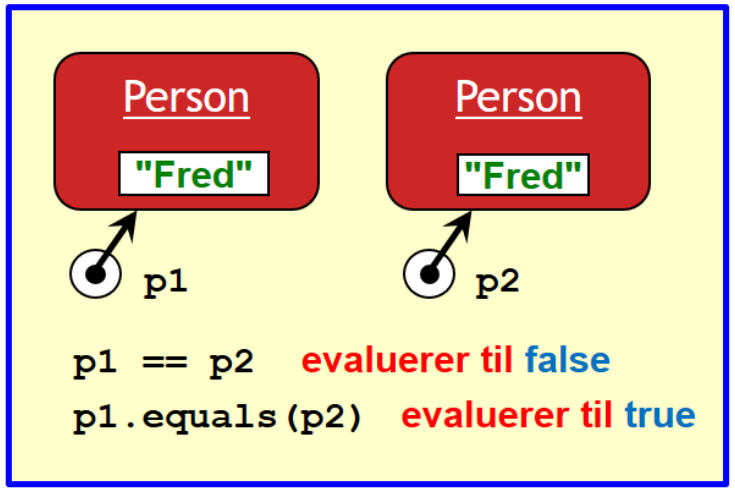
Forelæsning Uge 4 – Mandag

- **Algoritmeskabeloner**
  - Kan (ved simple tilretningerne) bruges til at **implementere metoder**, der **gennemsøger** en arrayliste (eller anden objektsamling) og finder objekter, der opfylder et givet **kriterium**
  - Fire forskellige skabeloner: findOne, findAll, findNoOf, findSumOf
- **Primitive typer (forfremmelse og begrænsning)**
- **Identitet versus lighed for objekter (herunder strenge)**
- **Afleveringsopgaver i uge 4**

- **En del studerende efterlyser mere "live" programmering ved forelæsningerne**
- **Det får I ved at se de ca. 75 videoer, der hører til kurset – de indeholder næsten alle "live" programmering**



Algoritmeskabeloner

- **Returnerer ét element, der opfylder en given betingelse**

```
public Pixel findOnePixel(int color) {
  for(Pixel p : pixels) {
    if(p.getColor() == color) {
      return p;
    }
  }
  return null;
}
```

Finder en pixel med den angivne farve

```
public Person findOnePerson(String q) {
  for(Person p : persons) {
    if(p.getName().contains(q)) {
      return p;
    }
  }
  return null;
}
```

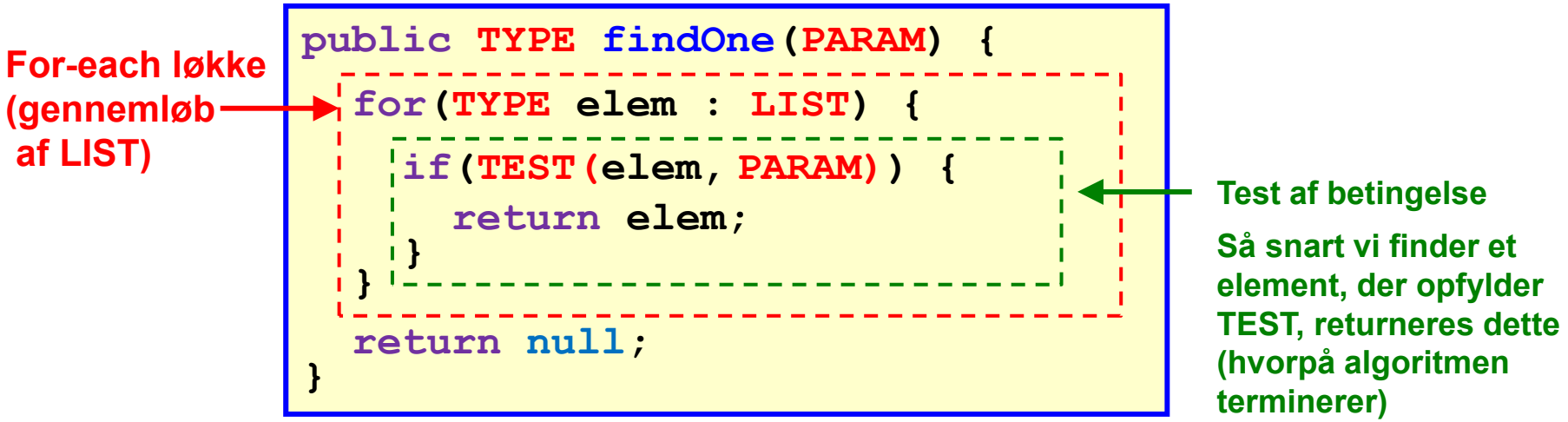
Finder en person, hvis navn indeholder den angivne tekststreng

De to metoder ligner hinanden rigtig meget

- De består begge af en if sætning inde i en for each løkke efterfulgt af return null
- Det eneste, der er forskelligt, er den betingelse, der testes, og de typer, der er involveret

Algoritmeskabelonen findOne

- **Gennemsøger en arraylisten LIST med elementer af typen TYPE og returnerer ét element, der opfylder TEST**



  - Hvis flere elementer opfylder **TEST**, returneres det første vi støder på
  - Hvis ingen elementer opfylder **TEST,** returneres **null** (den tomme pointer)

