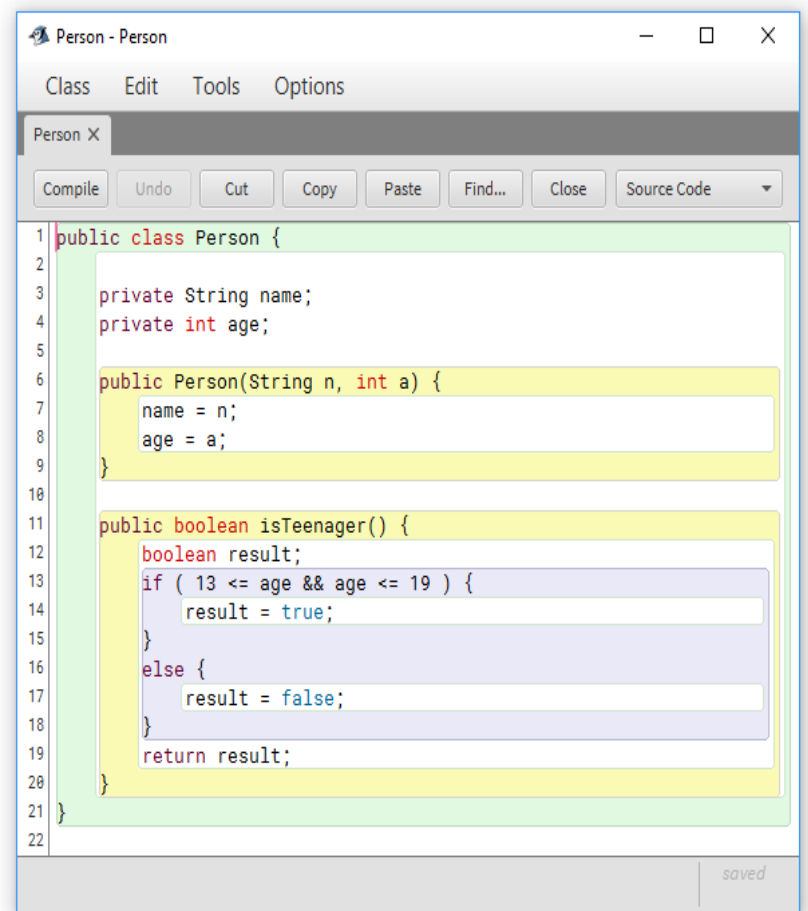


Forelæsning Uge 2 – Mandag

- **Sætninger**
 - Simple sætninger (assignment, interne og eksterne metodekald)
 - Sammensatte sætninger (blok, selektion, gentagelse)
- **Udtryk og operatorer**
- **Java syntax og style guide**
- **Afleveringsopgaver i uge 2**

- Husk at løse så mange som muligt af de opgaver, der er i BlueJ bogens kapitler
- Ved at løse en masse af disse i starten, sparer I tid, når I kommer til de lidt mere komplicerede opgaver
- Det er kun ved at programmere en masse, at I bliver gode til det

- På **Projekt Euler**, **CodingBats** og **Kattis** findes en masse ekstra opgaver, hvor I kan øve jer i Java programmering, hvis I har tid tilovers
- Links på ugeoversigten for Uge 3



The screenshot shows the BlueJ IDE window titled "Person - Person". The menu bar includes "Class", "Edit", "Tools", and "Options". Below the menu bar is a toolbar with buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Close", and a "Source Code" dropdown. The main text area displays the source code for the "Person" class, with line numbers 1 through 22 on the left. The code is as follows:

```
1 public class Person {  
2  
3     private String name;  
4     private int age;  
5  
6     public Person(String n, int a) {  
7         name = n;  
8         age = a;  
9     }  
10  
11     public boolean isTeenager() {  
12         boolean result;  
13         if ( 13 <= age && age <= 19 ) {  
14             result = true;  
15         }  
16         else {  
17             result = false;  
18         }  
19         return result;  
20     }  
21 }  
22
```

The bottom right corner of the window shows the word "saved".

● Simple sætninger

- **Assignment (ændring af variabels værdi)**

- Udregner værdien af udtrykket på højresiden og tildeler denne værdi til variabelen på venstresiden

```
v = exp;
```

```
name = n;
```

```
age = age + 1;
```

```
age += 1;
```

```
age++;
```

Udtrykkets type
skal matche
variablens type

- **Return sætning (typisk inde i accessor metode)**

```
return exp;
```

```
return age;
```

```
return name;
```

Udtrykkets type
skal matche
metodens returtype

Metodekald

- **Internt metodekald**

- Kald af metode i samme objekt

```
method (...);
```

```
isTeenager();
```

```
setName("Maria");
```

Det udtryk man bruger for en parameter skal matche parameterens type

- **Eksternt metodekald**

- Kald af metode i andet objekt
- Dot notation (dot = punktum på amerikansk)

```
object-reference.method(...);
```

```
p1.isTeenager();
```

```
p1.setName("Maria");
```

```
p1.setFarther(p2);
```

Det udtryk man bruger for en parameter skal matche parameterens type

Metoden skal være erklæret i objekt-referencens type (der er en klasse)

Sammensatte sætninger

- **Blok**

- Sekvens af sætninger (omgivet af krøllede parenteser)
- Parenteserne gør, at blokken opfattes som én sætning, og dermed kan bruges alle de steder, hvor man kan bruge en sætning

```
{ S1  S2  ... Sn }
```

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

Blok med to
assignment
sætninger

● Selektion (valg) – if sætning

`if (exp) S`

exp skal evaluere til en sandhedsværdi (boolean)

```
public void insertMoney(int amount) {  
    if (amount > 0 )  
    {  
        balance = balance + amount;  
    }  
}
```

