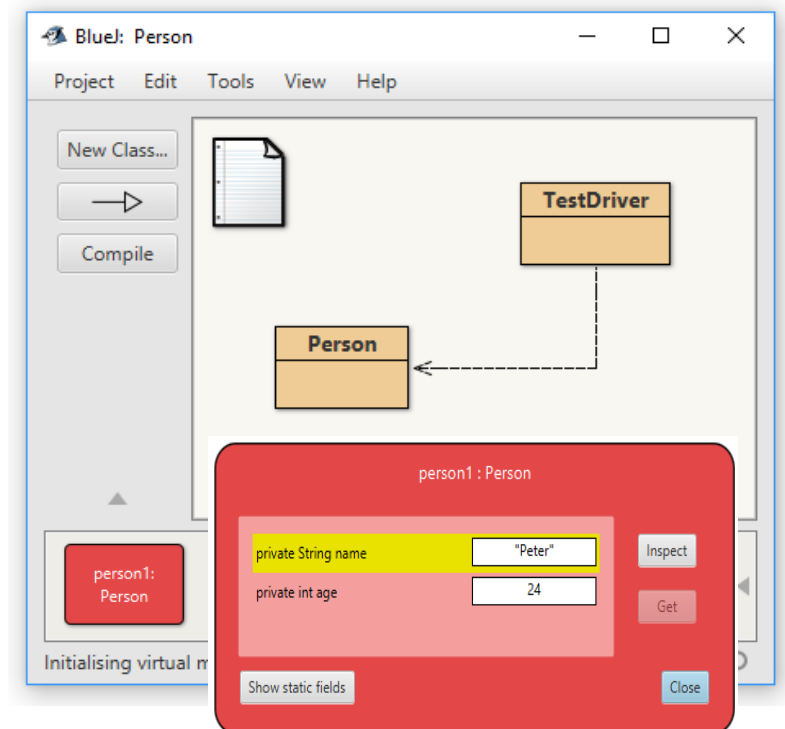


Forelæsning Uge 1 – Torsdag

- **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



● Objekters tilstand og opførsel i Java

- **Tilstand**

- Et objekts **tilstand** er defineret ved et sæt af feltvariabler (fields)
- Feltvariablerne er fastlagt i klassens erklæring (beskrivelse)
- **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 (beskrivelse)
- **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 int getAge() {  
        return age;  
    }  
  
    public void setName(String n) {  
        name = n;  
    }  
  
    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 + parametrenes typer

- Signaturen bestemmes af hovedet
- Returtypen indgår ikke i signaturen og det gør parametrenes 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 (fields)**
 - 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 referencer (pegepinde) til de objekter, der kan skabes (instancieres) 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)
- **En primitiv type er en type med "simple" værdier (der ikke er objekter)**
 - Heltal (int), reelle tal (double) og sandhedsværdier (boolean) er eksempler på primitive typer
 - Navne på primitive typer skrives med lille begyndelsesbogstav (int, double og boolean)

● Objekters tilstand og opførsel i BlueJ

The screenshot shows the BlueJ IDE with a 'Person' class selected. A 'Create Object' dialog box is open, showing the constructor `Person(String n, int a)`. The 'Name of Instance' is set to `person1`, and the constructor arguments are `"Peter"` and `24`. Below the dialog, a 'Person' class box is visible with a 'new Person(String n, int a)' button. At the bottom, an 'Inspector' window for `person1 : Person` shows the object's state: `private String name` with value `"Peter"` and `private int age` with value `24`. Red arrows point from yellow text boxes to specific parts of the interface:

- From `n` and `a` in the constructor signature to the `n` and `a` parameters in the constructor call.
- From `n` in the constructor signature to the `String` type in the constructor call.
- From `a` in the constructor signature to the `int` type in the constructor call.
- From the `person1` instance name to the 'Name of Instance' field.
- From the `new Person(String n, int a)` button to the 'Inspector' window.

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	<code>"Peter"</code>	Inspect
private int age	<code>24</code>	Get

Show static fields Close

Inspector, hvori vi kan se værdierne af feltvariablerne

Lad os lave endnu et Person objekt

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

new Person(String n, int a)

- Open Editor
- Compile
- Inspect
- Delete
- Convert to Stride
- Create Test Class

