

Ekstra opgaver 4

Niels Hallenberg

18. april 2000

Ekstra opgaver, sæt 4

Opgaverne er et supplement til løbesedlerne, og øver grundlæggende begreber gennemgået i kurset. Dette sæt fokuserer på klasser, objekter, klassefelter og kronstruktører.

1 JavaE13 — klasser og typer

```
class Punkt {
    double x,y;
}

class Pixel extends Punkt {
    private int color;
}

class Koordinat extends Pixel {
    public int kvadrant() {
        if (x>0 && y > 0)
            return 1;
        else if (x<0 && y>0)
            return 2;
        else if (x<0 && y<0)
            return 3;
        else
            return 4;
    }
}
```

Løs følgende opgaver givet programmet ovenfor.

1. Tegn et klassediagram for de tre klasser ovenfor.
2. Lad erklæringerne

```
Punkt punkt = new Pixel();
Pixel pixel = new Pixel();
Koordinat koord = new Koordinat();
```

være givet. Betragt følgende ordre. For hver ordre, angiv om ordren kan oversættes. Hvis ikke, angiv da hvorfor. Hvis ordren kan oversættes, angiv om programmet giver fejl på køretid, for eksempel ved at rejse en exception.

- (a) `pixel = punkt;`
- (b) `Pixel pixel0 = (Pixel)punkt;`
- (c) `Punkt punkt0 = pixel;`
- (d) `Punkt punkt1 = (Punkt)pixel;`
- (e) `pixel = (Punkt) punkt;`
- (f) `Pixel pixel1 = koord;`

```

(g) koord = pixel;
(h) Koordinat koord0 = (Koordinat)pixel;
(i) Punkt[] ps = {punkt, pixel, koord};
(j) Pixel[] pixs = {punkt, pixel, koord};
(k) Pixel[] pixs2 = {(Pixel)punkt, pixel};
(l) Koordinat[] koords = {pixel, koord};
(m) Koordinat[] koords2 = {(Koordinat)pixel, koord};
(n) punkt.x = 1.0;
(o) punkt.y = 2.0;
(p) punkt.color = 3;
(q) ((Pixel)punkt).color = 3;
(r) System.out.println("Farve: " + ((Pixel)punkt).color);
(s) pixel.color = 3;
(t) System.out.println("Kvadrant: " + pixel.kvadrant());
(u) System.out.println("Kvadrant: " + (Koordinat)pixel.kvadrant());
(v) System.out.println("Kvadrant: " + ((Koordinat)pixel).kvadrant());
(w) System.out.println("Kvadrant: " + koord.kvadrant());

```

2 Konstruktører

Betragt følgende klasse:

```

class Time {
    int hours, min;           // since midnight

    public Time(int hours, int min)
    { this.hours = hours; this.min = min; }

    public Time(int hours)
    { this.hours = hours; }

    public Time(Time t)
    { hours = t.hours; min = t.min; }
}

```

Betragt metoden main nedenfor. Angiv for hver ordre, om den kan oversættes. Hvis ikke, angiv da hvorfor.

```

public static void main(String[] args) {
    Time t0 = new Time(12);
    System.out.println("t0 = " + t0);
    Time t1 = new Time();
    Time t2 = new Time(12, 35);
    Time t3 = new Time(t2);
    Time t4 = new Time("12", 35);
    Time t5 = new Time(12.0, 35);
}

```

Vil print-ordren udskrive en dato på skærmen? Hvis du svarer nej, angiv da hvorfor og hvad du skal rette (tilføje) til klassen Time for at programmet udskriver en mere sigende dato?

3 JavaE14 — julefrokost

Lav en (skør) klasse som hedder `Julefrokost`. Klassen skal indeholde felter

- navn:** Festen skal have et navn
- velkomst:** Tid for velkomst.
- spisning:** Tid for spisning.
- oprydning:** Tid for oprydning efter spisning.
- dans:** Tid for dans.
- sove:** Tid for hvornår vi alle er trætte.

Du kan anvende klassen `Time` fra forelæsning 6.

Klassen skal have tre konstruktører. En der ikke tager nogen argumenter, og som opretter en standard plan for hvornår hvad sker samt en generel titel. Den anden konstruktør skal tage seks argumenter som initialiserer felterne. Hvad er de oplagte typer at anvende for hvert felt? Den tredje konstruktør skal tage et argument der er navnet på festen - de andre klasse felter initialiseres til "standardplanen". Dette kan gøres nemt ved at anvende den første konstruktør.

Klassen skal også have en metode "public" `toString` som returnerer en streng der indeholder planen for en julefrokost.