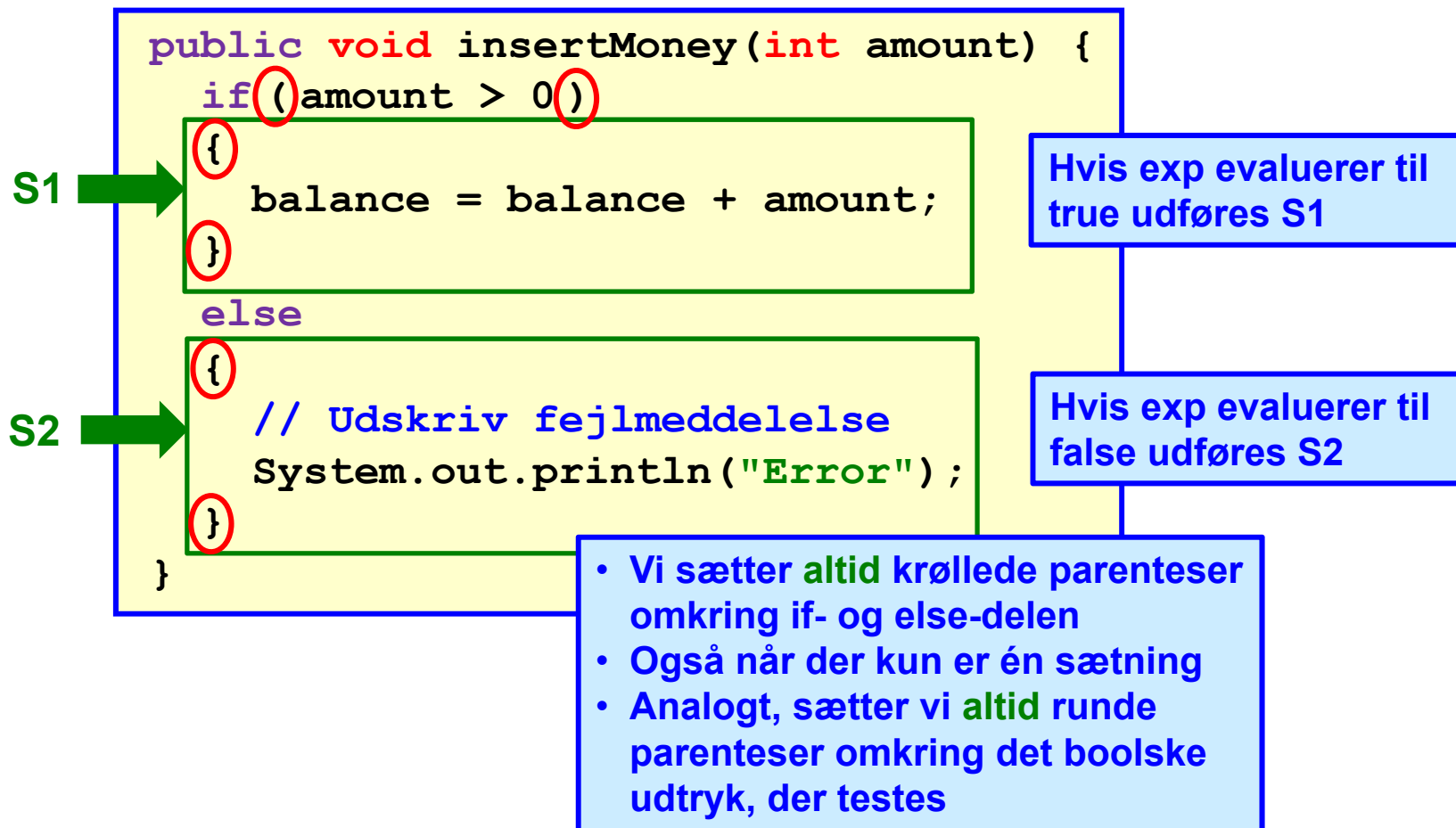
S →

S udføres kun, hvis exp evaluerer til true

Valg mellem to blokke – if-else sætning

```
if(exp) S1 else s2
```

exp skal evaluere til en sandhedsværdi (boolean)



Lidt mere kompakt

```
public void insertMoney(int amount) {  
    if(amount > 0) {  
        balance = balance + amount;  
    }  
    else {  
        // Udskriv fejlmeddelelse  
        System.out.println("Error");  
    }  
}
```

For at spare plads, kan startparentesen i en blok sættes på samme linje, som det der går forud

Hvis udtryk og sætning er kort, kan en if-sætning skrives på en enkelt linje (men husk parenteserne)

```
if ( amount > 0 ) { balance += amount; }
```

Indlejret selektion

- Datoer repræsenteres ved hjælp af tre feltvariabler (day, month, year)
 - Metoden **nextDay** ændrer de tre feltvariabler, så de repræsenterer den efterfølgende dag
 - Det antages, at alle måneder har 30 dage

```
public void nextDay() {  
    day = day + 1;  
    if (day > 30) {  
        day = 1;  
        month = month + 1;  
        if (month > 12) {  
            month = 1;  
            year = year + 1;  
        }  
    }  
}
```

Den **røde** if sætning er indlejret i den **grønne** if sætning

Forskellene på månedernes længde samt skudår, kan håndteres ved at indføre nogle flere if sætninger

Selektion mellem mange – switch sætning

Ugedag repræsenteret som heltal

```
int day;  
// 1 = Monday  
// 2 = Tuesday  
// 3 = Wednesday  
// 4 = Thursday  
// 5 = Friday  
// 6 = Saturday  
// 7 = Sunday
```

Metode, der konverterer fra heltal til tekststreng
(f.eks. 3 → "Wednesday", 6 og 7 → "Weekend", 0 → "Invalid day")

```
public String convertDay(int day) {  
    switch(day) {  
        case 1: return "Monday";  
        case 2: return "Tuesday";  
        case 3: return "Wednesday";  
        case 4: return "Thursday";  
        case 5: return "Friday";  
        case 6:  
        case 7: return "Weekend";  
        default: return "Invalid day";  
    }  
}
```

- Normalt afsluttes de enkelte cases med et **break**, der stopper udførelsen af switch sætningen
- Dette er ikke nødvendigt her, idet **return** også stopper udførelsen af sætningen

- Mere information om selektion: **Appendix D**

Selektion i udtryk – ved hjælp af ? og :

- Vi har set, hvordan man i en if (eller switch) sætning kan selekttere mellem forskellige (blokke af) sætninger
- Analogt kan man i et udtryk selekttere mellem to forskellige udtryk

`(exp ? exp1 : exp2)`

- exp skal være et boolsk udtryk, mens exp1 og exp2 skal have matchende typer (f.eks. begge være af typen int)
- Hvis exp evaluerer til true evalueres exp1 (og værdien af exp1 er værdien af det samlede selektionsudtryk)
- Hvis exp evaluerer til false evalueres exp2 (og værdien af exp2 er værdien af det samlede selektionsudtryk)
- Her kan man ikke udelade den sidste del (exp2) – et udtryk skal jo altid evaluere til en værdi

Selektion i udtryk (eksempler)

Bank eksempel (fra før)

```
if (amount > 0) {  
    balance = balance + amount;  
}
```

kan også programmeres sådan

```
balance += ( amount > 0 ? amount : 0 );
```

Evaluer højresiden og adder resultatet til venstresiden

Udskrift af to næsten identiske strenge

```
System.out.println("My mothers car is " +  
    (color == red ? "" : "not ") +  
    "red and has four doors.");
```

color == red → "My mothers car is red and has four doors."

color != red → "My mothers car is not red and has four doors."

