A20487

No calculator permitted in this examination

UNIVERSITY OF BIRMINGHAM

School of Computer Science

First Year – Degree of BSc with Honours
Artificial Intelligence and Computer Science
Computer Science
Computer Science with Study Abroad
Computer Science with Business Studies

First Year – Degree of BEng/MEng with Honours
Computer Science/Software Engineering

First Year – Joint Degree of BEng/MEng with Honours
Electronic and Software Engineering

First Year – Joint Degree of BSc/MSci with Honours
Mathematics and Computer Science
Pure Mathematics and Computer Science

06 18190

Software Workshop 1

Summer Examinations 2010

Time allowed: 3 hours

[Answer ALL Questions]

SECTION A

[Use a Separate Answer Book for THIS Section]

1. Answer the following questions about the Java programming language.

(a) What is the value of `z` after this line is executed? [1%]

```
double z = 9 * 3 / 2 - 0.5;
```

(b) How many possible values can a boolean variable take on? [1%]

(c) The following for loop has three parts missing from its first line. Write down that first line with the three parts filled in so that when the for loop runs it will print out the numbers 80, 40, 20, 10 and 5.

```
for (??? ; ??? ; ??? ) {
    System.out.println(i);
}
```

[3%]

(d) Suppose an interface is defined as follows:

```
public interface MessageSender {
    public double sendMessage();
}
```

Suppose also we want to use a declaration

```
MessageSender m = new UrgentMessage();
```

What words should appear in the gaps in the following sentences?

Class `UrgentMessage` must _____ MessageSender.
Class `UrgentMessage` must _____ `sendMessage`.

[2%]

2. The following Java code has mistakes. Given an integer `n`, the code was intended to print the largest integer `x` such that `x*x + x + 1 <= n`. In other words, `x*x + x + 1 <= n`, but `(x+1)*(x+1) + (x+1) + 1 > n`. It is assumed that `n >= 1`. Line numbers are included for convenience. Remember that integer division rounds towards zero.

```
(1)  int currentMin = 0;
(2)  int currentMax = n;
(3)  while (currentMin < currentMax) {
(4)     int mid = (currentMin + currentMax)/2;
(5)     if (mid*mid + mid + 1 <= n) {
(6)        currentMin = mid;
(7)     } else {
(8)        currentMax = mid;
(9)     }
(10) }
(11) System.out.println(currentMin);
```

(a) Suppose, in the NetBeans IDE, a breakpoint is put on line 5 and the above code (as part of a suitable main method) is run under the debugger with `n` having the value 40.

(i) For each of the first 4 hits, write down the values of currentMin, currentMax and `mid` that can be seen in the debugger. [4%]

(ii) What happens if you continue with 2 more hits and why? [4%]

(b) Consider the truth of the following conditions at the execution point immediately before the loop test is evaluated in line (3). (They are loop invariants.)

```
currentMin <= currentMax
(currentMin * currentMin) + currentMin + 1 <= n
(currentMax +1)*( currentMax +1) + (currentMax+1) + 1 > n
```

(i) Why are these conditions true on each iteration? (Your answer should be general and apply to every possible use of the code, not just to the example in part (a).) [4%]

(ii) If the conditions are still true when looping finishes, explain how they ensure that currentMin is the answer required. [3%]

3. Suppose a class `c` includes (amongst other things) a private field `n` of type `int` (assumed >=0) and a method `fact` for calculating n! (factorial of n). Note that 0!=1.

```
/**
 * Calculate factorial of n
 * @return n!
 */
public int fact() {
    int i = 0;
    int f = 1;
    /* loop invariant:
     *   0 <= i <= n and
     *   f = i!
     */
    while (i < n) {     //loop test
          i = i+1;
          f = f * i;
    }
    return f;
}
```

(a) Suppose (in NetBeans or using a similar debugger) a breakpoint has been set at the line containing the loop test. Execution will stop immediately before each evaluation of the loop test. The method `fact` is called on an instance of the class `c` in which `n` has value 6.

(i) How many times will the breakpoint be hit?
(ii) What will be the values of `i` and `f` on the *first 3* times and on the *last* time that the breakpoint is hit?
(iii) Why does the loop invariant say `i <= n`, while the loop test says `i < n`?