Tilføj en `main`-metode som laver en tabel af julefrokost-objekter (mindst to), og som udskriver dem.

Du kan lave felterne `public` og så ændre deres værdier i `main`-metoden; prøv dette.

3.1 Mere julefrokost

Tilføj metoden `move` til klassen `Time`:

```
public void move(int min) {
    int totalmin = 60 * this.hours + this.min + min;
    this.hours = totalmin / 60;
    this.min = totalmin % 60;
}
```

Lav følgende metoder i klassen `Julefrokost`:

- public void move(int min):** som flytter hele festplanen `min` minutter frem.
- public Julefrokost copy():** som returnerer en kopi af julefrokost-objektet.
- public void print():** som udskriver julefrokostplanen på skærmen. Bemærk, at du allerede har lavet en metode `toString`.

Prøv metoderne af i metoden `main`. Der er mange muligheder for udvidelser - nogle er svære og andre er nemme.

Du kan foreksempel lave metoder, som ændrer den tid hver aktivitet tager. For eksempel kan metoden `setTimeForVelkomst(int min)` sørge for at aktiviteten `velkomst` tager `min` minutter. Dette influerer på tiderne for hvornår de efterfølgende aktiviteter starter. Det influerer ikke på hvornår aktiviteten `velkomst` starter!

Du kan også lave en metode `setVelkomst(Time t)` som sætter starttidspunktet for aktiviteten `velkomst` uden at ændre de andre aktiviteter. Dette er meget nemmere end metoden `setTimeForVelkomst`.

4 JavaE15 — julefrokost på IT-højskolen

Klassen `Julefrokost` indeholder en generel skabelon for hvordan en julefrokost ser ud. På ITU har vi to julefrokoster. Lav en klasse `ITUJulefrokost` som er en udvidelse af `Julefrokost`. En julefrokost på ITU er kendetegnet ved at det er personer relateret til ITU som deltager. Vi vil holde styr på hvilke personer som er tilmeldt festen.

Vi lader en deltager være repræsenteret som en `ITUPerson`, se opgave Java62 på løbeseddel 8.

Klassen `ITUJulefrokost` skal indeholde følgende felter:

- max_deltagere:** Det maksimalt antal deltagere som kan deltage i festen.

ITUperson[] deltagere: En tabel af størrelse `max_deltagere` som indeholder hver deltager.
antal_tilmeldt: Angiver antal personer som pt. er tilmeldt festen.

Klassen `ITUjulefrokost` skal indeholde følgende konstruktører:

ITUjulefrokost(): Vi lader maksimalt 100 personer deltage, og vi anvender en standard julefrokostplan.

ITUjulefrokost(int n): Vi lader `n` personer deltage i julefrokosten. Derudover anvender vi en standard julefrokostplan.

Du kan udvide med flere konstruktører, som giver mulighed for at lave om på standardplanen.

Klassen `ITUjulefrokost` skal også indeholde en metode som kan tilmelde deltager. Lav en metode

```
public void tilmeld(ITUperson p)
```

som tilmelder person `p` til festen. Person `p` skal indsættes i tabellen `deltagere`. Pladsen i tabellen er givet ved feltet `antal_tilmeldt`, som skal tælles en op bagefter. Du bør checke, at der stadig er plads til en ny deltager.

Vi bør også have en metode `print_deltagere` som udskriver en deltagerliste på skærmen. Bemærk, at klassen `ITUperson` allerede har en metode `toString`.

Den sidste funktionalitet vi mangler er at kunne framelde personer til festen. Dette er lidt sværere idet vi gerne vil undgå "huller" i deltager tabellen `deltagere`. Betragt følgende tabel med tre deltagere

```
{p1, p2, p3}
```

og vi gerne vil framelde `p2`, det vil sige person på plads 1 i tabellen. Da, for at undgå "huller" i tabellen skal vi flytte `p3` fra felt 2 til felt 1 i tabellen således at vi får følgende tabel:

```
{p1, p3}
```

Lav metoden

```
void frameld(int n)
```

hvor `n` er indekset i tabellen `deltagere` for den person som skal frameldes.

4.1 Private og Public

Det er god programmeringsskik at lade felter være `private`. Ret klasserne `Julefrokost` og `ITUjulefrokost` så felterne er `private`. Idet felterne er `private` er det nødvendigt at tilføje metoder som kan opdatere og læse indholdet af de forskellige felter, f.eks. `setVelkomst` og `getVelkomst`.

5 Diverse

1. Kan man skrive følgende:

```
Object obj = "hek";  
obj = new Time(12, 45);
```

givet at klassen `Time` er tilgængelig.