Java and C# in depth

ETH Zurich

Date: 3 June 2010

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: 105 minutes.
- Use a pen (**not** a pencil)!
- Please write your student number onto **each** sheet.
- All solutions have to 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 allowed. Exchanging documents during the examination would mean failing the examination.
- Electronic equipment (laptops, cell phones, PDA, etc.) are **not** allowed. 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	16	
2	15	
3	17	
4	15	
5	10	
TOTAL	73	

1 C# and Java Fundamentals (16 Points)

All questions below refer to the Java language version 5.0 and C# language version 3.0 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 checked or unchecked box is worth 0.5 points. An incorrectly checked or unchecked box is worth 0 points.

Example. Which of the following statements are true?

a. The sun is a mass of incandescent gas.	☒	0.5 points
b. $2 \times 2 = 4$	☐	0 points
c. Britney Spears is a honoured doctor of ETH.	☐	0.5 points
d. "Rösti" is a kind of sausage.	☒	0 points

1. Consider the following Java code:

```
String a = "Good luck!";
String b = "Good";
String c = b;
b += " luck!";
System.out.print(a == b);
System.out.print(' ');
System.out.print(b == c);
System.out.print(' ');
System.out.print(a.equals(b));
System.out.print(' ');
System.out.print(a.equals(c));
```

What is printed?

a. true false true false	☐
b. false true true true	☐
c. false false false false	☐
d. false false true false	☐

2. Consider the following C# code (similar to the Java code from the previous question):

```
string a = "Good luck!";
string b = "Good";
string c = b;
b += " luck!";
Console.Write(a == b);
Console.Write(' ');
Console.Write(b == c);
Console.Write(' ');
Console.Write(a.Equals(b));
Console.Write(' ');
Console.Write(a.Equals(c));
```

What is printed?

a. true false true false	☐
b. false true true true	☐
c. false false false false	☐
d. false false true false	☐

3. You would like a method of your **public** C# class to be visible to other classes inside the same assembly, as well as to every subclass (including those outside the assembly). Which visibility modifier provides exactly the desired access?

   a. **internal** ☐
   b. **protected** ☐
   c. **protected internal** ☐
   d. It is not possible to express this level of visibility in C#. ☐
   e. This is not needed, because it is not allowed to add subclasses outside the assembly. ☐

4. Consider the following C# code:

```
1  class Test {
2      public static int x = 3;
3      public static int y;
4
5      public int a = x;
6      public int b;
7
8      static Test() {
9          x = 5;
10     }
11
12     public Test() {
13         x++;
14     }
15
16     static void Main() {
17         Test t1 = new Test();
18         Test t2 = new Test();
19         Console.Write(Test.x);
20         Console.Write(' ');
21         Console.Write(Test.y);
22         Console.Write(' ');
23         Console.Write(t2.a);
24         Console.Write(' ');
25         Console.Write(t2.b);
26     }
27 }
```

   What is printed as a result of running Main? (A question mark denotes an undefined value, i.e., any integer value can be printed.)

   a. 6 0 5 0 ☐
   b. 7 0 6 0 ☐
   c. 6 ? 5 ? ☐
   d. 3 0 3 ? ☐
   e. 4 0 4 0 ☐
   f. The code does not compile ☐

5. Consider the following Java code:

```
try {
        try {
                throw new RuntimeException();
        } catch (RuntimeException e) {
                System.out.println("Inner runtime exception");
                throw e;
        } catch (Exception e) {
                System.out.println("Inner exception");
        } finally {
                System.out.println("Finally!");
                throw new Exception();
        }
} catch (RuntimeException e) {
        System.out.println("Outer runtime exception");
} catch (Exception e) {
        System.out.println("Outer exception");
}
```

Which of the following lines are printed?

a. "Inner runtime exception" ☐
b. "Inner exception" ☐
c. "Finally!" ☐
d. "Outer runtime exception" ☐
e. "Outer exception" ☐
f. The code does not compile ☐

6. Which of the following declarations produce a compilation error in Java? (Remember that Integer is a subclass of Number and ArrayList is a subclass of List.)

