

**Written exam in**

**Introduction to Programming**  
**(Grundlæggende Programmering)**

IT-C, January 16th, 2002  
English version

*The is a 4 hour written open-book exam. It is graded according to the Danish 13-scale. Use of computers or electronic communications devices during the exam is prohibited.*

*The exam has 8 pages and consists of 4 questions, which are weighted according to the given percentages. All questions are obligatory.*

*If your exam answers make use of classes or methods from the textbook (Lewis and Loftus, Java Software), lecture notes or lecture slides you do not need to copy them, but can use precise references to them consisting of edition (of textbook), section, page number etc.*

*The exam questions are also available in Danish (separately). Eksamensspørgsmålene findes også på dansk (på separate ark).*

*Some good advice: Read the exam questions in their entirety before answering any single question.*

## Question 1 (25%)

1. What is printed to the terminal upon the invocation of `metode1()`, where the definition of method `metode1` is given below? The answer should consist of the lines that are printed by the invocations of the `System.out.println` method. Remember that  $5 - -4$  is  $5+4$ .

```
public static void metode1(){
    int sum = 0;
    int j = 5;
    int k = -4;
    sum = 0; j=5; k= -4;
    while (j>k){
        sum = sum + (j-k);
        j = j - k;
        k = k +2;
        System.out.println("sum: " + sum + " j: " + j + " k: " + k);
    }
}
```

2. What is printed to the terminal upon the invocation of `metode2()`, where the definition of method `metode2` is given below? The answer should consist of the lines that are printed by the invocations of the `System.out.println` method.

```
public static void metode2(){
    int sum = 10;
    for (int i = 10; i>=4; i -= 2){
        for (int j = i; j < 8; j++){
            sum = sum + (j-i);
            System.out.println("sum: " + sum + " i: " + i + " j: " + j);
        }
    }
    System.out.println("endelig sum: " + sum);
}
```

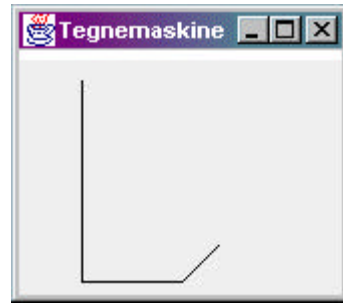
3. `Tegnemaskine` is a class with the following instance methods:

- ?? `void move(int afstand)`, which draws a line of length `afstand` (pixels) from the current point in the current direction.
- ?? `void turn(int grader)`, which changes the current drawing direction by `grader` degrees. Positive values for `grader` change the direction to the left (counterclockwise), negative values to the right (clockwise).
- ?? `void jump(int afstand)`, which changes the current point by moving it `afstand` pixels in the current direction without drawing anything.

The internal state of a `Tegnemaskine` object consists of its current drawing point and its current drawing direction. Note that the current drawing point is moved by both `move` and `jump`. The difference between the two methods is that the first draws a line while it does so, whereas the latter does not. A `Tegnemaskine` instance is created by `new Tegnemaskine()`, which automatically opens a windows for drawing and sets the current drawing point near the upper left corner with the current drawing direction downwards.

For illustration, invocation of `eksempel()` below gives rise to the drawing to the right.

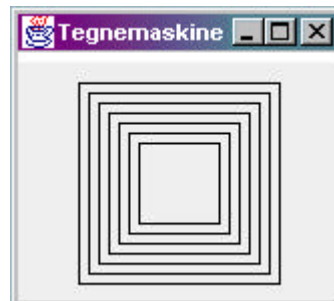
```
public static void eksempel(){
    Tegnemaskine tm = new Tegnemaskine();
    tm.move(100);
    tm.turn(90);
    tm.move(50);
    tm.turn(45);
    tm.move(25);
}
```



Using `Tegnemaskine` and its methods `move`, `turn`, `jump`, write a method with method header `void spiral(int n)` which draws a spiral as shown below. The parameter `n` specifies how long the first (outermost) line segment must be. Every line segment must be 2 pixels shorter than the previous line segment, and the last line segment must be at least 2 pixels long. Recall that a `Tegnemaskine` object starts at the upper left part of the drawing surface with drawing direction downwards, as shown below.



