Kursus 02199: Programmering
afsnit 4.1-4.4, (4.6)

Anne Haxthausen

IMM, DTU

1. Metoder (erklæring og kald af) (afsnit 4.3 (+ 4.6))
2. Klasser og objekter (afsnit 4.1-4.2, 4.4)
   - hvordan definerer man en klasse?
   - hvordan skaber man et objekt?
   - hvordan virker variable af klasse typer?
   - hvordan tilgår man et objekts felter, og hvordan kalder man et objekts metoder?
3. Indkapsling vha. synlighedsmodifikatorer (public og private) (afsnit 4.2)
4. Lidt om virkefelter (afsnit 4.3)
5. Opsummering

Funktioner

En (matematisk) *funktion* tager et tal som argument og giver et tal som resultat.

Eksempel på funktion: funktionen *square* tager et tal x og giver $x \cdot x$ som resultat:

$$square(x) = x \cdot x$$

Eksempler på anvendelse af funktion:

x	*square(x)*
1.2	1.44
4.4	19.36
3.0	9.00
43.0	1849.00

Metoder i Java

Metoder i Java ligner funktioner i matematik: de kan tage argumenter og producere resultater.

En Java-metode `square` svarende til funktionen $square(x) = x \cdot x$ kan erklæres sådan her:

```
static double square(double x) {
  return x * x;
}
```

Metoden tager et *argument* `x` af type **double** og giver et *resultat* af type **double**.

Resultatet er `x` ganget med sig selv.

En metode `syvgange`, der ganger et givet tal med syv, kan erklæres sådan her:

```
static double syvgange(double x) {
  return x * 7.0;
}
```

Metoden tager et argument `x` af type **double** og giver et resultat af type **double**.

Resultatet er syv gange `x`.

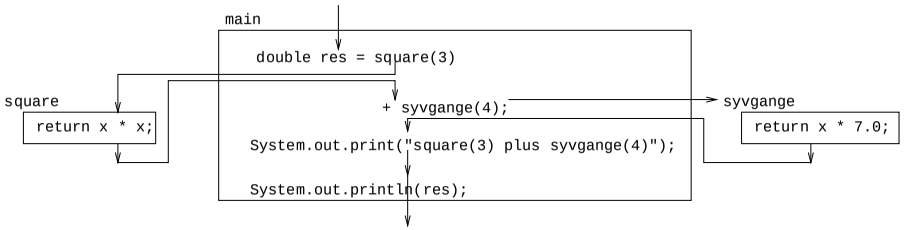
Metoder (eksempel: Metoder1.java)

En metode skal erklæres inde i en klasse.

```
public class Metoder1 {
  public static void main(String[] args) {
    double res = square(3) + syvgange(4);
    System.out.print("square(3) plus syvgange(4) er ");
    System.out.println(res);
  }
  static double square(double x) {
    return x * x;
  }
  static double syvgange(double x) {
    return x * 7.0;
  }
}
```

Et metodekald `square(3)` eller `syvgange(4)` er et udtryk. Det kan indgå i andre udtryk.

Udførelse af program med metodekald (Metoder1.java)



Generelt format for metode-erklæring:

adgang resulttattype metodenavn (*parametre*) {

sætninger og erklæringer

}

adgang er indtil videre bare **static** — senere møder vi andre muligheder.

resultattype er typen for værdier i metodens **return**-ordrer; **void** hvis ikke den returnerer noget.

metodenavn er metodens navn.

parametre er en liste af typer og navne på *formelle* parametre.

sætninger og erklæringer udgør *metodekroppen* og udføres når metoden *kaldes*.

Generelt format for metode-kald (som er et udtryk):

metodenavn (*argumenter*)

argumenter er en komma-separeret liste af argumentudtryk (også kaldet *aktuelle* parametre).

Effekt:

Argumentudtrykkene udregnes; deres værdier bindes til de formelle parametre; metodens sætninger og erklæringer udføres;

Når metoden udfører sætningen **return** *udtryk* udregnes *udtryk* og der returneres til metodekaldet; værdien af *udtryk* er da metodekaldets værdi.

Hvorfor bruge metoder?

En metode indkapsler og navngiver en samling beslægtede sætninger (og erklæringer).

