public class Turtle {  
    ...  
    // Tegn kvadrat med sidelængde size  
    public void square(double size) {  
        for( int i = 0; i < 4; i++ ) {  
            move(size);  
            turn(90);  
        }  
    }  
    ...  
}
```

Parameter i square

Argument til move

Det ville være smartere at lave en metode,
der kan tegne regulære figurer med et
vilkårligt antal sider (polygoner)

Metode: polygon med vilkårligt antal sider

```
public class Turtle {  
    ...  
    // Tegn regulær n-kant med sidelængde size  
    public void polygon(int n, double size) {  
        for( int i = 0; i < n; i++ ) {  
            move( size );  
            turn( 360.0 / n );  
        }  
    }  
}
```

To parametre

- Den første angiver antallet af sider
- Den anden angiver længden af siderne

Reelt tal (double)

- For at undgå nedrundingsfejl
- Division af to heltal giver et nyt heltal
- F.eks. evaluerer $360 / 7$ til heltallet 51
- Dvs. at man kun drejer $7 * 51 = 357$ grader
- Skildpadden kommer ikke helt tilbage til startposition og startvinkel

Hvad sker der, hvis n er negativ eller 0?

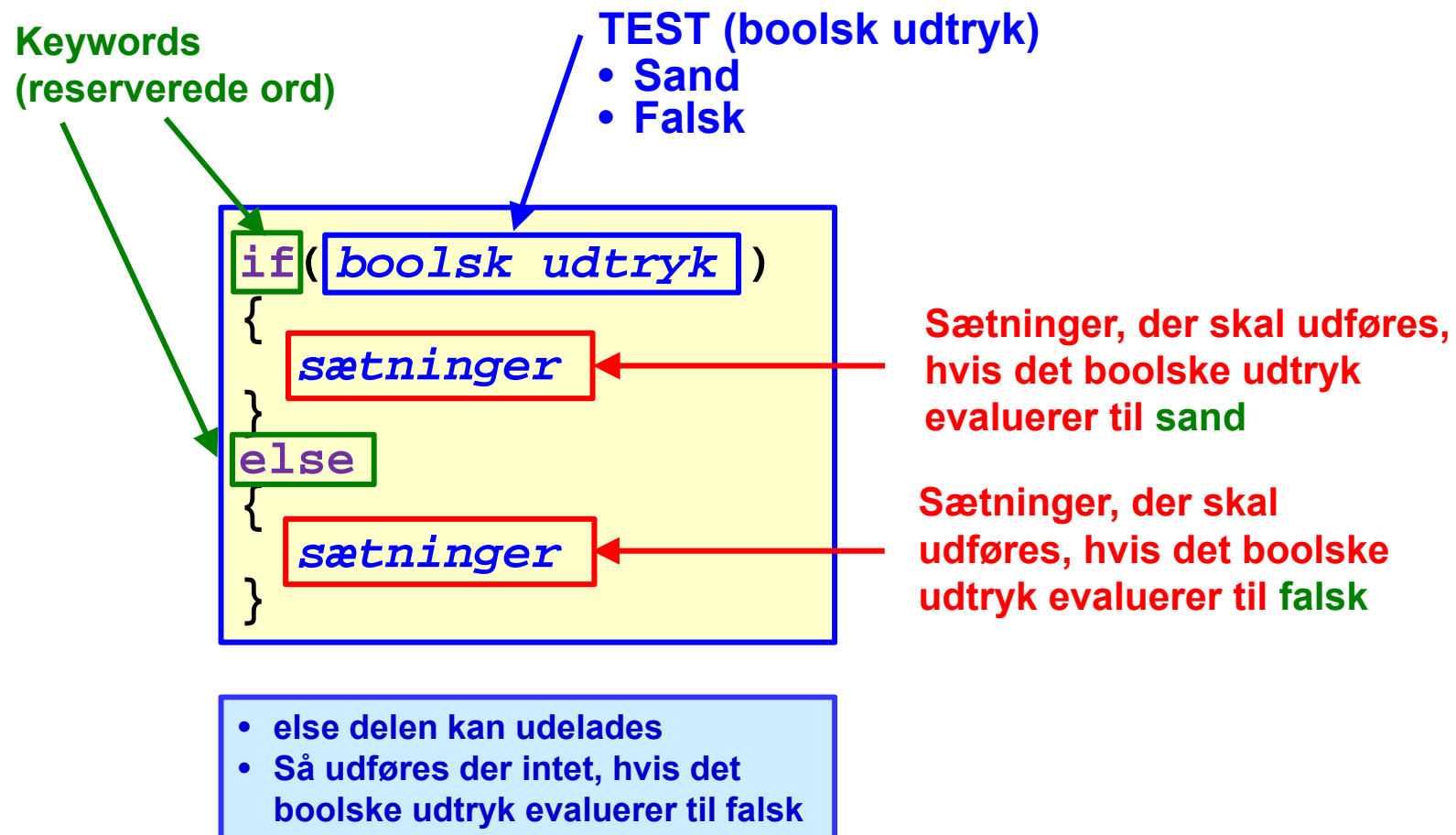
Hvad sker der, hvis n er 1?

Hvad sker der, hvis n er 2?

Om lidt vil vi lave en version, der tjekker, at parameteren n har en fornuftig værdi

Selektering (valg) mellem forskellige kode

- Ved hjælp af en if sætning kan man sikre, at noget kode kun udføres, når bestemte betingelser er opfyldt



Færdig polygon metode