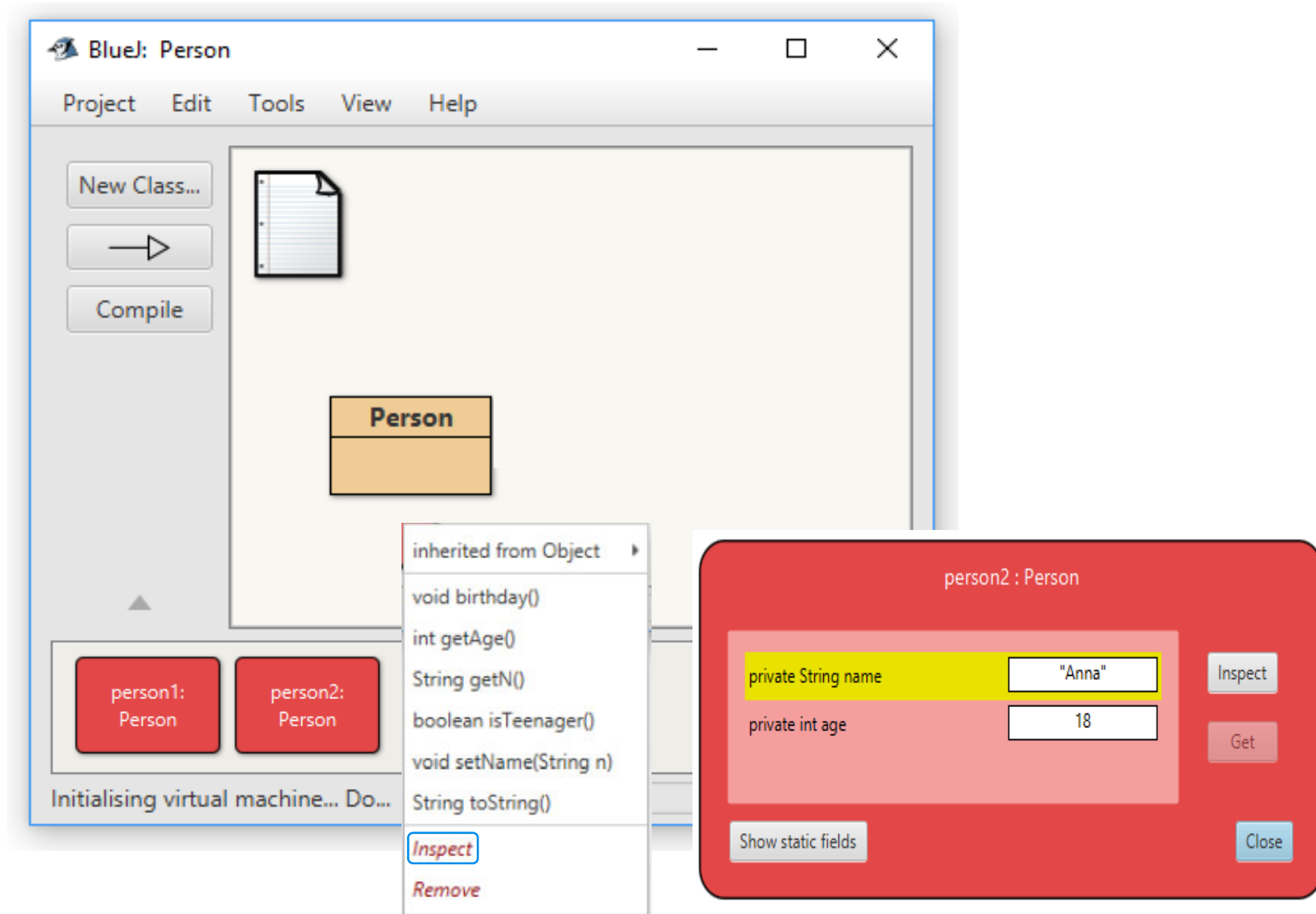
person1: Person

person2: Person

Initialising virtual machine... Do...

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) and a toolbar with buttons for "New Class...", a right-pointing arrow, and "Compile". In the center, there is a class diagram showing a class named "Person". Below the diagram, there are two instances of the "Person" class, labeled "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" instance, showing a list of methods: "inherited from Object", "void birthday()", "int getAge()", "String getN()", "boolean isTeenager()", "void setName(String n)", and "String toString()". The "int getAge()" method is highlighted. At the bottom of the menu, there are buttons for "Inspect" and "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 removing the `setName` method from the `Person` class in the BlueJ IDE.

BlueJ: Person Window: The main window shows the `Person` class. A context menu is open over the class, listing methods. The `void setName(String n)` method is highlighted with a blue box. The menu also shows `Inspect` and `Remove` options.

BlueJ: Method Call Window: A dialog box titled "BlueJ: Method Call" is open. It displays the following code:

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


Below the code, the method is being called: `person2.setName("Maria")`. The value `"Maria"` is entered in a text field. The window has `OK` and `Cancel` buttons.

Person Object Inspector: A red window titled `person2 : Person` shows the object's state. It lists private fields: `private String name` (value: `"Maria"`) and `private int age` (value: `18`). The `name` field is highlighted with a yellow background. Buttons for `Inspect`, `Get`, `Show static fields`, and `Close` are visible.

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 the diagram are two instances: "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" instance, showing a list of methods inherited from "Object". The method "void birthday()" is highlighted. At the bottom of the menu are the options "Inspect" and "Remove".

A "Terminal Window - Person" is open, displaying the output "Happy birthday Maria!".

A "person2 : Person" object monitor is shown, displaying the state of the object. It has two fields: "private String name" with the value "Maria" and "private int age" with the value 19. The "age" field is circled. Buttons for "Inspect", "Get", "Show static fields", and "Close" are visible.

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, 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. Below the class hierarchy, two instances of the "Person" class are shown: "person1: Person" and "person2: Person".

A "Method Result" window is open, showing the result of the "isTeenager()" method call on "person2". The window title is "BlueJ: Method Result". The code snippet shows the method signature and the return value:

```
// Is the person a teenager?  
// @return true is the person is a teenager, otherwise false  
boolean isTeenager()  
  
person2.isTeenager() returned:  
boolean true
```

The "Inspect" and "Get" buttons are visible next to the result. The "Close" button is at the bottom right.

A "Person" object window is also open, showing the instance "person2 : Person". It displays the private fields "name" (value "Maria") and "age" (value 19). The "Inspect" and "Get" buttons are visible next to the field values. The "Show static fields" button is at the bottom left, and the "Close" button is at the bottom right.

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-pointing arrow, and "Compile". The central workspace displays a class diagram for the "Person" class. Below the workspace, there are two red buttons labeled "person1: Person" and "person2: Person". A context menu is open over the "person2: Person" button, showing a list of methods inherited from Object: "void birthday()", "int getAge()", "String getN()", "boolean isTeenager()", "void setName(String n)", and "String toString()". The "void birthday()" method is highlighted with a blue border. At the bottom of the IDE, a status bar indicates "Initialising virtual machine... Do...".

Overlaid on the IDE is a "BlueJ: Terminal Window - Person" window. It has an "Options" tab and displays the output of the birthday method: "Happy birthday Maria!" and "Happy birthday Maria!". A message at the bottom of the terminal window states "Can only enter input while your programmi".

Another window, titled "person2 : Person", is shown in the foreground. It displays the state of the "person2" object. It has two input fields: "private String name" with the value "Maria" and "private int age" with the value "20". The "age" field is circled in blue. To the right of the input fields are buttons for "Inspect", "Get", and "Close". At the bottom left of this window is a button labeled "Show static fields".

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 is open, showing the code for the `isTeenager()` method and the result: `person2.isTeenager() returned: boolean false`. Below this, a "person2 : Person" object window is open, showing the object's state: `private String name` with value "Maria" and `private int age` with value 20. The "isTeenager()" method is also highlighted in this window.

