



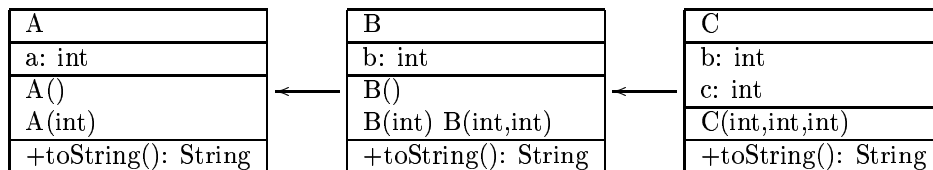
Examiner: Carsten Butz, IT University of Copenhagen
Censor: Fritz Henglein, DIKU, University of Copenhagen
Exam Form: Written Exam
Date: June 6, 2003
Time: 9:00 – 13:00

SOLUTIONS TO THE WRITTEN EXAM, JUNE 6TH, 2003

Please note the following: The solutions below only indicate one possible answer to the questions. Often, in particular when asked for code, various different answers are correct and will get full credit.

(Weight 20%) QUESTION 1

QUESTION 1.1



QUESTION 1.2

The output is

```
a = 1
a = 10 b = 10
a = 10 b = 2
a = 3 b = 4
a = 5 b = 6
```

QUESTION 1.3

Note: This question wasn't formulated well, so grading has been tolerant.

1. The scope of the identifier `a` (line 2) is all of the class A, i.e., everywhere inside the (curly) braces from line 1 to line 9.
2. The scope of the identifier `b` (line 12) is all of the class B.
3. The scope of the identifier `b` (line 23) is all of the class C.
4. The scope of the identifier `c` (line 24) is all of the class C.

QUESTION 1.4

The code is invalid. The reason is that `this` is a final variable where one cannot assign a value to. Note also that there is a type error. Thus, when compiling the class containing the suggested code the following error message will appear:

```
Test.java:20: cannot assign a value to final variable this
    B(A aa, int b){this = aa; this.b = b;}
                   ^
```

```
Test.java:20: incompatible types
found   : A
required: B
    B(A aa, int b){this = aa; this.b = b;}
                   ^
```

2 errors

QUESTION 1.5

When the constructor is called the variable `b` declared in the superclass B of C is set to 2. However, this variable is shadowed by the variable `b` declared in line 24, which is set upon initialization to the value 0. When printing the values of the variables `a`, `b`, and `c` it is the variable declared in line 24 that is accessed, not the one declared in line 12.

QUESTION 1.6

All classes automatically inherit from class `Object`, which already has a public method called `toString()`. Since we are not allowed to narrow the access to a method when overriding it, the method `toString()` has to be declared public.

(Weight 25%) QUESTION 2

QUESTION 2.1

```
class Segment{

    private Flat[] [] segment;

    Segment(int floors, int flatsPerFloor){
        segment = new Flat[floors][flatsPerFloor];
    }

    void setFlat(int floor, int flatNumber, Flat flat){
        segment[floor-1][flatNumber-1] = flat;
    }

    Flat getFlat(int floor, int flatNumber){
        return segment[floor-1][flatNumber-1];}

    void setTenant(int floorNumber, int flatNumber, Person p){
        segment[floorNumber-1][flatNumber-1].setTenant(p);
    }
}
```

QUESTION 2.2

```
class ApartmentBlock{

    private Segment[] block;

    ApartmentBlock(int numberOfSegments){
        block = new Segment[numberOfSegments];
    }

    void setSegment(int segmentNumber, Segment s){
        block[segmentNumber-1] = s;
    }

    Segment getSegment(int segmentNumber){return block[segmentNumber-1];}

    Flat getFlat(int segmentNumber, int floorNumber, int flatNumber){
        return block[segmentNumber-1].getFlat(floorNumber-1,flatNumber-1);}

    Person getTenant(int segmentNumber, int floorNumber, int flatNumber){
        return block[segmentNumber-1].getFlat(floorNumber-1,flatNumber-1).getTenant();
    }

    void setTenant(int segmentNumber, int floorNumber, int flatNumber, Person p){
        block[segmentNumber-1].setTenant(floorNumber,flatNumber,p);
    }
}
```