- Using `Tegnemaskine` and its methods `move`, `turn`, `jump`, write a method with method header `void kasse(int n, int k, int s)` (*kasse* means "box") that draws `n` squares inside of each other with a distance of `k` pixels between a square and its immediately enclosing square. The length of the outermost square must be `s` pixels. Use a nested loop, where the inner loop draws a square of a given size and the outer loop ensures that `n` squares are drawn in total. The result of invoking `kasse(7, 5, 100)` is shown below. Recall that a `Tegnemaskine` object starts at the upper left part of the drawing surface with drawing direction downwards, as shown below.



(End of Question 1)

## Question 2 (30%)

In this question you are to use a class that represents rivers. Here, a *river* is an object that has a *name* (a string), a *source* (where the river originates and how much water it has there, measured in liters per hour), a *length* (how long it is, measured in kilometers), a *width* (measured in meters) at its mouth (where it flows into the sea or another river). Finally, it has an array of *tributaries* (other rivers that flow into it along its way).

For example, the source of the river *Suså* is in *Gøgsmose*. *Suså* has *Kongskilde Bæk*, *Sorø Å* and *Ringsted Å* as tributaries. The source of *Kongskilde Bæk* is in *Kongskilden*. The source of *Sorø Å* is in *Sorø Sø*. The source of *Ringsted Å* is in *Gyrstinge Sø*. *Ringsted Å* has *Vigerdalså* as its sole tributary, whose source, in turn, is *Valsølille Sø*.

We represent sources (Danish: udspring) using the following class `Udspring`.

```
public class Udspring {
    private String navn; // name
    private double literPrTime; // liters per hour

    public Udspring(String n, double lpt){
        navn = n;
        literPrTime = lpt;
    }
    public String getNavn(){ return navn; }
    public double getLiterPrTime(){ return literPrTime; }
}
```

We represent rivers (Danish: vandløb) using the following – incomplete – class. (You will be asked to complete the definition of `Vandløb`.)

```
public class Vandløb {
    private String navn; // name
    private double bredde; // width – in meters
    private double længde; // length – in kilometers
    private Udspring udspring; // source

    private Vandløb[] sideGrene = new Vandløb[20]; // Max 20 tributaries
    private int antalSideGrene = 0; // number of tributaries.

    // Here should be the constructor declaration for Vandløb , see part 1 of this question

    public void tilføjSidegren(Vandløb v){
        // Here should be the body of method tilføjSidegren, see part 2 of this question
    }

    public double systemLængde(){
        // Here should be the body of method systemLængde, see part 5 of this question
    }

    // continued on next page...
```

// continued from previous page...

```
public double vandMængde(){
    double sum = udspring.getLiterPrTime();
    for (int sideGren = 0; sideGren < antalSideGrene; sideGren++)
        sum += sideGrene[sideGren].vandMængde();
    return sum;
}
}
```

1. Write the constructor for class `Vandløb`, which is to take name, width, length and source as parameters so that, e.g.  
`new Vandløb("Ringsted å", 7, 13, new Udspring("Gyrstinge sø", 250) )`  
creates an instance representing Ringsted å.

Furthermore, write two access methods that return the name and the number of tributaries, respectively, of a river.

