

**Skriftlig eksamen i  
Grundlæggende Programmering**

IT-højskolen i København, 11. juni 2001

*Alle hjælpemidler tilladt, dog ikke datamat.*

*Eksamen er skriftlig, fire timer, og bedømmes efter 13-skalaen.*

*Opgavesættet består af tre opgaver der alle ønskes løst.*

*Opgave 1 vægtes med 25 %, opgave 2 med 35%, og opgave 3 med 40%.*

*Hvis du i besvarelsen benytter klasser og metoder fra bogen og noterne, så behøver du ikke at skrive dem af, men du skal give en præcis henvisning med afsnitsnummer eller sidetal.*

*The exam questions are available also in English (separately).*

*Eksamensspørgsmålene findes også på engelsk (på separate ark).*

*Et godt råd: Gennemlæs opgavesættet inden du begynder at besvare de enkelte opgaver.*

**Opgave 1 (25 %)****Opgave 1.1**

Vis hvad dette Java-program udskriver på skærmen, når det køres:

```
class opg1_1 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 4; i >= 1; i=i-1) {
            sum = sum + 2*i;
            System.out.println(sum);
        }
    }
}
```

**Opgave 1.2**

Vis hvad dette Java-program udskriver på skærmen, når det køres:

```
class opg1_2 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < 6; i=i+1) {
            for (int j = 2; j < i; j=j+1) {
                sum = sum + j;
            }
            System.out.println(sum);
        }
    }
}
```

**Opgave 1.3**

Skriv en Java-metode `static void linie(int m, int s)`, som på skærmen udskriver en linie der indeholder `m` mellemrum efterfulgt af `s` stjerner, efterfulgt af linieskift.

For eksempel skal de to metodekald `linie(1, 3)` og `linie(0, 5)` tilsammen give følgende udskrift:

```
***
*****
```

**Opgave 1.4**

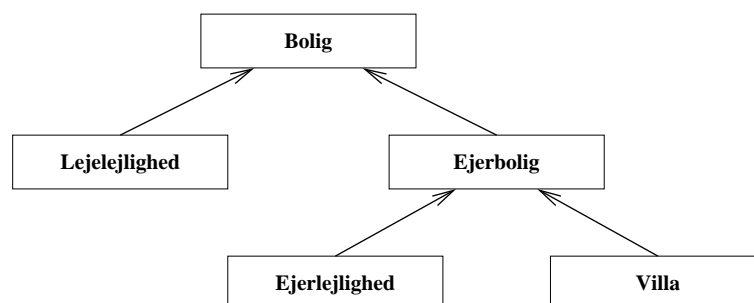
Skriv en Java-metode `static void diamant(int n)`, som på skærmen udskriver en diamant bestående af stjerner. Diamanten skal bestå af  $2*n+1$  linier, og den længste linie skal indeholde  $2*n+1$  stjerner.

For eksempel skal `stjerne(2)` udskrive en diamant bestående af  $2*2+1 = 5$  linier på skærmen, som følger:

```
  *
 ***
*****
 ***
  *
```

**Opgave 2 (35 %)**

I denne opgave skal du lave et program der modellerer et boligregister. Der findes en række forskellige typer af boliger og det vil vi her modellere ved brug af et klassehierarki af denne form:



Det vil sige at Lejelejlighed og Ejerbolig er subclasses af klassen Bolig, og at Ejerlejlighed samt Villa er subclasses af Ejerbolig. Klassen Bolig ser ud som vist her:

```
class Bolig {
    int areal;
    String kommune;

    Bolig(int areal, String kommune) {
        this.areal = areal; this.kommune = kommune;
    }

    public void print_info() {
        System.out.println("areal: " + areal);
        System.out.println("kommune: " + kommune);
    }
}
```

**Opgave 2.1**

En ejerbolig er en bolig, som har en kontantpris (et heltal) og en månedlig nettohusleje (et heltal). Skriv klassen Ejerbolig som en subclasse af Bolig. Klassen skal have en konstruktor der initialiserer alle dens felter (arealet, kommunen, kontantprisen, nettohuslejen). Metoden `print_info` fra superklassen Bolig skal overskrives således at metoden ikke blot udskriver arealet og kommunen, men også kontantprisen og nettohuslejen.

**Opgave 2.2**

En ejerlejlighed er en ejerbolig, som kan have en elevator, angivet ved sandhedsværdien af en boolean `elevator` (`true` hvis der er en elevator, `false` ellers). Skriv klassen Ejerlejlighed som en subclasse af Ejerbolig. Klassen skal have en konstruktor der initialiserer alle dens felter (arealet, kommunen, kontantprisen, nettohuslejen, elevator). Overskriv metoden `print_info` således at alle felterne udskrives.

