THIS PAPER IS NOT TO BE REMOVED FROM THE EXAMINATION HALLS

UNIVERSITY OF LONDON **CO2220 ZB**

BSc Examination

COMPUTING AND INFORMATION SYSTEMS and CREATIVE COMPUTING

Graphical object-oriented and internet programming in Java

Wednesday 2 May 2018: 10.00 – 13.00

Time allowed: 3 hours

Candidates should answer **FOUR** questions only. Full marks will be awarded for complete answers to **FOUR** questions. All questions carry equal marks and full marks can be obtained for complete answers to **FOUR** questions. The mark for each part of a question is indicated in [ ] brackets.

You must answer **TWO** questions from **Section A** and **TWO** questions from **Section B**.

There are 100 marks available on this paper.

Only your first **TWO** answers for Section A, and your first **TWO** answers for Section B, in the order that they appear in your answer book, will be marked.

No calculators should be used.

SECTION A

Answer TWO questions from this section.

Question 1

(a) Consider the class *Woo*, and its child class, *SonOfWoo,* below.
Note that *SonOfWoo* has FIVE numbered constructors. The constructor numbered 5 is a valid constructor, *i.e.* it will not cause any compilation errors.

```
public class Woo{
     private int i;
     private String s;

     public Woo(int i){
          this.i=i;
     }

     public Woo(String s){
          this.s = s;
     }

     public Woo(int i,String s){
          this.s = s;
          this.i = i;
     }
}


/***********************************/


class SonOfWoo extends Woo{
     private boolean b;

/*1*/     public SonOfWoo(int i, String s){
               this(s);
               super(i);
          }

/*2*/     public SonOfWoo(String s, boolean b){
               this.b = b;
               this(s);
          }

/*3*/     public SonOfWoo(float f){
               super(42);
          }
```

```
/*4*/      public SonOfWoo(boolean b){
               super("42");
               s = "woo";
           }

/*5*/      public SonOfWoo(String s){
               super(s);
           }
     }
```

(i) Which one of the following statements is correct? [2 marks]

(A) Constructor 1 is not valid because the `this` and `super` keywords cannot be used in the same constructor.

(B) Constructor 1 is not valid because the call to `super` must be the first statement if used in a constructor.

(C) Constructor 1 is valid.

(ii) Which one of the following statements is correct? [2 marks]

(A) Constructor 2 is not valid because the keyword `this` cannot be used twice in the same constructor.

(B) Constructor 2 is not valid because the keyword `this`, when used in a statement to call another constructor in same class, must come first.

(C) Constructor 2 is not valid because it attempts to directly access the private `boolean` variable *b.*

(iii) Which one of the following statements is correct? [2 marks]

(A) Constructor 3 is not valid because it has a `float` parameter but there is no `float` instance variable.

(B) Constructor 3 is not valid because the `float` parameter has not been referenced in any statements.

(C) Constructor 3 is not valid because the superclass instance variable *s* has not been initialised.

(D) Constructor 3 is valid.

(iv) Which one of the following statements is correct? [2 marks]

(A) Constructor 4 is not valid because the variable *s* has private access in *Woo* so cannot be directly accessed in *SonOfWoo.*

(B) Constructor 4 is not valid because the boolean parameter is not referenced in any statements.

(C) Constructor 4 is not valid because the instance variable *b* has not been initialised using the boolean parameter.

(D) Constructor 4 is valid.

(b) Consider the following class, *Orc*:

```
public class Orc{

     private String sound;
     private int age;

     public String getSound(){
          return sound;
     }

     public int getAge(){
          return age;
     }

     public static void main(String[ ] args){
          Orc orc = new Orc();
          System.out.println(orc.getSound());
          System.out.println(orc.getAge());
     }
}
```

The *Orc* class compiles and runs.