QUESTION 2.3

Add the following code to class ApartmentBlock

```
int numberOfInhabitants(){
    int noi = 0;
    for(int i = 0; i < block.length; i++)
```

```
        i+= block[i].numberOfInhabitants();
    return noi;
}
```

and add the following method to class Segment, returning for each segment the number of inhabitants living in this segment:

```
int numberOfInhabitants(){
    int noi = 0;
    for(int i = 0; i < segment.length; i++)
        for(int j = 0; j < segment[i].length; j++)
            noi += segment[i][j].getNumberInhabitants();

    return noi;
}
```

(Weight 30%) QUESTION 3

QUESTION 3.1

```
Storage(String fN){fileName = fN;}
```

QUESTION 3.2

```
void save(String[] arr) throws IOException{
    PrintWriter out = new PrintWriter(new FileWriter(fileName));

    int size = arr.length;

    out.println(size);

    for(int i = 0; i < size; i++)
        out.println(arr[i]);

    out.close();

    return;
}
```

QUESTION 3.3

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

    BufferedReader in = new BufferedReader(new FileReader(fileName));

    int size = Integer.parseInt(in.readLine());

    String[] data = new String[size];

    for(int i = 0; i < size; i++)
        data[i] = in.readLine();

    in.close();
    return data;
}
```

QUESTION 3.4

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

    Storage st = new Storage("data.txt");

    String[] arr;

    try{
        arr = st.load();
        for(int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }catch(FileNotFoundException fnfe){
        System.out.println("File not found!");
    }
}
```

QUESTION 3.5

Solution still to be provided.

QUESTION 3.6

```
import java.io.*;

class SortableStorage extends Storage{

    SortableStorage(String fileName){super(fileName);}

    void sort() throws IOException{

        String[] arr = super.load();

        /* Here follows one of the standard sorting algorithms,
         * this time sorting strings and using method compareTo
         * instead of sorting integers. The program is taken from
         * Peter Sestoft's notes, page 15.
         */
        selsort(arr,arr.length);

        super.save(arr);
    }
}
```

(Weight 25%) QUESTION 4

QUESTION 4.1

```
Student(String name, int sN, String sProgram){
    this.name = name;
    this.studentNumber = sN;
    this.degreeProgram = sProgram;
}

String getName(){return name;}

public String toString(){
    return name + " " + studentNumber + " " + degreeProgram;
}
```

QUESTION 4.2

```
import java.util.*;
import java.io.*;

class StudentStorage{

    private Storage storage;

    StudentStorage(String fileName){storage = new Storage(fileName);}

    Student[] load() throws IOException{

        String[] rawData = storage.load();
        int size = rawData.length;

        Student[] data = new Student[size];

        for(int i = 0; i < size; i++){
            data[i] = new Student();
            StringTokenizer sT = new StringTokenizer(rawData[i], " ");
            data[i].setName(sT.nextToken());
            data[i].setStudentNumber(Integer.parseInt(sT.nextToken()));
            data[i].setStudyLine(sT.nextToken());
        }

        return data;
    }

    void store(Student[] arr) throws IOException{

        int size = arr.length;
        String[] rawData = new String[size];

        for(int i = 0; i < size; i++)
            rawData[i] = arr[i].toString();

        storage.save(rawData);
    }
}
```

QUESTION 4.3

```
class MentoredStudent extends Student{

    private MentoredStudent mentor;

    MentoredStudent(String name, int sN, String sProgram){
        super(name,sN,sProgram);
        mentor = null;
    }

    void setMentor(MentoredStudent mentor){this.mentor = mentor;}
    MentoredStudent getMentor(){return mentor;}

    MentoredStudent guru(){
        if(mentor == null)
            return this;
        else
            return mentor.guru();
    }
}
```

As to the second question, the change results in an infinite regress. When `jill.guru()` is called then the method `jill.mentor.guru()` is called, which is `bob.guru()` because Bob is Jill's mentor. But to evaluate this method, the method `bob.mentor.guru()` is called, which is `jill.guru()` because Jill is Bob's mentor, etc. In particular, nothing is printed on the screen.