**Opgave 2.3**

En lejelejlighed er en bolig, som har en månedlig husleje (et heltal), og som kan have en elevator, angivet ved sandhedsværdien af en boolean `elevator`. Skriv klassen Lejelejlighed som en subclasse af Bolig. Klassen skal have en konstruktor der initialiserer alle dens felter (arealet, kommunen, huslejen, elevator). Overskriv metoden `print_info` således at felterne udskrives.

**Opgave 2.4**

Betragt følgende skelet til et program med en main-metode, der opretter et boligregister med fire boliger:

```
class Boligopgave {
    public static void main(String[] args) {
        Bolig bb = new Bolig(120, "København");
        Villa bv = new Villa(200, "Gentofte", 3000000, 20000, 800);
        Ejerlejlighed be = new Ejerlejlighed(100, "Århus", 2000000, 12000, false);
        Lejelejlighed bl = new Lejelejlighed(65, "Odense", 2500, false);

        Bolig[] boligregister = {bb,bv,be,bl};

        // Udskriv om bolig nummer 3 har en elevator // pp1
        // Udskriv information om alle boliger i boligregistret // pp2
        // Tæl og udskriv antallet af ejerboliger i boligregistret // pp3
    }
}
```

Ved programpunktet pp1 ovenfor ønsker man at udskrive om bolig nummer 3, dvs. `boligregister[3]`, har en elevator eller ej. Bemærk at `boligregister[3]` peger på et objekt af klassen `Lejelejlighed`, og at dette objekt altså har et elevator felt.

Her er to forslag pp1a og pp1b til ordrer i programpunkt pp1:

```
System.out.println("boligregister[3].elevator: " + // pp1a
    boligregister[3].elevator);

System.out.println("boligregister[3].elevator: " + // pp1b
    ((Lejelejlighed)boligregister[3]).elevator);
```

Hvilket eller hvilke af de ovenstående to forslag pp1a og pp1b bliver accepteret af Java-oversætteren hvis det ind sættes ved programpunkt pp1 ? Giv et kort begrundet svar.

**Opgave 2.5**

Tilføj noget programtekst til programpunkt pp2, som udskriver information om alle boligerne i boligregistret.

**Opgave 2.6**

Tilføj noget programtekst til programpunkt pp3, som tæller og udskriver antallet af ejerboliger i boligregistret.

Vink: Benyt Javas indbyggede operator `instanceof`. Udtrykket `(e instanceof K)` er `true` hvis objektet `e` tilhører klassen `K` eller en subclasses af `K`, og ellers er udtrykket `false`. Eksempel: når `bv` er et objekt af klasse `Villa`, så er udtrykkene `(bv instanceof Villa)` og `(bv instanceof Ejerbolig)` og `(bv instanceof Bolig)` alle sammen `true`, men `(bv instanceof Lejelejlighed)` er `false`.

**Opgave 2.7**

En lejlighed er en bolig, som har et areal (et heltal), en kommune (en `String`), samt eventuelt en elevator (en `boolean`). Skriv et interface `Lejlighed` som beskriver metoderne `get_areal`, `get_kommune`, `get_elevator`, samt `print_info`. Modificér klasserne `Lejelejlighed` og `Ejerlejlighed` således at de implementerer interfacet `Lejlighed`. Du behøver kun at angive ændringerne i forhold til tidligere og det er nok at vise løsningen for én af de to klasser.

**Opgave 2.8**

Med disse ændringer giver det mening at udvide ovenstående main-metode (efter pp3) med en erklæring af en variabel `lejlighedsregister`, som er en tabel hvis elementer er af type `Lejlighed`. For eksempel:

```
Lejlighed[] lejlighedsregister = {be,bl};
```

Vis hvordan main-metoden kan udvides til at finde og udskrive information om den lejlighed i en sådan variabel `lejlighedsregister` som har det største areal.

### Opgave 3 (40 %)

En afstandstabel kan for eksempel bruges til at angive hvor langt der er mellem nogle byer på Sjælland. For at lette programmeringen af afstandstabeller nummererer vi byerne 0, 1, ..., og vi kan få en afstandstabel der ser ud som neden for:

| By          | Nummer |
|-------------|--------|
| København   | 0      |
| Køge        | 1      |
| Hillerød    | 2      |
| Vordingborg | 3      |
| Slagelse    | 4      |
| Hundested   | 5      |
| Kalundborg  | 6      |

|   |     |    |     |     |     |     |
|---|-----|----|-----|-----|-----|-----|
|   | 0   |    |     |     |     |     |
| 1 | 42  | 1  |     |     |     |     |
| 2 | 41  | 72 | 2   |     |     |     |
| 3 | 97  | 60 | 128 | 3   |     |     |
| 4 | 92  | 56 | 97  | 64  | 4   |     |
| 5 | 67  | 80 | 34  | 136 | 106 | 5   |
| 6 | 103 | 85 | 99  | 100 | 37  | 108 |

Afstanden fra Slagelse (by nummer 4) til Køge (by nummer 1) fås ved at gå ind i række nummer 4 og hen til søjle nummer 1, hvor afstanden kan aflæses til 56 km.

