UNIVERSITY OF EDINBURGH course CS0077

FACULTY OF SCIENCE AND ENGINEERING

DIVISION OF INFORMATICS

SCHOOL OF COMPUTER SCIENCE

Computer Science 1 Bh

Degree Examination

Date: Saturday 25th May 2002

Time: 12:00–13:30 (one and a half hours)

Place: Adam House

Room: Ground Floor

Board of examiners

Chair: D.K. Arvind

External Examiner: R. Dyckhoff

Instructions to candidates

1. Check that the question paper contains pages 1–13 after this cover page. If it does not, inform the invigilator.
2. You should attempt as many questions as possible.
3. The questions in this paper vary widely in difficulty. During your first pass through the paper, you are advised not to dwell on questions to which the answer is not readily apparent.
4. Write your answer to each question in the box or table provided. If you wish to write more in answer to a question, continue on the blank page opposite, indicating that you have done so.
5. Please write *legible* and *concise* answers.
6. The marks allocated to each part of a question are indicated in the margin. There are 100 marks in total.

Question 1

(a) A useful technique when building large systems is the ability to *hide* parts of the system from the view of other parts. Java provides *visibility modifiers* to help manage hiding. Java's visibility modifiers define four *visibility levels*, shown in the first column of the table below. Complete the table to specify the visibility of methods or fields declared with the different visibility levels. [*6 marks*]

	visible in the defining class?	visible in another class in the same package?	visible in a subclass in another package?	visible in a non-subclass in another package?
public	✓	✓	✓	✓
protected	✓			
package	✓			
private	✓	×	×	×

(b) Java provides a notion of *package* to help structure large systems. Recall that a package consists of a collection of classes, and has a hierarchical name.

i. What line of Java would you add to the top of the `Student.java` file to declare the `Student` class to be part of the package `ed.informatics.cs1.studentdata`?

[*1 mark*]

ii. What line of Java would you add to the top of another class in a different package, to be able to use the `Student` class (and no other classes) without writing long identifiers?

[*1 mark*]

iii. What line of Java would you add to the top of another class in a different package, to be able to use all classes in the `studentdata` package, without writing long identifiers?

[*1 mark*]

iv. What is a disadvantage of using statements like those in **ii** and **iii** in your programs?

[*1 mark*]

Question 2

Consider the following Java code:

```
abstract class FishShopMenu {
    public int price() { return 0; }    // return price of menu item
    public abstract String toString();  // string name of menu item
}

class Fork extends FishShopMenu {
    public String toString() { return "Wooden fork"; }
}
```

Objects belonging to the class `FishShopMenu` represent items on the menu at a fish shop. The `toString` method provides the item name and the `price` method returns its price, as an integral number of pence.

i. How much does a wooden fork cost? [*1 mark*]

ii. Give the definition of a `Haddock` class, supposing that a haddock portion costs £1.85. [*3 marks*]

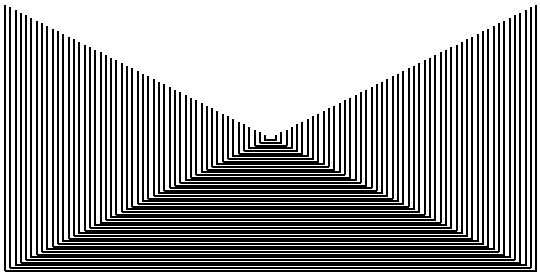
iii. What is wrong with the following definition of a `Haggis` class, supposing that a haggis portion costs £1.20? [*1 mark*]

```
class Haggis extends FishShopMenu {
   public int price() { return 120; }
}
```

iv. The fish shop is running a special offer, which allows any item to be "super-sized" for only £0.80 extra. Give the definition of a `Supersized` class which implements this behaviour. The constructor should take a `FishShopMenu` object which is to be super-sized. [*5 marks*]

Question 3

(a) *Infinity Software Systems* wish to display their corporate logo on their Web page in a Java applet. Their logo consists of 50 concentric open rectangles, in black on a white background, as shown below.



