

**Written exam in
Introductory Programming**

IT-C, January 17, 2001
English version

All materials are permitted during the exam, except computers.

The exam questions must be answered in writing within four hours. The answers will be graded using the Danish '13-grade' scale.

There are four main questions, all of which should be answered. The four main questions have equal weight.

Your answers may use classes or methods from the textbook, the lecture notes, and the lecture slides. You do not need to copy them to your answers, but you must give a precise reference, stating section, page number etc.

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

You are advised to read all the questions before you start answering any of them.

Question 1

Question 1.1

Show what is printed on the display when the Java program below is being executed. Be careful to indicate newlines and blanks (spaces) correctly:

```
class Opgave1_1 {
    public static void main(String[] args) {
        line(5); line(2); line(5);
        System.out.println();
        line(5); line(5); line(5);
    }

    static void line(int i) {
        switch (i) {
            case 0: System.out.println("  "); break;
            case 1: System.out.println(" *"); break;
            case 2: System.out.println(" * "); break;
            case 4: System.out.println("* "); break;
            case 5: System.out.println("* *"); break;
        }
    }
}
```

Question 1.2

Write a method `static void show(int d)` that can be called with an argument `d` between 1 and 6, and which prints a face of a die whose number of spots is `d`. You may use the method `line`. For instance, the call `show(2)` should print

```
*
*

```

and the call `show(3)` should print

```
*
*
*
```

Question 1.3

Write a complete Java program `Opgave1_3` which prints a hollow box, 5 lines high and 5 plusses wide:

```
+++++
+   +
+   +
+   +
+++++
```

Question 1.4

Write a method `static void fill(char c, int n)`, which given the parameters `c` and `n` prints a box `n` lines high and `n` plusses wide, where the box is filled with the character `c` instead of blanks (spaces).

For instance, the call `fill('o', 5)` should print this filled-in box:

```
+++++
+ooo+
+ooo+
+ooo+
+++++
```

Question 2

This question concerns calendar dates in the year 2001. The months are numbered 1–12 (January–December) and the weekday are numbered 0–6 (Monday–Sunday). January 1, 2001 was a Monday (weekday 0). Year 2001 is not a leap year, so February has 28 days, and all problems concerning leap years can be ignored.

Here is a skeleton of a program for computing with calendar dates in year 2001:

```
class Opgave2 {
    static int[] days =
        { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334, 365 };

    static int monthlen(int m) { ... }

    static String dayname(int wd) { ... }

    static int dayno(int m, int d) { ... }

    static int weekday(int m, int d) { ... }

    static void prmonth(int m) { ... }
}
```

The array `days` contains 13 numbers, so that `days[m]` is the total number of days in the months up to and including month `m`. For instance, `days[2]` equals 59 which is the total number of days in January and February.

In the questions below you must complete the methods `monthlen`, `dayname`, `dayno`, `weekday` and `prmonth`. When answering a question you may use methods from previous questions (even if you have not answered those questions).

Question 2.1

Write the method `monthlen`, so that the call `monthlen(m)` returns the number of days in month number `m`. Use the array `days`. For instance, the call `monthlen(2)` must return 28, the number of days in February 2001.

Question 2.2

Write the method `dayname`, so that the call `dayname(wd)` returns the name of the weekday whose number is `wd`. (It is up to you whether it should be in Danish or in English. For instance, `dayname(0)` may return "mandag" or "Monday" or "Mon").

Question 2.3

Write the method `dayno`, so that the call `dayno(m, d)` returns the number of days since January 1 (in year 2001). For instance, `dayno(1, 1)` must give 0, `dayno(1, 17)` must give 16, `dayno(1, 31)` must give 30, `dayno(2, 1)` must give 31, `dayno(12, 1)` must give 334, and `dayno(12, 31)` must give 364.

Question 2.4

Write the method `weekday`, so that the call `weekday(m, d)` returns the number of the weekday for day `d` in month `m` (in year 2001). For instance, `weekday(1, 1)` must give 0 (Monday); `weekday(1, 17)` must give 2 (Wednesday); `weekday(2, 1)` must give 3 (Thursday); `weekday(12, 1)` must give 5 (Saturday), and `weekday(12, 31)` must give 0 (Monday). Hint: Use the modulo operator (%), since $0\%7$ is 0, $16\%7$ is 2, $31\%7$ is 3, and $334\%7$ is 5.

