Java and C# in depth

ETH Zurich

Date: 27 May 2013

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.
- 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.
- 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.
- 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 exam supervisors if you feel disturbed during the exam.

Good luck!

Question	Number of possible points	Points
1	16	
2	12	
3	12	
4	15	
5	10	
TOTAL	68	

1 Java and C# Fundamentals (16 Points)

All questions below refer to the Java language version 7 and C# language version 5.0 as taught in the course. 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. Casting and Autoboxing

   a. In C#, all value types (built-in or user-defined) can be boxed. ☐
   b. In C#, no explicite type cast is necessary to box and unbox value types. ☐
   c. In Java, all conversions between built-in primitive types are implicit. ☐
   d. In Java and C#, no cast is necessary to convert an `int` to a `float`, even though you loose precision. ☐

2. Method redefinition and dispatching.

   Consider the following C# classes:

```
public class A
{
  public virtual string m() { return "A"; }

  public string T { get; set; }

  public A() { T = m(); }
}

public class B : A
{ public new string m() { return "B"; } }

public class C : A
{ public override string m() { return "C"; } }
```

   and the following corresponding declarations:

```
A a = new A();
B b = new B();
C c = new C();
A x = new B();
A y = new C();
```

   (a) What does the following statement print?

```
Console.WriteLine(a.m() + b.m() + c.m() + x.m() + y.m());
```

   (Spaces between letters are added for readability.)

   a. A A C A C ☐
   b. A B C A C ☐
   c. A B C B C ☐
   d. A B C A A ☐

(b) What does the following statement print?

```
Console.WriteLine(a.T + b.T + c.T + x.T + y.T);
```

(Spaces between letters are added for readability.)

a. A A C A C ☐
b. A B C A C ☐
c. A A C A A ☐
d. The statement does not compile ☐

3. Genericity (in Java).

Consider the following declaration of a generic class `GC` in Java:

```
import java.util.*;

public class GC <T extends List<Number>> {}
```

Which of the following definitions produce a compilation *error* (not just warning) in Java?
(Remember that, in the Java typesystem, `Integer` is a subclass of `Number`, and `ArrayList` is a subclass of `List`.)

a. `GC ex_a = new GC();` ☐
b. `GC<Number> ex_b = new GC<Number>();` ☐
c. `GC<List<Number>> ex_c = new GC<List<Number>>();` ☐
d. `GC<List<Number>> ex_d = new GC<>();` ☐
e. `GC<List<Integer>> ex_e = new GC<List<Integer>>();` ☐
f. `GC<List<Number>> ex_f = new GC<List<Integer>>();` ☐
g. `GC ex_g = new GC<List<Number>>();` ☐
h. `GC<List<Number>> ex_h = new GC<ArrayList<Number>>();` ☐
i. `GC<ArrayList<Number>> ex_i = new GC<ArrayList<Number>>();` ☐

4. Exception (in Java).

Consider the following Java code:

```
class Test{
  public static void main(String[] args) {
    try {
      try {
        throw new Exception();
      }
      catch (RuntimeException e) {
        System.out.println("Inner runtime exception");
      }
      finally {
        System.out.println("Finally");
      }

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

```
    }
  }
}
```

Which of the following statement(s) about the program behavior is/are true?

a. The program prints "Inner runtime exception" ☐
b. The program prints "Finally" ☐
c. The program prints "Exception caught" ☐
d. The program prints "Outer runtime exception" ☐
e. The program prints "Outer exception" ☐
f. The program terminates with an exception ☐

5. Value and Reference types (in C#). Consider the following C# snippet:

```
struct Sample {
        private int x;
        public int Increment() {
        this.x = this.x + 1;
        return this.x;
        }
}

class Foo
{
        public readonly Sample s = new Sample();
}

class Program{
        static void Main(string[] args) {
        Foo f = new Foo();
        System.Console.WriteLine(f.s.Increment());
        System.Console.WriteLine(f.s.Increment());
        System.Console.WriteLine(f.s.Increment());
        }
}
```

What is behaviour of this program?

a. The program prints "1 2 3" ☐
b. The program prints "0 0 0" ☐
c. Some exception will be thrown ☐
d. The program prints "1 1 1" ☐
e. The program won't compile ☐

6. Java Serialization and versioning. Consider the following two versions of the Java class `Test`:

```
class Test implements Serializable{
        //Version 1
        private int data = 5;
        String[] stringArray =  {"a","b","c"};
        transient double temp = 1.1;
}

class Test implements Serializable{
        //Version 2
        private int data = 5;
        String[] stringArray =  {"a","b","c"};
        double temp = 1.1;
}
```

What can you do to make sure that you can deserialize objects of version 1 from within objects of version 2?

a. Doing nothing works: the default deserialization mechanism will accept objects of version 1 from within objects of version 2. ☐
b. Insert in version 1 a `private static final long serialVersionUID` attribute and initialize it to the right value e.g. using the Eclipse IDE support. ☐
c. Insert in version 1 a `private static final long serialVersionUID` attribute and providing a custom value, e.g. `1L`. ☐
d. Insert in version 2 a `private static final long serialVersionUID` attribute and initialize it to the right value e.g. using the Eclipse IDE support. ☐
e. Insert in version 2 a `private static final long serialVersionUID` attribute and providing a custom value, e.g. `1L`. ☐

7. Reflection (in C#). Which of the following combinations of binding flags will work to invoke method
`internal void AMethod (double, int)`
passing default values?

a. `BindingFlags bf = BindingFlags.InvokeMethod | BindingFlags.NonPublic;` ☐
b. `BindingFlags bf = BindingFlags.InvokeMethod | BindingFlags.Internal;` ☐
c. `BindingFlags bf = BindingFlags.CreateInstance | BindingFlags.NonPublic;` ☐
d. `BindingFlags bf = BindingFlags.CreateInstance | BindingFlags.Internal` ☐
e. `BindingFlags bf = BindingFlags.DefaultArgs | BindingFlags.NonPublic;` ☐

Old questions start

8. 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 ☐

9. 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 ☐

10. 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. ☐

11. 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 ☐

12. 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 ☐

13. 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 Java Fundamentals (12 Points)

The class `Address` is used to store street addresses consisting of a street and postal code. The subclass `ExtendedAddress` has an additional field for a country code, and can be used for international addresses. Finish the implementation of these two classes, so that we can use them to index hashtables.

2.1 Class Address

```
public class Address {

    // Street part of address.
    public final String street;

    // Postal code of address.
    public final int postalCode;

    // Initilize street to s, and postal code to p.
    public Address(String s, int p) {
        street = s; postalCode = p;
    }

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....

    .....
```

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

    ..........

    ..........

    ..........
}
```

2.2 Class ExtendedAddress

```
public class ExtendedAddress extends Address {

    // Country code of address.
    public final String country;

    // Initilize street to s, postal code to p, and country code to c.
    public ExtendedAddress(String s, int p, String c) {
        super(s, p); country = c;
    }

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

    ..........

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

```
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
    .....
}
```

3 Bounded queue(12 points)

Bounded queues are widely spread in the multithreading. In this question you need to implement a bounded thread-safe queue with 2 essential operations: enqueue and dequeue. Enqueue blocks the calling thread if queue stores more than K elements. Dequeue blocks the calling thread, if queue is empty. There can be many threads blocked on one operation. When a blocking condition is gone (an element is added or removed), waiting threads should be waked up. In this question we ask you to implement the singnaling constructs with Monitors & single lock object. The implementation should be thread-safe.

Quick reference for Monitor's methods

- Monitor.Enter(object lock) Enters the critical sections, assosiated with lock.
- Monitor.Exit(object lock) Leaves the previously entered critical section.
- In C# locks are recursive (or reentrant), i.e. one thead can enter the several critical section without any problems.
- Monitor.Wait(object lock) suspends the executioning thread, ceases all previosly obtained locks. Maybe waked upon notification.
- Monitor.Pulse(object lock) and Monitor.PulseAll(object lock) notifies (awakens) one (all) waiting on the lock.

Your tasks:

- Add necessary private fields to your implementation
- Implement constructor, which accepts a maximum queue capacity. Note, that capacity should be positive - the implementation should fail on incorrect input by throwing an appropriate exception.
- Implement Enqueue and Dequeue methods. Note, that there can be more than one thread enqueing and dequeing at the same time. You should use a *single* lock for your signaling constructs.
- Implement Properties Max and Count.

Listing 1: Selector class

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace Concurrency
{
  public sealed class BoundedQueue<T>
  {
    private readonly Queue<T> _queue;
    //define here the neccessary fields

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

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

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

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

    //define here the constructor,
    //which accepts the maximum amount of elements.
```