The centre of the logo is at co-ordinate (200,200). The width of the rectangle increases by 4 pixels and its height by 2 pixels in going from each rectangle to its nearest enclosing neighbour. Implement the `paint()` method for this applet. Recall that a *`Graphics`* context provides methods `setColor()`, `drawRect()` and `drawLine()` and that Java defines `Color.white` and `Color.black`. [*5 marks*]

(b) Some colours are pre-defined in Java, but others can be defined by the programmer. What is the method by which colours are specified? [*2 marks*]

(c) What are the attributes of a *font* which a Java programmer can specify? [*3 marks*]

Question 4

(a) Describe the use of *host names* and *port numbers* in client/server systems. [*2 marks*]

(b) What role do *proxy servers* play in client/server systems? [*2 marks*]

(c) What are *race conditions* in client/server systems? [*1 mark*]

(d) Complete the following client template code to enable it to communicate with a server which is listening on port 5055 of the machine `java.ed.ac.uk`.

```
import java.io.*;
import java.net.*;

class Client {
   public static void main (String[] args) throws IOException {
      Socket sock =
           [                                                    ];   [1 mark]

      // Communication has been established
      InputStreamReader is = new InputStreamReader
         (  [                                  ]  );                 [1 mark]
      BufferedReader input = new BufferedReader(is);
      PrintWriter server = new PrintWriter
         (  [                                  ]  );                 [1 mark]

      [                          ]("Message sent");                  [1 mark]
      server.flush();

      System.out.println("Reply: " + [                          ]); [1 mark]

   }
}
```

Question 5

Consider the following *container stuffing* problem:

> *Parcels range in size from 1 to 10 units and containers can hold up to 10 units. Given a sequence $p_1, \ldots, p_n$ of parcels, find a way of stuffing them into containers so that the minimum number of containers is used to hold the parcels.*

(a) One proposed solution is the *greedy* approach where we stuff containers one at a time always choosing the largest unstuffed parcel that will still fit in the container. When all the remaining parcels are too big to fit in the current container we go on to a new one, and so on. Can you find a sequence of parcels for which greedy approach fails to find a stuffing that minimises the number of containers used? Explain how your example causes the greedy approach to fail. *[Hint: you don't need to use parcels bigger than size 4.]* [*4 marks*]

(b) Consider the following recursive program that solves the container stuffing problem. The method invocation `stuffIt(s,parcel)`, where `parcel` is the vector of parcel sizes and `s` indicates we should solve the problem for the parcels between index `s` and the end of the vector. This invocation returns a `Pair`, where the `containers` instance variable counts the number of containers used and `spare` counts the amount of spare space left in the last container. After reading the program you should answer the questions below.

```
public class Pair  {
    public int containers;
    public int spare;

    public Pair () {
        containers = 0;
        spare = 0;
    }
}
```

```
import java.util.*;

class Stuff {

    public static int size = 10;

    public static Pair add(Pair r, int p) {
        if (r.spare<p) {
            r.containers++; r.spare = size-p;
        } else { r.spare = r.spare-p;}
        return(r);
    }

    public static Pair min(Pair r1, Pair r2) {
        if (r1.containers<r2.containers ||
            (r1.containers==r2.containers
             && r1.spare>=r2.spare)) {
            return(r1);
        } else { return(r2); }
    }
```

```
    public static Pair stuffIt(int start, Vector parcel) {
        Pair result;
        Integer first;

        if (start==parcel.size()) { .... }                          [A]

        start++;
        result = add(stuffIt(start,parcel),
                     ((Integer)parcel.elementAt(start-1)).intValue()) ;

        for(int i = start; i<parcel.size(); i++) {
            first = (Integer)parcel.elementAt(i);
            parcel.setElementAt(parcel.elementAt(start-1),i);
            result = min(result,add(stuffIt(start,parcel),first.intValue()));
            parcel.setElementAt(first,i);
        }
        return(result);
    }
}
```

i. In the program the line marked `[A]` is incomplete. Complete it in this box: [*2 marks*]

iii. Provide a, reasonably accurate, estimate of the running time of the program for n parcels. Justify your answer. [*4 marks*]

Question 6

(a) The class below contains two Java methods, `eight()` and `add()`. Give the equivalent Java byte code which the Java compiler would produce for these methods. [*5 marks*]

```
class Methods {
    static int eight() {
        return add(add(1, 3), 4);
    }
    static int add(int x, int y) {
        return x + y;
    }
}
```