- **Algoritmeskabelon → Konkret metode**
  - Indsæt hvad de **RØDE** ting skal være (**TYPE**, **LIST, PARAM** og **TEST**)
  - Kopiér resten, som det står uden modifikationer

En anden slags metoder

- **Returnerer alle elementer, der opfylder en given betingelse**

```
public ArrayList<Pixel> findAllPixels(int color) {
  ArrayList<Pixel> result = new ArrayList<>();
  for(Pixel p : pixels) {
    if(p.getColor() == color) {
      result.add(p);
    }
  }
  return result;
}
```

Finder alle pixels med den angivne farve

```
public ArrayList<Person> findAllPersons(String q) {
  ArrayList<Person> result = new ArrayList<>();
  for(Person p : persons) {
    if(p.getName().contains(q)) {
      result.add(p);
    }
  }
  return result;
}
```

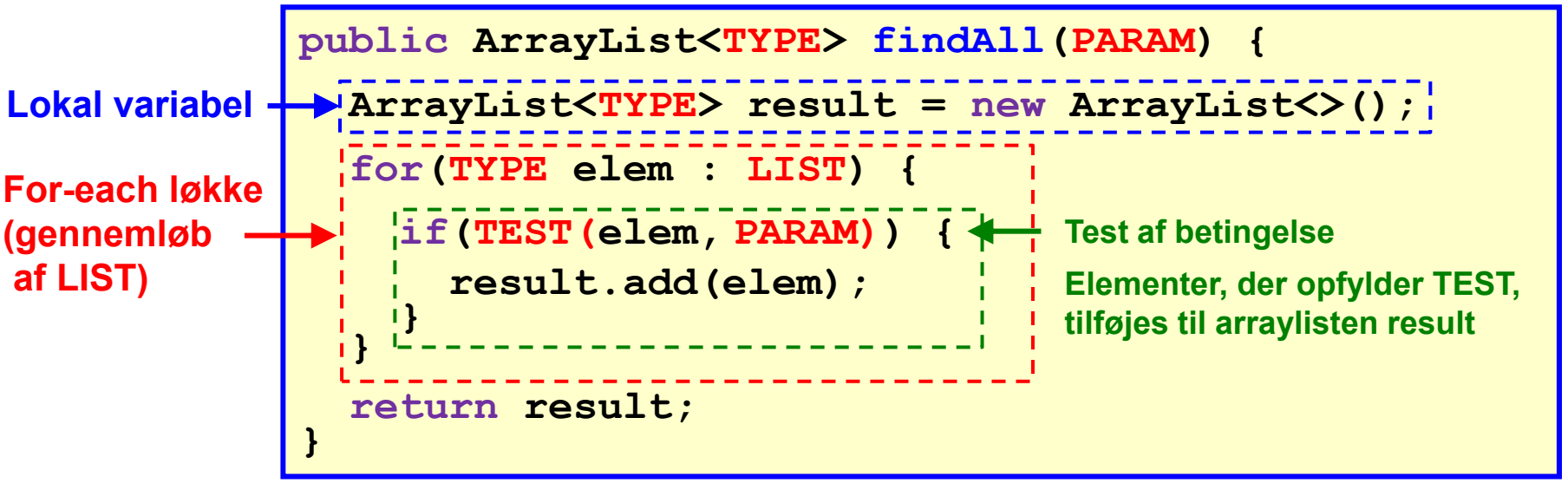
Finder alle personer, hvis navn indeholder den angivne tekststreng

De to metoder ligner hinanden rigtig meget

- Det eneste der er forskelligt er den betingelse, der testes, og de typer, der er involveret

Algoritmeskabelonen findAll

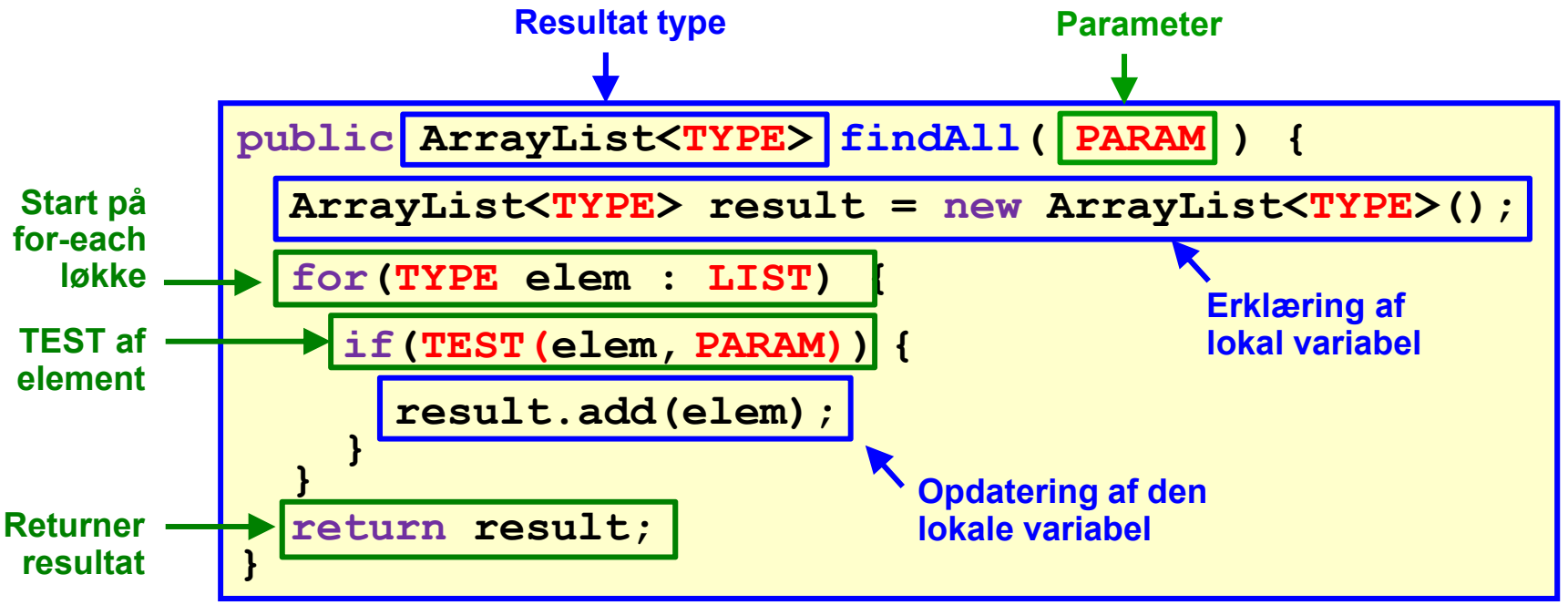
- **Gennemsøger en arrayliste LIST med elementer af typen TYPE og returnerer en arrayliste med alle elementer, der opfylder TEST**