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

 ..........

 ..........

 ..........

 ..........

 ..........

 ..........

//Enqueues a new item to the queue.
//Blocks, if the queue already contains Max elements.
public void Enqueue(T item)
{

   ..........

   ..........

   ..........

   ..........

   ..........

   ..........
}

//Dequeues the last item from the queue.
//Blocks, if the queue is empty
public T Dequeue()
{

   ..........

   ..........

   ..........

   ..........

   ..........

   ..........
}

// Gets the maximum amount of items.
public int Max
{

   ..........

   ..........

   ..........
}
```

```
    // Gets the current amount of items, stored in the queue
    public int Count
    {
        ..........................................................................
        ..........................................................................
        ..........................................................................
        ..........................................................................
        ..........................................................................
    }
  }
}
```

4 Delegate (13 points)

In C#, one way to represent a criterion regarding certain type of objects is to use delegates. For example, a delegate of type `delegate bool IntCriterion(int t);` can encode a criterion regarding integers, and be used to select the integers satisfying the criterion from a list.

In this question you need to program a library to support composite criteria construction using existing delegates and logical connectives like negation and conjunction. The library should also facilitate the application of such composite criteria in selecting objects from enumerable containers.

Note:

- To keep the question text short, the design of the class hierarchy is not the optimum;
- Using of namespaces `System`, `System.Collections.Generic`, and `System.Linq` are omitted from all the listings.

4.1 Composite criterion construction

Listing 2 shows a use case of composite criterion construction, and Listing 3 gives the sketch of the abstractions for different criteria and their visitors. Add code to the dotted lines to complete the abstractions in a way that supports the use case.

Your tasks:

- Provide *a lambda expression* at location Ⓐ. The expression takes a single argument of type `int` and returns `true` if and only if the argument can be divided by 3;
- Provide *an anonymous method* at location Ⓑ. The method takes a single argument of type `int` and returns `true` if and only if the argument can be divided by 4;
- Add code at locations Ⓒ, Ⓓ, Ⓔ, and Ⓕ so that objects of type `CriterionVisitor` could be used to visit `Criterion` objects (the Visitor Pattern).

Listing 2: Use case for composite criterion construction

```
namespace CompositeCriterion{

    class Program{

        static void Main(string[] args){
            PrimitiveCriterion<int> p1 = new PrimitiveCriterion<int>(

                ..................................................................Ⓐ
                );
            PrimitiveCriterion<int> p2 = new PrimitiveCriterion<int>(

                ..................................................................Ⓑ
                );
            NegCriterion<int> n1 = new NegCriterion<int>(p2);

            // 'a1' is satisfied by integers that can be divided by 3
            // but not by 4.
            AndCriterion<int> a1 = new AndCriterion<int>(p1, n1);
        }
    }
}
```

Listing 3: Composite criterion classes

```
namespace CompositeCriterion{
```

```
public delegate bool criFunc<T>(T t);

interface Criterion<T>{

    ..... ©

    .....
}

interface CriterionVisitor<T>{

    ..... Ⓓ

    .....

    .....
}

class PrimitiveCriterion<T>: Criterion<T>{
    private criFunc<T> criterionFunc;

    public criFunc<T> CriterionFunc{
        get { return criterionFunc; }
        protected set { criterionFunc = value; }
    }

    public PrimitiveCriterion(criFunc<T> cf){
        CriterionFunc = cf;
    }

    ..... Ⓔ

    .....