Lav en ny metode:

- Hvis der er en naturlig operation som metoden svarer til.
- Hvis den metode du arbejder på ellers bliver længere end én side (40–50 linier).
  (Læs selv afsnit 4.6 om metode dekomposition.)
- Hvis det samme — eller næsten det samme — problem skal løses to steder i dit program.
  Programmering med copy-and-paste giver uoverskuelige programmer der er umulige at vedligeholde.
  Motto: hvert problem skal løses (højst) én gang!

At opfinde nye metoder og beslutte hvad deres argumenter og resultater skal være er en kreativ proces.

Det kræver evne til at generalisere og se fællestræk mellem operationer.

Klasser og objekter

Uformelt

- En *klasse* repræsenterer et begreb: tidspunkt, aftale, bil, ko, person, ...
- Et *objekt* repræsenterer en ting, en instans af et begreb: et bestemt tidspunkt, en bestemt bil, en bestemt ko, en bestemt person, ...
- En klasse har tilhørende *metoder*, der er de operationer (funktioner), der kan udføres på dens objekter.

I programmeringssproget Java

- En *klasse* er en type, svarende til `int`, `double`, `boolean`, ...
- Et *objekt* er en værdi, ligesom `17`, `18.01`, `false`, ...
- En *metode* er en operation (funktion), ligesom $+$, $-$, ...

Tre trin i brugen af klasser, objekter og metoder

1. Definer en *klasse* (inkl. dens metoder).
2. Skab objekter af klassen.
3. Brug objekterne (tilgå dets felter og kald dets metoder).

Nogle klasser (`String`) er allerede defineret i et klassebibliotek. Så kan trin 1 springes over.

Hvis en klasse (f.eks. `Keyboard`) har *statiske* metoder, kan disse bruges uden trin 2-3.

Definition af klasser

En klasse består af erklæringer af:

- data (konstanter og variable) (kaldet *felter*)
- metoder

Ved definition af en klasse gives den et navn.

Eksempel

```
class Time {
  private int hours, min; //timer og minutter siden midnat

  public Time(int h, int m) {hours = h; min = m;}
  public int getmin() { return min; }
}
```

Et objekt af klassen `Time` har felterne `hours` og `min` til at angive et tidspunkt, samt en metode `Time`, der kan bruges til at initialisere disse felter.

Skabelse af objekter

Et objekt er en instans af en bestemt klasse. Det har felter og metoder, som specificeret i klassen. Ved objektets *tilstand* forstås indholdet af dets felter. Objektets metoder kan bruges til at ændre og læse dets tilstand.

I Java kan et objekt skabes vha. **new** operatoren. Den kalder en *konstruktor* (en metode med samme navn som klassen), som initialiserer objektet.

Eksempel

new `Time(12, 35)`
skaber et objekt af klassen `Time` med `hours == 12` og `min == 35`.

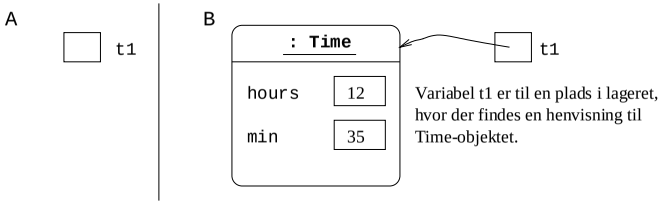
Variable af klasse typer

En variabel indeholder enten

- en primitiv værdi (når dens type er primitiv), eller
- en *reference* til et objekt (når dens type er en klasse).

Eksempel

```
Time t1;                 //A   erklaering
t1 = new Time(12, 35); //B   initialisering
```



Erklæring og initialisering kan slås sammen:

```
Time t1 = new Time(12, 35);
```


Tilgang til et objekts felter

gøres via prik-notationen:

```
t1.min = (t1.min + 2) % 60 //kun lovligt, hvis min ikke er private
```

Kald af et objekts metoder

gøres via prik-notationen:

```
... t1.getmin() ...
```


Beskrivelse af metoder i udvidet Time klasse

Antag givet Time-objekt t1.

`Time(h, m)` skaber et nyt `Time` objekt med tidspunktet givet ud fra `hours` og `min`.

`t1.gethours` returnerer for `t1` antal timer siden midnat

`t1.getmin` returnerer for `t1` antal minutter (udover hele timer) siden midnat

`t1.toString()` returner tidspunktet som f.eks. `12.27`.

`t1.passtime(m)` øger tidspunktet `t1` med `m` minutter.

`t1.plus(m)` returnerer et nyt tidspunkt, som er det øjeblikkelige tidspunkt `t1` plus `m` minutter.