- Mere information om selektion i udtryk: **Sektion 7.5.1**

Eksempel på dårlig kode

```
public boolean isTeenager() {  
    boolean result;  
    if( 13 <= age && age <= 19 ) {  
        result = true;  
    }  
    else {  
        result = false;  
    }  
    return result;  
}
```

Metoden tjekker om
personen er teenager

Den gør det rigtige,
men er unødvendig
lang og kompliceret



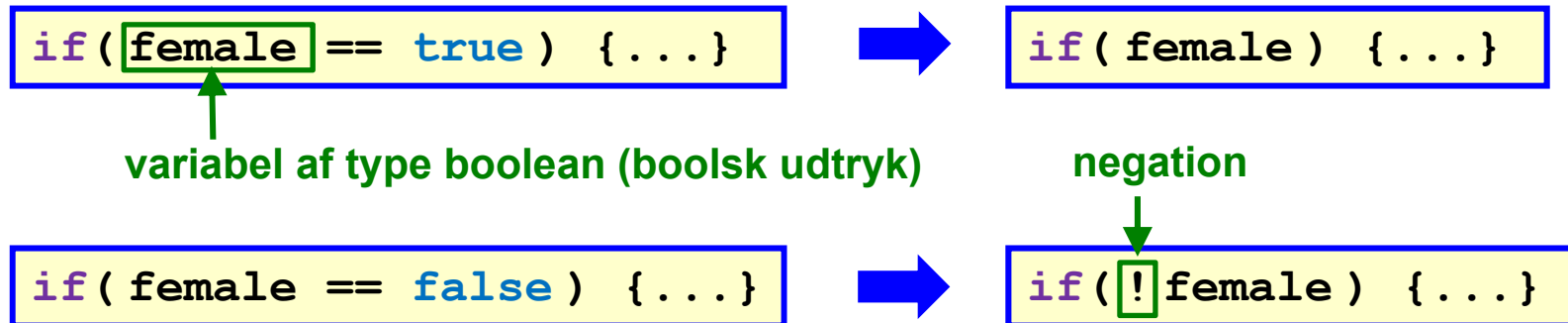
```
public boolean isTeenager() {  
    return (13 <= age && age <= 19);  
}
```

8 linjer kode



1 linje kode

Andet eksempel på dårlig kode



- Helt galt går det, hvis man kommer til at skrive

```
if( female = true ) { ... }
```

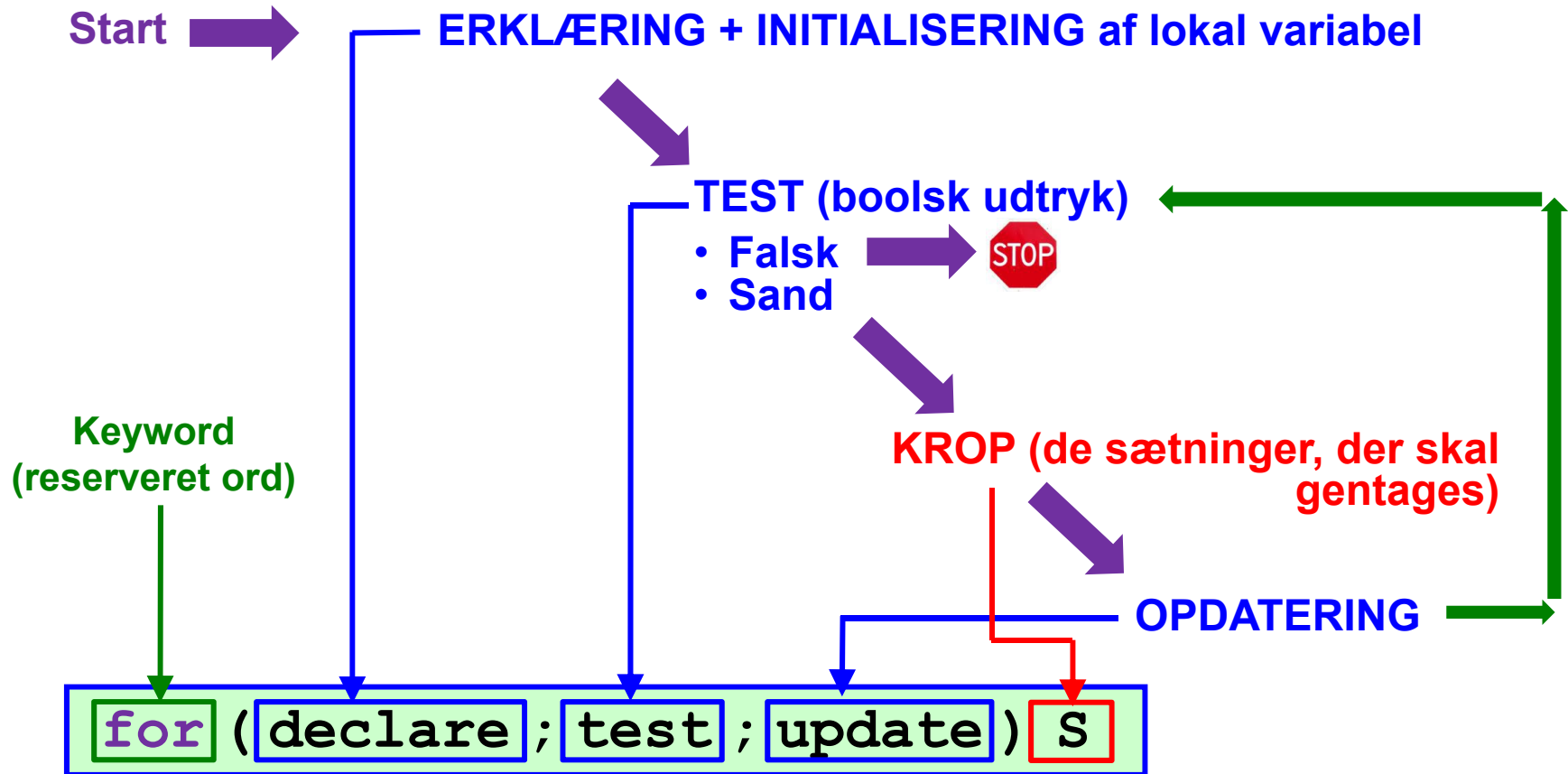
- Assignment, som ændrer værdien af `female` til `true`
- Assignmentet er selv et boolsk udtryk, så oversætteren er tilfreds
- Udtrykket evaluerer altid til `true`, hvorfor kroppen i `if` sætningen altid udføres

- Hvorfor er et assignment et udtryk?

