



Examiner: Carsten Butz, IT University of Copenhagen  
Censor: Fritz Henglein, DIKU, University of Copenhagen  
Exam Form: Written Exam  
Date: June 6, 2003  
Time: 9:00 – 13:00

## WRITTEN EXAM, JUNE 6, 2003

Please read the following instructions carefully before starting answering the questions.

- The exam is a written exam. You have four (4) hours to answer the questions.
- The exam is graded on the Danish 13 scale.
- The question set is available in English only. You are allowed to answer the questions in either *English* or *Danish*.
- The exam is an open book exam, which for this exam means that the following aids may be used:
  - The text book *Java by Dissection* or any other Java book.
  - Additional course notes, like copies of slides, the notes by Peter Sestoft on searching and sorting, and on software testing.
  - Handwritten notes.
  - A language dictionary.
- No electronic devices are permitted. Electronic dictionaries are not allowed.
- The exam consists of four main questions. The weight of each question is given (in percentage points) next to the question. All four questions should be answered.
- This question set consists of nine pages.
- When using classes or methods from the book or from the notes by Peter Sestoft then it is not necessary to copy them. However, you have to give the precise location of those (for example, source, edition, section number, page number). It is your responsibility that we can identify that location.
- You are advised to read all questions before you start answering any of them.

(Weight 20%) QUESTION 1

For this question consider the following fragment of code:

```
1 class A{
2     int a;
3
4     A(){a = 10;}
5     A(int a){this.a = a;}
6
7     public String toString(){return "a = " + a;}
8
9 }
10
11 class B extends A{
12     int b;
13
14     B(){b = a;}
15     B(int b){a = 10; this.b = b;}
16     B(int a, int b){super(a); this.b = b;}
17
18     public String toString(){return super.toString() + " b = " + b;}
19
20 }
21
22 class C extends B{
23     int b;
24     int c;
25
26     C(int a, int b, int c){super(a,b); this.c = c;}
27
28     public String toString(){return "a = " + a + " b = " + b + " c = " + c;}
29
30 }
```

QUESTION 1.1

Draw the UML class diagram for the above three classes.

QUESTION 1.2

What does the following test program print on the screen?

```
class Test{
    public static void main(String[] args){

        A one = new A(1);
        B two = new B();
        B three = new B(2);
        B four = new B(3,4);
        A five = new B(5,6);

        System.out.println(one.toString());
        System.out.println(two.toString());
        System.out.println(three.toString());
        System.out.println(four.toString());
        System.out.println(five.toString());
    }
}
```

### QUESTION 1.3

State what the scope of each of the following identifiers is:

1. the variable `a` in class `A` (line 2);
2. the variable `b` in class `B` (line 12);
3. the variable `b` in class `C` (line 23);
4. the variable `c` in class `C` (line 24);

### QUESTION 1.4

Someone suggests the following constructor for class `B`:

```
B(A aa, int b){this = aa; this.b = b;}
```

State, with reason, whether or not the constructor definition above is correct Java code.

### QUESTION 1.5

The following main method

```
class Test{  
  
    public static void main(String[] args){  
  
        C five = new C(1,2,3);  
        System.out.println(five.toString());  
  
    }  
}
```

produces as output the line

```
a = 1 b = 0 c = 3
```

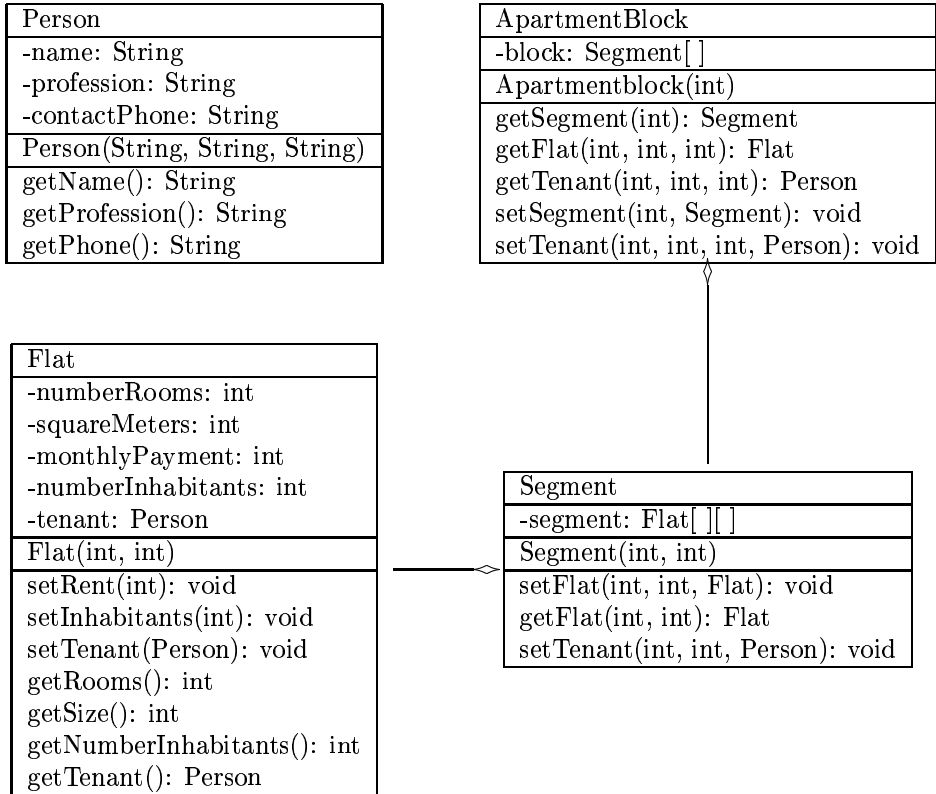
Explain why.