(b) The following Java byte code method was written by hand by an inexperienced Java byte code programmer and will cause an error when it is executed. Identify the error by showing the intermediate steps in the execution of the method [*5 marks*]

Method *int six()*
0 iconst_1
1 iconst_2
2 iconst_3
3 iadd
4 iadd
5 iadd
6 ireturn

Question 7

(a) In constrast to locally compiled Java main programs, *applets* are restricted programs that can be downloaded from the Internet, for example. List three kinds of operation that a locally compiled program is allowed to carry out, but that a Java applet is not. [*3 marks*]

(b) Why are these sorts of *sandboxing* restrictions on applets considered to be a good idea? [*2 marks*]

(c) Give an example of a problem which a hostile applet *could* cause, despite sandboxing. [*3 marks*]

(d) The following method is special for Java applications because it is never normally invoked by the programmer.

```
public static void main (String[] args)
```

Give the heading of a method which is similarly special for Java applets. As above, include the access control modifiers, the return type and the formal parameter list. [*2 marks*]

Question 8

(a) Explain what is meant by the *fetch-execute cycle* of a von Neumann machine. [*3 marks*]

(b) Explain the distinction between **i)** a *machine instruction* and **ii)** a *micro-instruction*. [*2 marks*]

(c) The MIPS R2000 instruction `add $24,$4,$9` requests the processor to add the contents of Register 4 and Register 9, and place the result in Register 24. Using a register-transfer notation, or diagrams if you prefer, describe the sequence of operations that would be carried out by the datapath under the direction of the control unit. [*5 marks*]

Question 9

(a) Show the following equivalences are true using chains of logical equivalences.

i. $\neg(\neg P \wedge Q) \Leftrightarrow \neg Q \vee P$

[2 marks]

ii. $P \Rightarrow (Q \wedge P) \Leftrightarrow P \Rightarrow Q$

[2 marks]

(b) Consider the following truthtable for a 3 argument propositional operator X.

P	Q	R	$X(P,Q,R)$
f	f	f	f
t	f	f	t
f	t	f	f
t	t	f	t
f	f	t	t
t	f	t	t
f	t	t	t
t	t	t	f

Write down the CNF (Conjunctive Normal Form) formula for X suggested by this truthtable.

[2 marks]

[Question continues on next page]

(c) Use a semantic tree to show that the propositional formula $((P \vee Q) \Rightarrow Q) \Rightarrow Q$ is not a tautology. If you have a choice of formulas to discharge, discharge first the formula closest to the tree root. Explicitly state a counter-example truth assignment suggested by the tree.

[*4 marks*]

Question 10

(a) Consider the static Java method

```
public static int sPower(int a, int n) {
  if (n == 0)
    return a;
  else
    return sPower(a, n-1) * sPower(a, n-1);
}
```

Show by induction on n that

$$\forall n : \mathbb{N}.\ \texttt{sPower}(a, n) = a^{2^n}$$

where $\mathbb{N}$ is the non-negative integers $\{0, 1, 2, \ldots\}$.

i. How does the base case argument go?

[*1 mark*]

ii. For the step case, what is the inductive hypothesis and what is the statement to be shown?

[*2 marks*]

iii. What is the proof of the step case?

[*2 marks*]

[Question continues on next page]

(b) Complete the the *requires* (precondition), *ensures* (postcondition) and *loop invariant* assertions for the following Java method which tests if integer `i` is contained in integer array `seq`:

```
//@ requires
```

[*1 mark*]

```
//@ ensures
//@   \result <==>
```

[*2 marks*]

```
static boolean member(int i; int[] seq) {
  boolean val = false;

  //@ loop_invariant
  //@ val <==>
```

[*2 marks*]

```
  for (int j = 0; j != seq.length; j++) {
    if (i == seq[j]) val = true;
  }
  return val;
}
```

Recall, a *loop invariant* assertion concerns the state of the program just before each evaluation of the loop termination test. The special specification variable `\result` is used in `ensures` statements to refer to the return value of a method.

Write your assertions using the extended Java-like syntax illustrated in class, or, if you prefer, more logic-like syntax.