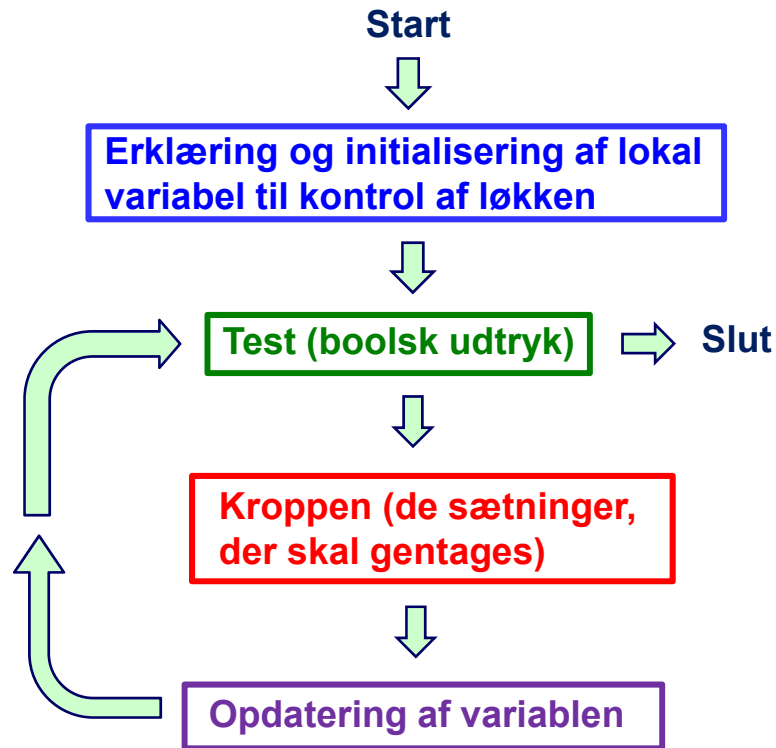
```
x = y = 37;
```

Assignment, men også et udtryk med værdien 37, som så assignes til `x`

● Iteration (gentagelse) – for løkke



Eksempel på for løkke



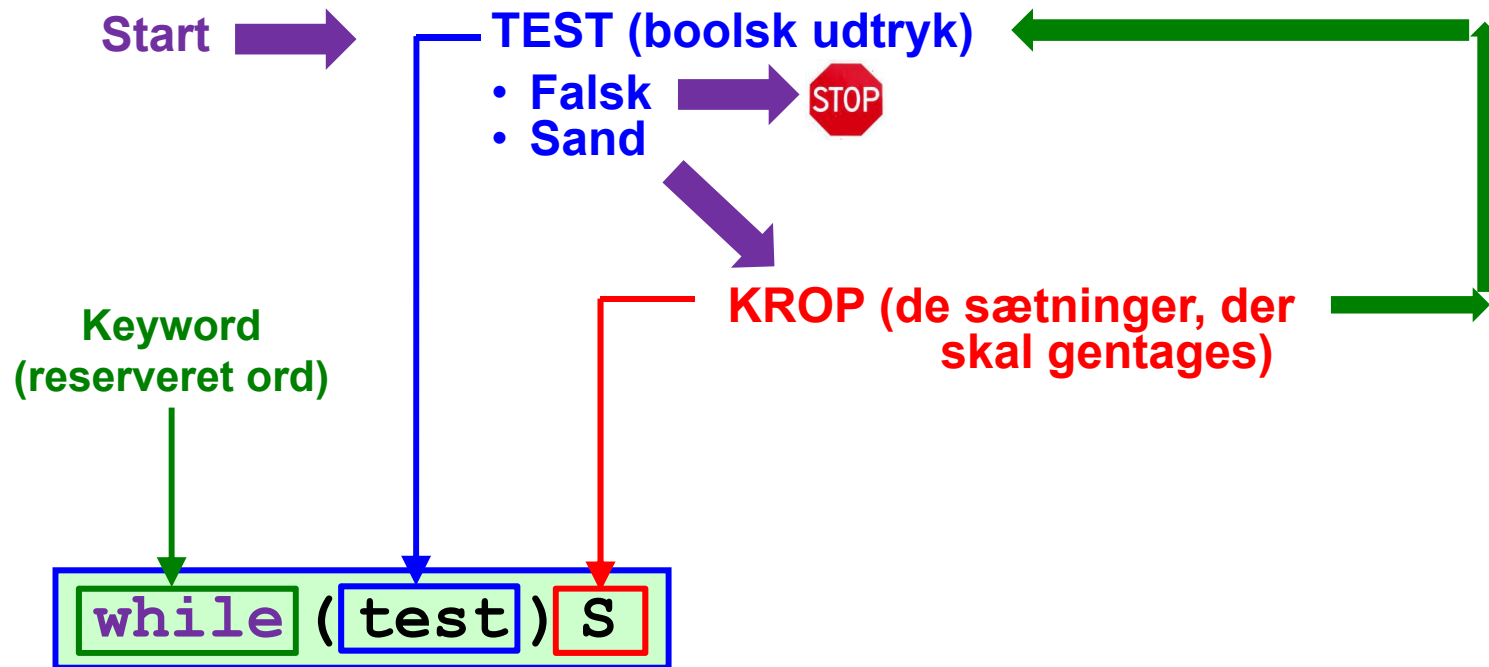
Tegn ligesidet n-kant

```
for (int i = 0; i < n; i++) {  
    move(size);  
    turn(360.0 / n);  
}
```

Kroppen (og opdateringen) gentages så længe den boolske betingelse er opfyldt

- I dette tilfælde gentages **move** og **turn** operationerne **n** gange, hvorved man tegner en ligesidet n-kant

while løkke



- **while** løkken er simplere og mere fleksibel end for løkken
- **Vi skal selv huske at**
 - erklære og initialisere en passende variabel (som indgår i vores test)
 - opdatere variablen i kroppen

Eksempel på while løkke



Tegn ligesidet n-kant

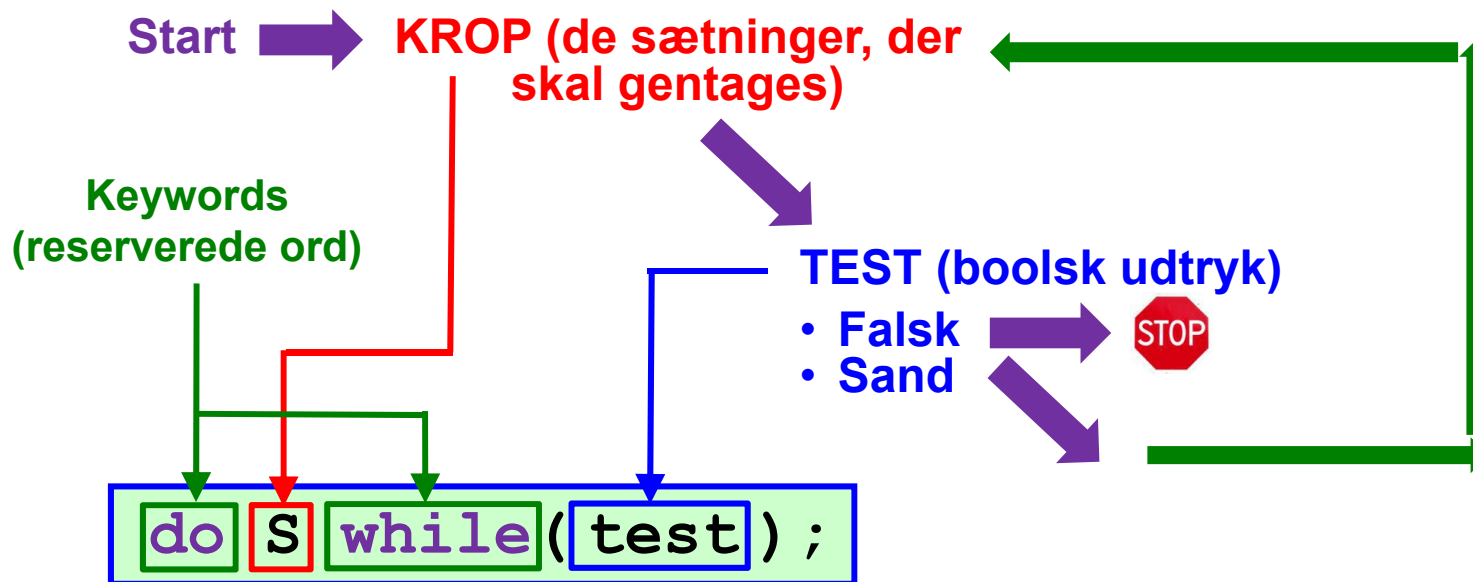
```
int i = 0;
while (i < n) {
    move(size);
    turn(360.0 / n);
    i++;
}
```

