Techniques of Java Programming

ETH Zurich

Date: 21 June 2007

Family name, first name: .....

Student number: .....

I confirm with my signature, that I was able to take this exam under regular circumstances and that I have read and understood the directions below.

Signature: .....

Directions:

- Exam duration: 90 minutes.
- Use a pen (**not** a pencil)!
- Please answer in English.
- Please write your student number onto **each** sheet.
- All solutions can be written directly onto the exam sheets. If you need more space for your solution ask the supervisors for a sheet of official paper. You are **not** allowed to use other paper.
- You must answer all questions (no questions are optional).
- All personal documents are authorized. Exchanging documents during the examination would mean failing the examination.
- Only one solution can be handed in per question. Invalid solutions need to be crossed out clearly.
- Please write legibly! We will only correct solutions that we can read.
- Manage your time carefully (take into account the number of points for each question).
- Please **immediately** tell the supervisors of the exam if you feel disturbed during the exam.

Good luck!

Question	Number of possible points	Points
1	11	
2	15	
3	26	
4	29	
5	26	
6	27	
Total	134	

1 Java Basics (11 Points)

Put checkmarks in the checkboxes corresponding to the correct answers. Zero, one or more correct answers are possible per block. A correctly set checkmark is worth 1 point, an incorrectly set checkmark is worth -1 point. If the sum of your points is negative, you will receive 0 points.

1. A method...

   a. implements one of more interfaces. ☐
   b. is used to provide global state. ☐
   c. is always part of a class. ☐
   d. is something that can be executed (unless it is abstract). ☐

2. A class...

   a. always has a name. ☐
   b. is always the target of methods calls. ☐
   c. can contain other classes. ☐
   d. can inherit from multiple other classes. ☐
   d. can be generic. ☐

3. A constructor...

   a. is used to create classes. ☐
   b. must not take any arguments. ☐
   c. can only occur once per class. ☐
   d. cannot be called from outside the class it is contained in. ☐

4. An interface...

   a. is used to specify the dynamic properties of a program. ☐
   b. can be instantiated to create objects. ☐
   c. can inherit from at most one class and multiple interfaces. ☐
   d. must not contain fields or methods-bodies. ☐

5. An object...

   a. can be of primitive type (int, char, ...) or the instance of a class. ☐
   b. is an instance of a class. ☐
   c. can be thrown as an exception if it inherits from class Exception. ☐
   d. can be marked as static in order to be globally visible. ☐

6. Java: the platform

   a. Java programs can load classes at runtime. ☐
   b. Java programs must be interpreted in order to be run. ☐

2 Test Driven Development and Extreme Programming (15 Points)

List and describe the advantages and disadvantages of test driven development (compared to more traditional development processes) (3 points)

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Describe the difference between system level tests and unit testing. Would you do both in the same project or choose only one? When are system level tests run and when unit tests (6 points)

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

In Test Driven Development, explain the different roles a test case has? (6 points)

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

3 Streams (26 Points)

1) Summarize the similarities and differences between the classes `InputStream` and `Reader` available in the official Java API: *(6 Points)*

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2) What are the advantages of using a stream infrastructure for input and output operations when compared to more "traditional" API that is directly working on a `File` or `Console` class with `read` and `write` operations. *(6 Points)*

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

3) Implement a *filter* by inheriting from `InputStream` operating on another `InputStream` passed to it on creation. The filter should convert Unix to Windows ASCII text files by replacing the character `'\n'` with the character sequence `'\r\n'`. *(14 Points)*

Hint 1: ASCII values: `'\n'` = 10 (linefeed), `'\r'` = 13 (carriage return);

Hint 2: `InputStream` has a single abstract method with the signature '`int read() throws IOExecption`', returning the next byte of data, or `-1` when the end-of-file has been reached.

(continued on the next page)

You **may** use the following skeleton:

```
class UnixToWindowsStream extends InputStream {

        .....

        .....

       .....
        public UnixToWindowsStream (InputStream in) {

                .....

                .....

       }
        public int read() throws IOException {

                .....

                .....

                .....

                .....

                .....

                .....

                .....

                .....

               .....
       }
        .....

        .....

        .....

        .....

        .....

        .....

}
```

4 SWING (29 Points)

4.1 Basics (6 Points)

1) What do people mean when they say that AWT is *heavyweight* whereas SWING is *lightweight*?

Heavyweight means: ..... *(1)*

.....

Lightweight means: ..... (1)

.....

2) What are the advantages of a "lightweight" SWING? Do you also see potential disadvantages?

Advantages: ..... (2)

.....

.....

Disadvantages: ..... (2)

.....

.....

4.2 Event Handling (6 Points)

1) What is the *Model-View-Controler* (MVC) paradigm?
..... (3)

.....

.....

.....

2) With what part of MVC would you associate a *listener* and why? What *listener* interface would you use for reacting on button clicks?

Listeners in MVC: ..... (2)

.....

.....

Listener for button clicks: ..... (1)

4.3 Layout Managers (7 Points)

1) What is the task of *layout managers* in SWING and what kind of components they are (usually) associated with.

