

Model solutions to
Written exam in
Introductory Programming

IT-C, June 11, 2001

Question 1

Question 1.1

The program prints the following four lines:

```
8
14
18
20
```

Question 1.2

The program prints the following six lines:

```
0
0
0
2
7
16
```

Question 1.3

The problem is solved using two for-loops:

```
static void linie(int m, int s) {
    for (int k=0; k<m; k=k+1)
        System.out.print(" ");
    for (int k=0; k<s; k=k+1)
        System.out.print("*");
    System.out.println();
}
```

Question 1.4

Using the above method, we can print the upper n lines, then the middle line, then the lower n lines:

```
static void diamant(int n) {
    for (int i=0; i<n; i=i+1)
        linie(n-i, 2*i+1);
    linie(0, 2*n+1);
    for (int i=n-1; i>=0; i=i-1)
        linie(n-i, 2*i+1);
}
```

But it could also be done using only one for-loop, with i going from -n to n, and taking appropriate absolute values:

```
static void diamant(int n) {
    for (int i=-n; i<=n; i=i+1)
        linie(Math.abs(i), 2*(n-Math.abs(i))+1);
}
```

Question 2

Question 2.1

The constructor in class Ejerbolig should call the superclass constructor, and the overriding print_info method should call print_info from the superclass:

```
class Ejerbolig extends Bolig {
    int kontantpris;
    int nettohusleje;

    Ejerbolig(int areal, String kommune, int kontantpris, int nettohusleje) {
        super(areal, kommune);
        this.kontantpris = kontantpris; this.nettohusleje = nettohusleje;
    }

    public void print_info() {
        super.print_info();
        System.out.println("kontantpris: " + kontantpris);
        System.out.println("nettohusleje: " + nettohusleje);
    }
}
```

Question 2.2

```
class Ejerlejlighed extends Ejerbolig {
    boolean elevator;

    Ejerlejlighed(int areal, String kommune, int kontantpris, int nettohusleje,
                 boolean elevator) {
        super(areal, kommune, kontantpris, nettohusleje);
        this.elevator = elevator;
    }

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

Question 2.3

```
class Lejelejlighed extends Bolig {
    int husleje;
    boolean elevator;

    Lejelejlighed(int areal, String kommune, int husleje, boolean elevator) {
        super(areal, kommune);
        this.husleje = husleje; this.elevator = elevator;
    }

    public void print_info() {
        super.print_info();
        System.out.println("husleje: " + husleje);
        System.out.println("elevator: " + elevator);
    }
}
```

Question 2.4

Only the latter (pp1b) statement will be accepted by the compiler. The reason is that `boligregister` has compile-time type `Bolig[]`, so that `boligregister[3]` has compile-time type `Bolig`, but class `Bolig` has no `elevator` field. Hence the cast to class `Lejelejlighed` is necessary for the compiler to 'see' that the object referred to by `boligregister[3]` has an `elevator` field.

Question 2.5

```
for (int i=0; i<boligregister.length; i=i+1)
    boligregister[i].print_info();
```

Question 2.6

```
int ejerboliger = 0;
for (int i=0; i<boligregister.length; i=i+1)
    if (boligregister[i] instanceof Ejerbolig)
        ejerboliger = ejerboliger + 1;
System.out.println("Antal ejerboliger er " + ejerboliger);
```

Question 2.7

```
interface Lejlighed {
    int get_areal();
    String get_kommune();
    boolean get_elevator();
    void print_info();
}

class Ejerlejlighed extends Ejerbolig implements Lejlighed {
    boolean elevator; /* as before */

    Ejerlejlighed(...) { /* as before */ }

    public void print_info() { /* as before */ }

    public int get_areal() { return areal; }
    public String get_kommune() { return kommune; }
    public boolean get_elevator() { return elevator; }
}
```

Question 2.8

```
int j = 0;
for (int i = 1; i < lejlighedsregister.length; i++) {
    if ((lejlighedsregister[i].get_areal()) >
        (lejlighedsregister[j].get_areal()))
        j = i;
}
System.out.println("Lejlighed med størst areal er:");
lejlighedsregister[j].print_info();
```

Question 3

Question 3.1

```
int rutelængde(int[] byer) {
    int længde = 0;
    for (int i=1; i<byer.length; i=i+1)
        længde = længde + afstand(byer[i-1], byer[i]);
    return længde;
}
```

Question 3.2

```
int rutelængde(String[] byer) {
    int længde = 0;
    for (int i=1; i<byer.length; i=i+1)
        længde = længde + afstand(bynummer(byer[i-1]), bynummer(byer[i]));
    return længde;
}
```

Question 3.3

```
int rutelængde(String[] byer) {
    try {
        int længde = 0;
        for (int i=1; i<byer.length; i=i+1)
            længde = længde + afstand(bynummer(byer[i-1]), bynummer(byer[i]));
        return længde;
    } catch (UkendtBy e) {
        return -1;
    }
}
```

Question 3.4

```
class AfstandsTabel {
    int antalByer;
    int[][] afstande;
    String[] byer;

    AfstandsTabel(String[] byNavne){ /* as shown in the question */ }

    void sætAfstand(int by1, int by2, int afst) {
        if (by1 > by2)
            afstande[by1][by2] = afst;
        else if (by2 > by1)
            afstande[by2][by1] = afst;
    }

    int afstand(int by1, int by2) {
        if (by1 > by2)
            return afstande[by1][by2];
        else if (by2 > by1)
            return afstande[by2][by1];
        else
            return 0;
    }
}
```

Question 3.5

This methods throws `ArrayIndexOutOfBoundsException` if `bynavn` is not found:

```
int bynummer(String bynavn) {
    int bynr = 0;
    while (!bynavn.equals(byer[bynr]))
        bynr = bynr + 1;
    return bynr;
}
```

Question 3.6

We assume there are at least two towns; otherwise it is not possible to find a nearest neighbour (in which case the following method will return -1):

```
int findNærmeste(int by) {
    int nmstafst = 1000000; // Larger than the diameter of Sjælland
    int nmstby = -1;
    for (int by2=0; by2<antalByer; by2=by2+1)
        if (by2 != by && afstand(by, by2) < nmstafst) {
            nmstafst = afstand(by, by2);
            nmstby = by2;
        }
    return nmstby;
}
```