Kroppen (inklusive opdateringen) gentages så længe den boolske betingelse er opfyldt

- I dette tilfælde gentages **move** og **turn** operationerne **n** gange, hvorved man tegner en ligesidet n-kant

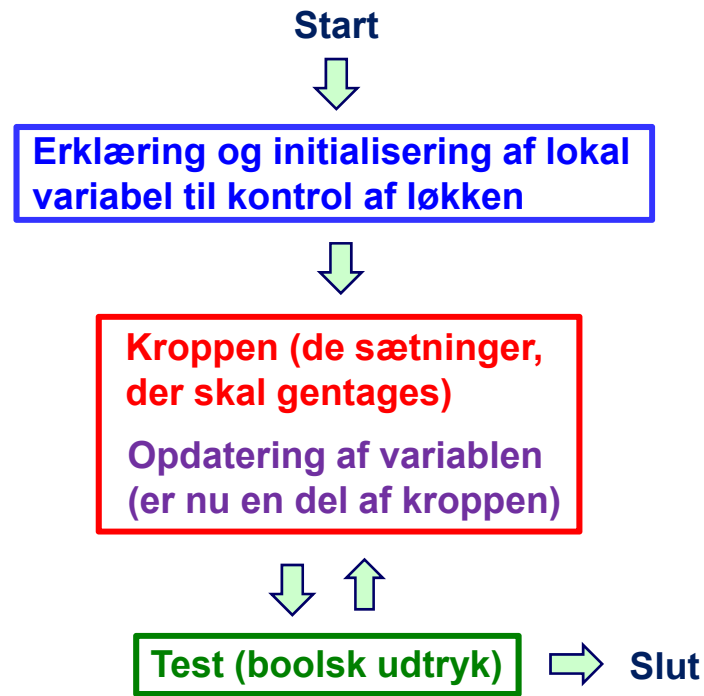
do-while løkke

- **Fungerer på samme måde som while løkken**
 - Men nu kommer udførelsen af kroppen **før** testet
 - Det betyder at kroppen altid udføres **mindst én gang**



- **Vi skal (også her) selv huske at**
 - erklære og initialisere en passende variabel (som indgår i vores test)
 - opdatere variablen i kroppen

Eksempel på do-while løkke



Tegn ligesidet n-kant

```
int i = 0;
do {
    move(size);
    turn(360.0 / n);
    i++;
} while (i < n);
```

do-while løkken skal afsluttes med et semikolon (det skal de andre løkker ikke)

do-while løkken ligner while løkken, men kroppen udføres nu før testet

– Det betyder, at kroppen altid udføres mindst én gang

Kroppen (inklusive opdateringen) gentages så længe den boolske betingelse er opfyldt

– I dette tilfælde gentages **move** og **turn** operationerne **n** gange, hvorved man tegner en ligesidet n-kant

Sammenligning af de tre slags løkker

- **Hvilken løkke skal man vælge?**
 - for løkker bruges, når man før udførelsen ved, hvor mange gange løkken skal gennemløbes
 - De to andre slags løkker er mere fleksible, men her skal man selv huske at erklære, initialisere og opdatere variablen
- **Vi har set, at disse tre løkker alle tegner en ligesidet n-kant**

for løkke

```
for(int i=0; i<n; i++) {  
    move(size);  
    turn(360.0 / n);  
}
```

Er der situationer, hvor de tre løkker **ikke** gør helt det samme?

while løkke

```
int i = 0;  
while(i < n) {  
    move(size);  
    turn(360.0 / n);  
    i++;  
}
```

do-while løkke

```
int i = 0;  
do {  
    move(size);  
    turn(360.0 / n);  
    i++;  
} while(i < n);
```

- Om en uges tid møder vi en fjerde slags løkker, som kaldes **for-each løkker**
- Mere information om iteration: **Appendix D**

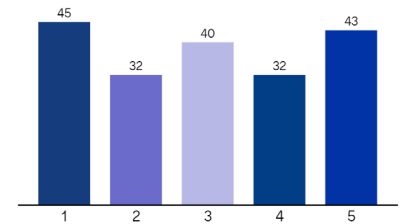
● Quizzer i forelæsningerne?

- **De næste forelæsninger vil der være en quiz midt i hver time**
 - Altså to quizzer i hver forelæsning
 - Hver quiz indeholder 3-10 spørgsmål
- **Formålet med quizzerne er**
 - at afveksle undervisningsformen – så I ikke falder i søvn
 - at få jer til at være aktive – både individuelt og sammen med dem, som sidder ved siden af jer
 - at repetere stof – typisk fra foregående forelæsning
- **Quizzerne giver os feedback**
 - De fortæller instruktorerne og mig, hvad I har forstået, og hvor der skal sættes ekstra ind

Afvikling af quizzerne

- **Quizzerne afvikles på følgende måde**

- Spørgsmålet præsenteres
- I får 1-2 minutter til at finde svaret på spørgsmålet (gerne i samarbejde med dem, som sidder ved siden af)
- I stemmer via jeres mobil eller jeres bærbare
- De mulige svar gennemgås (eventuelt med input fra jer)
- Resultatet af afstemningen vises (som et søjlediagram)



- **Mine slides viser de rigtige svar og forklarer disse**

- Det betyder, at I kan bruge quizzerne, selv om I ikke har været til den pågældende forelæsning
- I skal blot huske jeres svar og tjekke, om det var det rigtige
- Quizzerne ligger sammen med slidsene fra forelæsningen

- **Nogle af forklaringerne introducerer nye ting – f.eks. eksempler på, hvordan syntaks og run-time fejl kan se ud**

Hvordan stemmer man?

- **Det nemmeste er at stemme via jeres mobil ved hjælp af **Mentimeter** app (som er gratis)**
 - Hentes samme sted, som I henter jeres andre applikationer
 - I kan også stemme via en webbrowser ved hjælp af linket **www.menti.com** (eller blot **menti.com**)
- **Ved afstemningens start indtastes en kode, der identificerer den afstemning, som I ønsker at deltage i**
- **Afstemningen er anonym**
 - Det registreres ikke, hvem der stemmer på hvad

Pause

Brug eventuelt lidt af den til at downloade **Mentimeter** app'en

● Udtryk

- **Et udtryk er bygget op af variabler, konstanter og operatorer**

- x, y og z er variabler
(feltvariabler eller lokale variabler)

```
x + 2 * y / z
```

- 2 er en konstant
- +, * og / er operatorer for addition, multiplikation og division

- **En operator tager nogle operander og leverer et resultat**

- F.eks. kan + operere på
 - to heltal (int), hvilket resulterer i et heltal
 - to reelle tal (double), hvilket resulterer i et reelt tal (double)
 - et heltal og et reelt tal, hvilket resulterer i et reelt tal
 - to tekststreng, hvilket resulterer i en tekststreng

- **Et udtryk kan også indeholde metodekald, der returnerer en værdi**

- "Name: " er en konstant
- getName() er et metodekald, der returnerer en tekststreng

```
"Name: " + getName()
```