  - Hvis ingen elementer opfylder **TEST**, returneres den tomme liste
  - Video 4.2 fra BlueJ bogen behandler et eksempel på denne algoritmeskabelon

- **Algoritmeskabelon → Konkret metode**
  - Indsæt hvad de **RØDE** ting skal være (**TYPE**, **LIST**, **PARAM** og **TEST**)
  - Kopiér resten, som det står uden modifikationer

Lad os kigge lidt nærmere på findAll



- **Lad os bibeholde de grønne dele, men udskifte de blå**
  - Det giver os to nye algoritmeskabeloner

Algoritmeskabelonen findNoOf

- Gennemsøger arraylisten LIST med elementer af typen TYPE og returnerer antallet af elementer, der opfylder TEST

```
public int findNoOf(PARAM) {
  int result = 0;          <- Erklæring
  for(TYPE elem : LIST) {
    if(TEST(elem, PARAM)) {
      result++;            <- Opdatering
    }
  }
  return result;
}
```

- Algoritmeskabelon → Konkret metode
  - Indsæt hvad de RØDE ting skal være (TYPE, LIST, PARAM og TEST)
  - Kopiér resten, som det står uden modifikationer

Algoritmeskabelonen findSumOf

- **Gennemsøger arraylisten LIST med elementer af typen TYPE og summerer værdien af de elementer, der opfylder TEST**

```
public int findSumOf(PARAM) {
  int result = 0;
  for(TYPE elem : LIST) {
    if(TEST(elem,PARAM)) {
      result += VALUE(elem, PARAM);
    }
  }
  return result;
}
```

Erklæring

Opdateringen beregner elementets værdi ved hjælp af VALUE , f.eks.
- farven i et Pixel-objekt
- længden af navnet i et Person-objekt

  - Hvis man undlader **TEST** (og fjerner if sætningen), finder man værdien af **alle** elementer i **LIST**

- **Algoritmeskabelon → Konkret metode**
  - Indsæt hvad de **RØDE** ting skal være (**TYPE**, **LIST**, **PARAM** og **TEST**)
  - Kopiér resten, som det står uden modifikationer

Sammenligning af de fire algoritmeskabeloner

- **Alle skabeloner gennemsøger en arrayliste (eller en anden objektsamling)**
  - Hvert enkelt element i listen tjekkes op mod en **angiven betingelse**
  - Betingelsen involverer **kun det element** i listen, der pt. undersøges

- **Forskelle**

  - Skobutik med en arrayliste med sko
  - Betingelsen tester skoenes farve

  - findOne returnerer **ét** element, der opfylder den angivne betingelse (og stopper så snart et sådant element er fundet) ÉN RØD SKO
  - findAll returnerer en arrayliste med **alle** de elementer, der opfylder den angivne betingelse ALLE RØDE SKO
  - findNoOf returnerer **antallet** af elementer, der opfylder den angivne betingelse ANTALLET AF RØDE SKO
  - findSumOf returnerer **summen** af værdierne af de elementer, der opfylder den angivne betingelse SAMLEDE PRIS FOR ALLE RØDE SKO

Skabelon	Lokal variabel	Initialisering	Opdatering
`findAll`	`arrayliste`	`tom liste`	`add`
`findNoOf`	`heltal`	`0`	`+= 1`
`findSumOf`	`heltal`	`0`	`+= VALUE`

Køreprøven indeholder opgaver, som kan løses ved hjælp af algoritmeskabeloner

Eksempler på findNoOf

- Returnerer antallet af elementer, der opfylder en given betingelse

```
public int findNoOfPixels(int color) {
  int result = 0;
  for(Pixel p : pixels) {
    if(p.getColor() == color) {
      result++;
    }
  }
  return result;
}
```

Finder antal pixels med den angivne farve

```
public int findNoOfPersons(String q) {
  int result = 0;
  for(Person p : persons) {
    if(p.getName().contains(q)) {
      result++;
    }
  }
  return result;
}
```