```
public class Turtle {  
    ...  
    // Tegn regulær n-kant med sidelængde size  
    public void polygon( int n, double size ) {  
Test → if( n >= 3 ) {  
        for( int i = 0; i < n; i++ ) {  
            move( size );  
            turn( 360.0 / n );  
        }  
    }  
    else {  
        System.out.println("n must be >= 3");  
    }  
}  
}
```

Tegn polygon med n sider

Udskriv fejlmeddelelse på terminalen

Bør vi også tjekke værdien af size?

Generel metode → specifikke metoder

Vi kan benytte den generelle metode `polygon` til at konstruere mere specifikke metoder, der kan tegne kvadrater og cirkler.

```
public class Turtle {  
    ...  
    // Tegn regulær n-kant med sidelængde size  
    public void polygon(int n, double size) {  
        ...  
    }  
  
    // Tegn kvadrat med sidelængde size  
    public void square(double size) {  
        polygon(4, size);  
    }  
  
    // Tegn cirkel med den angiven radius  
    public void circle(double radius) {  
        polygon(100, 2 * radius * Math.PI / 100);  
    }  
}
```

Konstanten π (fra klassen `Math`)

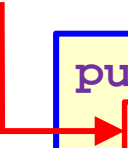
Vigtige principper for god programmering

- **Det kan betale sig at lave gode generelle metoder, som kan genbruges i mange situationer**
 - Parametrisering er nøglen hertil
 - Det er svært at "opfinde" gode generelle metoder, dvs. at gå fra det konkrete til det generelle – men forsøg!
- **Skeln mellem anvendelse og implementation**
 - Når man anvender en metode, er det vigtigt at forstå, hvad operationen gør
 - Når man implementerer en metode, skal man tage stilling til, hvordan den skal gøre det
 - I skal også skelne – selv om I både er anvender og implementør

● Forskellige slags variabler

- **Klasser har feltvariabler**

- Tilhører objektet
- Lever og dør med dette
- Bruges til værdier der skal gemmes mellem metodekald
- **private** som access modifier



```
public class Turtle {  
    private String color;  
    ...  
    public void polygon( int n, double size ) {  
        double angle = 360.0 / n;  
        for( int i = 0; i < n; i++ ) {  
            move(size);  
            turn(angle);  
        }  
    }  
    ...  
}
```

Forskellige slags variabler (fortsat)

- **Metoder og konstruktører har lokale variabler**
 - Tilhører metoden/konstruktøren
 - Lever og dør med det enkelte kald af metoden/konstruktøren
 - Kan ikke bruges til at gemme resultater mellem metodekald
 - Ingen access modifier (kan aldrig tilgås udenfor metoden/konstruktøren)

**Demo af
Date klassen
i Blue J**