- + konkatenerer de to tekststreng (concatenation ≈ sammensætning)

Brug af udtryk

- **Udtryk bruges mange steder, f.eks.**

- Højresiden af assignment
- Return sætning (inde i accessor metoder)
- Argumenter til metodekald

```
i = 2 * j;
```

```
return i + j;
```

```
move ( x / y );
```

- **Alle udtryk har en type**

- Typen beskriver, hvilke slags (type) værdier udtrykket kan evaluere til
- Antag at i og j er variabler af type int og x og y variabler af type double

```
2 * j
```

int

```
x / y
```

double

```
i + j
```

int

```
2.45 * j
```

double

- Typen for et udtryk bestemmes af typerne for de variabler, konstanter, og metodekald, der indgår i det

Udtrykkets type skal matche brugen

- **I et assignment skal udtrykkets type matche variabelens type**

- Ok, hvis v og w er variabler af samme type

```
v = 2 * w;
```

- **I en return sætning skal udtrykkets type matche metodens returtype**

- Ok, hvis returtypen er identisk med v's type

```
return v + 1;
```

- **I et metodekald skal arguments type matche parameterens type**

- Ok, hvis parameterens type er den samme som argumentets type

```
move(size / 3);
```

- **I en if sætning skal betingelsen være et boolsk udtryk**

- Dvs. et udtryk der evaluerer til sand eller falsk

```
if (v < w) {...};
```

Matchende typer

- **Hvad betyder det at to typer matcher hinanden?**
 - Det er trivielt opfyldt, hvis de to typer er identiske
- **Man kan godt have et match uden at typerne er helt identiske**
 - F.eks. kan man bruge en **int**, de steder hvor der kræves en **double**
 - Argumentet er af type **int**, mens parameteren er af type **double**

```
move(100);
```
 - Første operand er en **int**, men bruges som en **double** (resultatet er en **double**)

```
14 / 7.0 → 2.0
```
- **Om nogle uger skal vi se på subklasser / subtyper**
 - Så bliver tingene mere komplekse
 - De steder man skal bruge et udtryk af en bestemt type, kan man i stedet bruge et udtryk, hvor typen er en subtype af den krævede

Javas typebegreb

- **Java sproget har stærkt type check**
 - Dvs. at mismatch mellem typer (i langt de fleste tilfælde) opdages, når programmet oversættes (undtagelsen er brug af type-cast)
 - Andre sprog opdager først typefejl, når programmet køres – eller opdager dem slet ikke
- **Stærkt type check er en stor fordel for programmøren**
 - Mange programmeringsfejl opdages under oversættelsen, hvor de som regel er lette at rette
- **Andre sprog**
 - Funktionelle sprog har også stærkt type check
 - JavaScript er et sprog til webbrowsere
 - Syntaksen ligner Java, men der er **ikke** stærkt type check, hvilket kan gøre det vanskeligt at lokalisere visse slags fejl

Udvalgte operatoren

Aritmetiske operatoren

+ - * / % ...

Logiske operatoren

&& || ! ...

^

bitvis
eksklusiv
OR

Relationelle operatoren

== != < > <= >=

new er også en operator

`new Class (...);`

- **Nogle operatoren er overloadede**

- Det betyder, at de kan bruges på argumenter af forskellig type
- + kan betyde læg sammen (for heltal, reelle tal eller en blanding)
- + kan også betyde konkatination (sammensætning af strenge)

- **Præcedens regler**

- Bestemmer rækkefølgen, som operatorerne udføres i
- $4 + 3 * 5$ evaluerer til 19

- **Java har 15 niveauer**

- Brug parenteser, når I er i tvivl

- **Mere om udtryk og operatoren: Appendix C**

● Syntaktiske elementer i Java

- **Reserverede ord (keywords)**

- `class, new, public, if, for, while, private, ...`

- **Navne (identifiers)**

- `int, boolean, double, String, Person, Date, p1, age, turn, move, day, month, year, ...`

Farverne er dem, som BlueJ og mine slides bruger

- **Konstanter (literals)**

- `"Aarhus Universitet", 1928, 5.78, true, false, ...`

- **Specialtegn (special characters)**

- `; () { } < > = + - * / < <= == != ? : && || ! ...`

- `// /* */ /** @`

- **Luft (white space)**

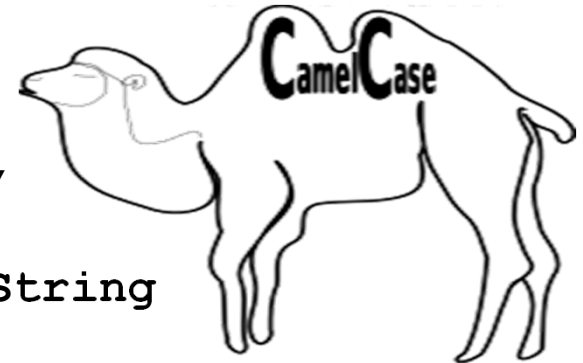
- linjeskift, mellemrum, tab, ...

- kan indsættes vikårligt **mellem** syntaktiske elementer uden at betydningen påvirkes

Java style guide (regler for pæn kode)

- **Navngivning**

- Navne skrives på **engelsk** (eller amerikansk) og skal være velvalgte (beskrivende)
- Klasser: med **stort** CamelCase
 - eks.: `Person`, `String`, `NumberDisplay`,
- Variabler og metodenavne: med **lille** camelCase
 - eks.: `firstName`, `trackName`, `displayString`



- **Indrykning**

- Alt mellem { og } rykkes ét 'hak' ind
- For hvert ekstra niveau af parenteser rykkes endnu et 'hak' ind

```
public class Person
{
    private int age;
    public Person()
    {
        age = 32;
    }
}
```

BlueJ book

```
public class Person {
    private int age;
    public Person() {
        age = 32;
    }
}
```