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

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 the 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)

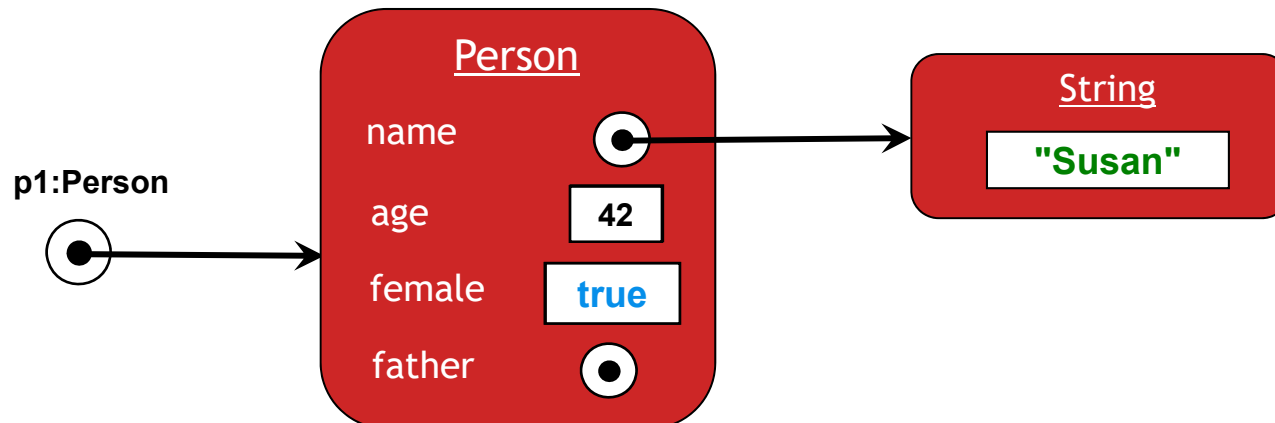
```
public class Person {  
    private String name;  
    private int age;  
    private boolean female;  
    private Person father;
```

Nu med 4 feltvariabler

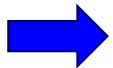
```
    public Person(String n, int a, boolean sex) {  
        name = n;  
        age = a;  
        female = sex;  
    }  
    ...  
}
```

Konstruktøren
initialiserer 3 af
feltvariablerne

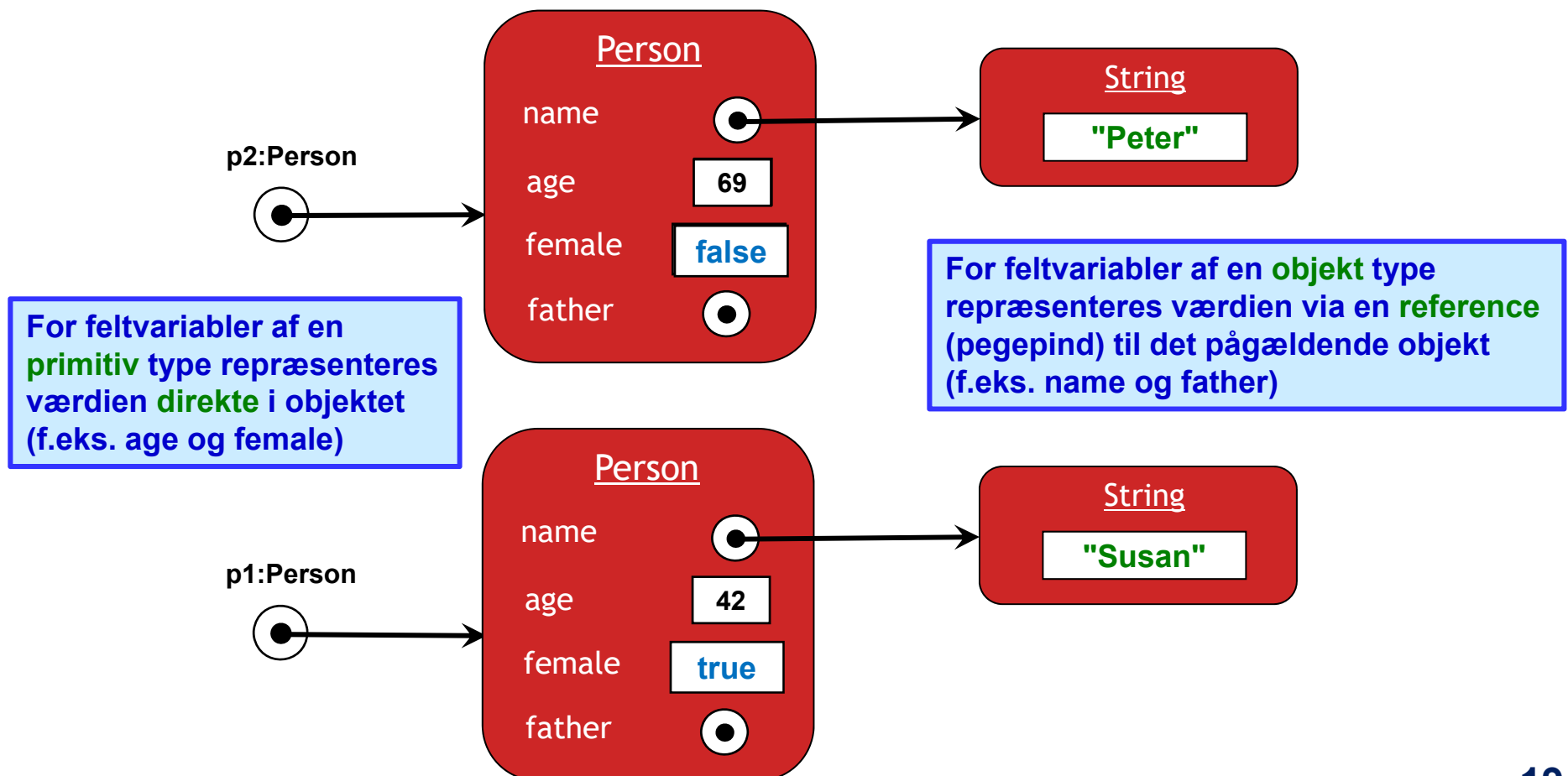
```
private Person p1;  
p1 = new Person("Susan", 42, true);
```