Finder antal personer, hvis navn indeholder den angivne tekststreng

Eksempler på findSumOf

- Returnerer summen af de elementer, der opfylder en given betingelse

```
public int findSumOfDarkPixels(int color) {
  int result = 0;
  for( Pixel p : pixels ) {
    if( p.getColor() <= color ) {
      result += p.getColor();
    }
  }
  return result;
}
```

Finder summen af mørke pixelværdier

```
public int findSumOfTeenagers() {
  int result = 0;
  for( Person p : persons ) {
    if( 13 <= p.getAge() && p.getAge() <= 19 ) {
      result += p.getAge();
    }
  }
  return result;
}
```

Finder summen af teenagernes alder

Primitive typer i Java

Heltal

byte ≤ short ≤ int ≤ long

8 bit 16 bit 32 bit 64 bit

≤

Reelle tal

float ≤ double

32 bit 64 bit

VI

Sandhedsværdier

boolean

1 bit

Tegn

char

16 bit

X ≤ Y angiver at et udtryk af type X kan assignes til variabler af type Y

Udtryk kan assignes til variabler hvis type er større eller lig udtrykkets type

Parameterværdier kan være mindre end parametrenes type

Eksempel:

```
double d;
int i;
```

Man må gerne assigne en "lille" værdi til en "stor" variabel

Lovligt: → `d = 7;`

int ≤ double

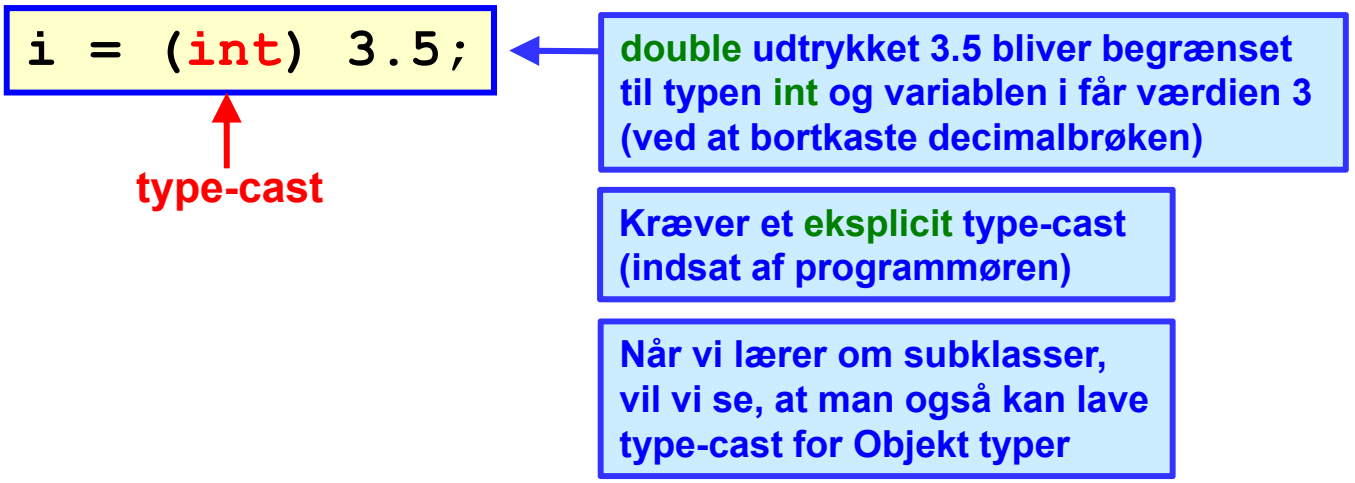
Ulovligt: → `i = 3.5;`

Man kan ikke proppe en "stor" værdi ind i en "lille" variabel

Forfremmelse og begrænsning

- En værdi kan forfremmes til en "større type"

- En værdi kan begrænses til en "mindre type"



Eksempler på forfremmelse og begrænsning

- **Hvad er værdien af dette udtryk?**

```
(int) (12 / 2.5)
(int) (12.0 / 2.5)
(int) 4.8
4
```

12 forfremmes til 12.0

4.8 begrænses til heltallet 4
ved at smide decimalbrøken væk

```
9 + 6 * 2 / (int) -3.5
```

= 5

```
6 / (int) (2 / 2.5)
```

Exception: java.lang.ArithmeticException (/ by zero)

- **Er nedenstående erklæringer lovlige?** **NEJ**

```
double d = 7;
int i = d;
```

- Oversætteren kigger kun på typerne
- De aktuelle værdier kendes først, når programmet køres

Error: incompatible types: possible lossy conversion from double to int

Konstanter og wrapper typer

Primitive typer

Type	Konstanter	Wrapper type
byte	15	Byte
short	-3215	Short
int	45320	Integer
long	45320L	Long
float	15.03e5F	Float
double	15.03e5	Double
char	'h'	Character
boolean	false	Boolean

Objekt typer

String	"hello"

Detaljer kan findes i Appendix B

String er en objekt type

- Derfor behøver den ingen wrapper type
- Tekststrenge bruges så ofte, at Java tillader, at String konstanter kan skabes ved a skrive "…" (i stedet for at bruge new operatoren)
- String objekter er imutable (deres værdi kan ikke ændres)