(i) Explain why the attempt to make an object of the class works, even though the class does not have a constructor. [3 marks]

(ii) Give the output of the *Orc* class. [2 marks]

(iii) Add methods to the *Orc* class, and statements to the main method, such that the value of the *sound* variable is set to "Grunt" and that of the *age* variable to 102. Note that the changes to the values of the instance variables should be made in the main method. [4 marks]

(c) Consider the *Novel* class, below:

```
public class Novel {
     protected String title;
     protected String author;
     protected String publisher;

     public Novel(String title,String author,String
     publisher){
          this.title = title;
          this.author = author;
          this.publisher = publisher;
     }
}
```

Extend the class to an *AudioBook* class. The *AudioBook* class has a `String` *narrator* instance variable, a constructor, and a method to return the value of the *narrator* variable. [8 marks]

Question 2

(a)

(i) Say whether each of the following statements are TRUE or FALSE: [6 marks]

(A) All variables contained in an `ArrayList` are objects; primitives are automatically wrapped into objects, and unwrapped back to primitives on insertion and retrieval.

(B) An `ArrayList` resizes to whatever size is appropriate.

(C) Arraylists use random access and are about as fast as arrays.

(D) The enhanced *for* loop can be used for iterating through what Java calls collections. It cannot be used with an `ArrayList` because Java does not considers the class to be a collection.

(E) Arraylists can be parameterised using a type parameter in angle brackets < >. This means that any object inserted into the list will be wrapped to the type in the brackets.

(F) Arraylists can be parameterised to primitive or object variables.

(ii) Given that when *short-circuit operators* are used in boolean expressions, the JVM will not check both sides of the boolean expression under certain conditions, say whether the following statements are TRUE or FALSE: [2 marks]

(A) & and | are short-circuit operators.

(B) It is good practice **not** to use the short-circuit operators in Boolean expressions.

(b)

(i) What error will the compiler find with class *Q2a*? [2 marks]

```
interface Trev{

     public abstract int sum();
}

class Q2a implements Trev{
     public static void main(String[] args){
          System.out.println("Initialising...");
     }
}
```

(ii) Consider the following:

```
abstract class Rex{
     String name;
     public abstract String getName();
     public abstract void setName(String name);

     public String toString(){
          return ("name is "+name);
     }
}
```

Say which one of the following statements about the above interface is TRUE: [3 marks]

(A) The *Rex* class will produce a compilation error because the *toString()* method is not abstract.

(B) The *Rex* class will produce a compilation error because an abstract class cannot have non-abstract instance variables.

(C) The *Rex* class will compile.

(iii) Given the following definition:

```
abstract class Vehicle{
     abstract boolean hasEngine();
     abstract int passengers();
}
```

say which of the following classes will compile successfully and which will not: [4 marks]

```
/*1*/      abstract class Bicyle implements Vehicle{
               abstract int noOfWheels();
           }

           /*********************************/

/*2*/      abstract class MountainBike extends Bicyle{
               public boolean hasEngine(){
                    return false;
               }

               public int passengers(){
                    return 0;
               }
               public int noOfWheels(){
                    return 2;
               }
           }

           /*********************************/


/*3*/      class Truck implements Vehicle{
               public boolean hasEngine(){
                    return true;
               }
           }

           /*********************************/

/*4*/      class Spaceship implements Vehicle{

               public boolean hasEngine(){
                    return true;
               }

               public int passengers(){
                    return 0;
               }
           }
```

(c) A *set* is an object where we can do the following: [8 marks]

(1) Take the intersection of two sets.

(2) Take the union of two sets.

(3) Check whether an Object is in a set.

(4) Check whether a set is empty.

Write an interface for the *set* object, using appropriate method names where required.

Question 3

(a) Say which of the following statements are TRUE, and which are FALSE:

(A) An example of event handling is listening for events such as a clicked button in a GUI, and taking some action in response. [8 marks]

(B) When a class needs more than one `JButton`, in order to implement different actions when different `JButtons` are clicked, the accepted solution is to implement the Listener interface (*e.g.* `ActionListener`) once for each button using inner classes.