Endnu et objekt



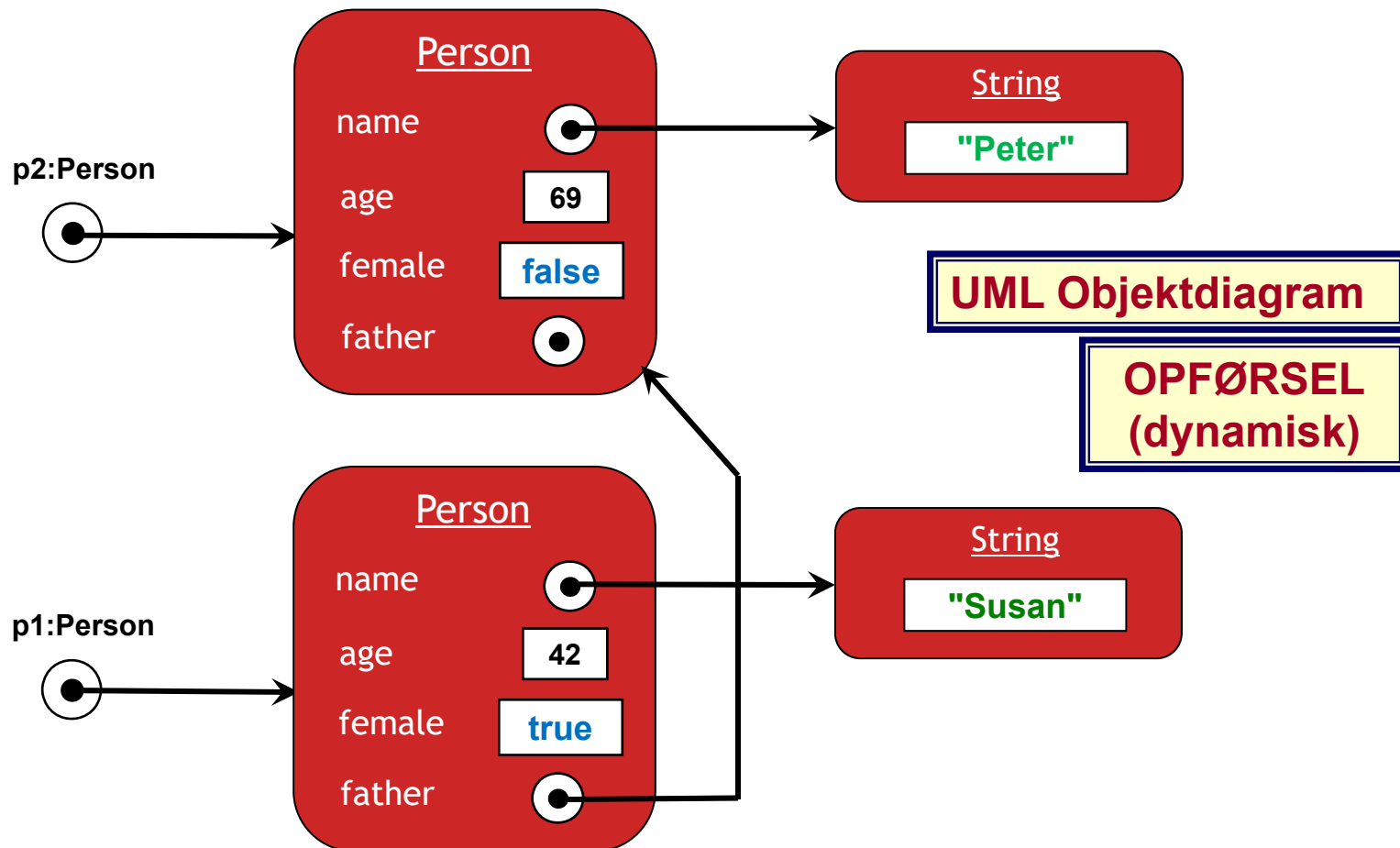
```
private Person p2;  
p2 = new Person("Peter", 69, false);
```



Metoden setFather (mutator metode)

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

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



Metoden birthday (mutator metode)

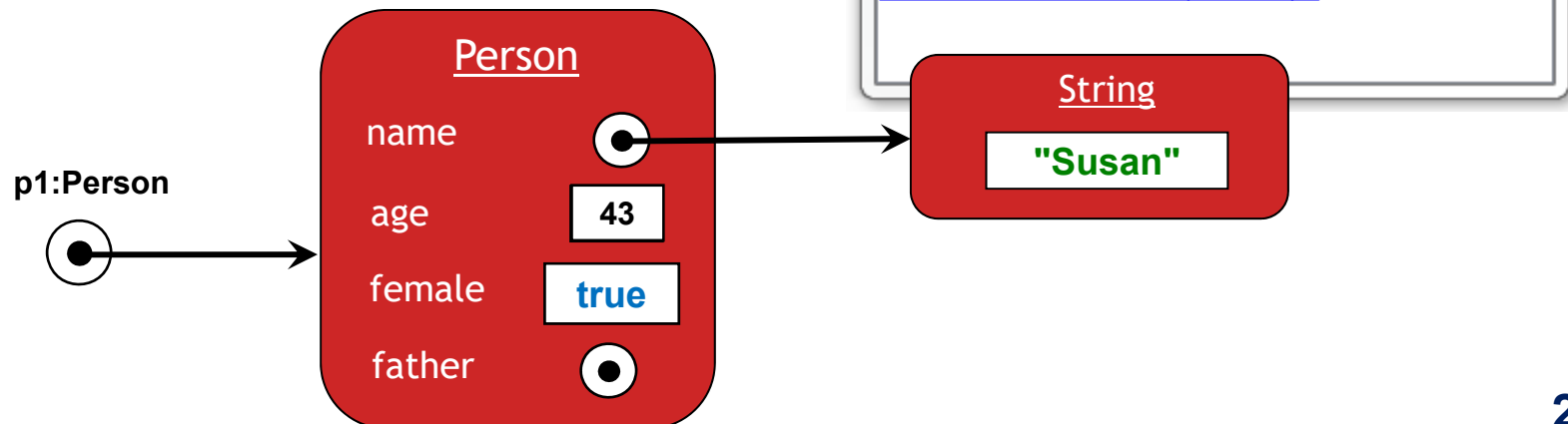
```
p1.birthday();
```

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

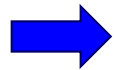
Kald af metode i Java's klassebibliotek
(udskriver linje på BueJ's terminal)