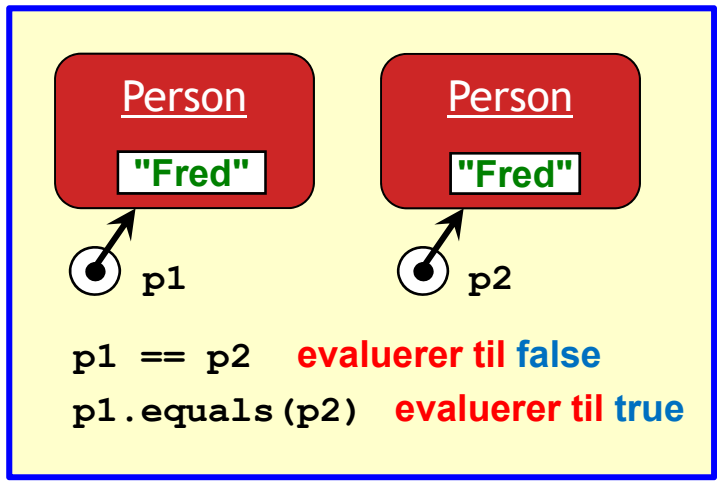
Pause

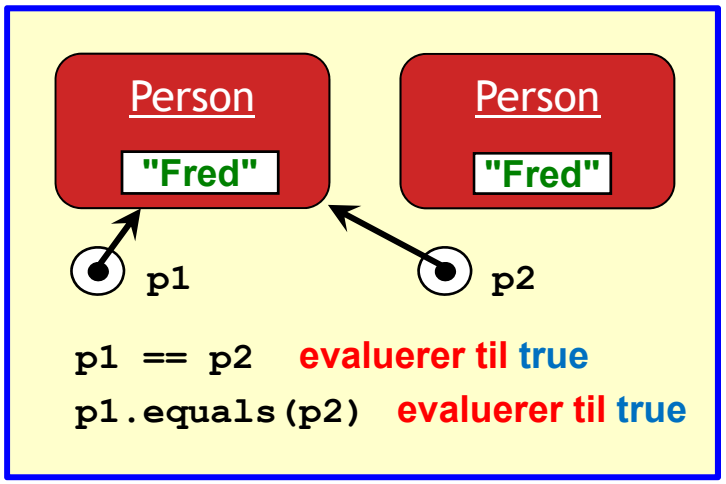
Identitet versus lighed (magen til)

- **I det virkelige liv skelner vi mellem objekter, der er identiske, og objekter, der ligner hinanden**
  - To personer er ikke identiske, selvom de hedder det samme og er født samme dag (feltvariablerne har samme værdier)
  - Hvis man fortæller tjeneren, at man vil have samme pizza som dem ved nabobordet, kommer han med en der ligner (dvs. er magen til)

Java	Type	Semantik
== operatoren	Primitiv typer	Samme værdi
	Objekt typer	Samme objekt (det er ikke nok at feltvariablerne har samme værdier)
equals metoden	Objekt typer	Lighed (feltvariablerne har samme værdier)

== operatoren versus equals metoden





Sammenligning af tekststrenge

BlueJ's code pad

```
String s1 = "Peter";
String s2 = "Petersen".substring(0,5);
s1
      "Peter"  (String)
s2
      "Peter"  (String)
s1 == s2
      false   (boolean)
s1.equals(s2)
      true   (boolean)
s2.equals(s1)
      true   (boolean)
String s3 = "Peter";
s1 == s3
      true   (boolean)
```

== operatoren tester identitet (samme objekt)

equals metoden tester lighed (magen til)

Oversætteren har fundet ud af at s1 og s3 er ens og har kun oprettet ét String objekt

Tekststrenge skal altid sammenlignes ved hjælp af equals metoden

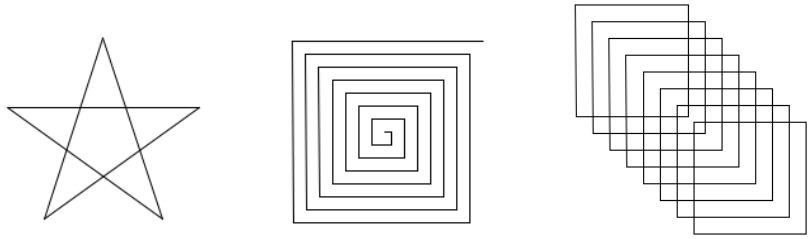
Aldrig ved hjælp af ==

Quiz

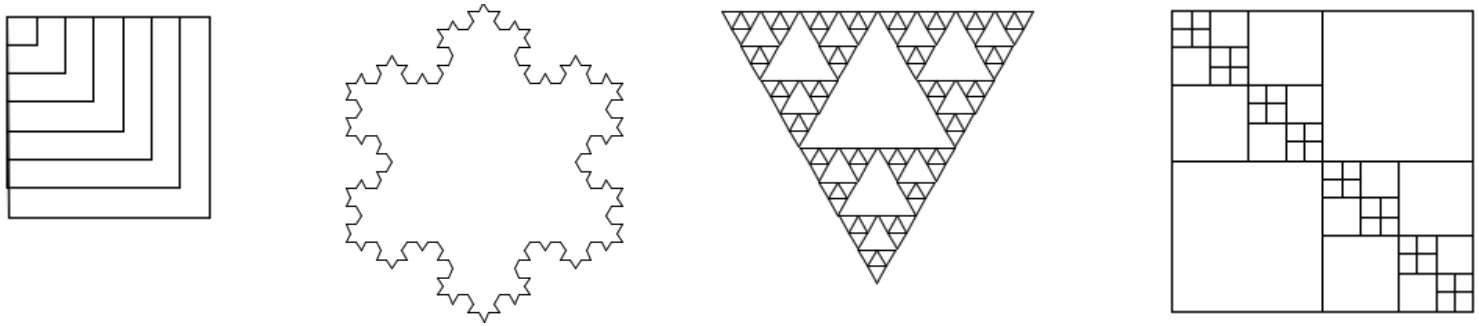
Det er OK fordi String objekter er immutable, dvs. at deres indhold ikke kan ændres

