UNIVERSITY OF LONDON

GOLDSMITHS COLLEGE

B. Sc. Examination 2012

DEPARTMENT OF COMPUTING

IS52028A Principles and Applications of Programming

Duration: 3 hours

There are four questions in this paper. You should answer ALL FOUR questions. Each question carries 25 marks. The marks for each part of a question are indicated at the end of the part in [.] brackets.

There are 100 marks available on this paper.

No calculators should be used.

THIS PAPER MUST NOT BE REMOVED FROM THE EXAMINATION ROOM

Question 1

(a) i. A prime number is a positive integer, larger than 1, that can only be divided, without remainder, by itself and 1. Provide an English language procedure for testing if a given integer $p > 1$ is prime. [2]

ii. Draw a flow chart for the procedure described in part a.(i). [3]

iii. Write a *procedural* Java program `Prime` which implements the procedure of part a.(ii). Your program should test program arguments for primality e.g if your program is invoked with the command line instruction `java Prime 12 97 2`, your program responds with

```
12 is not a prime
97 is prime
2 is prime
```

You may wish to use the following information taken from the Java api:

```
class java.lang.Integer

public static int parseInt(String s)
    Parses the string argument as a signed decimal integer.
```

[10]

(b) i. What is the string pool and how does it differ from the garbage collectable heap? Your answer should consider advantages and disadvantages. [5]

ii. Consider the class StringTest below.

```
package exam;

public class StringTest {

    public static void main(String[] args) {

        String s = "42";
        String t = "42";
        String u = new String("42");

        System.out.println(s == t);
        System.out.println(s.equals(t));
        System.out.println(s == u);
        System.out.println(s.equals(u));
    }
}
```

Its output is:

```
true
true
false
true
```

Explain these results. [5]

Question 2

(a) i. Write a class `Broadcast` with a single method `public void userIn()` which, using Java input streams, endlessly reads in strings typed at the command line. [4]

ii. Write an interface `Listener` with a single abstract method `public void inform(String s)`. [2]

iii. Modify your `Broadcast` class so that it includes a method `public void register(Listener l)` which enables `Listener`'s to register their interest.
Also, adapt the method you wrote in part a.(i) so that messages typed at the command line are distributed to all registered listeners. [6]

(b) i. The JVM is able to run more than one call stack in order to give the appearance of multi-tasking. Explain what a call stack is and how the JVM accomplishes multi-tasking. Illustrate your answer with code snippets and with appropriate diagrams. [8]

ii. The multi-tasking described in part b.(i) can create a problem known as deadlock. Explain what deadlock is, and provide an example of how it might occur. [5]

Question 3

(a) A stack is an ordered collection of elements. Items can be placed onto the top of the stack (pushing) and can be removed from the top (popping). For example if a stack is represented by $(3, 5, 4, 2)$ with the most recent additions to the left, then pushing 8 and then 13 will leave the stack as $(13, 8, 3, 5, 4, 2)$. Popping the stack will retrieve 13 and leave the stack as $(8, 3, 5, 4, 2)$. Apart from popping and pushing, a client might enquire if a stack is empty.

i. Write an API specification for a stack of generic elements. [5]

ii. Provide an implementation of a stack by following your API in part a.(i). Include code to test your class by constructing a stack of `SimpleBooks` and calling methods on that stack. [10]

(b) The rest of this question concerns programs written for register machines. The available instructions and language syntax is given in the appendix of this exam and is exactly the language that was studied in the lectures. In the following program listing, instruction line numbers `n:` have been added; they play no role in the execution of the program.

The program

```
1:    [A] X <- X - 1
2:        Y <- Y + 1
3:        IF X != 0 GOTO A
```

has the effect of copying the contents of register `X` into `Y`, unless `X` is initialised to zero, in which case the program terminates with `Y = 1`.

i. Provide a snapshot sequence of the program when `X` is initialised to the value 2. [3]

ii. Alter the program of part b.(i) so that `X` is copied into `Y` for all values of `X`, including `X = 0`. [3]

iii. Write a program that copies register `X` into `Y` and leaves the value of `X` unchanged. [4]

Question 4

This is an example of an android UI specification:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="vertical" >
  <TextView android:id="@+id/text"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Hello, I am a TextView" />
  <Button android:id="@+id/button"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Hello, I am a Button" />
</LinearLayout>
```

(a) Give an advantage and a disadvantage of using an xml user interface definition rather than defining the UI programmatically. [2]

(b) Describe the User Interface that is created. [3]

(c) Describe the LinearLayout element and what it does in the above sample? [3]

(d) The above example simply creates a graphical layout. How would you make the button have an effect on your program? [8]

(e) How would you write an android app that allows the user to enter a thermometer reading of a human temperature (constrained to be between 35 and 42 degrees) and feedsback to the user if it is high (over 37.5) or very high (over 40)? Explain your design decisions. [9]

Appendix

The register machine considered here has four different instructions, an increment instruction, a decrement instruction, a dummy instruction and a conditional branch (jump) instruction:

```
V <- V + 1
V <- V - 1
V <- V
IF V != 0 GOTO L
```

Increment, dummy and jump instructions perform as expected but, since variable values are restricted to the non-negative integers i.e. 0, 1, 2, ..., `V <- V - 1` will decrement `V` unless `V = 0`, in which case `V` is left unchanged.

There are also labelled instructions of the form `[L] instruction` e.g.

```
[L]  Z <- Z + 1
```

is a labelled increment instruction.

Each instruction is separated by a new line and within each line 'tokens' (`V, =, +, -, 0, 1, IF, != , GOTO, L, [L]`) are separated by one or more whitespace characters.

`V` is either an input variable `X1, X2, X3`..., a local variable `Z1, Z2` ...or the output variable Y. Additionally, `X` and `Z` are synonymous with `X1` and `Z1`.

The label `L` is one of `A1, B1, C1, D1, E1, A2, B2` ...and once more `A1` is synonymous with `A` etc.

The local variables Z and the output variable Y are initialised to zero. Some or all of the input variables may be initialised to non zero values. Otherwise, input variables are also initialised to zero.

A program is a list of instructions of finite (but unlimited) length. A computation commences at the first instruction and proceeds to successive instructions in the list until a jump to an undefined label is encountered, or until the last instruction is reached and there is no jump to a preceding instruction. Conventionally, the label `E` is undefined so that an instruction `IF V != 0 GOTO E` will terminate the program.

A label may be repeated: in this case any jump will be to the first occurrence of the label in the program.