(C) When a class needs more than one `JButton`, in order to implement different actions when different `JButtons` are clicked, the accepted solution is to have the Listener call back method (*e.g. actionPerformed()*) query the event source.

(D) Inner classes have access to all of their containing classes variables, **except** for private variables.

(E) Event sources (such as a `JButton`) can only be registered with one event handler.

(F) If a region is not specified then the single parameter *add()* method of *BorderLayout* defaults to the SOUTH region.

(G) The BorderLayout manager has 6 regions.

(H) Swing applications should **not** directly call the *paintComponent()* method.

(b) Consider the class *CirclesGUI* class below:

```
imimport javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class CirclesGUI implements MouseListener{
     JFrame frame;
     JButton dButton;
     CircleDrawPanel drawPanel;
     Color color = Color.red;
     int X; int Y;
     int diameter = 100;
     Random r= new Random();

     public static void main (String[] args){
          CirclesGUI gui = new CirclesGUI();
          gui.go();
     }

     //this method sets up the JFrame, including adding any
     components and listeners.
     public void go(){
          frame = new JFrame();
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

          drawPanel = new  CircleDrawPanel();
          frame.getContentPane().add(BorderLayout.CENTER,drawPanel);
          drawPanel.addMouseListener(this);

          frame.setSize(600,600);
          frame.setVisible(true);
     }

          public void mousePressed(MouseEvent e) {}
          public void mouseReleased(MouseEvent e) {}
          public void mouseEntered(MouseEvent e) {}
          public void mouseExited(MouseEvent e) {}

          public void mouseClicked(MouseEvent e) {
               X = e.getX();
               Y = e.getY();
               X = X - diameter/2;
               Y = Y - diameter/2;
               drawPanel.repaint();
          }
```

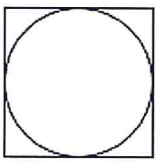
//CirclesGUI class continues on next page

```
//CirclesGUI class continued

     class CircleDrawPanel extends JPanel{
          public void paintComponent (Graphics g){
               super.paintComponent(g);
               Graphics2D g2=(Graphics2D)g;
               g2.setColor(color);
               g2.fillOval(X,Y,diameter,diameter);
          }
     }

     class DiameterListener implements ActionListener{
          public void actionPerformed (ActionEvent e){
          }
     }
}
```

(i) The circle can be viewed as being inside a square, with the square just large enough to contain the circle, as in the image below. The *X* and *Y* coordinates specify one of the corners of the square that the circle is notionally drawn inside. Which corner? [2 marks]



(A) The top right corner.

(B) The top left corner.

(C) The bottom left corner.

(D) The bottom right corner.

(ii) When *CirclesGUI* class is compiled and run it will show a red circle in a frame. When the user clicks anywhere in the frame, the circle will be redrawn such that the centre of the circle is now determined by where the user clicked in the frame. [2 marks]

In your answer book write down the statements that are centring the position of the circle around the place where the user clicked.

(iii) Add a `JButton` to the NORTH of the `JFrame` using `BorderLayout`. Set the text on the button to read "Click me to resize the circle." [3 marks]
NOTE: In your answer you can indicate what statements you are adding to the class and where you are adding them. You do not need to write out the entire class in your answer book.

(iv) The *CirclesGUI* class contains an inner class called *DiameterListener*. Make the inner class listen to the `JButton`. [3 marks]

(v) Complete the *actionPerformed(ActionEvent)* method of the *DiameterListener* class such that when the user clicks on the `JButton`, the circle is redrawn with a random diameter. The size of the random diameter is limited by the width of the *drawPanel* variable. With each redrawn circle, the circle's *X* and *Y* coordinates should be recalculated such that the centre of the circle remains in the same place. [7 marks]

Each time the user clicks the button the diameter changes, and the circle is redrawn once more.

SECTION B

Answer TWO questions from this section.

Question 4

(a)

(i) Say whether each of the following statements are TRUE or FALSE: [5 marks]

(A) A static method (such as main) can directly access a non-static (*i.e.* instance) variable.