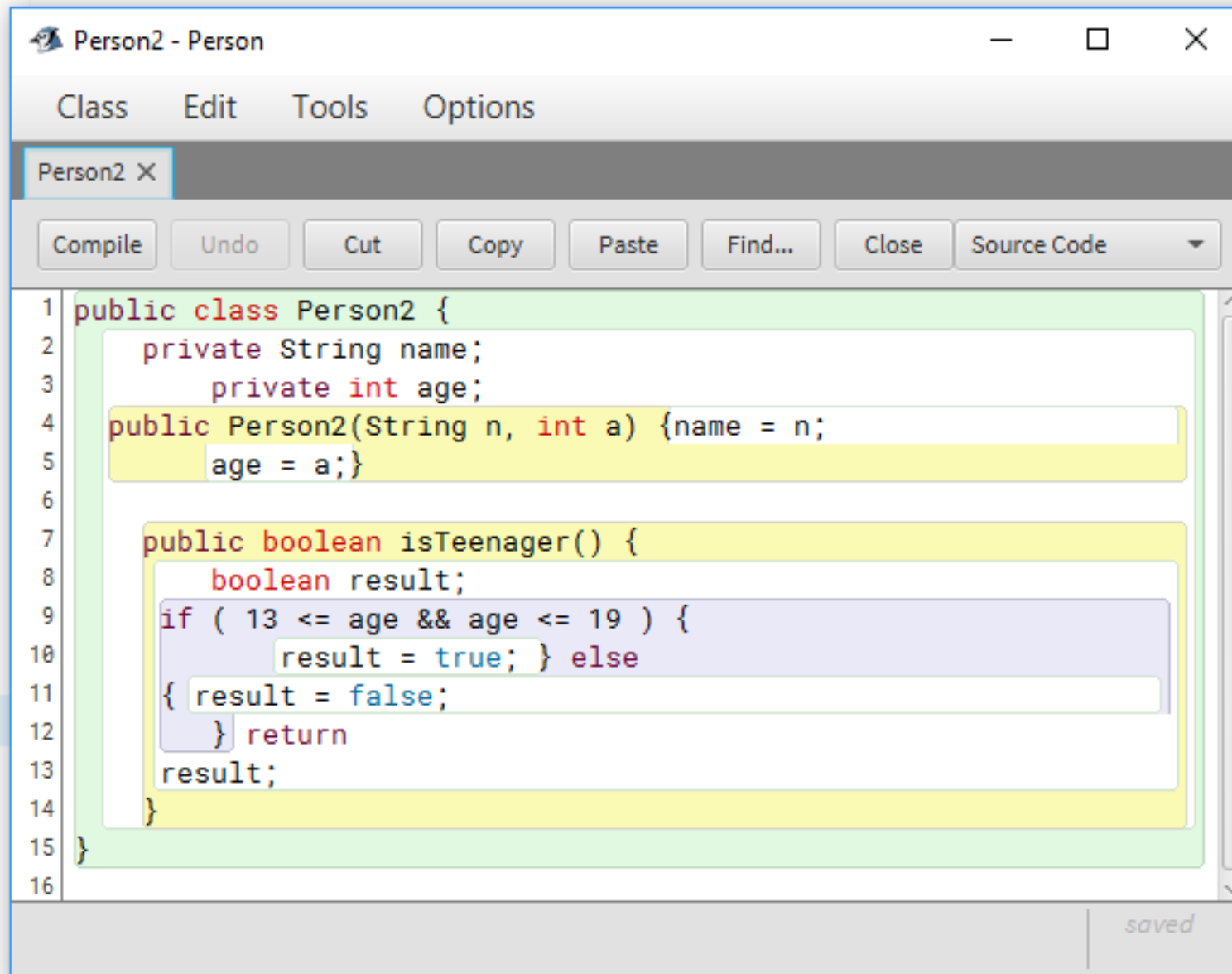
Mine slides

- **BlueJ styleguide: Appendix J**

Quiz

Åben jeres **Mentimeter** app eller gå ind på websiden **menti.com**

Hvad gør nedenstående kode?

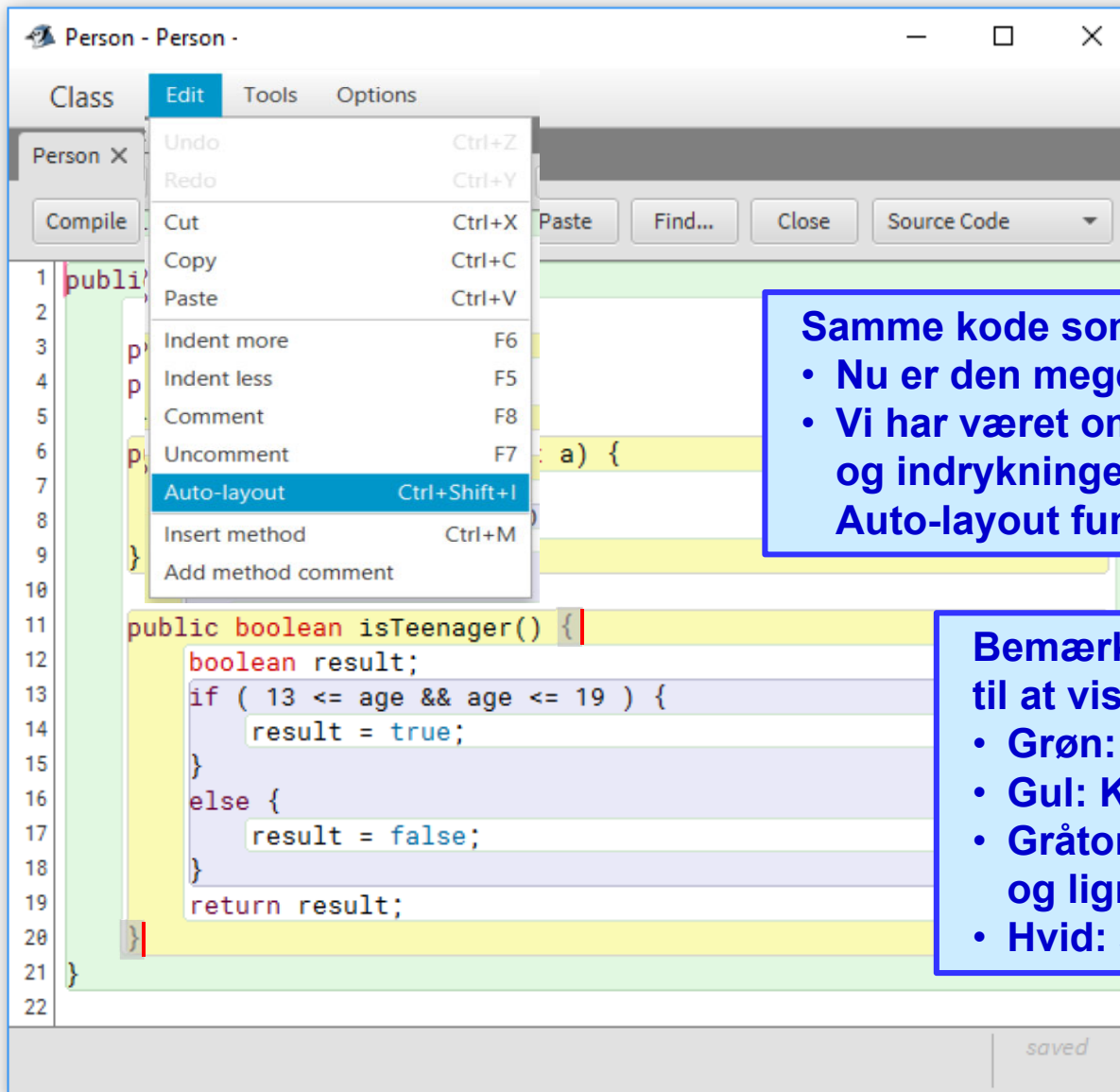


The screenshot shows a Java IDE window titled "Person2 - Person". The menu bar includes "Class", "Edit", "Tools", and "Options". The toolbar contains buttons for "Compile", "Undo", "Cut", "Copy", "Paste", "Find...", "Close", and "Source Code". The code editor displays the following Java code for the `Person2` class:

```
1 public class Person2 {  
2     private String name;  
3     private int age;  
4     public Person2(String n, int a) {name = n;  
5         age = a;}  
6  
7     public boolean isTeenager() {  
8         boolean result;  
9         if ( 13 <= age && age <= 19 ) {  
10             result = true; } else  
11             { result = false;  
12             } return  
13             result;  
14     }  
15 }  
16
```

The code defines a `Person2` class with two private attributes: `String name` and `int age`. It includes a constructor `Person2(String n, int a)` that initializes these attributes. The `isTeenager()` method returns `true` if the age is between 13 and 19, and `false` otherwise. The IDE interface also shows a "Person2 X" tab and a "saved" status at the bottom right.