a. Number[] a = **new** Integer[10]; ☐
b. ArrayList<**int**> list1 = **new** ArrayList<**int**>(10); ☐
c. List<Integer> list2 = **new** ArrayList<Integer>(10); ☐
d. ArrayList<Number> list3 = **new** ArrayList<Integer>(10); ☐
e. ArrayList<?> list4 = **new** ArrayList<Integer>(10); ☐
f. ArrayList<? **extends** Number> list5 = **new** ArrayList<Integer>(10); ☐
g. ArrayList list6 = **new** ArrayList<Integer>(10); ☐

2 Inheritance, polymorphism, dynamic binding: a comparison between C# and Java (15 Points)

Illustrate and compare the characteristics of inheritance, polymorphism, and dynamic binding in Java and C#, specifying differences and common features. The presentation should be concise, clear and self contained. You can assume a general knowledge of object-oriented concepts. You are welcome to include a few clear examples.

3 Java Persistence (17 Points)

Consider the evolution of a simple class BankAccount, and the corresponding management of its stored instances.

1. The following method Main.main creates a BankAccount object, performs some operations on it, prints some information on the console, and finally serializes the object previously created.
   In comment (–1–), provide the console output produced after executing line 9.

```
1  //Used with class BankAccount version 1 only
2  public class Main
3  {
4          public static void main(String[] args) throws Exception
5          {
6                  BankAccount ba = new BankAccount("001");
7                  ba.deposit(new BigDecimal("1.50"));
8                  ba.withdraw(new BigDecimal("3"));
9                  System.out.println(ba);
10
11                 // -1 -.....
12
13                 MySerializer s = new MySerializer();
14                 s.serialize(ba);
15         }
16 }
```

```
1  public class MySerializer
2  {
3          public void serialize(Serializable target) throws Exception{
4                  OutputStream os =
5                          new FileOutputStream("exam_ser_test.txt");
6                  ObjectOutput oo = new ObjectOutputStream(os);
7                  oo.writeObject(target);
8                  oo.close();
9          }
10
11         public Object deserialize() throws Exception {
12                 InputStream is =
13                 new FileInputStream("exam_ser_test.txt");
14                 ObjectInput oi = new ObjectInputStream(is);
15                 Object obj = oi.readObject();
16                 oi.close();
17                 return obj;
18         }
19 }
```

```
1  public class BankAccount implements Serializable //Version 1
2  {
3
4          private static final long serialVersionUID = 1L;
5          private String code;
6          private BigDecimal total_withdrawals;
7          private BigDecimal total_deposits;
8          private transient String lastOperation;
9
10         public BankAccount(String custCode)      {
11                 setCode(custCode);
12                 total_deposits = BigDecimal.ONE;
13                 total_withdrawals = BigDecimal.ZERO;
14                 setLastOperation("creation");
15         }
16
17         public void setCode(String code) {
18                 this.code = code;
19                 setLastOperation("set code");
20         }
21
22         public BigDecimal getBalance() {
23                 setLastOperation("view balance");
24                 return new BigDecimal(
25                 total_deposits.subtract(total_withdrawals)
26                 .toString());
27         }
28
29         public void deposit(BigDecimal amount)  {
30                 total_deposits = total_deposits.add(amount);
31                 setLastOperation("deposit");
32         }
33
34         public void withdraw(BigDecimal amount) {
35                 total_withdrawals = total_withdrawals.add(amount);
36                 setLastOperation("withdraw");
37         }
38
39         public String toString() {
40                 return "Account: " + code + " .Balance: " +
41                 getBalance().toString() +
42                 ". Last op.: " + lastOperation;
43         }
44
45         public void setLastOperation(String lastOperation) {
46                 this.lastOperation = lastOperation;
47         }
48 }
```

2. The following method Main2.main retrieves the object previously stored.
   In comment (–2–), provide the console output produced when the execution (on the object serialized by method Main1.main) reaches line 7, assuming the same version 1 of class BankAccount is available to Main2.main

```
1  public class Main2
2  {
3          public static void main(String[] args) throws Exception
4          {
5                  MySerializer s = new MySerializer();
6                  BankAccount ba2 = (BankAccount)(s.deserialize());
7                  System.out.println(ba2);
8
9                  // -2- ..........................................
10
11                 // -3- ..........................................
12
13                 // -5- ..........................................
14         }
```

3. Consider now re-executing the same method Main2.main with a new version 2 of class BankAccount, shown in the following page.
   In comment (–3–), provide the console output produced when the execution (on the object serialized by method Main1.main with version 1 of BankAccount) reaches line 7, assuming version 2 of class BankAccount is available to Main2.main.