`t1.to(t)` angiver hvor lang tid, der er til tidspunktet `t` fra `t1`.

`t1.before(t)` er sand, hvis `t1` er tidligere på dagen end `t`.

En implementering af Time klassen

```
class Time {
  private int hours, min; //timer og minutter siden midnat

  public Time(int h, int m) {hours = h; min = m;}

  public int gethours() {return hours ;}

  public int getmin() { return min; }

  public String toString() { return hours + "." + min; }
```

```
    public void passtime(int m) {
      int totalmin = 60 * hours + min + m;
      hours = (totalmin / 60) % 24; min = totalmin % 60;
    }

    public Time plus(int m) {
      int totalmin = 60 * hours + min + m;
      return new Time( (totalmin / 60) % 24, totalmin % 60);
    }

    public int to(Time t)
     { return 60 * t.hours + t.min - 60 * hours - min; }

    public boolean before(Time t)
     { return (hours < t.hours) ||
              (hours == t.hours && min <= t.min);
     }
  }
```


Eksempel på program, der bruger `Time`

```
public class TestofTime {
  public static void main(String[] args) {
    Time ur1, ur2;

    ur1 = new Time(9,10);
    System.out.println("ur1 viser " + ur1);

    ur2 = ur1.plus(60);
    System.out.println("ur2 viser " + ur2);

    ur1.passtime(40);
    System.out.println("efter 40 minutter viser ur1 " +ur1);
  }
}
```


Uddata fra `TestofTime`

```
ur1 viser 9.10
ur2 viser 10.10
efter 40 minutter viser ur1 9.50
```


Indkapsling

Ved *indkapsling* af et objekt forstås, at brugeren kun må tilgå objektets data gennem de metoder, som objektet tilbyder.

Eksempel `ur1.min` bør være ulovlig, men `ur1.getmin()` ok

Brugeren skal kun at kende interfacet af den tilhørende klasse:
for hver metode: navn, argument typer, resultat type, og hvad der sker, når den *kaldes*.
Ved at gemme den interne data repræsentation og algoritmer (metodekroppe) for brugeren kan vi ændre implementering uden problemer.

Eksempel

Vi kunne ombestemme os og repræsentere et tidspunkt som antal minutter siden midnat:

```
class Time {
  private int min; //minutter siden midnat
  public Time(int h, int m) {min = (h * 60 + m) % 1440;}
  ...
}
```

uden at kald af `Time` metoder i `TestofTime` klassen behøver at ændres.

Synlighedsmodifikatorerne private og public

Indkapsling i Java opnås ved brug af *synlighedsmodifikatorer* i data og metode erklæringer:

- **private**: ved ting, som kun må bruges inde i klassen.
- **public**: ved ting, som alle frit kan bruge.

Anbefaling:

- Erklær variable **private**, så objekternes tilstand indkapsles.
- Erklær hjælpe-metoder **private**.
- Erklær øvrige metoder (herunder konstruktorer) **public**.
- Konstanter, som det giver mening omgivelserne kender til, erklæres **public** (og **static**), ellers **private**.

Klassebiblioteker, pakker og import

Blev omtalt ved 2. forelæsning.

Grundbegreb: virkefelter

Definition

En variabels eller konstants *virkefelt* er den del af programmet, hvor den kan bruges (læses og skrives i).

Virkefeltsregler for variable og konstanter i Java

I Java skelnes mellem:

- *felter*: variable og konstanter erklæret i en klasse
- *lokale data*: variable og konstanter erklæret i en metode

Regler:

- *Felterne* i en klasse har virkefelt i *hele* deres klassen.
- *Lokale data* har virkefelt fra punktet *efter* deres erklæring til slutningen af den blok, hvor de er erklæret.
- *Formelle parametre* af en metode virker, som var de erklæret øverst i metode-kroppen.
- Variable erklæret i hovedet af en for-løkke har virkefelt i resten af for-løkken.
- Det er ulovligt at erklære en ny lokal variabel/konstant inde i virkefeltet for en anden lokal variabel/konstant med det samme navn.
- Det er lovligt at erklære en lokal variabel/konstant inde i virkefeltet for et felt med det samme navn. I så fald vil den lokale *skygge* feltet, så det ikke er synlig.

