Tentti: T-106.1200 Basics of Programming T

Date: December 14th, .2006

Note! This is the exam for the course Basics of Programming T.
There is a separate question sheet for T-106.1203 Basics of Programming L.

General Info

The questions 1 and 2 are obligatory. You must get at least one point from each of these two questions in order to pass the exam. On the other hand, if you get at least one point from each of the obligatory questions, you will pass the exam with at least a grade of 1. In this case, your exam grade will be determined by the total sum of points from all questions.

Question 1 (2 points)

Examine the classes given in Appendix 1 carefully. The class `Building` represents (large) buildings, which may have multiple floors (numbered from zero upwards) and multiple parallel elevators. A user of a building's elevators can summon the nearest available elevator by using a method in class `Building`. Elevators are represented by the class `Elevator`. The class `Test` can be used to experiment with the other two classes.

Write a detailed description of what happens when this elevator program is executed starting from the main method in class `Test`. You must clearly and precisely indicate the order in which program execution proceeds from one line to another in the three given classes. You must also carefully list all the values that each variable in the program receives during program execution. The variables that you need to pay attention to include parameter variables and other local variables as well as instance variables.

When answering, make use of the line numbers shown in Appendix 1.

Note: it is not enough that you only explain what the main method in class `Test` does. You have to explain the behavior of all the code that is executed after the program is started. This does mean that you have to write quite a bit.

Hint: the program prints out:

```
Elevator #1: floor 4, available: true
Elevator #2: floor 0, available: true
```

The purpose of this question is to make sure that every student who passes the course understands the basics of Java syntax and program execution. The question is obligatory and you must receive at least one point from it to pass the exam. This question is graded on a scale of: 0 (failed) / 1 (passed) / 2 (excellent).

Question 2 (2 points)

Edit the class `Elevator` presented in the previous question in such a way that it can be used to keep track of elevators' passengers and their weight. Use the class `Passenger` from Appendix 2 to represent elevator passengers.

To be more specific, you need to make the following changes to the class `Elevator`:

1. Add two more instance variables for elevators: a list of the passengers in the elevator, and a weight limit which determines how heavy a load the elevator can carry at most. Also make any appropriate changes to the constructor.
2. Add a method `addPassenger`, which can be used to add a new passenger to an elevator. Adding a passenger is only successful if the elevator's weight limit would not be exceeded as a result.

(QUESTION CONTINUES OVERLEAF)

You should come up with sensible variable names, parameters, etc. yourself. You do not need to implement any other functionality (e.g. removal of passengers from elevators) than the things listed above.

Write down all the code changes that need to be made. You don't need to make any changes to the ready-made methods of the class `Elevator` or to the other given classes.

The purpose of this question is to make sure that every student who passes the course has at least some basic skills in writing Java program code. The question is obligatory and you must receive at least one point from it to pass the exam. This question is graded on a scale of: 0 (failed) / 1 (passed) / 2 (excellent).

Question 3 (4 points)

State your opinion of the following claims and justify it using ***at most a couple of sentences***.

a) Claim: Visibility modifiers such as `private` and `public` are used in order to prevent errors in programs.

b) Let us take a look at the method:

```
public static int doStuff(int number, int another) {
  if (number < another) {
    return 0;
  } else {
    return doStuff(number - 1, another + 1);
  }
}
```

Claim: When this method is called with the parameter values 5 and 1, a so-called infinite recursion occurs and eventually depletes the memory resources available to the program.

c) Claim: In case a method is supposed to merely calculate and return a result based on the parameters given to the method, then defining the method as a static method may be appropriate

d) Claim: Instance variables defined for an abstract class can not be used from within the program code of that abstract class.

Question 4 (2 points)

Let us assume that we have a program which contains the line:

```
this.stringArray[index] = input;
```

Let us further assume that we have noticed that the program crashes with a run-time exception. To be more specific, the program crashes with a `NullPointerException` caused by the line quoted above. What could be the problem? Where in the program would you look for the cause of the error? Note that the rest of the program code is not given here; answer the question briefly, in general terms.

Question 5 (2 points)

a) Bytecode is a language associated with Java programming. Which of the following does a Bytecode program resemble the most: a functional program, a logic program, a machine-language program, a Java program, a procedural program? Justify your answer using at most a couple of sentences.

b) Define, using at most a couple of sentences, what is meant by the term *frame* (or *activation record*) when it is used in association with Java programming.