### QUESTION 1.6

The method `toString()` must be declared `public`. Why is this the case?

(Weight 25%) **QUESTION 2**

An apartment block is a building containing a number of apartments (flats in American English). Here, an apartment block consists of a number of segments, each with a separate entrance. Each segment has a certain number of floors. The number of flats (apartments) on each floor is fixed for each segment, but may vary from segment to segment. A flat has a certain number of people living in it. If the flat is unoccupied there are zero people living in it. Each occupied flat has a main tenant, a person with attributes like name, profession, and contact details (phone number). The above is captured in the following UML class diagram.



In the following you may assume that implementations of the classes `Person` and `Flat` are given. The constructor `Flat(int nR, int sM)` takes as integers arguments the number of rooms of the flat, and the size in square meters.

**QUESTION 2.1**

Give the implementation of class `Segment`.

- The constructor `Segment(int floors, int flatsPerFloor)` creates a new segment with `floors` many floors and `flatsPerFloor` many flats on each floor.
- The method `void setFlat(int floor, int flatNumber, Flat flat)` assigns to the flat with number `flatNumber` on floor `floor` the flat `flat`.
- The method `Flat getFlat(int floor, int flatNumber)` returns a reference to the flat with number `flatNumber` on floor `floor`.
- The method `void setTenant(int floor, int flatNumber, Person p)` assigns the person `p` as tenant of the flat `flatNumber` on floor `floor`.

**QUESTION 2.2**

Give the implementation of class `ApartmentBlock`.

- The constructor `ApartmentBlock(int numberOfSegments)` creates a new apartment block with `numberOfSegments` many different segments.
- The method `void setSegment(int segmentNumber, Segment s)` sets the segment with number `segmentNumber` to `s`.
- The method `Segment getSegment(int segmentNumber)` returns a reference to the segment with number `segmentNumber`.
- The method `Flat getFlat(int segmentNumber, int floor, int flatNumber)` returns a reference to the flat `flatNumber` on floor `floor` of segment `segmentNumber`.
- The method `Person getTenant(int segmentNumber, int floor, int flatNumber)` returns a reference to the person living in flat `flatNumber` on floor `floor` of segment `segmentNumber`.
- The method `void setTenant(int segmentNumber, int floor, int flatNumber, Person p)` sets the tenant of flat `flatNumber` on floor `floor` of segment `segmentNumber` to `p`.

### QUESTION 2.3

Add a method `int numberOfInhabitants()` to class `ApartmentBlock` which should return the number of Inhabitants in that block. Show all code that you need to add this method.

(Weight 30%) **QUESTION 3**

You are supposed to implement a simple storage class `Storage` that allows to store arrays of strings in a file given by `fileName`, and to retrieve that data. The UML class diagram of class `Storage` is given as follows:

Storage
-fileName: String
Storage(String)
save(String[ ]): void
load(): String[ ]

The files associated with a storage have the following structure: The first line of the file consists of the (integer) number of strings stored in the file. The following lines consist of the actual data, with each string stored on one line. Thus, a storage file with 4 strings might look as follows:

```
4
Miller Peter
Smith George
Becker Phil
Becker Paul
```

**QUESTION 3.1**

Write the constructor for this class, which takes as argument the file name that is associated to this storage. No file operations (open, close and similar) must occur in the constructor.

**QUESTION 3.2**

Write the method

```
void save(String[] arr) throws IOException
```

which takes as argument an array of strings representing the individual records, and which writes the correct data to the file.

**QUESTION 3.3**

Write the method

```
String[] load() throws FileNotFoundException, IOException
```

which retrieves the strings from the file, and returns an array of strings holding the individual records.

**QUESTION 3.4**

The following method `main()` creates access to storage `st` that might have been created and filled by another program.

Fill in code that will retrieve the data stored in the file `data.txt`, and print it line by line on the screen. In case the file `data.txt` does not exist a message "File not found!" should be printed on screen.

```
public static void main(String[] args){
    Storage st = new Storage("data.txt");
    // fill in your code here
}
```

### QUESTION 3.5

Write a test program for class `Storage` that should do the following: When used as

```
java TestNewStorage data.txt
```

the program should check whether the file `data.txt` already exists. If the file does not exist, an error message is printed. If the file does exist, records

```
Miller Peter  
Smith George  
Becker Phil  
Becker Paul
```

are created and those records are stored correctly in file `data.txt`.

