

Written exam in Introductory Programming

IT-C, June 12, 2002
English version

All materials are permitted during the exam, except electronic devices.

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).

Advice: Check that you have all pages. There are 10 pages in total (including this one.) Read all the questions before you start answering any of them. Do not work on Questions 4.4 and 3.4 before you have finished work on all the other questions. The answer to Question 3.4 is only taken into account in deciding whether to award grade 13. So do it last, if at all. Watch your time consumption!

Question 1

Consider the following definition of class Loop.

```
class Loop {
    public static void loop(int x, int y, int z) {
        for (int c = x; c > y; c = c-z)
            System.out.println("c: " + c + " x: " + x + " y: " + y + " z: " + z);
    }

    public static void nestedLoop(int x, int y, int z) {
        while (x >= y)
            loop(x--, ++y, z++);
    }
}
```

Question 1.1

Show what is printed on the display upon invocation of `Loop.loop(9, 3, 2)`.

Question 1.2

Show what is printed on the display upon invocation of `Loop.nestedLoop(6, 0, 2)`.

Question 1.3

Write the body of method `classify` below, which takes as input a 2-dimensional array (an array of arrays) of integers and prints the number of zeros (0), ones (1) and other integers that occur in the array.

```
public class ZeroesOnes {
    public static void classify(int[][] arr) {
        ...
    }

    public static void test() {
        int[][] a = {{0, 1, 8, 9}, {4, 4, 1, 0}, {1, 1, 0}};
        classify(a);
    }
}
```

E.g., the output resulting from executing `ZeroesOnes.test()` should be

```
0: 3
1: 4
other: 4
```

Question 2

Consider the following incomplete definition of class Hat.

```
class Hat {
    protected char printSymbol;

    public Hat(char printSymbol) {
        this.printSymbol = printSymbol;
    }

    // prints a number of characters in a row
    private void draw(char symbol, int times) {
        for (int i = 0; i < times; i++)
            System.out.print(symbol);
    }

    // prints a new line
    private void draw() {
        System.out.println();
    }

    public void print() {
        ... // see Question 2.1
    }

    public void print(int topWidth, int bottomWidth, int height) {
        ... // see Question 2.2
    }

    public boolean valid(int topWidth, int bottomWidth, int height) {
        ... // see Question 2.3
    }

    public void printValid(int topWidth, int bottomWidth, int height) throws Exception {
        if (valid(topWidth, bottomWidth, height))
            print(topWidth, bottomWidth, height);
        else
            throw new Exception("Invalid input arguments");
    }
}
```

Question 2.1

Write the definition of method `print` with method signature `print()`. When invoked it must print the following shape, representing a hat, on the display, using the value of `printSymbol`. Use the 2 overloaded private methods named `draw` in `Hat` for drawing a number of characters in a row and for drawing a newline, respectively.

For example,

```
Hat hat0 = new Hat('*');
hat0.print();
```

should print

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

on the display. Note that this “hat” is precisely 5 asterisks wide at the top, 11 asterisks wide at the bottom and 5 asterisks high.

[Hint: Use, e.g., method invocation `draw(' ', 3)` to print 3 blanks in a row. Your method only needs to be able to print hats of that particular shape, printing the value of `printSymbol`.]

Question 2.2

Write the definition of method `print` with method signature `print(int, int, int)` where the parameters indicate how wide the hat should be at the top, how wide at the bottom, and finally how high it should be. E.g.,

```
Hat hat1 = new Hat('+');
hat1.print(5, 11, 5);
System.out.println();
hat1.print(4, 6, 6);
```

should print

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

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

In this question your code need not test the values of `topWidth`, `bottomWidth` and `height`, but may assume that they are sensible.

Question 2.3

Write the body of method `valid`, which checks whether the arguments to be passed to the `print`-method are valid. To be valid the arguments must satisfy the following properties:

1. The hat must be at least 3 symbols wide at the top, and at least 3 lines high.
2. The bottom line of the hat must be at least 2 symbols wider than its top line.
3. The hat must be symmetric; that is, the bottom line must extend equally much to the left and to the right of the “body” of the hat above it.