Task: ..... (2)

.....

Layout Managers are associated with: ..... (1)

2) Implement the following method to create *and* open a JFrame with a BorderLayout in its content pane and labels with the text "North", "South", and "Center" in the corresponding regions. (4)

```
private void constructUI() {
    JFrame frame = new JFrame("My Frame");

    Container content = .....;

    content.setLayout(.....);
    //add the contents

    .....;

    .....;

    .....;
    //show the frame

    .....;

    .....;
}
```

4.4 SWING Multithreading (10 Points)

1) What is the Event Dispatching Thread?

. . . . . (2)

. . . . . .

2) What is the purpose of the methods SwingUtilities.invokeAndWait() and SwingUtilities.invokeLater() and what is their difference?

Purpose: . . . . . (2)

. . . . . .

. . . . . .

Difference: . . . . . (1)

. . . . . .

3) Complete the following code template such that the caption of the status label is set to `"Please wait"` before the "long-running" computation is started. Assume that the method is not called within the Event Dispatching Thread and make sure that the label really shows the waiting message *before* the computation starts. (3)

```
public int doComputation(JLabel status) {
    try {
        SwingUtilities. . . . . .(
            new . . . . . .() {
                public void run() {
                    . . . . . .;
                }
            }
        ) ;
        //do long running computation here
        return 5 + 7;
    } catch (Exception ex) {
         ex.printStackTrace();
    }
}
```

4) Show the life-cycle of SwingWorkers in terms of its *four* most important meth-

ods `execute()`, `doInBackground()`, `done()`, and `get()`, and state for each in what thread it is executed.

..... (2)

.....

.....

.....

5 Threads (26 points)

```
/**
 * Class supposed to give a result between 0 and 9
 * chosen at random by the JVM.
 **/
public class Lottery extends Thread{

private int num;
private static volatile int result=-1;
public Lottery(int num){
        this.num=num;
}

public void run(){
        result=num;
}

public static void main(String []args){
        Lottery l[]=new Lottery[10];
        for (int i=0; i<10; i++){
                l[i]=new Lottery(i);
                l[i].start();
        }
        System.out.println(result);
}
}
```

1) What can be printed on the command-line (6 points):

-1	☐
0	☐
1	☐
2	☐
3	☐
4	☐
5	☐
6	☐
7	☐
8	☐
9	☐
10	☐
11	☐

2) Do you think that the algorithm shows the different numbers with the same probability? Explain your point of view (4 points).

.....

.....

.....

.....

.....

.....

3) How can a programmer influence the threads scheduler of the JVM? (2 points)

.....

.....

.....

4) How would you use these mechanisms to make it fair to any possible number between 0 and 9 (4 points)?

.....

.....

.....

5) Show a possible implementation (10 points):

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

6 RMI (27 Points)

6.1 Stub and Skeleton (8 Points)

Explain how remote procedure calls work in general (e.g. using a sequence diagram): (4)

What roles play *Client Stub* and *Skeleton* in the calling sequence?

Client Stub: ..... (2)

. . . . . .

. . . . . .

Skeleton: ..... (2)

. . . . . .

. . . . . .

6.2 Marshalling/Unmarshalling (6 Points)

Marshalling and *Unmarshalling* are important functions regarding remote procedure calls. Describe how marshalling/unmarshalling works with *Java RMI* for

Primitive Type Arguments: ..... (1)

. . . . . .

. . . . . .

Object Type Arguments: ..... (2)

. . . . . .

. . . . . .

Exceptions: ..... (1)

. . . . . .

. . . . . .

Give an example: ..... (2)

. . . . . .

. . . . . .

6.3 Argument Passing: Java vs. RMI (4 Points)

As you know from other programming languages such as c/c++, arguments can be passed *by value*, *by Pointer* or *by reference*. Note that both Pointer and reference are addresses to the actual value or object. Pointers can be `null`, references always refer to an existing value or object.

Which of these techniques are conceptually used in Java and Java RMI:

Type	*Java*	*Java RMI*
Primitive Type (argument or return value)		
Object Type (argument or return value)		
Exceptions (catch block)		
Remote Objects (argument or return value)	*n/a*	

6.4 Exporting and Binding (9 Points)

Java RMI uses the terms *export* and *bind* for remote objects.

6.4.1 Export (4)

What does the *export* function do? (2)

. . . . . .

. . . . . .

. . . . . .

What is the *input*, what the *ouptut* of the *export* method? (1)

Input: . . . . . .

Output: . . . . . .

Who calls the *export* method? . . . . . (1)

6.4.2 Bind (5)

Which object offers the *bind* method? . . . . . (1)

What is it used for? . . . . . (1)

. . . . . .

. . . . . .

. . . . . .

Who calls the *bind* method? . . . . . (1)

Who gets access to *bound objects*, and how? . . . . . (1)

. . . . . .

. . . . . .

Accessing *bound objects*, what type of object will be returned? . . . . . (1)

. . . . . .