(B) A static variable has the same value for every instance of the class.

(C) A static method can be called before an instance of the method's class has been made.

(D) When marked as final, static variables are constants. By agreed convention, they are named in uppercase with underscores separating words.

(E) An instance variable cannot be marked final.

(ii) Consider the classes *Happy* and *Happier*, below. When the *Happier* class is compiled, what error will the compiler find? [3 marks]

```
public class Happy{
     private String name;

     public Happy(String n){
          name = n;
     }

     final void askToDance(){
       System.out.println("Shall we dance
     the Foxtrot?");
     }
}

/************************************/
```

```
public class Happier extends Happy{

     public Happier(String n){
          super(n);
     }

     void askToDance(){
       System.out.println("Shall we dance
     the Rumba?");
     }
}
```

(b)

(i) Say whether the following statements are TRUE or FALSE: [3 marks]

(A) In order to make the best use of memory `StringBuilder` or `StringBuffer` should **not** be used when writing classes with a lot of `String` concatenation.

(B) `String` objects are immutable.

(C) *format()* is a `String` method that allows precise control over the format of printed Strings.

(ii) Consider the following class, *StringThang*

```
class StringThang{
     public static void main(String[] args){
          String eg = "hello/greetings/wassup=salutation";
          String[] r1 = eg.split("=");
          for (String token:r1) System.out.println(token);
          String[] r2 = r1[0].split("/");
          for (String token:r2) System.out.println(token);
     }
}
```

Give the output of the class when it is run. [3 marks]

(iii) Consider the *TestBox* class:

```
public class TestBox{

     Integer i;
     int j;

     public TestBox(){
          i = 5;
          j = 3;
     }

     public static void main(String[] args){
          TestBox t = new TestBox();
          t.go();
     }

     public void go(){
          Integer intObj = new Integer(j);
          int k = Integer.valueOf(intObj);
          j=i;
     }
}
```

Identify autoboxing in the above class. [3 marks]

(c) Consider the *ProspectiveStudent* and the *AdmissionsFileManager* classes, together with the contents of the *admissions.txt* file below:

```
public class ProspectiveStudent{
    private String name;
    private String gender;
    private String phoneNumber;
    private String emailAddress;
    private int id;
    private int progCode;
    private boolean offerMade;

    public ProspectiveStudent(int id, String name, String gender,
     String phoneNumber, String emailAddress, int progCode,
     boolean offerMade) {

       this.id = id;
       this.name = name;
       this.gender = gender;
       this.phoneNumber = phoneNumber;
       this.emailAddress = emailAddress;
       this.progCode = progCode;
       this.offerMade = offerMade;
    }

    //Some getters and setters here

    public String toString() {
       StringBuilder sb = new StringBuilder();

       sb.append("ID: ").append(id).append("\n");
       sb.append("Name: ").append(name).append("\n");
       sb.append("Gender: ").append(gender).append("\n");
       sb.append("Phone No: ").append(phoneNumber).append("\n");
       sb.append("Email: ").append(emailAddress).append("\n");
       sb.append("Programme: ").append(progCode).append("\n");
       sb.append("Offer made: ").append(offerMade).append("\n");

       return sb.toString();
    }
}
/*************************************************************/
```

```
import java.io.*;
import java.util.*;

public class AdmissionsFileManager {
    private static String filename = "admissions.txt";

    public static ArrayList<ProspectiveStudent> read() {
        Scanner in;
        try {
            in = new Scanner(new FileInputStream(filename));
        } catch (FileNotFoundException e) {
            System.err.println("Could not find " + filename);
            return null;
        }
        ArrayList<ProspectiveStudent> list = new
                            ArrayList<ProspectiveStudent>();
        while(in.hasNextLine()) {
          String id = in.nextLine();
          if (id.startsWith("ID: ")) {
          //since the line contains "ID" we can be sure that there
          are the required six extra lines following it

                ArrayList<String> details = new
                     ArrayList<String>();
                details.add(id);
                for (int i = 0; i < 6; i++) {
                    details.add(in.nextLine());
                }
                ProspectiveStudent ps = parsePS(details);
                    list.add(ps);
          }
        }
        in.close();
        return list;
    }

    private static ProspectiveStudent parsePS(ArrayList<String>
     details) {
        ArrayList<String> data = new ArrayList<String>();

        for (String s : details) {
            String[] split = s.split("\\:\\s+", 2);
            data.add(split[1]);
        }
        //some statements missing here
    }
}

/*************************************************************/
```

```
//contents of admissions.txt file, below
ID: 202
Name: Some Guy
Gender: MALE
Phone Number: 1234567890
Email Address: guy@example.com
Programme code: 100302
Offer made: false

ID: 501
Name: Jane Doe
Gender: FEMALE
Phone Number: +447000000
Email Address: jdoe@example.co.uk
Programme code: 100250
Offer made: true

ID: 693
Name: Mx Smith
Gender: OTHER
Phone Number: +331234567
Email Address: mxsmith@example.fr
Programme code: 100140
Offer made: true
//end of admissions.txt file
```