E.g., the following invocations of `valid` should return false:

1. `valid(4, 5, 4)`, because the bottom line is not sufficiently longer than the top line;
2. `valid(2, 8, 7)`, because the top line is not wide enough;
3. `valid(4, 6, 2)`, because the hat is not high enough;
4. `valid(5, 10, 7)`, because the hat’s bottom line cannot be made to extend an equal number of symbols to the left and to the right of the body of the hat.

On other hand, the following invocations of `valid` should return true:

1. `valid(4, 6, 4)`,
2. `valid(3, 9, 7)`,
3. `valid(4, 6, 3)`,
4. `valid(5, 11, 7)`.

Question 2.4

Consider the following definition of class SubHat.

```
class SubHat extends Hat {
    public SubHat(char printSymbol) {
        super(printSymbol);
    }
    public void draw(char symbol, int times) {
        System.out.println('o');
    }
    public boolean valid(int t, int b, int h) {
        return true;
    }
}
```

What does execution of the following Java code print on the display?

```
Hat hat2 = new SubHat('x');
try {
    hat2.printValid(2, 2, 2);
}
catch (Exception e) {
    System.out.println("An exception has been thrown.");
}
```

Give a brief explanation of the execution that leads to the result.

Question 3

This question is about understanding a class (questions 3.1 and 3.2) and extending it (question 3.3 and 3.4). Consider the following definition of class SimpleVector.

```
class SimpleVector {
    protected Object[] arr;
    protected int length;

    public SimpleVector(int size) {
        arr = new Object[size];
        length = 0;
    }

    public Object get(int index) throws Exception {
        if (0 <= index && index < length)
            return arr[index];
        else
            raise Exception("Illegal index argument to get method");
    }

    public void set(int index, Object obj) {
        arr[index] = obj;
        if (index >= length) length = index + 1;
    }

    public int length() {
        return length;
    }

    public void print() {
        for (int i=0; i < length; i++)
            System.out.println(arr[i]);
    }
}
```

Note that the length of new SimpleVector(17) (or any other argument for the constructor) is initially 0.

Question 3.1

What does execution of the following Java code print on the display?

```
SimpleVector a0 = new SimpleVector(4);
a0.set(0, "Freddy");
a0.set(2, "George");
a0.set(1, "Madonna");
try {
    a0.print();
}
catch (Exception e) {
    System.out.println(e.getMessage());
}
```

Question 3.2

The following Java code prints "r", the 4th letter in the string "George". What would happen if we omitted the downcast operation (String) in line 3?

```
SimpleVector a1 = new SimpleVector(5);
a1.set(2, "George");
String name = (String) a1.get(2); // contains downcast operation
System.out.println(name.charAt(3));
```

Question 3.3

Write a subclass `FlexibleVector` of `SimpleVector` that has a method `add` for adding an element in the next available slot of a vector instance. For example, execution of

```
FlexibleVector a2 = new FlexibleVector(10);
a2.add("Freddy");
a2.add("Madonna");
a2.add("George");
System.out.println("a2 has " + a2.length() + " elements.  They are:");
a2.print();
```

should print

```
a2 has 3 elements.  They are:
Freddy
Madonna
George
```

Just as `SimpleVector`, its subclass `FlexibleVector` should have a constructor which takes as argument the maximum number of elements a particular instance can have.

Question 3.4

(This is a continuation of Question 3.3. You are strongly advised *not* to work on it before you have answered *all* other questions.)

Write a subclass `UnboundedFlexibleVector` of `SimpleVector`. `UnboundedFlexibleVector` is to behave just as `FlexibleVector`, but, unlike `FlexibleVector`, it should not set any limit on how many elements can be in an instance.