Hvis man kun skriver print,
får man intet linjeskift

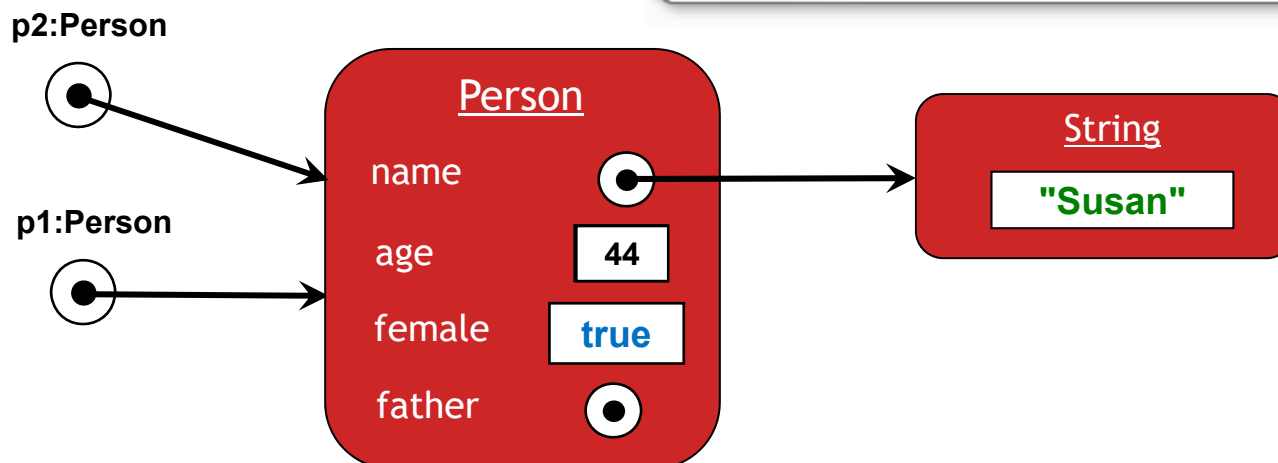
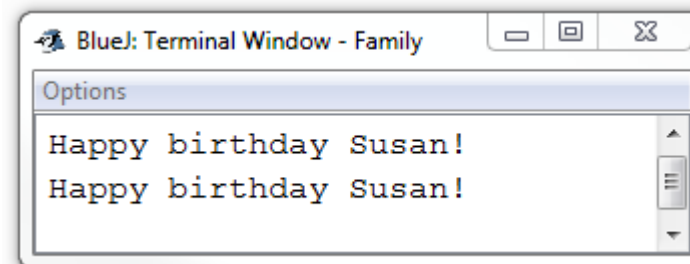
Sammensætning af
tekststreng
(konkatenering)



En person – to referencer



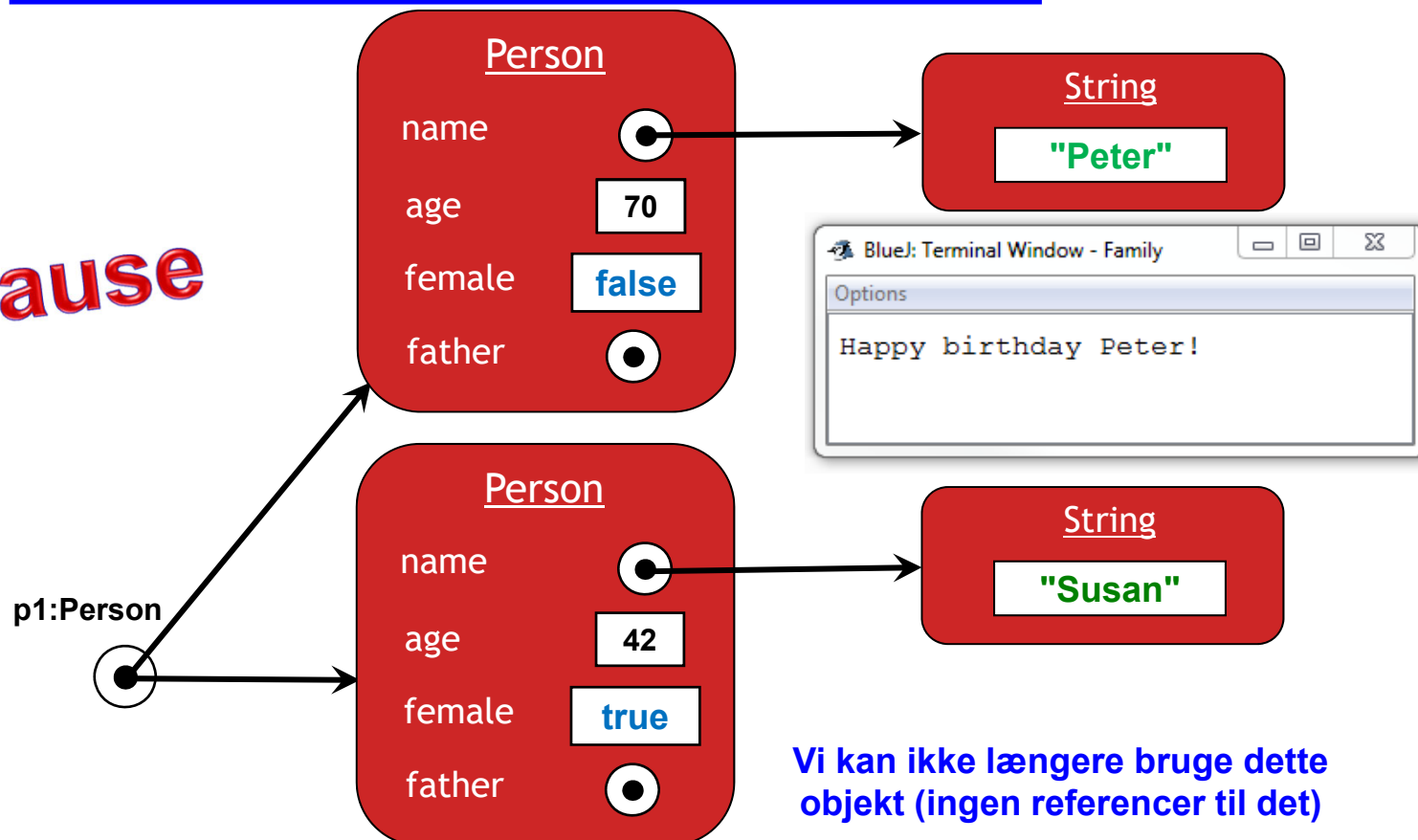
```
private Person p1, p2;  
p1 = new Person("Susan", 42, true);  
p2 = p1;  
p1.birthday();  
p2.birthday();
```