Afleveringsopgave: Skildpadde 2 (Turtle 2)

I Skildpadde 1 tegnede I forskellige figurer i stil med nedenstående

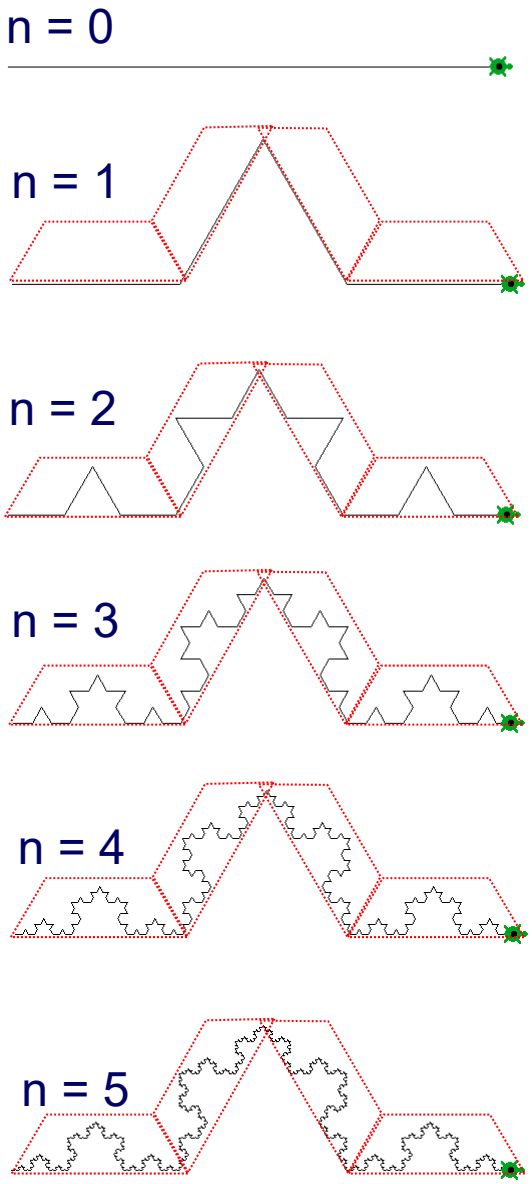


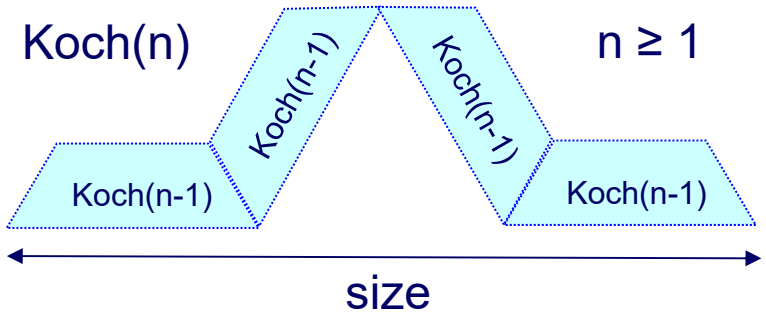
Nu skal I, ved hjælp af rekursive metoder, tegne mere komplekse figurer i stil med nedenstående



Koch kurver

En Koch kurve af grad n (hvor n ≥ 1) kan tegnes ved at tegne fire Koch kurver af grad n-1