### QUESTION 3.6

Design and implement a subclass `SortableStorage` that extends `Storage` with a method `sort()`, which sorts the strings in the storage in alphabetical order. E.g., in the example above, if `st` is instead a `SortableStorage` object with the storage file contents above (see introduction to Question 3) then the storage file should contain

```
4  
Becker Paul  
Becker Phil  
Miller Peter  
Smith George
```

after invocation of `st.sort()`.

To compare two strings in the alphabetical order use the method

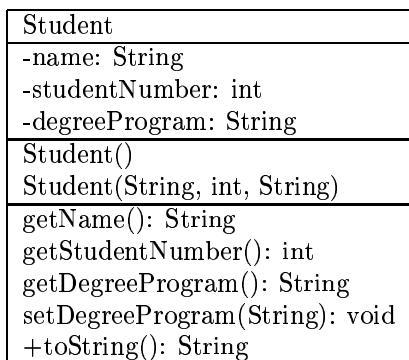
```
public int compareTo(String anotherString)
```

of class `String`. The result of `oneString.compareTo(anotherString)` is negative if `oneString` precedes `anotherString`, zero if the two strings are equal, and positive if `oneString` follows the string `anotherString` in the alphabetical order.

(Weight 25%) QUESTION 4

*Note: This question has to be answered independently from the previous one. Any reference to your particular implementation in the previous question will be penalized.*

Consider class `Student` with the following UML class diagram:



QUESTION 4.1

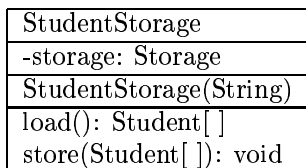
Show the implementation of the methods

```
Student(String, int, String)
String getName()
public String toString()
```

where the constructor takes as arguments the student's name, the student number, and the degree program, and where the method `getName()` returns the name of the student. The method `toString()` returns a single string representation of the student record, consisting of name, student number, and degree program, separated by single space characters.

QUESTION 4.2

Use class `Storage` with its constructor and methods `load` and `save` from Question 3 to implement a class `StudentStorage` with the following UML class diagram:



- Implement the constructor `StudentStorage(String)`, which takes as argument the name of the file which will be associated to the student storage.

- Implement the method

```
void store(Student[] arr) throws IOException
```

which takes an array of student records as parameter, converts each entry to a single string, and stores these strings in the storage.

- Implement the method

```
Student[] load() throws FileNotFoundException, IOException
```

which returns an array of student records. Note that from `storage` you retrieve an array of strings, each string consisting of the string representation of a single student like

```
Miller 875643 INT
```

Use a `StringTokenizer` as discussed in class to retrieve the individual components of the student records. Useful methods from class `StringTokenizer` are

- StringTokenizer(String str, String delim), a constructor which creates a new StringTokenizer with delim a list of symbols separating the individual tokens;
- countTokens(): int, returning the number of tokens;
- nextToken(): String, returning the next token;
- hasMoreTokens(): boolean, returning true exactly if there are more tokens available.

### QUESTION 4.3

Define class `MentoredStudent` as a subclass of `Student`:

<code>MentoredStudent</code>
-mentor: <code>MentoredStudent</code>
<code>MentoredStudent(String, int, String)</code>
<code>setMentor(MentoredStudent): void</code>
<code>getMentor(): MentoredStudent</code>
<code>guru(): MentoredStudent</code>

The mentor of a student is another (mentored) student. The constructor takes as arguments the name, student number, and the name of the degree program of the student and sets the fields accordingly. The methods `setMentor(MentoredStudent)` and `getMentor()` allow to set and retrieve the mentor of a mentored student. A student is his or her own guru if he or she has no mentor, i.e., if the mentor field is null. Otherwise, if the student has a mentor, mentor's guru is the student's guru. For example, execution of

```
MentoredStudent fred = new MentoredStudent("Fred", 2111, "K-DKM");
MentoredStudent bob = new MentoredStudent("Bob", 2898, "K-DKM");
MentoredStudent jill = new MentoredStudent("Jill", 1189, "K-DKM");

jill.setMentor(bob);
bob.setMentor(fred);

System.out.println("Jill's guru is " + jill.guru().getName() + ".");
```

prints the following line:

Jill's guru is Fred.

- Show the implementation of class `MentoredStudent`.
- What happens during execution, if we change the example above to the following?

```
MentoredStudent fred = new MentoredStudent("Fred", 2111, "K-DKM");
MentoredStudent bob = new MentoredStudent("Bob", 2898, "K-DKM");
MentoredStudent jill = new MentoredStudent("Jill", 1189, "K-DKM");

jill.setMentor(bob);
bob.setMentor(jill); // Only line that is changed!

System.out.println("Jill's guru is " + jill.guru().getName() + ".");
```