The method with the heading
public static ArrayList<ProspectiveStudent> read()
in the *AdmissionsFileManager* class has been written to read a text file (*admissions.txt*), and convert what it finds into a *ProspectiveStudent* object or objects. In order to parse the contents of the file to *ProspectiveStudent* objects, the method first checks that the line it has read contains "ID". If the line does contain "ID" the method puts that line, and the following six lines into an `ArrayList<String>`. It then sends the `ArrayList<String>` to the *parsePS(ArrayList<String>)* method.

The *parsePS(ArrayList<String>)* method's task is to turn the `ArrayList<String>` parameter it is given, into a *ProspectiveStudent* object, and return it.

Write the missing statements from the *parsePS(ArrayList<String>)* method of the *AdmissionsFileManager* class. You may assume that the method receives correct input so no error checking is necessary. [8 marks]

You may find the following extract from the Java API helpful:

parseBoolean

```
public static boolean parseBoolean(String s)
```

Parses the string argument as a boolean. The `boolean` returned represents the value `true` if the string argument is not `null` and is equal, ignoring case, to the string "true".

Example: `Boolean.parseBoolean("True")` returns `true`.
Example: `Boolean.parseBoolean("yes")` returns `false`.

Parameters:
`s` - the `String` containing the boolean representation to be parsed.

Returns:
the boolean represented by the string argument.

Question 5

(a)

(i) Java has two types of stream; connection streams for the connection itself, and chain streams. Chain streams can only connect to other streams. Connection streams are low level and read byte by byte.

`BufferedWriter` fills its buffer, and does not write to the file until the buffer is full. `BufferedReader` fills its buffer and does not read again until the buffer has been processed. Why is chaining a connection stream to a `BufferedWriter` or `BufferedReader` preferred to using a connection stream alone?

(A) Because using a buffer is more efficient as reading/writing to file is expensive compared to manipulating data in memory. [2 marks]

(B) Because using a buffer is less dangerous as reading/writing to file can be a source of I/O errors.

(C) Because using a buffer is more secure as it makes it harder for a third party to intercept.

(D) None of the above.

(ii) How could the `BufferedWriter` be forced to send data before its buffer is full? [2 marks]

(iii) Consider the *SourceViewer* class below:

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

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

public class SourceViewer{

     public static void main(String[] args){
          try{
               URL u = new URL("https://www.gold.ac.uk/");
               Reader r = new BufferedReader(new
                       InputStreamReader(u.openStream()));
               int c = -1;
               while ((c=r.read()) !=-1) System.out.print((char)c);
          }

          catch(Exception e){
               System.err.println(e);
          }

          finally{
               System.out.println("That's all folks!");
          }
     }
}
```

Say which of the following statements about the program are TRUE, and which are FALSE. [4 marks]

(A) It navigates to the given URL (if valid) and prints out any text it finds there, ignoring any images.

(B) It navigates to the given URL (if valid) and prints the source code that it finds there, with the original formatting of the HTML code.

(C) If an exception is thrown the only output is `"That's all folks!"`

(D) If an exception is thrown it prints an error message. The program always ends by printing `"That's all folks!"`

(b)

(i) Which of the following is saved when an object is *serialized*? [2 marks]

(A) An object's state, given by its instance variables is saved. Any objects that are referenced by the instance variables, and in turn any further objects referenced by their instance variables *etc.* are also saved.

(B) An object's static and instance variables are saved. Any objects that are referenced by the instance variables, and in turn any further objects referenced by their instance variables *etc.* are also saved.

(C) The object's source code.

(ii) Which of the following statements are TRUE, and which are FALSE? [4 marks]

(A) An object is serializable if its super class is serializable.

(B) Static variables **can** be serialized.

(C) A transient variable will assume the default/null value when its containing object is deserialized.

(D) Serialization can throw runtime exceptions.

(iii) At what point in the serializing/deserializing process could a `ClassNotFoundException` be thrown? [3 marks]

(c) Write a method with the heading: [8 marks]
public static ArrayList<String> deserialize(String filename)
The method attempts to open a file and deserialize its contents into an `ArrayList<String>`. If successful it returns the `ArrayList<String>`, if unsuccessful it prints an error message to standard output and returns null.

Question 6

(a)

(i) Why are some Java exceptions checked and some unchecked? Choose one of the statements below that best describes the reason: [2 marks]

(A) Because exception checking is memory heavy, only those most critical to successful internet and network programming are checked.

(B) Unchecked exceptions are a legacy issue from earlier versions of Java. They are now deprecated.

(C) Exceptions should only be used for problems caused by an application's interactions with the outside world, and not by the internal logic of the code. Therefore exceptions that may arise due to poor logic on the part of the developer are not checked.

(ii) Identify the checked exceptions in the following: [6 marks]

```
ArithmeticException
ArrayIndexOutOfBoundsException
ClassCastException
ClassNotFoundException
EOFException
FileNotFoundException
IllegalArgumentException
IOException
NumberFormatException
NullPointerException
SocketException
UnknownHostException
```

(b)

(i) A thread can be in several different states: one of them is BLOCKED.

There are a number of reasons why a thread may become BLOCKED. State which of the following are genuine reasons why the thread scheduler may move a thread to the BLOCKED state:

(A) the thread is waiting for data. [4 marks]

(B) the thread's *run()* method has completed.

(C) the thread is trying to access a locked object.

(ii) Which of the following four words describe genuine thread states? [2 marks]

NEW
RUNNABLE
STARTED
STOPPED

(iii) Synchronization locks the methods of an object, threads cannot enter without a key, and only one key is available per object. Identify which of the following statements are genuine reasons that synchronization should be used sparingly: [4 marks]

(A) because method access is slowed.

(B) because it increases the likelihood of the object failing unpredictably because of memory issues.

(C) because synchronization brings with it the hazards of thread deadlock (where each thread has the key that the other needs).

(c) Consider the following Java class, which has 3 missing code fragments:

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

public class SimplestServer{
     private int port = 6006;

     public void go (){
        try{
             ServerSocket serverSocket = new ServerSocket(port);
             System.out.println("Server started on port "+port);
             Socket socket =        /* missing 1 */
             PrintWriter writer =   /* missing 2 */
             writer.println("Hello client");
             writer /* missing 3 */
             serverSocket.close();
             socket.close();
             System.out.println("Server closed");
        }

        catch (IOException e){
             e.printStackTrace();
        }
     }

     public static void main (String[] args){
        new SimplestServer().go();
     }
}
```

Can you give each of the 3 missing code fragments such that the completed *SimplestServer* class will reply to connection requests with the message "Hello client" and then terminate the connection? [7 marks]

END OF PAPER