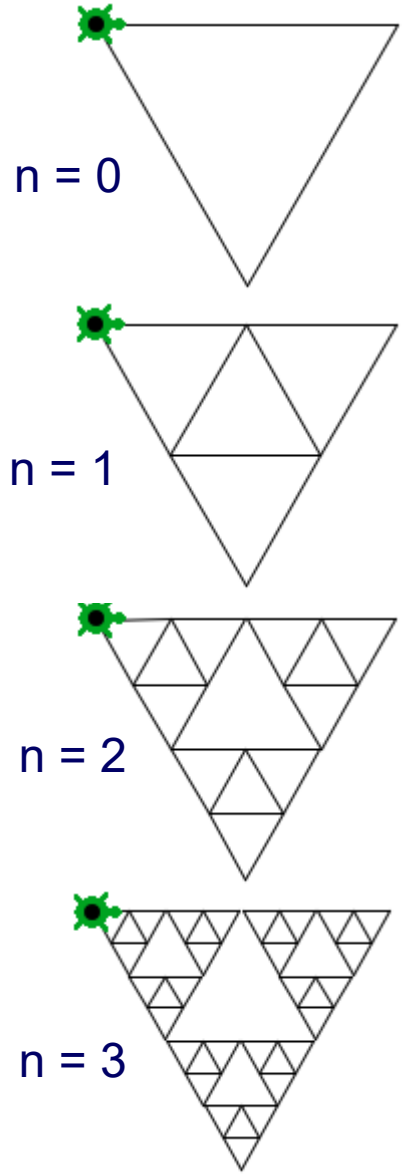


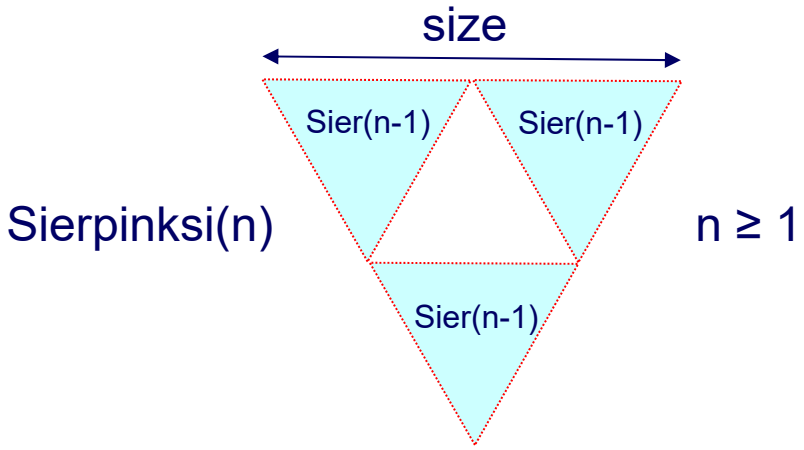
Rekursiv metode

```
public void kochCurve(int n,double size) {
  if( n >= 1 ) {
    kochCurve(n-1, ……); turn(……);
    kochCurve(n-1, ……); turn(……);
    kochCurve(n-1, ……); turn(……);
    kochCurve(n-1, ……);
  }
  else {
    move(size);
  }
}
```

Sierpinksi kurver

En Sierpinksi kurve af grad n (hvor n ≥ 1) kan tegnes ved at tegne tre Sierpinski kurver af grad n-1





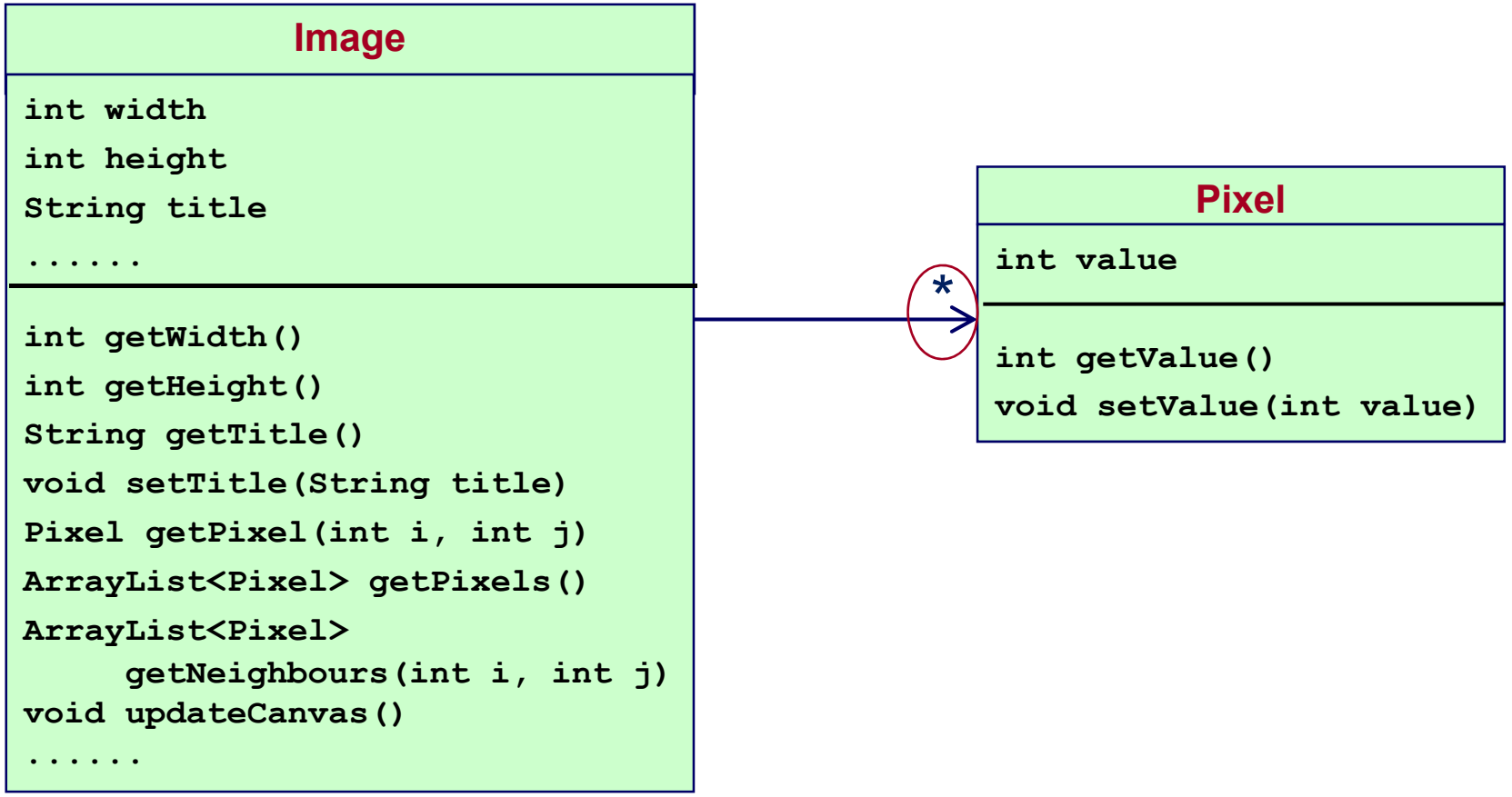
Rekursiv metode

```
public void sierpinskiCurve(int n, double size) {
  if( n >= 1 ) {
    sierpinskiCurve(n-1, ……); // Draw first triangle
    …… // Go to start position for second triangle
    sierpinskiCurve(n-1, ……); // Draw second triangle
    …… // Go to start position for third triangle
    sierpinskiCurve(n-1, ……); // Draw third triangle
    …… // Go back to start position and start angle
  }
  else {
    triangle(size);
  }
}
```