Pænt layout



Samme kode som før

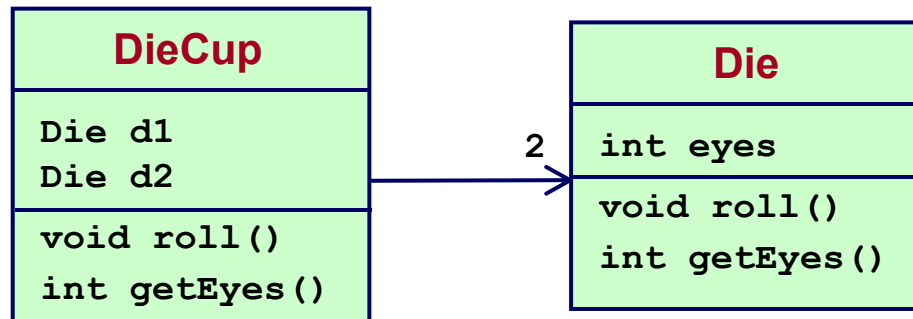
- Nu er den meget lettere at læse og forstå
- Vi har været omhyggelige med linjeskift og indrykningerne, og brugt editorens Auto-layout funktion

Bemærk at editoren bruger farver til at vise kodens komponenter

- Grøn: Klassen
- Gul: Konstruktører og metoder
- Gråtoner: if sætninger, for løkker og lignende
- Hvid: simple sætninger

● Afleveringsopgave: Raflebæger 2 (DieCup 2)

- I Raflebæger 1 lavede I et raflebæger med to terninger



- Nu skal I lave en TestDriver klasse med to metoder, der kan bruges til en mere systematisk afestning af DieCup og Die klasserne

`test()`

Skaber et raflebæger med to terninger, laver et kast med bægeret og udskriver resultatet af kastet i BlueJ's terminalvindue

`testMultiple(int noOfRolls)`

Skaber et raflebæger, laver et specificeret (positivt) antal kast og udskriver resultatet af disse i BlueJ's terminalvindue

Raflebæger 2 (DieCup 2) – fortsat

- **Dernæst skal I generalisere situationen, således at terninger nu kan have et vilkårligt antal sider (≥ 2)**
 - Det kræver bl.a. andet, at I ændrer konstruktøren for Die klassen, så den får en parameter, der angiver antallet af sider i terningen

```
// Skaber terning med noOfSides sider  
Die(int noOfSides){...}
```

- I skal også ændre konstruktøren for DieCup klassen, så den får to parametre, der angiver antallet af sider i de to terninger

```
/**  
 * Skaber et raflebæger med to terninger,  
 * hvor den første har sides1 sider og  
 * den anden sides2 sider  
 */  
DieCup(int sides1, int sides2){...}
```

- Endelig skal I tilpasse metoderne i TestDriver klassen, således at de kan anvendes til raflebægre, hvor terningerne har et variabelt antal sider

● Afleveringsopgaver om Studieteknik

- Hver af de næste fire uger vil I have en **obligatorisk afleveringsopgave om studieteknik**
 - Studievaner (uge 2)
 - Læsegruppe (uge 3)
 - Læsning (uge 4)
 - Noter (uge 5)
- Ved at bruge lidt tid på at læse om og eksperimentere med disse ting, kan I blive langt mere effektive til jeres studier
 - På den måde kan I spare tid og få mere ud af jeres anstrengelser.
- Nogle synes, at det er dumt / tidsspilde at bruge tid på den slags ting
 - Det er en **stor misforståelse**
 - God studieteknik er en væsentlig forudsætning for at blive en god og effektiv studerende – det kommer ikke af sig selv
- Opgaverne om studieteknik løses i jeres **læsegruppe**

● Opsummering

- **Sætninger**

- Simple sætninger (assignment, interne og eksterne metodekald)
- Sammensatte sætninger (blok, selektion og iteration)

- **Udtryk**

- Operatorer, overloadning og præcedens regler / brug af parenteser

- **Java styleguide**

- Navngivning
- Indrykning
- Brug af parenteser

SKAL følges

ellers kommer
instruktorerne
efter jer, og I får
genaflevering

- Her gælder den "sorte skole"
- Ingen plads til kreativitet
- I SKAL gøre som vi siger



- **Afleveringsopgaver i uge 2**

- Raflebæger 2 (par)
- Studievaner (læsegruppe)
- Quiz 2 (individuelt) – **sidst på ugen**
- Fristen for alle tre er mandag kl. 14.00 (lørdag 22.00 for IT-holdene)

- Brug eventuelt ugebrevne
- De giver en oversigt over, hvad der skal ske i ugens løb

- Skal overholdes (med mindre andet på forhånd er aftalt med instruktoren)

Programmeringscafé

- **Tilbud til studerende, som ikke tidligere har programmeret (eller kun har programmeret en lille smule)**
 - 2-3 timer om ugen
 - Det er frivilligt, om man ønsker at deltage
- **Ledes af Magnus Madsen (tenure-track adjunkt)**
 - En time hvor man programmerer i fælleskab
 - Magnus programmerer på projektoren og de studerende skriver med på egen PC
 - Diskussion af problemstillinger undervejs
 - Spørgsmål til/fra de studerende
 - En time hvor hver studerende arbejder videre på programmet, mens Magnus går rundt og hjælper
- **De der deltog sidste år siger, at det var en særdeles stor hjælp for dem**
 - Caféen gjorde det meget letter at komme i gang med afleveringsopgaverne



Programmeringscafé (fortsat)

- **Ingen forberedelse – tager kun den tid som caféen varer**
 - Intet nyt materiale – man får alt materiale via de almindelige forelæsninger, ved at læse bogen, se videoerne og deltage i øvelserne
 - Programmeringscaféen er et **supplement**, som forklarer de vigtigste principper i et langsommere tempo og med flere eksempler
 - Hvis man har let ved at programmere og/eller man er super ambitiøs, så er programmeringscaféen ikke det rigtige sted at komme
 - I stedet kan man overveje at deltage i pre talent-track, som beskrives ved en senere forelæsning
- **Tid og sted for programmeringscaféen**
 - Mandag kl. 18.15-21.00 eller onsdag kl 15.15-18.00 (man deltager kun én af gangene)
 - Finder sted i Aabogade (i nærheden af Storcenter Nord)
- **Caféen starter allerede i indeværende uge**
 - Tilmelding er i princippet slut, men hvis du er interesseret kan du kontakte Christina Sanne Gøttsche (csg@cs.au.dk)

Så er vi klar til at forlade lokalet

- **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 afsprittingsansvarlige bliver tilbage og hjælper med at afspritte alle borde og stole (dog ikke stofoverflader)
 - 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 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