    .....
}

class NegCriterion<T> : Criterion<T>{
    private Criterion<T> subCriterion;

    public Criterion<T> SubCriterion{
        get { return subCriterion; }
        protected set { subCriterion = value; }
    }

    public NegCriterion(Criterion<T> sc){
        SubCriterion = sc;
    }

    // Other details omitted
}

class AndCriterion<T> : Criterion<T>{
    private Criterion<T> leftCriterion;
    private Criterion<T> rightCriterion;

    public Criterion<T> LeftCriterion{
        get { return leftCriterion; }
        private set { leftCriterion = value; }
    }
```

```
        public Criterion<T> RightCriterion{
            get { return rightCriterion; }
            private set { rightCriterion = value; }
        }

        public AndCriterion(Criterion<T> lc, Criterion<T> rc){
            LeftCriterion = lc;
            RightCriterion = rc;
        }

        ..... (F)

        .....

        .....
    }
}
```

4.2 Checking fulfillment of criterion

Class `CriterionChecker` in Listing 4 implements the visitor pattern to check if an object fulfills an criterion, and method `check` is the interface for the check.

Please provide necessary code at locations (G), (H), and (I) to realize the intended semantics of `CriterionChecker.check`.

Note: No need to provide the `visit` method for `NegCriterion` at location (I).

Listing 4: Criterion fulfillment checker class

```
namespace CompositeCriterion{

    class CriterionChecker<T> : CriterionVisitor<T>{

        // Declare necessary attributes here.

        ..... (G)

        .....

        // Check if 'subject' satisfies 'criterion'.
        // Return true if yes, otherwise false.
        public bool check(T subject, Criterion<T> criterion){
            this.subject = subject;

            ..... (H)

            .....
        }

        // Define necessary methods here.
        // Note:  No need to provide the visit method for NegCriterion.

        ..... (I)

        .....

        .....

        .....

        .....
```

```
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
        ..........
    }
}
```

4.3 Using criterion objects in element selection

When using `Criterion` objects to select elements from a container, there are different choices as for how to select and return the result selection. For example, *eager selection* checks all the elements from the container and stores the ones satisfying the criterion into a separate object, which is then returned to the client, while *lazy selection* searches for and returns the next element satisfying the criterion in the container, *only* when the element is accessed by the client through the return selection.

Class `Selector` in Listing 5 intends to support both lazy and eager selection. Code in Listing 6 uses the criterion object from Listing 2 in element selection, where the two `foreach` loops should output the same sequence of integers.

Your tasks:

- Please briefly compare the performance of eager selection and lazy selection (Describe at least one advantage for each.)

  Lazy selection:

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

  Eager selection:

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

- Please add necessary code at locations Ⓙ and Ⓚ, so that the return objects from both methods `LazySelected` and `EagerSelected` could be enumerated to access all the objects satisfying `cri` from `source`, in the lazy and the eager way respectively.

Listing 5: Selector class

```
namespace CompositeCriterion{
    class Selector<T>{
        public static CriterionChecker<T> checker =
                new CriterionChecker<T>();

        public static IEnumerable<T> LazySelected(
                IEnumerable<T> source, Criterion<T> cri){

            ..........Ⓙ
```

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

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

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

        public static IEnumerable<T> EagerSelected(
                IEnumerable<T> source, Criterion<T> cri){
            List<T> result = new List<T>(10);

            ..........................................................................Ⓚ

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

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

            ..........................................................................
        }
    }
}
```

Listing 6: Use case for using criterion objects in element selection

```
namespace CompositeCriterion{
    class Program{

        static void Main(string[] args){
            // Continuing from Listing 2

            // Generate an array of integers [1, 2, 3, ..., 20]
            int[] values = Enumerable.Range(1, 20).ToArray();

            // Should print: 3,6,9,15,18,
            foreach(int i in Selector<int>.LazySelected(values, a1)){
                Console.Write(i);
                Console.Write(",");
            }
            Console.WriteLine();

            // Should print: 3,6,9,15,18,
            foreach(int i in Selector<int>.EagerSelected(values, a1)){
                Console.Write(i);
                Console.Write(",");
            }
        }
    }
}
```