To personer – én reference

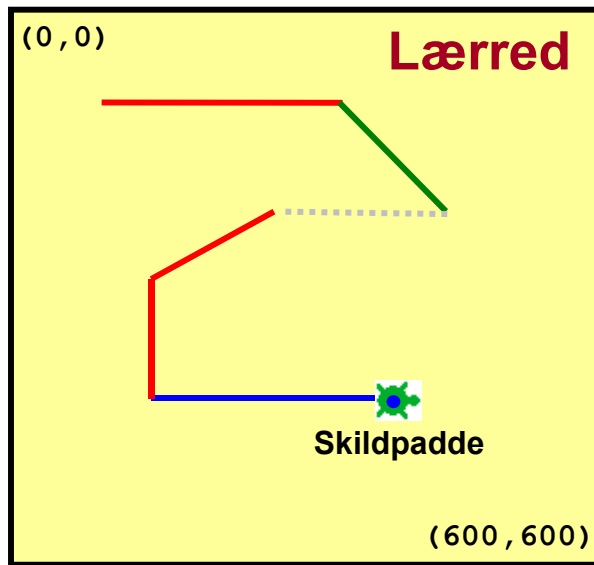
```
private Person p1;  
p1 = new Person("Susan", 42, true);  
p1 = new Person("Peter", 69, false);  
p1.birthday();
```

Pause

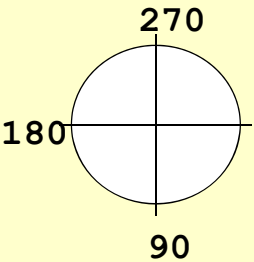



● Iteration, selektion og parametrisering

- 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, ...

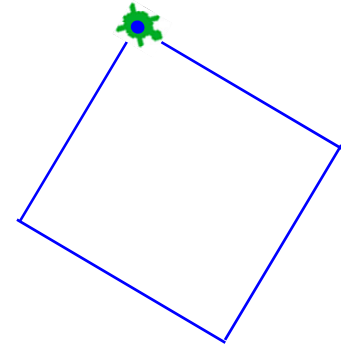
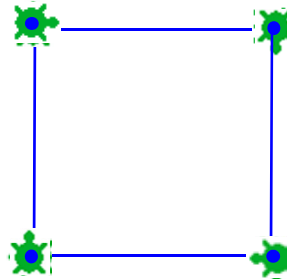
Turtle
<pre>move(double distance) turn(double degrees) penUp() penDown() ... square(double size) polygon(int n, double size) circle(double radius) ...</pre>

- **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);
```

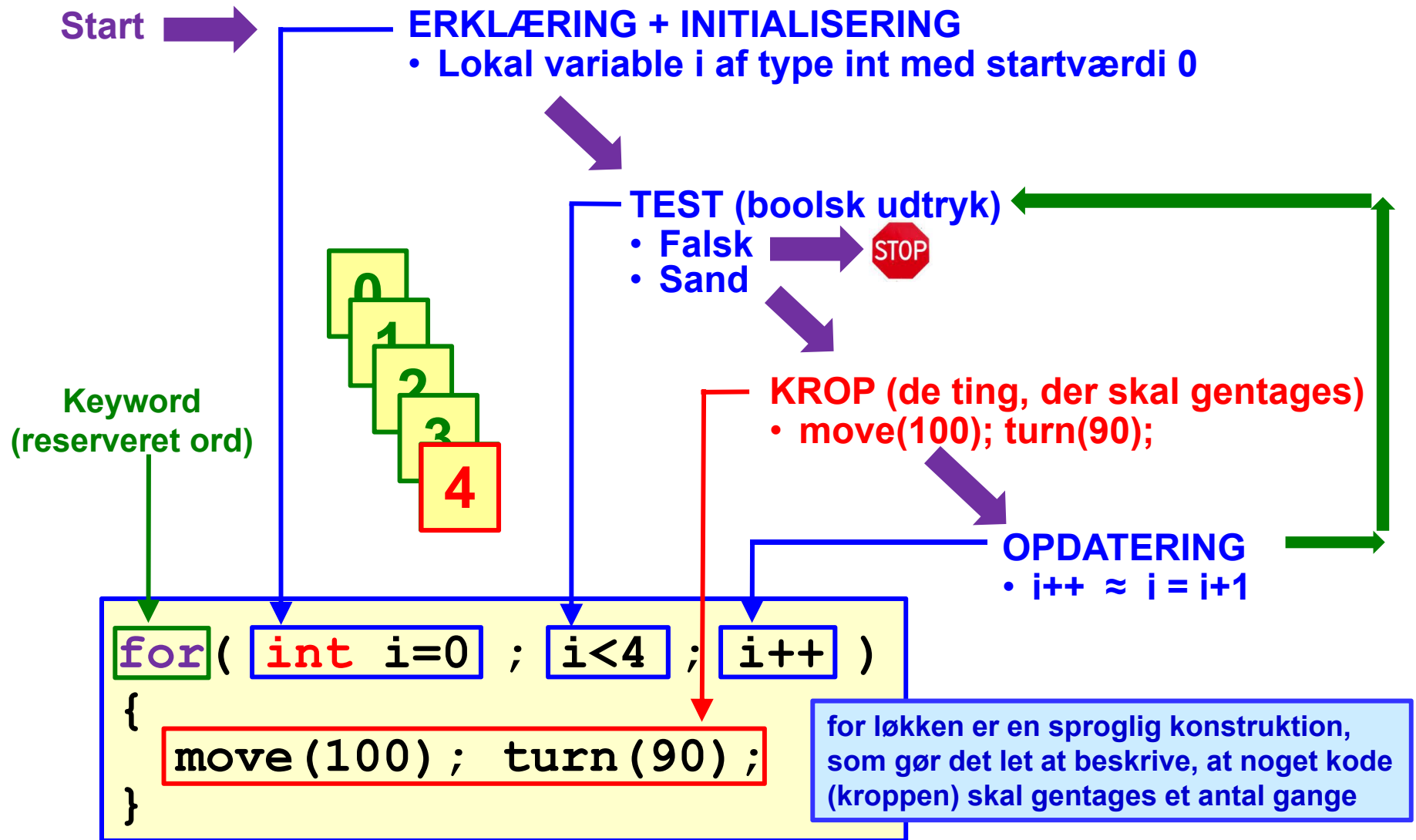
```
// Tegn kvadrat  
gentag 4 gange {  
    move(100);  
    turn(90);  
}
```

```
// 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?