Vigtigt at man husker at gå tilbage til udgangsposition og –vinkel

Afleveringsopgave: Billedredigering

- **I får "udleveret" et projekt med nedenstående to klasser**
  - Samme som i billedredigeringsdelen af sidste forelæsning (bortset fra, at vi har tilføjet feltvariablen **title** og metoderne **getTitle** og **setTitle**)

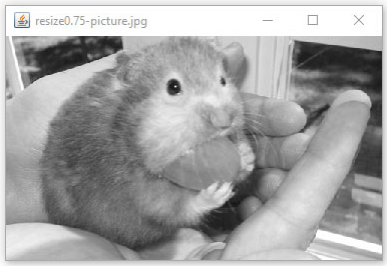


Billedredigeringsopgave – fortsat

- I skal implementere en række billedoperationer på gråtonebilleder heriblandt

Original

Lysere, mørkere, invertering



Skalering

Spejling vandret og lodret

Uskarpt (blur)

Køreprøveopgaverne

- **De opgavesæt, som vi bruger ved køreprøven, ligner hinanden til forveksling**
  - Man skal først lave en klasse, der beskriver en specificeret slags objekter (f.eks. mobiltelefoner eller pirater)
  - Dernæst skal man lave en klasse (f.eks. en webshop eller et piratskib), som ved hjælp af en arrayliste referer til et antal objekter af den første klasse
  - Så skal man lave to metoder, som arbejder på objektsamlingen (f.eks. finder en mobiltelefon i et givet prisinterval, den billigste mobiltelefon, de pirater der har mest guld eller den samlede mængde af guld hos de pirater, der er ombord på skibet)

Her er → algoritme-skabeloner nyttige

  - Dernæst skal man lave en metode, der udskriver de objekter man har (f.eks. web-shoppens ejer og de mobiltelefoner den indeholder eller piratskibets navn og de pirater der er ombord på det)
  - Til sidst skal man ved hjælp af funktionel programmering lave yderligere to metoder, som arbejder på objektsamlingen (disse to opgaver findes kun i sættene fra 2018 og frem)

Køreprøveopgaverne (fortsat)

- **Da opgaverne ligner hinanden så meget, er det ikke særligt vanskeligt at løse dem – det kræver ingen gode idéer**
  - Men til gengæld kræver det masser af træning, således, at I kan bruge Javas sprogkonstruktioner **hurtigt og korrekt**
  - I skal kunne huske, hvordan man skriver de forskellige ting i Java uden brug af hjælpemidler (bortset fra Javas klassebibliotek)
  - Hvis I ikke har trænet det igen og igen, kan I ikke nå det på de 30 minutter, der er til rådighed
- **Videoer**
  - For at hjælpe jer, har vi lavet nogle videoer, som detaljeret viser, hvordan man løser forskellige køreprøveopgaver
  - Videoerne om Phone, Pirate, Car og Turtle viser hvordan de løses ved hjælp af imperative programmering (det som I har lært indtil nu)
  - Videoerne om Penguin viser, hvordan de løses ved hjælp af funktionel programmering (som I vil lære om ved den sidste forelæsning i uge 5)
- **Det er utrolig vigtigt, at I ser disse videoer inden øvelserne i uge 5 (gerne flere gange)**

Opsummering

- **Fire algoritmeskabeloner, som alle tjekker elementer i en arrayliste op mod en angiven betingelse**
  - findOne returnerer **ét** element
  - findAll returnerer en arrayliste med **alle** elementer
  - findNoOf returnerer **antallet** af elementer
  - findSumOf returnerer **summen** af værdierne

Køreprøven indeholder opgaver, som kan løses ved hjælp af algoritmeskabeloner

- **Primitive typer**
  - Regler for assignments og parametre (bestemt via ≤ relation)
  - Forfremmelse (til større type) og begrænsning (til mindre type)
  - Konstanter og wrapper typer

Ved den første forelæsning i uge 5 vil der være mere information om køreprøven

- **Identitet versus lighed**
  - For objekter generelt
  - Tekststrenge (objekter af typen String) skal **altid** sammenlignes ved hjælp af **equals** metoden

- **Afleveringsopgaver i uge 4**

Det var alt for nu….. … spørgsmål