Question 2.5

Write the method `prmonth` so that the call `prmonth(m)` prints a calendar for month `m` (in year 2001).

For instance, the call `prmonth(1)` must print the calendar for January 2001:

```
Mon 1
Tue 2
Wed 3
Thu 4
Fri 5
Sat 6
Sun 7
Mon 8
...
Wed 17
Thu 18
...
Wed 31
```

Similarly, the call `prmonth(2)` must print the calendar for February 2001:

```
Thu 1
Fri 2
...
Wed 28
```

Question 3

This question concerns courses and students at the IT University. Courses are represented as objects of the class `Course`, shown below. A course has a name and a particular weekday, which is an integer: 0 means Monday, 1 means Tuesday, ..., 6 means Sunday.

There are two kinds of students: ITC students (who have a student id and follow zero or more courses), and single-course students (who do not have a student id and follow exactly one course).

The abstract class `Student`, shown below, describes what is common to all students: a student has a name (the field `name`), and one can ask how many courses that student follows (using the method `getCourseCount`). The student's courses are numbered 0, 1, ..., so one can ask for the student's course number 0, 1, ... (using the method `getCourse`), and finally one can print all information about the student (using the method `print`).

```
class Course {
    String name;
    int weekday;

    Course(String name, int weekday)
    { this.name = name; this.weekday = weekday; }

    int getDay() { return weekday; }

    public String toString() { return name; }
}

abstract class Student {
    String name;
    abstract int getCourseCount();
    abstract Course getCourse(int i);
    abstract void print();
}

... declaration of the class SingleCourseStudent ...

... declaration of the class ITCStudent ...
```

In the questions below you must write complete declarations of the classes `SingleCourseStudent` and `ITCStudent` so that they can be used as shown in this program:

```
class Opg3 {
    public static void main(String[] args) {
        Course gp = new Course("Introductory Programming", 2);
        Course dbd = new Course("Design of User Interfaces and Data", 4);
        Course dbw = new Course("Database-based Web Publishing", 4);

        Student s1 = new SingleCourseStudent("Svend Bergstein", gp);

        Course[] courses1 = { gp, dbd, dbw };
        Student s2 = new ITCStudent("Birthe Weiss", courses1, 4711);
        Student s3 = new ITCStudent("Jan Trøjborg", courses1, 666);

        s1.print();
        s2.print();
        s3.print();
    }

    static boolean dayClash(Student s) { ... }
}
```

Question 3.1

Declare the class `SingleCourseStudent` as a subclass of `Student`, with one field `course` (of type `Course`), and a constructor which takes as arguments name and course, so that it may be called as shown in the main method above.

The class must contain also suitable implementations of the methods `getCourseCount` and `getCourse`. The method `getCourse` should return the student's (unique) course, regardless what argument the method has been called with.

Question 3.2

Declare the class `ITCStudent` as a subclass of `Student`, with fields `courses` (of type `Course[]`) and `studentId` (of type `int`), and a constructor which takes as arguments `name`, `course array`, and `student id`, so that it may be called as shown in the `main` method above.

The class must contain also suitable implementations of the methods `getCourseCount` and `getCourse`.

The call `s.getCourseCount()` must return the number of courses followed by student `s`. The call `s.getCourse(i)` must return course number `i` of student `s`, where possible values for `i` are `0, 1, ...` up to the number of courses minus one. For instance, if the student follows 3 different courses, then it must be possible to call `getCourse` with the arguments `0, 1, and 2`.