[5%]

(b) Consider a new method, similar to `fact`, but instead of multiplying `1*2*3*...*n` we only multiply consecutive numbers from interval `[from,n]`. In other words, we calculate: `from*(from+1)*(from+2)*...*n`. (Assume `from>=1`). Write a non-defensive, private method `factPartND` implementing such a calculation.

```
private int factPartND(final int from, final int n)
```

Write a definition of `factPartND`. It should use a while loop, as in `fact`, with a suitable loop invariant. Also include a Javadoc heading, with a requires condition.

[5%]

(c) Now write a defensive, public method `factPart` for calculating `from*(from+1)*(from+2)*...*n`. It should call `factPartND` but throw an `IllegalArgumentException` if the parameters are invalid. Describe how the Javadoc from part (b) needs to be modified. [5%]

4. A public class `Power` is to have a private double field `x`. The field can be initialized from a constructor parameter. If no parameter is supplied, the field is initialized with 0. The class has a public getter method `getX` for `x`. It also has a public method `powN` that takes an integer argument `n`, calculates the n-th power of `x` (x multiplied with itself n times) and returns the result.

(a) Write the entire Java definition of the class `Power`. In body of the powN method use a `for(...) { ... }` loop. Use a non-defensive definition of powN. [6%]

(b) Answer the following questions:

(i) What is the meaning of: x is a *private* field? [1%]

(ii) Write Java code, to be used outside the class of `Power`, that will declare a variable `z` of type `Power`, initialize it to a new instance of `Power` with `x` taking value 5, calculate 5*5*5*5 using `powN` method, and print the result to `System.out`. [2%]

(c) We require a subclass `BoundedPower` of `Power`, partly written as

```
public class BoundedPower extends Power {

    public static final double MAX_X = 1000000;
    // invariant: x <= MAX_X;

    //constructor and powN needed here

}
```

It should behave just like `Power`, except that it never allows `x` to exceed 1000000.

Complete this by writing Java definitions for the relevant constructor and `power`. [6%]

5. We would like to sum up elements of a sequence given by s.valAtIndex(i), i=0,1,2,... Here s is of type Seq. Seq is an interface that has a method valAtIndex, with an integer parameter (the index 0,1,2,...) and double result.

   (a) Write the *interface* `Seq`. [1%]

   (b) Write a class `Geometric`, implementing `Seq`, so that each instance `s` represents a geometric series `s.valAtIndex(0), s.valAtIndex(1), s.valAtIndex(2), ...` such that the i-th element `s.valAtIndex(i)` is equal to the i-th power of the base `b`, i.e. b^i (recall that $b^0 = 1$).

   Use a private field for `b`. The field is initialised in the constructor. [6%]

   (c) Write code that calculates the sum of the first 16 elements of the series 1, 3, 9, 27, 81... The code does so by first creating a suitable instance of `Geometric` that will be of type `Seq`. [2%]

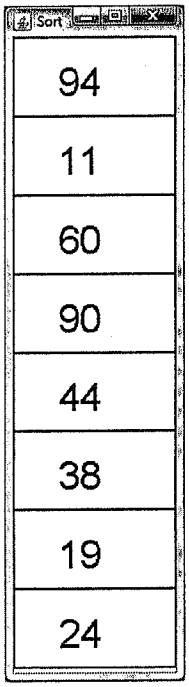
   (d) We now want to sum up the first 1000 elements of the series 1, 1/2, 1/4, 1/8, 1/16, ...
   Demonstrate the power of using interfaces by re-using the code created in (c). What needs to be changed? Write down the new code. [6%]

For reference, *java.lang.Math* defines a method *Pow(double a, double b)* that returns value of the first argument raised to the power of the second argument.

SECTION B

[Use a Separate Answer Book for THIS Section]

6. The window to the right is displayed by a program that is going to be developed to demonstrate how sort methods work. The window contains a Panel described by the following Java class:




```
import java.awt.*;

public class SortPanel extends Panel {

   private SortCanvas topCanvas=null,
                      clickedCanvas=null;

   public SortPanel () {
      setLayout(new GridLayout(8,0));
      for (int c=0; c<8; c++) {
         int nextValue = (int)(Math.random()*100);
         SortCanvas nextCanvas =
                      new SortCanvas(nextValue);
         add(nextCanvas);
         if (c==0) topCanvas=nextCanvas;
      }
   }
}
```