4. In comment (–4–) in the following page, implement a method readObject() of class BankAccount that helps to correctly retrieve a BankAccount object previously stored in version 1 into a BankAccount object based on version 2. "Correctly" means that no exception is raised and all attributes are initialized to their correct values.

5. Consider now one more execution of method Main2.main.
   In comment (–5–), provide the console output produced when the execution (on the object serialized by method Main1.main with version 1 of BankAccount) reaches line 7, assuming version 2 of class BankAccount is available to Main2.main, including your implementation of method readObject().

Hints: The following API of class ObjectInputStream will be useful:

- The abstract static nested class ObjectInputStream.GetField of class ObjectInputStream provides access to the persistent fields read from the input stream.
- The method **abstract** Object get(String name, Object val) of the nested class ObjectInputStream.GetField gets the value of the field referenced by name from the stream. Argument val is the default value to use if the field referenced by name does not have a value.
- The method ObjectInputStream.GetField readFields() of class ObjectInputStream reads the persistent fields from the stream and makes them available by name.

```
1  public class BankAccount implements Serializable //Version 2
2  {
3          private static final long serialVersionUID = 1L;
4          private String code;
5          private transient String lastOperation;
6          private BigDecimal balance;
7
8
9          public BankAccount(String custCode)     {
10                 setCode(custCode);
11                 balance = BigDecimal.ONE;
12                 setLastOperation("creation");
13         }
14
15         public BigDecimal getBalance() {
16                 setLastOperation("view balance");
17                 return balance;
18         }
19
20         public void setCode(String code) {
21                 this.code = code;
22                 setLastOperation("set code");
23         }
24
25         public void deposit(BigDecimal amount) {
26                 balance = balance.add(amount);
27                 setLastOperation("deposit");
28         }
29
30         public void withdraw(BigDecimal amount) {
31                 balance = balance.subtract(amount);
32                 setLastOperation("withdraw");
33         }
34
35         public String toString() {
36                 return "Account: " + code + " .Balance: " +
37                 getBalance().toString() +
38                 ". Last op.: " + lastOperation;
39         }
40
41         public void setLastOperation(String lastOperation) {
42                 this.lastOperation = lastOperation;
43         }
44 }
```

```
1  public class BankAccount implements Serializable {
2          //Version 2, continued
3
4          private void readObject(ObjectInputStream ois)
5                  throws IOException, ClassNotFoundException {
6
7                  // - 4 - .....
8
9                  .....
10
11                 .....
12
13                 .....
14
15                 .....
16
17                 .....
18
19                 .....
20
21                 .....
22
23                 .....
24
25                 .....
26
27                 .....
28
29                 .....
30         }
31 }
```

4 C# Threads (15 Points)

Suppose you are writing a C# program that simulates some physical process and visualizes the results.

A class Model encapsulates the functionalities related to simulation, whereas another class Viewer is for the visualization tasks. The interface of Model consists of method **int** NextValue(), which performs a simulation step and returns the resulting value (of type **int**) of the observed quantity. The interface of Viewer consists of method **void** Display(**int** value), which includes a new value to the graphical representation of the process under simulation. Assume that both methods require significant CPU time to execute.

Class MainProgram uses Model and Viewer to perform a specified number of simulation steps. On each step the new value of the quantity of interest is calculated using NextValue and then visualized using Display. Your initial implementation calls these two methods sequentially (see MainProgram.ExecuteSequentially), but you are not satisfied with the performance. You speculate that on your two-core machine you could get some speedup if you let NextValue start executing the next simulation step while Display is still processing the previous value (you implement this behavior in MainProgram.ExecuteConcurrently).

Moreover, the Main method should compare execution times of the sequential and concurrent versions to check that some speedup is achieved.

Your task

Fill in the blanks in the following code templates so that:

- ExecuteConcurrently executes methods CalculateAll and DisplayAll in different threads.
- ExecuteConcurrently displays the same results as ExecuteSequentially.
- Method Main indeed measures the execution times of the two variants (sequential and concurrent).

For simplicity, our implementation of NextValue and Display does not perform any actual computation, but just lets some time elapse.

Note that the number of simulation steps numSteps is unspecified and could be arbitrarily large. Correspondingly, your implementation should *not* store a collection of all previous simulation results.

You may add methods and/or attributes to the class MainProgram as appropriate.

```
class Model {
    private int value = 0;

    public int NextValue() {
        // Let some time elapse:

        ..........................................................................
        value++;
        return value;
    }
}

class Viewer {
    public void Display(int value) {
        // Let some time elapse:

        ..........................................................................
        Console.WriteLine(value);
    }
}

class MainProgram {
    static Model model;
    static Viewer viewer;
    static int numSteps = // unspecified constant

    static void ExecuteSequentially() {
        model = new Model();
        viewer = new Viewer();
        for (int i = 0; i < numSteps; i++) {
            viewer.Display(model.NextValue());
        }
    }


    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................


    static void CalculateAll() {
        for (int i = 0; i < numSteps; i++) {

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................
```

```
        }
    }

    static void DisplayAll() {
        for (int i = 0; i < numSteps; i++) {

            ..........

            ..........

            ..........

            ..........

            ..........
        }
    }

    static void ExecuteConcurrently() {
        model = new Model();
        viewer = new Viewer();

            ..........

            ..........

            ..........

            ..........

            ..........

            ..........

            ..........

            ..........

            ..........

            ..........
    }

    static void Main(string[] args) {
        DateTime start;
        TimeSpan span;

        start = DateTime.Now;
        ExecuteSequentially();
        span = DateTime.Now - start;
        Console.WriteLine(span.TotalMilliseconds);

        start = DateTime.Now;
        ExecuteConcurrently();
        span = DateTime.Now - start;
        Console.WriteLine(span.TotalMilliseconds);
    }
}
```

5 Methods as Objects in C# and Java (10 Points)

Sometimes we might want to use methods as objects, e.g. to pass them as arguments to other methods, as illustrated by the example in Figure 1.

In Figure 1, method quickSortCData sorts an array of CData objects (its first argument). To allow for different sorting criteria (e.g., descending order of attribute ID or alphabetical order of attribute name) quickSortCData takes a second argument sorter, from which the ordering of any two CData objects could be queried when necessary.

In the example, class CData has in particular the attributes ID, of type **int**, and name, of type String; both classes OrderInAlphName and OrderInDescID have a method is_less_than providing the ordering of two CData objects according to their name and ID, respectively; method sortCDataInOrder relies on quickSortCData to sort.

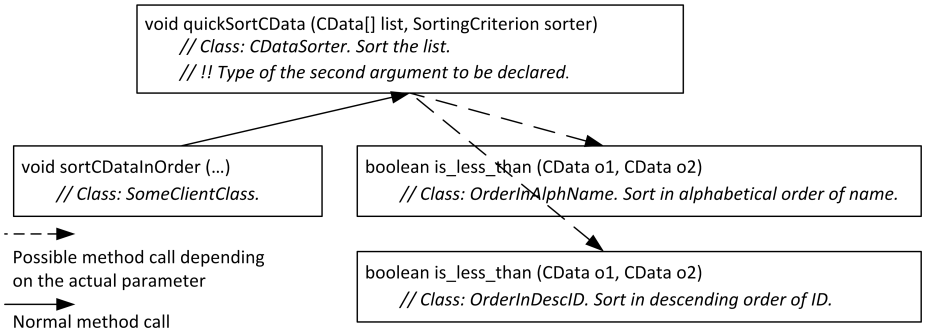

Figure 1: An example of passing methods as arguments.

For each of the following points, provide solutions both in Java and in C#. In the C# solution you have to use delegates.

1. Provide a declaration for the type SortingCriterion, i.e. the type of the second parameter of quickSortCData;

.....

.....

.....

.....

.....

.....

.....

.....

2. Describe the relationship among types SortingCriterion, OrderInAlphName, and OrderInDescID;

.....

.....

.....

.....

.....

.....

.....

.....

3. Assume that we have an array of CData named dataArray, and that we want to sort the objects in dataArray in descending order of ID. Provide an example of call to quickSortCData which achieves this result. Make sure to properly initialize the parameter sorter before the actual call to quickSortCData.

.....

.....

.....

.....

.....

.....

.....

.....