Question 3.3

Write implementations of the method `print` in class `SingleCourseStudent` as well as `ITCStudent`. The implementations must be written so that the `main` method shown above prints this when executed:

```
Svend Bergstein, single-course student:
Introductory Programming
```

```
Birthe Weiss, id 4711:
Introductory Programming
Design of User Interfaces and Data
Database-based Web Publishing
```

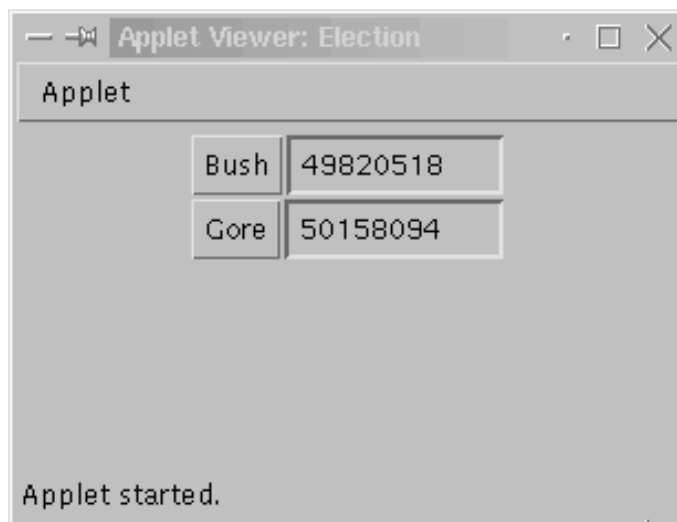
```
Jan Trøjborg, id 666:
Introductory Programming
Design of User Interfaces and Data
Database-based Web Publishing
```

Question 3.4

Write the method `static boolean dayClash(Student s)` in class `Opg3`. The method must return `true` if the given student follows two courses on the same weekday; otherwise it must return `false`. Hint: Use the methods `getCourseCount` and `getCourse` from the `Student` class.

Question 4

This question concerns an applet for counting votes in an election. The applet has two buttons labelled 'Bush' and 'Gore', and two corresponding textfields (with room for 10 characters) that show how many times each of the buttons has been pressed. Here is an example snapshot of the applet:



Here is a skeleton of the Applet class:

```
import java.applet.Applet; import java.awt.*; import java.awt.event.*;

public class Election extends Applet {
    int cntBush = 0, cntGore = 0;
    Button butGore, butBush;
    TextField txtGore, txtBush;
}

public void init() {
    Panel p = new Panel();
    ...
    add(p);
}

class BushListener implements ActionListener { ... }

class GoreListener implements ActionListener { ... }
}
```

The comments /* a */, /* b */ etc. have no effect; there are inserted only to name program points in the questions below.

Question 4.1

Write Java declarations and statements for program point /* b */ in method `init` to create the four components, give the applet a suitable layout manager, and add the components to the applet, so that it appears as shown in the figure above.

Use a drawing to indicate possible auxiliary Panels and the placement of the components on these panels.

Hint: It is useful to place the four components on a Panel `p` that has `GridLayout` or `BorderLayout` (and possibly some auxiliary Panels). The textfields must not be editable by the user.

Question 4.2

Write the contents of the two listener classes `BushListener` and `GoreListener` (at program points `/* c */` and `/* d */`), so that a click on a button increases the corresponding number of votes by one and displays it in the corresponding textfield. The fields `cntBush` and `cntGore` must contain the current number of votes.

Also add statements to `init` (at program point `/* b */`) to create the listeners and associate them with the buttons.

Question 4.3

Add a textfield `status` with room for 30 characters, and place it after the panel `p`. When a button has been pressed, the new textfield must display one of these three messages: 'Bush is ahead by x votes' or 'Gore is ahead by y votes', or 'Bush and Gore are equal', where x and y must be the actual difference in the number of votes. In the example on the preceding page, the text would be 'Gore is ahead by 337576 votes'. Carefully describe all required additions and changes to the applet, and where they go (at which program points).

Question 4.4

Make sure that the text in the status textfield is correct even when the difference in votes is one. In that case, the text in the status textfield should be (for instance) 'Gore is ahead by 1 vote', rather than 'Gore is ahead by 1 votes'.

Question 4.5

Add a histogram with two horizontal bars showing the relative number of votes for Bush and Gore. Place it after the other components on the applet. The canvas containing the graphics must be 250 pixels wide and 50 pixels high. Scale the bars so that the length of the bar for the leading candidate equals the width of the canvas. (If both candidates have zero votes, nothing should be shown). Hint: Create a subclass of `Canvas` as an inner class in `Election`, so that the `paint` method of the canvas can access the fields `cntBush` and `cntGore` in the `Election` class.

Here is an example snapshot of the applet very early in the count:

