Techniques of Java Programming

ETH Zurich

Date: 29 May 2008

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 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 should answer all questions (no questions are optional).
- All personal documents are authorized. Exchanging documents during the examination would mean failing the examination.
- Electronic equipment (laptops, cell phones, PDA, etc.) are **not** authorized. Using them 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	9	
2	10	
3	13	
4	10	
5	12	
TOTAL	54	

1 Java Fundamentals (9 Points)

All questions below refer to the Java language version 5.0 as taught in the course. Put checkmarks in the checkboxes corresponding to the correct answers. Multiple correct answers are possible; there is at least one correct answer per question. 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. Consider the following classes:

```
1  package ch.ethz.inf.java;
2
3  public class CustomInteger {
4
5          private int anInt;
6
7          public CustomInteger(int i) {
8                  anInt=i;
9          }
10
11         public int getAnInt() {
12                 return anInt;
13         }
14 }
```

```
1  package ch.ethz.inf.java;
2
3  public class Main {
4
5          public static void main(String[] args) {
6                  CustomInteger ci1 = new CustomInteger(6);
7                  CustomInteger ci2 = new CustomInteger(6);
8                  Integer i = new Integer(6);
9                  System.out.print(ci1.equals(ci2));
10                 System.out.print(ci1==ci2);
11                 System.out.print(ci1.equals(i));
12         }
13 }
```

Which of the following result from the print statements?

a. truefalsefalse. ☐
b. falsefalsefalse. ☐
c. truefalsetrue. ☐
d. falsetruetrue. ☐

2. How could you declare the visibility of a method that has to be visible both in its own class package and by descendant classes?
   a. The defaut (package) visibility best addresses the need. ☐
   b. Both the defaut (package) and the protected visibility can be used. ☐
   c. The protected visibility best addresses the need. ☐
   d. In Java doesn't exist a visibility level that addresses the mentioned specific need , so I have to declare the method public. ☐

3. Consider the following classes:

```
1  package ch.ethz.inf.java;
2
3  public class Creature {
4
5          public void attack(int i){
6                  System.out.print("Creature attacks:" + i +" ");
7          }
8  }
```

```
1  package ch.ethz.inf.java;
2
3  public class Orc extends Creature {
4
5          public void attack(int i){
6                  System.out.print("Orc attacks:" + i + " ");
7          }
8  }
```

```
1  package ch.ethz.inf.java;
2
3  public class Goblin extends Creature {
4
5          public void attack(long i){
6                  System.out.println("Goblin attacks:" + i + " ");
7          }
8  }
```

```
1  package ch.ethz.inf.java;
2
3  public class GameTest {
4
5          public static void main(String[] args) {
6                  Creature c1 = new Orc();
7                  Creature c2 = new Goblin();
8                  Goblin g = new Goblin();
9                  c1.attack(7);
10                 c2.attack(5);
11                 g.attack(3);
12                 g.attack(4L);
13         }
14 }
```

Which of the following result from the print statements?

a. Orc attacks:7 Creature attacks:5 Goblin attacks:3 Goblin attacks:4 ☐

b. Creature attacks:7 Creature attacks:5 Creature attacks:3 Goblin attacks:4 ☐

c. Orc attacks:7 Creature attacks:5 Goblin attacks:3 Creature attacks:4 ☐

d. Orc attacks:7 Creature attacks:5 Creature attacks:3 Goblin attacks:4 ☐

4. Consider the following classes:

```
1  package ch.ethz.inf.java;
2
3  public class Creature2 {
4
5          protected Weapon weapon=new Weapon("Sword");
6
7          public Creature2(Weapon w){
8                  weapon = w;
9          }
10
11         public Weapon getWeapon(){
12                 return weapon;
13         }
14
15 }
```

```
1  package ch.ethz.inf.java;
2
3  public class Goblin2 extends Creature2 {
4
5          public Goblin2(Weapon w) {
6                  super(w);
7          }
8
9          public MeleeWeapon getWeapon(){
10                 return (MeleeWeapon) weapon;
11         }
12 }
```

```
1  package ch.ethz.inf.java;
2
3  public class Weapon {
4
5          private String name="Name not assigned";
6
7          public Weapon(String name){
8                  this.name=name;
9          }
10
11         public String getName(){
12                 return name;
13         }
14 }
```

```
1  package ch.ethz.inf.java;
2
3  public class MeleeWeapon extends Weapon {
4
5          public MeleeWeapon(String name) {
6                  super(name);
7          }
8  }
```

```
1  package ch.ethz.inf.java;
2
3  public class WeaponTest {
4
5          public static void main(String[] args) {
6                  Creature2 c=new Goblin2(new MeleeWeapon("Axe"));
7                  System.out.println(c.getWeapon().getName());
8          }
9  }
```

Which of the following are true?

a. The code does not compile. The method getWeapon() is duplicated. ☐
b. The code compiles correctly and prints "Name not assigned". ☐
c. The code compiles correctly and prints "Axe". ☐
d. The code compiles correctly and prints "Sword". ☐

5. Consider the following class:

```
1  package ch.ethz.inf.java;
2
3  public class ExceptionTest {
4
5          public static void main(String[] args) {
6                  ExceptionTest et = new ExceptionTest();
7                  try{
8                          et.method();
9                  }
10                 catch (Exception e){
11                         System.out.println("Inside catch");
12                 }
13                 finally{
14                         System.out.println("Inside finally");
15                 }
16                 System.out.println("After finally");
17         }
18
19         private void method() {
20
21                 throw new RuntimeException("Runtime Exception");
22         }
23 }
```

What happens when trying to compile/execute the code?

a. The code compiles correctly. ☐
b Exactly three messages are printed. ☐
c. The code does not compile. The method "method" must "throws" the exception. ☐
d. Exactly two messages are printed. ☐
e. The code does not compile. The compiler signals "unreachable code" after the finally clause. ☐

6. Consider the following class:

```
1  package ch.ethz.inf.java;
2  import java.util.ArrayList;
3  import java.util.Collection;
4
5  public class GenericsTest {
6
7          public static void main(String[] args) {
8                  //ArrayList<Creature> critArray=null;
9                  //ArrayList<?> critArray=null;
10                 ArrayList<Goblin> goblinArray =
11                         new ArrayList<Goblin>();
12                 //goblinArray.add(new Goblin());
13                 //critArray = goblinArray;
14                 System.out.print(critArray.get(0).toString());
15         }
16 }
```

Which line(s) would you uncomment to compile and execute the following code without raising exceptions?

a. Line 8. ☐
b Line 9. ☐
c. Line 12. ☐
d. Line 13. ☐

2 Exceptions (10 Points)

Discuss advantages and disadvantages of checked and unchecked exceptions, with particular respect to data exceptions. .....

3 Threads (13 Points)

Suppose we want to simulate a simple connection pool which can handle a maximum of 10 connections. We have a connection factory that produces connections, and a client that consumes them. Because we are writing a simulation, you don't have to create any real connection object; instead it is sufficient to only keep track of the number of currently active connections making use of the attribute and constants in class ConnectionPool. The situations in which the maximum or minimum number of available connections is reached should be handled by letting the connection factory or the client wait alternatively, rather than by raising an exception. Provide the missing code for the following classes, filling the dots where indicated. Methods or attributes may be missing. The elaboration will start from the ConnectionPoolTest class. There may be more dotted lines then needed.

```
1  package ch.ethz.inf.java;
2
3  public class ConnectionPoolTest
4  {
5          private ConnectionFactory connectionFactory;
6          private Client client;
7
8          public static void main(String[] args)
9          {
10                 ConnectionPoolTest cpt = new ConnectionPoolTest();
11                 cpt.start();
12                 System.out.println("End main thread");
13         }
14
15         private void start() {
16                 ConnectionPool connectionPool= new ConnectionPool();
17                 connectionFactory =
18                         new ConnectionFactory(connectionPool);
19                 client = new Client(connectionPool);
20
21                 //create and start the two threads here
22
23                 //.....
24
25                 //.....
26
27                 //.....
28
29                 //.....
30         }
31 }
```

```
1  package ch.ethz.inf.java;
2
3  public class ConnectionFactory implements Runnable
4  {
5          private ConnectionPool connectionPool;
6
7          //.....................................
8
9          //.....................................
10
11         //.....................................
12
13         //.....................................
14
15         //.....................................
16
17         //.....................................
18
19         //.....................................
20
21         //.....................................
22
23         //.....................................
24
25         //.....................................
26
27         //.....................................
28
29         //.....................................
30
31         //.....................................
32
33
34         /**
35          * Simulates connection requests
36          */
37         public void run()
38         {
39                 for(int i=0;i<100;i++)
40                 {
41                         createConnection();
42                 }
43         }
44 }
```

```java
package ch.ethz.inf.java;

public class Client implements Runnable
{
	private ConnectionPool connectionPool;


	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	//.....

	/**
	 * Simulates connection releases
	 */
	public void run()
	{
		for(int i=0;i<100;i++)
		{
			releaseConnection();
		}
	}
}
```

```java
package ch.ethz.inf.java;

public class ConnectionPool
{

	private int activeConnectionsCounter;
	private final int MAX_NUMBER_CONNECTIONS=10;
	private final int MIN_NUMBER_CONNECTIONS=0;
	//the actual reference to a container of connections
	//can be omitted here, because it is not relevant
	//to understand the whole mechanism

	//.....

	//.....

```

```
17          //.....
18
19          //.....
20
21          //.....
22
23          //.....
24
25          //.....
26
27          //.....
28
29          //.....
30
31          //.....
32
33          //.....
34
35          //.....
36
37          //.....
38
39          //.....
40
41          //.....
42
43          //.....
44
45          //.....
46
47          //.....
48
49          //.....
50
51          //.....
52
53          //.....
54
55          //.....
56
57          //.....
58
59          //.....
60
61          //.....
62
63          //.....
64
65          //.....
66
67          //.....
68
69          //.....
70
71          //.....
72
73          //.....
74
75          //.....
76
77          //.....
78  }
```

4 Reflection (10 Points)

Suppose we are dealing with an application that draws images embedded in documents. In terms of resources, we judge too expensive to draw all the images when a document is opened for the first time, thus we want to draw them lazily (on demand), as soon as their visualization is needed. In particular, when the method draw() in class Graphics is invoked, we want to load the image first and then pass the invocation to the actual Graphics object. We believe that the Java dynamic proxy facility can be useful to solve this problem. Analyze the code of the following four classes, filling the dots where indicated. There may be more dotted lines than needed.

```
1  package ch.ethz.inf.java;
2  /**
3   * The interface all the images
4   * have to conform to
5   */
6  public interface ImageHandlingInterface {
7
8          /**
9           * Draws an image
10          */
11         void draw();
12         /**
13          * Loads an image
14          */
15         void load();
16 }
```

```
1  package ch.ethz.inf.java;
2
3  public class Graphics implements ImageHandlingInterface{
4
5          public void draw() {
6                  System.out.println("Drawing image...");
7          }
8
9          public void load() {
10                 System.out.println("Loading image...");
11         }
12 }
```

```java
package ch.ethz.inf.java;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class ImageInvocationHandler implements InvocationHandler {

        private ImageHandlingInterface target;

        public ImageInvocationHandler(Object obj) {

                //sets the target image object
                //.............................
        }

        public Object invoke(Object proxy, Method method,
                        Object[] args)  throws Throwable {


                //.............................

                //.............................

                //.............................

                //.............................

                //.............................

                //.............................

        }

}
```

```java
package ch.ethz.inf.java;
import java.lang.reflect.Proxy;

public class ProxyTest {

        public static Object createProxy(Object obj)
        {
                //.............................................

                //.............................................

                //.............................................

                //.............................................
        }

        public static void main(String[] args) {
                ImageHandlingInterface image =new Graphics();
                ImageHandlingInterface image_proxy =
                        (ImageHandlingInterface) createProxy(image);
                //draw the image here, to test the program

                //.............................................
        }
}
```

5 Design Patterns (12 Points)

1. Consider the following extract from an informal meeting:
   "We have to program an application for a POS (Point Of Sale). The stakeholders tell us that that we should be taking into account that for every purchase the price of the comprised items needs to be computed according to particular details regarding the purchase and the purchaser. More precisely, a discount has to be offered in case the purchase total price is above a certain amount, or a certain number of identical items are bought. Also, there are certain categories of customers eligible for discounts. Finally, the conditions affecting the final purchase price are mutually exclusive and might change in time."
   Which design pattern(s) would you take into consideration to model the described situation and why?

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

2. Suppose you have to re-engineer an e-commerce application.
   Every time a customer registers to use the application, the application processes the customer data and assigns him/her to a certain profile, saves the data into a database, and sends a welcome email. At the moment, the implementation is centralized in class Customer, which has a method addDataToDB(), that processes the customer data, adds them to the database and sends a welcome email, using objects of classes DBHandler, ProfileHandler and WelcomeEmail.
   Which design pattern(s) would you take into consideration to refactor the described situation and why?