Denne opgave handler om en Java-klasse `AfstandsTabel`, der kan indeholde navnene på nogle byer, og den tilsvarende afstandstabel.

Til at begynde med antager vi at klassen `AfstandsTabel` har en metode `int afstand(int by1, int by2)`, som giver afstanden mellem to byer. For eksempel giver kaldet `afstand(4, 1)` afstanden mellem Slagelse og Køge. Metoden `afstand` vil ligeledes give afstanden mellem Slagelse og Køge ved kaldet `afstand(1, 4)`, altså rækkefølgen af de to byer er uden betydning.

Desuden antager vi at klassen har en metode `bynummer`, som givet et bynavn returnerer byens nummer. Metoden skal have følgende erklæring: `int bynummer(String bynavn)`.

#### Opgave 3.1

Skriv en metode `rutelængde`, som tager en tabel af bynumre og beregner længden af en rejse gennem disse byer (i den givne rækkefølge). Et kald kunne f.eks. være

```
int[] minRute = {0, 1, 4, 6}; // København, Køge, Slagelse, Kalundborg
int længde = rutelængde(minRute);
```

Der skal altså laves en metode med erklæringen `int rutelængde(int[] byer)`. Hvis man giver den en rute (en tabel af bynumre) med 0 eller 1 element skal den returnere 0 km. Metoden skal bruge metoden `afstand`, omtalt ovenfor.

#### Opgave 3.2

Lav en `rutelængde`-metode der i stedet tager en tabel af bynavne som argument. Den har altså følgende erklæring: `int rutelængde(String[] byer)`. Du må gerne bruge metoden `bynummer`, omtalt ovenfor. Du må også gerne bruge metoden `rutelængde` fra opgave 3.1, også selvom du ikke kunne løse den opgave.

#### Opgave 3.3

Antag nu at metoden `bynummer` kaster en exception af klassen `UkendtBy` hvis man beder om en by der ikke er med i afstandstabellen. Metoden er altså erklæret som `int bynummer(String bynavn) throws UkendtBy`. Det er *ikke* en del af denne opgave at skrive metoden `bynummer` eller erklære klassen `UkendtBy`.

Omskriv metoden `rutelængde` fra opgave 3.2 så den fanger en sådan exception, og returnerer `-1` (altså minus én) som `rutelængde` hvis der er en ukendt by i ruten. Hvis du ikke har lavet opgave 3.2, angiver du blot med en kommentar hvor i din besvarelse du anvender løsningen fra opgave 3.2.

**Opgave 3.4**

Skriv nu erklæringen `class Afstandstabel { ... }` af selve klassen `Afstandstabel`, idet du her kun skal medtage følgende aspekter af klassen:

- Klassen skal indeholde denne konstruktor:

```
AfstandsTabel(String[] byNavne){
    antalByer = byNavne.length;
    afstande = new int[antalByer][];
    for (int i=1; i<antalByer; i=i+1)
        afstande[i] = new int[i];
    byer = byNavne;
}
```

Du skal angive korrekte erklæringer af de tre felter `antalByer`, `afstande` og `byer`, sådan at ovenstående konstruktor kan oversættes og virker.

- Du skal skrive en metode `void sætAfstand(int by1, int by2, int afst)`, der indsætter afstanden mellem to byer i tabellen `afstande`. Man skal forestille sig at en række kald til denne metode bruges til at udfylde afstandstabellen. Metoden skal ikke indsætte nogen afstand hvis `by1` er lig `by2` (for så er afstanden jo nul, og der er ingen grund til at gemme den i tabellen).
- Du skal skrive metoden `int afstand(int by1, int by2)` som beskrevet i indledningen til opgave 3.

**Opgave 3.5**

Skriv metoden `int bynummer(String bynavn)`, som givet et bynavn returnerer byens nummer. Metoden skal tilhøre klassen `AfstandsTabel` og skal benytte klassens tabel `byer`. Det er ligegyldigt hvad der sker når det angivne bynavn ikke findes (metoden behøver altså ikke kaste exception `UkendtBy` som antaget i delopgave 3.3).

**Opgave 3.6**

Tilføj en metode `findNærmeste` til klassen `AfstandsTabel`, som finder den nærmeste naboby til en given by. For eksempel er Hundesteds nærmeste naboby Hillerød, så `findNærmeste(5)` skal give 2. Metoden skal have følgende erklæring: `int findNærmeste(int by)`, og skal returnere nummeret på den by der ligger nærmest på byen med nummer `by`.