Virkefelter: eksempel 1

```
class Time
{
  public Time(int h, int m) { hours = h; min = m; }

  private int hours, min; //timer og minutter siden midnat

  ...

  public Time plus(int m) {
      int totalmin = 60 * hours + min + m;
      return new Time( (totalmin / 60) % 24, totalmin % 60);
  }

}
```


Virkefelter for loop variable: eksempel

```
public void loop(){
  ...
  for (int n=1;                 // n usynlig
       n<=10;                   // n synlig
       n = n + 1)               // n synlig
     System.out.println(n * n); // n synlig
  ...                           // n usynlig
}
```

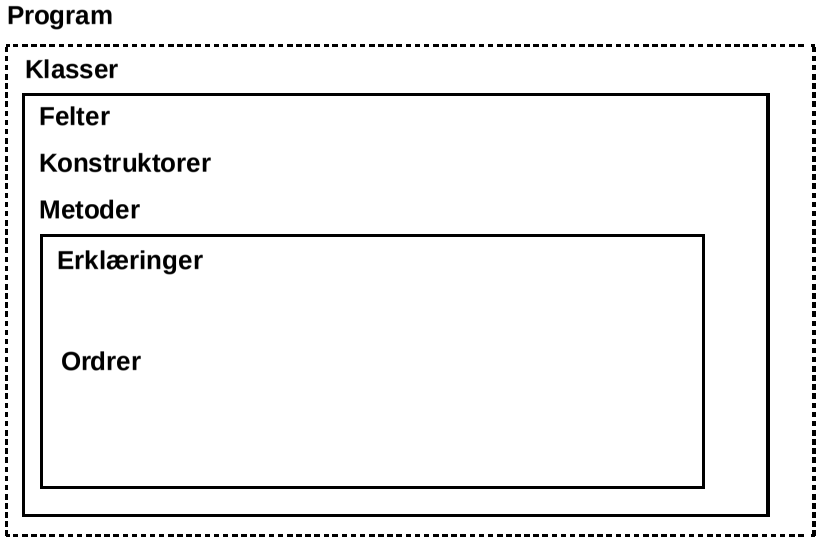

Virkefelter: eksempel på skygge

```
class Time
{
  private int hours, min; //min #1

  ...

  public Time plus(int min) { //min #2
    int totalmin = 60 * hours + getmin() + min; //min #2 synlig
    return new Time(..., totalmin % 60);        //min #2 synlig
  }
}
```


Opsummering: Java program struktur



Opsummering: klasser

To formål med klasser i Java

(1) som indpakning for nogle metoder (f.eks. `Keyboard` indpakker `readInt` m.fl.):

I så fald er alle felter og metoder **static**.

(2) som skabelon for objekter (f.eks. `Time` som skabelon for `ur1` og `ur2`):

I så fald er nogle felter og metoder ikke-**static**.

Felter i objektet indeholder objektets tilstand.
Konstruktorer initialiserer objektet (ved at initialisere felterne).
Metoder i objektet gør det muligt at ændre tilstanden eller kigge på tilstanden.

Synlighed af felter og metoder angives med **public** og **private**.

Opsummering: objekter

Et objekt er en (sammensat) værdi hørende til en bestemt klasse.

Et objekt kan *skabes* med **new** operatoren (der kalder en konstruktor fra klassen):

```
ur1 = new Time(12, 35);
```

eller med en metode, der returnerer et objekt:

```
ur2 = ur1.plus(60);
```

Man kan tilgå (aflæse eller ændre) et **public** (men ej et **private**) felt i et objekt via prik-notationen:

```
System.out.println(ur1.min);
ur1.min = 13;      //kun lovligt, hvis min ikke er private
```

Man kalder en metode i et objekt via prik-notationen:

```
ur1.passtime(40);
```


Opsummering: erklæringer af metoder

En metoder erklæres med navn, parametre og resultat type, samt krop.

Specielle metoder: konstruktorer (har samme navn som klassen, ingen resultattype)

Opsummering: kald af metoder

Generelt format for kald af metoder defineret i samme klasse

metodenavn`(` *argumenter* `)`

Eksempel (fra `Metoder1`): `syvgange(4)`

Generelt format for kald af en anden klasses statiske metoder

klassenavn.metodenavn `(` *argumenter* `)`

Eksempel: `Keyboard.readInt()`

Generelt format for kald af et objekts metoder

objektnavn.metodenavn `(` *argumenter* `)`

Eksempel: `ur1.passtime(60)`