2. Write the body of the method `tilføjSidegren` in class `Vandløb`. Invoking the method should add the argument to the array of tributaries. (You may assume that there are at most 20 tributaries to any single river. You need not check for a value greater than 20 in your code.)
3. The following declarations create objects representing Ringsted å and Vigerdals å and assign references to them to the variables `ringstedÅ` and `vigerdalsÅ`, respectively:

```
Vandløb ringstedÅ = new Vandløb("Ringsted å", 7, 13, new Udspring("Gyrstinge sø", 250) );
Vandløb vigerdalsÅ = new Vandløb("Vigerdals å", 3, 6, new Udspring("Valsøllille sø", 130) );
ringstedÅ.tilføjSidegren( vigerdalsÅ);
```

Write corresponding declarations for Kongskilde bæk, Sorø å, and Suså. Add Kongskilde bæk, Sorø å, and Ringsted å as tributaries to Suså. (You may use arbitrary values for information that is missing from the example above, e.g., for length, width and the like.)

4. To compute the amount of water per hour at the mouth of a river, method `vandMængde` takes the amount of water at its source and adds to it the amount of water at the mouth of each of its tributaries. (See definition of `vandMængde` above.)
  - a. Method `vandMængde` recursively invokes `vandMængde` inside its for-loop. How can one be sure that the method terminates; that is, that it doesn't keep on calling itself forever?
  - b. How many times will `vandMængde` be invoked during the invocation of `vandMængde(Suså)`?
5. Write a method `double systemLængde()` which computes the total length of a river system. Given a river, the total length of its river system is the length of the river itself, plus the lengths of all its tributaries, plus the length of all tributaries to those tributaries and so on.

(End of Question 2.)

### Question 3 (30%)

An apartment block (see picture to the right) is a building containing apartments. Here, an apartment block consists of a number of sections, each with a separate entrance. Each section has the same number of floors. There is precisely one apartment on any given floor in any section of the apartment block. Each apartment is inhabited by zero or more persons. An empty apartment is one in which zero persons live. Each person has a number of attributes: name, age and profession. Each apartment has at most one telephone. The above is captured by the following Java classes.



```
public class Person {
    public String navn; // name
    public int alder; // age
    public String profession;
    Person (String n, int a, String p){
        navn = n; alder = a; profession = p;
    }
}

public class Lejlighed { // Apartment
    public String telefonNr; // phoneNo
    public Person[] beboere; // Inhabitants
    public Lejlighed(){
        beboere = new Person[0]; telefonNr = null;
    }
    public void flytInd(Person[] b, String tlf){ // Move in.
        beboere = b; telefonNr = tlf;
    }
    public int antalBørn(){ // number of children
        // Here should be the body of method antalBørn (of class Lejlighed), see part 5 of this question.
    }
}

public class BoligBlok // Apartment block
{
    Lejlighed[][] lejligheder; // apartments
    BoligBlok(int opgange, int etager){ //sections, floors
        // Here should be the body of the constructor for BoligBlok, see part 1 of this question
    }
    public Lejlighed getLejlighed(int opgang, int etage){ // getAppartment
        // Here should be the body of method getLejlighed, see part 2 of this question.
    }
    public void flytInd(Person[] personer, String tlf, int opgang, int etage){ // move in
        Lejlighed lejlighed = getLejlighed(opgang, etage);
        lejlighed.flytInd(personer, tlf);
    }
    public int antalPersoner(){ // number of persons
        // Here should be the body of method antalPersoner, see part 3 of this question
    }
}
// continued on next page...
```

// continued from previous page...

```
public Lejlighed flest(){ // the most
    // Here should be the body of method flest, see part 4 of this question
}
```

```
public int antalBørn(){ // number of children
    // Here should be the body of method antalBørn (of Class BoligBlok), see part 5 of this question
}
}
```

1. Write the constructor for class Boligblok. The apartments must be initialized correctly with empty apartments, not with null values. As an example, the following code creates (an object representing) an apartment block with 7 floors and 20 sections, where the Hansen family moves into the apartment on the 4<sup>th</sup> floor in section 3.

```
BoligBlok betonBo = new BoligBlok(20, 7);
Person[] familienHansen = { new Person("Anna Hansen", 32, "Smed"),
                             new Person("Jonas Hansen", 35, "Portør"),
                             new Person("Amalie Hansen", 4, "Forældre plager") },
betonBo.flytInd(familienHansen, "12 13 14 15", 3, 4);
```