Please remember to fill in the course questionnaire on the course web site by December 22nd! Your feedback is invaluable when we develop the course further. Filling in the questionnaire is required before you can receive a grade for this course.

```
1: public class Elevator {
2:
3:   private int currentFloor;   // most-recent holder
4:   private int topFloor;       // fixed value
5:   private boolean doorOpen;   // most-recent holder, "flag"
6:
7:   public Elevator(int topFloor) {
8:     this.currentFloor = 0;
9:     this.topFloor = topFloor;
10:    this.doorOpen = false;
11:  }
12:
13:  private void openDoor() {
14:    this.doorOpen = true;
15:  }
16:
17:  public void closeDoor() {
18:    this.doorOpen = false;
19:  }
20:
21:  public boolean isAvailable() {
22:    return this.doorOpen == false;
23:  }
24:
25:  public int getFloor() {
26:    return this.currentFloor;
27:  }
28:
29:  public int getDistance(int destination) {
30:    //  Math.abs returns the absolute value (Finnish: itseisarvo) of its parameter.
31:    return Math.abs(this.currentFloor - destination);
32:  }
33:
34:  // "A button is pressed inside the elevator, so it travels
35:  // to the given destination."
36:  public boolean travelTo(int destination) {
37:    if (destination >= 0 && destination <= this.topFloor) {
38:      this.closeDoor();
39:      this.currentFloor = destination;
40:      this.openDoor();
41:      return true;
42:    } else {
43:      return false;
44:    }
45:  }
46:
47:  // "A button is pressed outside, ordering the elevator to
48:  // come to the given destination."
49:  public boolean orderTo(int destination) {
50:    if (this.isAvailable() && destination >= 0 && destination <= this.topFloor) {
51:      this.currentFloor = destination;
52:      this.openDoor();
53:      return true;
54:    } else {
55:      return false;
56:    }
57:  }
58:
59: }
60:
61:
62:
63:
64:
65:
66:
67:
```

Appendix 1

```
68: import java.util.ArrayList;
69:
70: public class Building {
71:
72:   private String name;                        // fixed value
73:   private int height;                         // fixed value
74:   private ArrayList<Elevator> elevators;     // container
75:
76:   public Building(String name, int height) {
77:     this.name = name;
78:     this.height = height;
79:     this.elevators = new ArrayList<Elevator>();
80:   }
81:
82:   public void addElevator() {
83:     Elevator newElevator = new Elevator(this.height - 1);
84:     this.elevators.add(newElevator);
85:   }
86:
87:   public String getName() {
88:     return this.name;
89:   }
90:
91:   public Elevator orderElevatorToFloor(int destination) {
92:     Elevator closest = null;                  // most-wanted holder
93:     for (Elevator current : this.elevators) { // current: most-recent holder
94:       if ((closest == null || current.getDistance(destination) <
95:                               closest.getDistance(destination))) {
96:         if (current.isAvailable()) {
97:           closest = current;
98:         }
99:       }
100:    }
101:    if (closest != null) {
102:      closest.orderTo(destination);
103:    }
104:    return closest;
105:  }
106:
107:  public Elevator orderElevatorToLobby() {
108:    return this.orderElevatorToFloor(0);
109:  }
110:
111:  public void printElevatorDescription() {
112:    int elevatorNumber = 1;                   // stepper
113:    for (Elevator current : this.elevators) {   // current: most-recent holder
114:      System.out.println("Elevator #" + elevatorNumber +
115:                         ": floor " + current.getFloor() +
116:                         ", available: " + current.isAvailable());
117:      elevatorNumber++;
118:    }
119:  }
120: }
121:
122: public class Test {
123:   public static void main(String[] args) {
124:     Building office = new Building("Test Office", 7);
125:     office.addElevator();
126:     office.addElevator();
127:     Elevator test1 = office.orderElevatorToFloor(3);
128:     test1.travelTo(6);
129:     test1.closeDoor();
130:     Elevator test2 = office.orderElevatorToFloor(4);
131:     test2.closeDoor();
132:     office.printElevatorDescription();
133:   }
134: }
```

APPENDIX 2

```
1:  public class Passenger {
2:
3:      private double weight; // most-recent holder
4:
5:      public Passenger(double weight) {
6:          this.setWeight(weight);
7:      }
8:
9:      public void setWeight(double newWeight) {
10:         this.weight = newWeight;
11:     }
12:
13:     public double getWeight() {
14:         return this.weight;
15:     }
16:
17: }
```