Intet

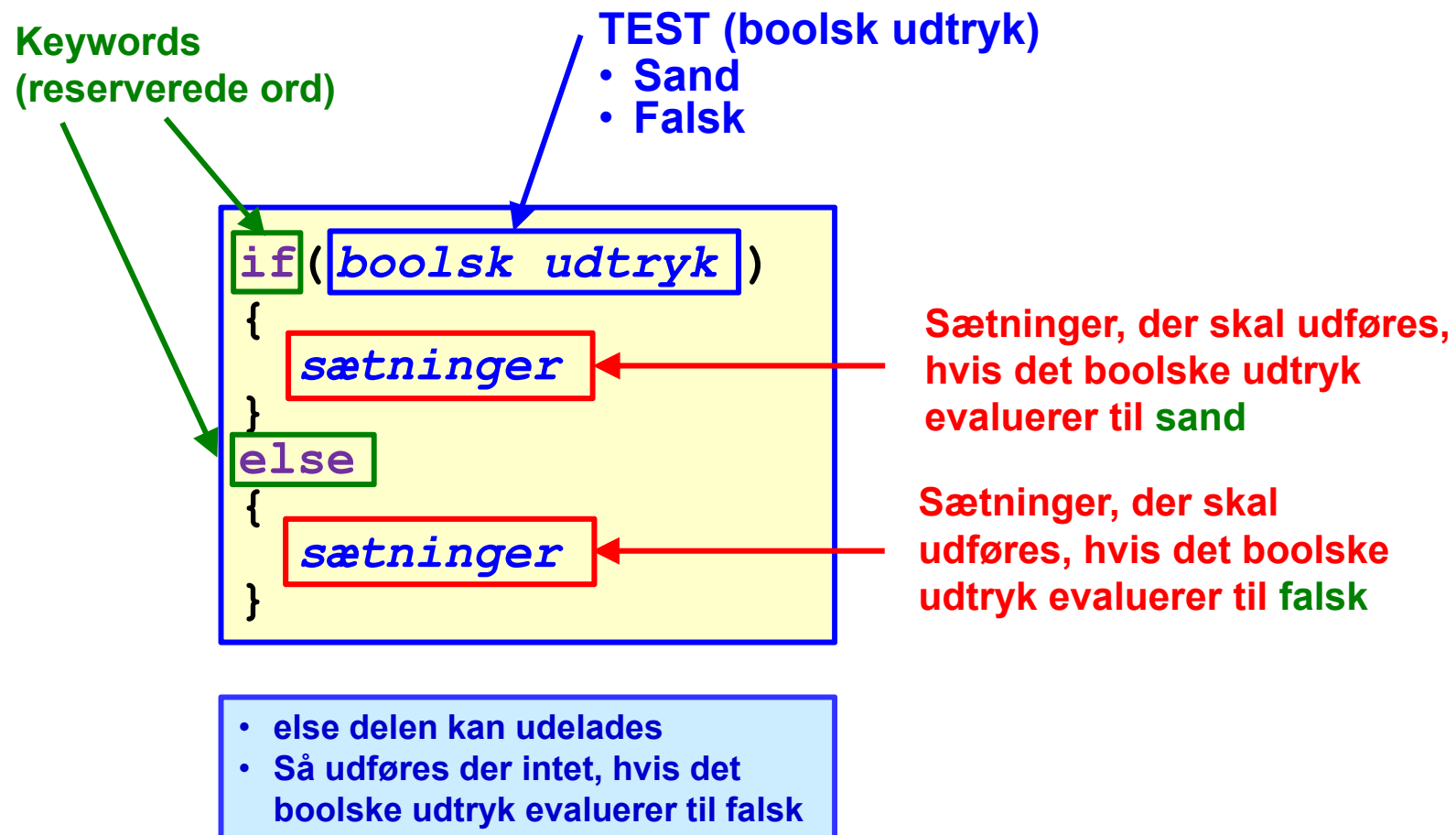
Hvad sker der, hvis n er 1?

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?

Hvad sker der, hvis size er 0?

Hvad sker der, hvis size er negativ?

Intet

Skildpadden bakker, men tegner
en korrekt n-kant og returnerer
til udgangspositionen