```
public class Turtle {  
    private String color;  
    ...  
    public void polygon(int n, double size) {  
        double angle = 360.0 / n;  
        for(int i = 0; i < n; i++) {  
            move(size);  
            turn(angle);  
        }  
    }  
    ...  
}
```

Parametre

Hjælpevariabel

Kontrolvariabel

- Det er vigtigt at skelne mellem feltvariabler og lokale variabler
- De tjener to helt forskellige formål

● Opsummering

- **Objekters tilstand og opførsel**
 - Java og BlueJ
- **Skabelse af objekter (via new-operatoren)**
 - Objekt referencer og objektdiagrammer
- **Iteration (gentagelser) og selektering (valg)**
 - Java's for løkke
 - Java's if sætning
- **Parametrisering**
 - Lav gode generelle metoder
 - Skeln mellem anvendelse og implementation
- **Forskellige slags variabler**
 - Feltvariabler
 - Lokale variabler

Objektorienteret programmering

- I objektorienteret programmering opfattes et program som en **model**, der beskriver (simulerer) opførslen af en del af verden
 - **Klasser** modellerer **begreber** (f.eks. Student)
 - **Objekter** der er **instanser** af klasser (f.eks. Peter Hansen, Anna Petersen,)
 - Det man beskriver kan være noget der eksisterer eller noget, som man gerne vil bygge

Ovenstående definitioner stammer fra sproget Simula 67 og er dermed 50 år gamle

Kristen Nygaard (1926-2002)

- grundlægger af objektorienteret modellering og programmering (sammen med Ole-Johan Dahl)
- gæsteprofessor på Aarhus Universitet i en årrække, hvor han havde stor betydning for opbygningen af datalogi
- Nygaard-bygningen i IT-parken er opkaldt efter Kristen



Studiestartsprøve

- **Gælder alle nye bachelorstuderende**
 - Prøvens hovedformål er at identificere de studerende, der ikke har påbegyndt studiet, så de kan udmeldes inden det officielle sommeroptag opgøres
- **I begyndelsen af september vil I modtage en mail på jeres e-mailadresse**
 - Mailen indeholder et link til et spørgeskema, der handler om studievalg og studiestart
 - Det er **obligatorisk** at svare og på den måde vise, at I er studieaktive
 - Hvis I ikke svarer (inden for få dage) bliver I **automatisk frmeldt** jeres studie

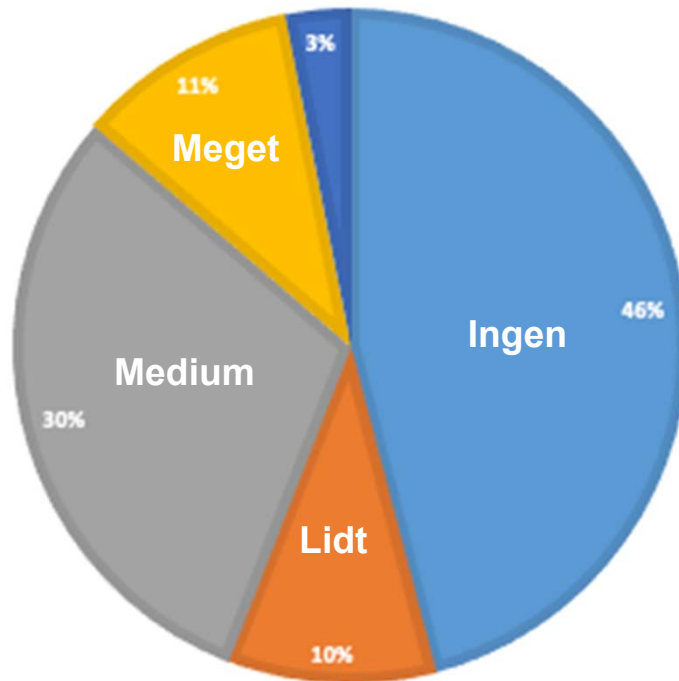
Husk at forberede jer til øvelserne

- **Ved øvelserne i Uge 2 skal I beskæftige jer med opgaverne i kapitel 2 og 3 i BlueJ bogen**
 - Opgaverne skal løses, mens I læser kapitlerne
 - Ved øvelserne vil instruktorerne så tage fat i de opgaver, hvor I har problemer
 - Der vil selvfølgelig også være muligt at få hjælp til tvivlsspørgsmål i kapitlernes tekst og i de tilhørende videonoter (som I også skal se, mens i læser kapitlerne)
- **Derudover skal I (ved anden øvelsesgang) arbejde videre med raflebæger projektet**