A `SortPanel` object contains eight `SortCanvas` objects described as follows:

```
import java.awt.*;

public class SortCanvas extends Canvas {
   private int value;

   public SortCanvas(int value) {
      setValue(value);
   }

   public void paint(Graphics g) {
      Dimension d = getSize();
      int canvHeight = d.height;
      g.drawRect(0,0,d.width-1,canvHeight-1);
      Font f = new Font("arial",0,canvHeight/2);
      g.setFont(f);
      g.drawString(String.valueOf(value), 40, canvHeight*3/4);
   }

   public void setValue(int value) { this.value=value; }

   public int getValue() { return value; }

   public void background(Color c) {
      setBackground(c);  repaint();
   }
}
```

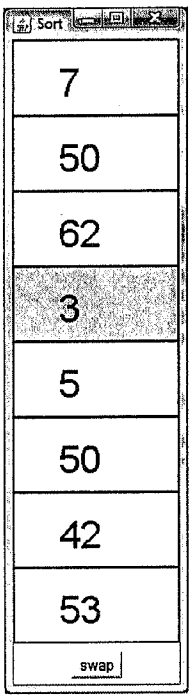
Question 6 continues over the page

Question 6 continued

(a) Write a Java class `SortFrame` that can be used to create windows like the one illustrated above (of size 160x600 pixels), and write a main method to display one such window. [4%]

(b) Now modify the `SortPanel` class so that if one of the canvases in a `SortPanel` object is *clicked*, that canvas should be highlighted with a background colour of pink (`Color.PINK`) and any previously highlighted canvas being set back to a background colour of white. (use the variable `clickedCanvas` to remember which canvas was clicked last). [5%]

(c) Now modify your Frame class so that the window displayed includes a bottom `Panel` with a 'swap' button as shown. Your `SortPanel` class should also be modified to implement `ActionListener` so that it can respond to a click on the button by swapping the value displayed in the top `SortCanvas` with the value displayed in the most recently clicked `SortCanvas` (if any).



The following is a listing of the interface class ActionListener from the Java API:

```
package java.awt.event;

import java.util.EventListener;

public interface ActionListener extends
EventListener {

    public void actionPerformed(ActionEvent e);

}
```

[7%]

7. (a) The heading for the class `TreeMap` in the Java library package `java.util` includes the following elements:

```
public class TreeMap<K,V> ...
```

The method `put` defined in the class `TreeMap` starts with:

```
public V put(K key, V value) ...
```

Explain the the use of the types `K` and `V` in the above code fragments.

[3%]

(b) Write a declaration for a `TreeMap` variable and construct a `TreeMap` object that could be used to store a simple dictionary for mapping a word in one language (String) to the corresponding word in another language (String).

[2%]

(c) Write a class `Dictionary` that can be used to represent an English-French dictionary. Your dictionary class should use two `TreeMap` objects, one for mapping an English word to a corresponding French word and the other for mapping a French word to a corresponding English word. Your class should include the following methods

`add` for adding a given pair of words to the dictionary (the pair should be added to both mappings);

`getFrench` returns the French word corresponding to a given English word

`getEnglish` returns the English word corresponding to a given French word

Your answer should include a main method that tests your class by adding three pairs of words to the dictionary and calling the `getFrench` and `getEnglish` methods.

[6%]

Question 7 continues over the page

Question 7 continued

(d) A small DVD shop has a very simple stock control system. Each DVD is currently represented by a Java object constructed from the class:

```
public class DVD
{
   private String stockCode;
   private String title;
   private double price; // price per item
   private int quantity; // no of items in stock

   public DVD(String s, String t, double p, int q)
   {  setStockCode(s);  setTitle(t);
      setPrice(p);  setQuantityInStock(q);  }

   // plus get and set methods for
   // stockCode, title, price, quantity

}
```

The entire stock is going to be stored as a `TreeMap` using the `stockCode` as a key. Write a java class `DVDStock` that *extends* the class `TreeMap` appropriately, thus inheriting all the methods defined for a `TreeMap`. Your class should include the following methods

`add` that adds a new DVD object to a DVDStock map;

`getPrice` that returns the price of the object with a given stock code.

Your answer should include a main method that tests your class by adding three DVD objects to a DVDStock map and calling the `getPrice` method.

[6%]