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 (fields)**
 - 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 (eller den enkelte udførelse af en løkke)
 - Kan **ikke** bruges til at gemme resultater mellem metodekald
 - Ingen access modifier (kan aldrig tilgås udenfor metoden/konstruktøren)

```
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

Demo af
Date klassen
i Blue J

● 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
 - I dag har vi f.eks. set på, hvordan vi kan **modellere** personer (og deres familierelationer) samt hvordan vi kan modellere tegnende skildpadder
 - **Klasser** modellerer **begreber** (f.eks. Person og Turtle)
 - **Objekter** er **instanser** af klasser (f.eks. forskellige personer)
 - 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 mere end 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

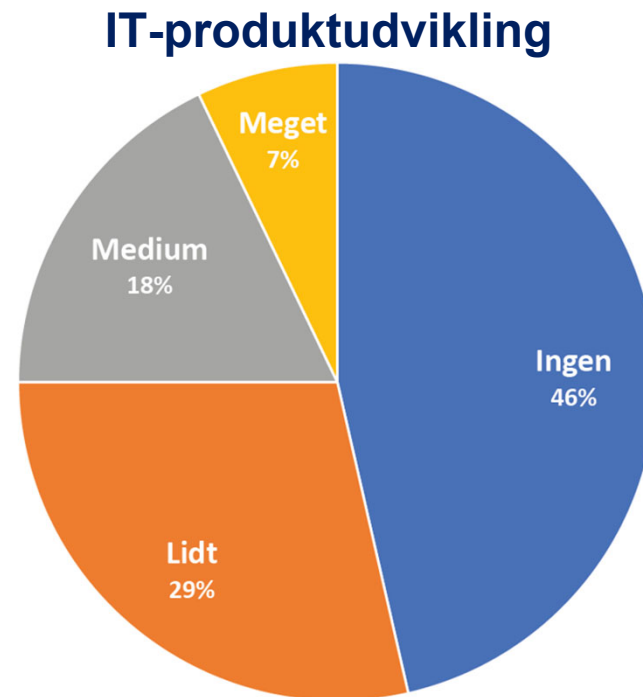
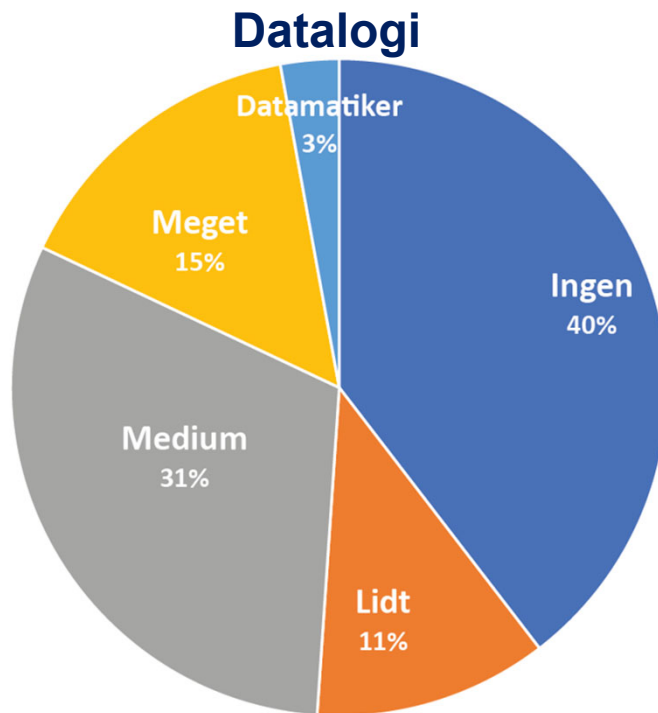
Husk at forberede jer til øvelserne

- **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 jeres model, således at terninger kan have et vilkårligt antal sider (større end eller lig med 2)

- Husk at aflevere Raflebæger 1 og Quiz 1 inden mandag kl. 14.00 (lørdag kl. 22.00 for IT 'ere)

Programmeringserfaring

- **Stor spredning med hensyn til programmeringserfaring**
 - For datalogi er det godt halvdelen, der har ingen eller lille erfaring
 - For it-produktudvikling er det tre fjerdedele



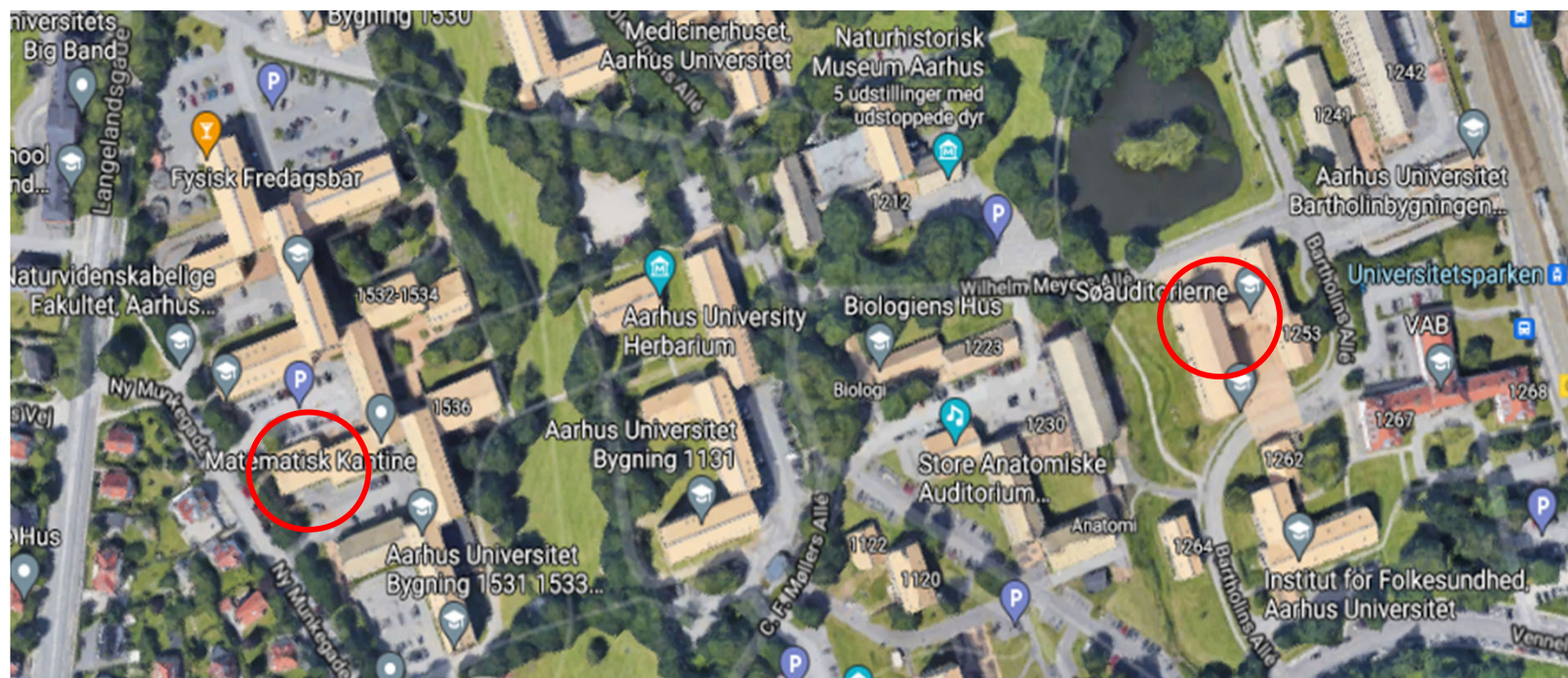
- **Det betyder, at nogle af jer vil synes, at det går lidt langsomt her i starten**
 - Det er nødvendigt af hensyn til dem, der har ingen eller lille programmeringserfaring (mere end halvdelen af jer)

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 Brightspace sider
- **På websiderne Projekt Euler, CodingBats og Kattis findes en masse opgaver, hvor I kan øve jer i Java programmering**
 - Links på ugeoversigten for Uge 3
- **Deltage i 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
 - Mere information på cs.au.dk/talent og ved en senere forelæsning

Vaccination mod COVID-19 på Campus

- **Alle medarbejdere og studerende kan blive vaccineret mod COVID-19 i Søauditorierne i Universitetsparken.**
 - Åbningstider: mandag, tirsdag, torsdag og fredag kl. 9-13 i perioden 30. august til 10. september
 - **Ingen krav om tidsbestilling**
 - Du kan både påbegynde dit vaccineforløb og få dit andet stik
 - Du vil blive vaccineret med Pfizer-BioNTechs vaccine.



Så er vi klar til at forlade auditoriet

- **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 afspritningsansvarlige bliver tilbage og hjælpes ad med afspritningen
 - Start med at desinficere jeres hænder, før I rører ved sprayflasken
 - Brug sprayflaske til overflader på borde og stole (efterlades fugtige men ikke våde)
 - Husk alle berøringspunkter (bordkant, underside, armlæn mv.)
- **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**

Det var alt for nu.....

... spørsmål