 - Nu skal I lave nogle metoder til aftenstning af raflebægeret
 - Derudover skal I generalisere løsningen, således at en terning kan have et vilkårligt antal sider (større end eller lig med 2)

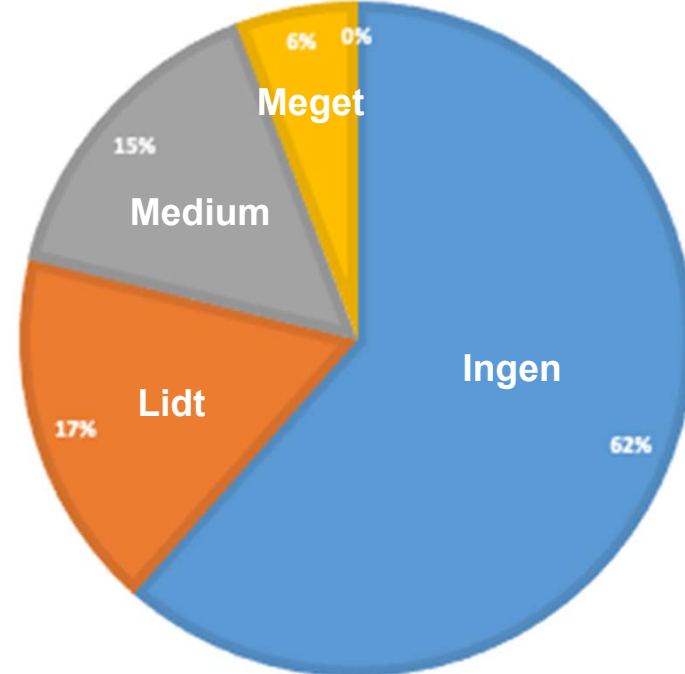
- Husk at aflevere Raflebæger 1 og Quiz 1 inden mandag kl. 13.00 (lørdag kl. 24.00 for IT 'ere)

Programmeringserfaring

Datalogi



IT produktudvikling



- Ingen programmeringserfaring. (brug af Excel eller HTML tæller ikke)
- 10-50 linjers programmer i rigtige sprog, scripting sprog, Scratch, AppMaker eller lignende
- 50-250 linjers programmer i generelle sprog som Java, C, C#, Pascal, BASIC, Python eller Javascript
- Mere en 250 linjer eller mere end 100 linjer i flere forskellige sprog

Hvis I har tid til overs

- **Bruge mere tid på de andre kurser**
- **Begynde på afleveringsopgaverne til de kommende uger**
 - De ligger parat til jer på kursets Blackboard sider
- **På websiderne Projekt Euler, CodingBats og Kattis findes en masse opgaver, hvor I kan øve jer i Java programmering**
 - Links under uge 3 på ugeoversigten Uge 1-7
- **Tilmelde sig instituttets præ-talentforløbet**
 - Tilbud til studerende, der har overskud til at lave lidt ekstra udover de normale kurser
 - Her i efteråret tilbydes et 5 ECTS kursus med nogle spændende foredrag og opgaver
 - Man kan følge hele kurset eller dele af det
- **Ved starten af 2. år, kan man søge om at blive optaget på det egentlige talentforløb**
 - Her kan man kun blive optaget, hvis man har 10 i snit på sine 1. års kurser
 - Det vigtigste er altså at gøre det godt på kurserne
 - Man kan sagtens blive optaget på talentforløbet uden at have deltaget i præ-talentforløbet
 - Mere information på cs.au.dk/talent og ved forelæsningen den 10. september

Afspritning

- **Afspritnings-ansvarlige**

- På hvert øvelseshold er der udpeget 2-3 studerende, som er afspritnings-ansvarlige
- Hold 1 skynder sig at vælge 2-3 stykker

- **Afspritnings-instruks**

- Start med at desinficere jeres hænder, før I rører ved sprayflasken
- Desinficer borde og stole (ikke stofoverflader)
- Husk alle berøringspunkter (bordkant, underside, armlæn mv.)
- Brug sprayflaske til overflader, som skal efterlades fugtig, men ikke våd

Så er vi klar til at forlade lokalet

- **Bliv siddende indtil I får besked på andet**
- **I Auditorium E går man ud af døren til venstre for tavlerne**
 - Bliv siddende indtil jeg har fået den åbnet og sikret
 - Vi starter med den side af auditoriet, der er nærmest døren
 - Rækkerne tømmes nede fra og op
- **I Peter Bøgh-Andersen går man ud af de øverste døre**
 - Rækkerne tømmes oppe fra og ned
 - Brug den dør der er nærmest ved jer
- **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