2. Write the body of method getLejlighed. Both sections and floors must be numbered starting with 1.
3. Write the body of antalPersoner. The method must return the total number of persons living in the apartment block. Note that the number of persons living in an apartment can be obtained using beboere.length since beboere is an array holding the inhabitants of the apartment.
4. Write the body of method flest. The method must return a reference to the object in the apartment block where most persons live amongst all apartments in the same apartment block. If there is more than one such apartment, any one of them may be chosen.
5. Write the body of method antalBørn in class Lejlighed. The method must return the number of children in the apartment. A child is a person under the age of 12 years. Using method antalBørn in class Lejlighed, write the body of method antalBørn in class BoligBlok, which returns the number of children living in the whole apartment block.

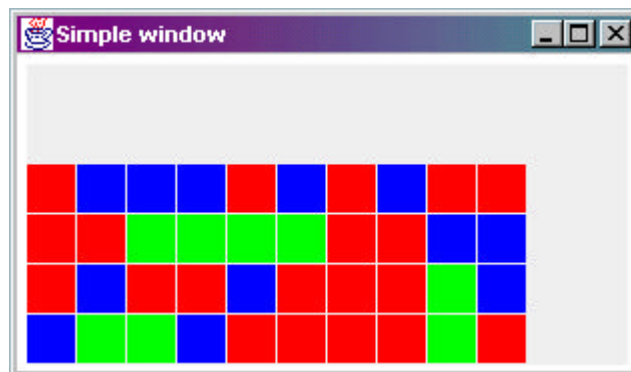
(End of Question 3.)

## Question 4 (15%)

Note: This question reuses the introductory explanations and the definitions of classes and methods in Question 3. The answers to Question 3, however, are not required to answer this question.

1. Design a functional (external) test of method `flest` in class `Boligblok`. Note that `flest` has no parameters. You can vary the objects the method is invoked on and thus the distribution of inhabitants and size of an apartment block. One possible test case could, for example, be an invocation of `flest` on an apartment block with 3 floors and 2 sections with, respectively, 0,1,2,3,4,5 inhabitants in its apartments.
2. Write a method `tegnBoligblok` with method header `public void tegnboligblok(Boligblok blok, Graphics gr)` which draws apartment block `blok` using the graphics object referenced by `gr`. An apartment block is drawn schematically as follows: A single apartment in `blok` is to be drawn as a square of size 25 \* 25 pixels. If it has no (zero) inhabitants it is to be drawn as a green square, if it has 1 or 2 inhabitants it is to be drawn as a blue square, and if it has more than 2 inhabitants it is to be drawn as a red square. You may assume that the graphics object `gr` is 300 pixels wide and 150 pixels high. The drawing may fill the whole drawing area all the way to its edges.

For example, an apartment block with 10 sections and 4 floors will be drawn as shown below. It shows the apartment block from the side, with section no. 1 leftmost and section no. 10 rightmost.



3. The drawing showing an apartment block is to be used in a graphical user interface (GUI) with two graphical components, where the drawing of the apartment block is one of the components. The other component is to contain a label and text field for displaying a telephone number. Upon clicking into a colored square in the apartment block drawing, the text field is to display the telephone number (if any) of the apartment represented by the clicked square. You may assume here that the GUI is already programmed such that the coordinates of the mouse click are converted to the section number `opgang` and floor number `etage` of the apartment clicked and that the method `lejlighed_clicked` with method header `public void lejlighed_clicked(BoligBlok blok, int opgang, int etage)` is invoked upon each mouse click. Your job is to declare a label containing text "Telephone number: " and a text field with space for 8 characters, and to write the body of method `lejlighed_clicked` such that the telephone number of the chosen apartment in apartment block `blok` is displayed in the text field upon invocation. (You need not concern yourself with the layout of the GUI containing the components here.)

(End of Question 4.)