Question 4

This question concerns employees in a company. An employee has a name and a unique identification number. This is represented by the class `Employee` below.

```
class Employee {
    private String name;
    private static int nextId = 1;
    private int id;
    public Employee(String name) {
        this.name = name;
        this.id = nextId++;
    }
    public String getName() {
        return name;
    }
    public int getId() {
        return id;
    }
}
```

Question 4.1

The class `Employee` could have been defined alternatively, namely by the class `Employee2` below, which is much shorter.

```
class Employee2 {
    public String name;
    public static int nextId = 1;
    public int id;
}
```

Question 4.1.1

`Employee2` does not have the accessor methods `getName` and `getId`, nor an explicit constructor. Show how the following Java statements are coded using class `Employee2` instead of class `Employee`; that is, give Java code that first constructs an instance of class `Employee2` and then prints its name followed by its identification number.

```
Employee henrik = new Employee("Henrik");
System.out.println("Employee: " + henrik.getName() + ", " + henrik.getId());
```

Question 4.1.2

Give a rationale (reasons) for choosing `Employee` for representing employees instead of `Employee2` even though `Employee` involves more code.

Question 4.2

A *managed employee* is an employee who has a manager, who in turn is a managed employee.

Question 4.2.1

Write a subclass `ManagedEmployee` of `Employee` so that:

- the subclass has a new private instance variable `manager` of type `ManagedEmployee`;
- the subclass has (public) accessor methods `setManager` and `getManager` for writing and reading the value of `manager`; `getManager` must return `null` for a new instance of `ManagedEmployee` (that is, before `setManager` has been executed).

Question 4.2.2

Write a list of Java statements that creates employees “Fritz”, “Henrik” and “Mads” (in this order) and then makes Mads the manager of Henrik, and Henrik the manager of Fritz.

Question 4.3

Let `ManagedEmployee` be the class defined in Question 4.2. The *director* of a managed employee `emp` is defined as follows:

- if `emp.getManager()` returns `null` (the null reference), then `emp` itself is the director of `emp`;
- otherwise, the director of `emp` is whoever the director of `emp`'s manager is.

Question 4.3.1

Add to the definition of `ManagedEmployee` a method `getDirector` which returns the director for a managed employee.

Question 4.3.2

Explain what happens when executing the following code (using your definition of method `getDirector`):

```
ManagedEmployee hans = new Employee("Hans");
ManagedEmployee per = new Employee("Per");
hans.setManager(per);
per.setManager(hans);
System.out.println("Hans's director: " + hans.getDirector());
```

Question 4.4

Question 4.4.1

We say `emp1` is a *subordinate* of `emp2` if `emp2` is the manager of `emp1`; that is, if `emp2 == emp1.getManager()` is true. Define the body of class method `printNumSubordinates` below

```
public class EmployeeInf {
    public static void printNumSubordinates(ManagedEmployee[] emps) {
        ...
    }
}
```

which prints on the display how many subordinates each managed employee has, with one line per managed employee. You may assume that all employees have unique employee identification numbers (employee id) in the range 1–10000 and you may use the employee id to identify an employee in the output. Here is an example of the output that is to be produced by `printNumSubordinates`:

```
Employee 1 has 0 subordinates.
Employee 2 has 4 subordinates.
Employee 3 has 1 subordinates.
Employee 4 has 2 subordinates.
Employee 5 has 0 subordinates.
...
Employee 10000 has 0 subordinates.
```

Note: In this example we have omitted 9994 lines (indicated by `...`) — your code has to produce all 10000 lines!

[Hint: Use an array `subords` of length 10001 such that `subords[i]` eventually contains the number of subordinates of the managed employee with identification number `i`.]

Question 4.4.2

We want method `printNumSubordinates` of Question 4.4.1 to print the lines in sorted order, with the managed employee with most subordinates printed first; e.g.,

```
Employee 6433 has 87 subordinates.  
Employee 127 has 55 subordinates.  
Employee 14 has 37 subordinates.  
Employee 8814 has 22 subordinates.
```

and so on (10000 lines in total).

Describe and justify what sorting algorithm you would use. Use words, do not write Java code.