THIS PAPER IS NOT TO BE REMOVED FROM THE EXAMINATION HALL

UNIVERSITY OF LONDON

CO2220 ZA

BSc EXAMINATION

COMPUTING AND INFORMATION SYSTEMS and CREATIVE COMPUTING

Graphical object-oriented and internet programming in Java

Thursday 2 May 2019: 10.00 – 13.00

Time allowed: 3 hours

DO NOT TURN OVER UNTIL TOLD TO BEGIN

This paper is in two parts: **PART A** and **PART B**. There are a total of **THREE** questions in each part. Candidates should answer **TWO** questions from **PART A** and **TWO** questions from **PART B**.

Full marks will be awarded for complete answers to a total of **FOUR** questions, **TWO** from **PART A** and **TWO** from **PART B**. The marks for each part of a question are indicated at the end of the part in [ ] brackets.

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

There are 100 marks available on this paper.

Calculators are not permitted in this examination.

UL19/0312

Part A

Question 1

(a) Consider the class *PetDog*, below.

```
class PetDog{
     private int age;
     private String name;

     public int getAge(){
        return age;
     }

     public String getName(){
        return name;
     }

     public static void main (String[] args){
        PetDog one = new PetDog();
        System.out.println("Dog name is " + one.getName());
        System.out.println("Dog age is " + one.getAge());
     }
}
```

(i) The *PetDog* class compiles and runs without error. Give the output of the class. [4 marks]

(ii) The *PetDog* class compiles and runs despite not having a constructor. Is this because: [4 marks]

(A) The class does not need a constructor because it has public methods to return the value of the instance variables.

(B) When a class does not have a constructor, the JVM automatically adds a default, no-argument constructor.

(C) The class does not need a constructor to set the value of its instance variables, because instance variables have default values.

(D) None of the above.

(b) Consider the class *Woo*, below.

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

     public Woo(){
          i = 1000;
          s = "The impossibility of now";
     }

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

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

     public String toString(){
          String p = i+" "+s;
          return p;
     }
}
```

The class *SonOfWoo* below is a child class of *Woo*. Give the output of the *SonOfWoo* class. [9 marks]

```
class SonOfWoo extends Woo{

     public SonOfWoo(String s, int i){
          super(i, s);
     }

     public SonOfWoo(int i){
          super(i);
     }

     public SonOfWoo(String s){
          super(42);
     }

     public SonOfWoo(){
     }
     //SonOfWoo continued on next page
```

```
//SonOfWoo continued


            public static void main(String[] args){
               SonOfWoo sow1 = new SonOfWoo("House 6", 636);
               SonOfWoo sow2 = new SonOfWoo(636);
               SonOfWoo sow3 = new SonOfWoo("House 6");
               SonOfWoo sow4 = new SonOfWoo();

               System.out.println(sow1);
               System.out.println(sow2);
               System.out.println(sow3);
               System.out.println(sow4);
            }
}
```

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

```
public class Machine{

     private String name;
     private int pin;

     public Machine (String name, int pin){
          this.name = name;
          this.pin = pin;
     }

     public Machine (String name){
          this.name = name;
          pin = 00000000;
     }

     public void speak(){
          System.out.println("The machines are rising");
     }
}
```

The *Robot* class below is a child class of *Machine*.

```
public class Robot extends Machine{

     public Robot (String name, int pin){
          super(name, pin);
     }

     public void broadcast(String s){
          System.out.println(s);
     }
}
```

(i) In the *Robot* class override the *speak()* method with a method that outputs [4 marks]

```
I am your robot overlord
```

(ii) In the *Robot* class overload the *broadcast()* method, with a method that outputs [4 marks]

```
the robot uprising has begun
```

Note that in your answer you need only write your two new methods and do not need to give the complete *Robot* class.

Question 2

(a)

(i) State whether each of the following is TRUE or FALSE: [4 marks]

(A) An abstract class can be instantiated.

(B) A class can implement any number of interfaces, but can only extend one abstract class.

(C) An abstract class cannot have concrete (*i.e.* non-abstract) methods.

(D) An abstract class must have the abstract key word in its class declaration.

(ii) Consider the abstract class, *Zed*, below:

```
public abstract class Zed{
     abstract boolean lessThanZero(int x, int y);
}
```

Write a concrete class, *ZedStuff,* that extends the *Zed* class and implements the *lessThanZero(int, int)* method. The method returns true if $x + y$ is less than 0, and false otherwise. [4 marks]

(b)

(i) Consider the *XXX* class.

```
import java.util.*;

public class XXX{

  public static void main(String[] args){
    /*1*/ ArrayList<String> b = new ArrayList<String>();
    /*2*/ b.add("hello");
    /*3*/ String s = b.get(0);
    /*4*/ ArrayList  a = new ArrayList();
    /*5*/ a.add("hello");
    /*6*/ s = a.get(0);
  }
}
```

Identify which of the lines numbered 1-6 will give a compilation error. [3 marks]

(ii) Consider the *HB* class below.

```
import java.util.*;

public class HB {

     static ArrayList<String> answers =
       new ArrayList<String> (Arrays.asList("Without a
       doubt", "Yes", "Concentrate and ask again",
       "Don't count on it", "Outlook unclear"));

     public static String magic8Ball() {
          Random r = new Random();
          int index = r.nextInt(/*1*/);
          return /*2*/
     }

     public static void main(String[] args) {
          String m8ball1 = magic8Ball();
          System.out.println(m8ball1);
     }
}
```

The *magic8Ball()* method has parts of two statements missing, denoted by `/*1*/` and `/*2*/`

Complete the *magic8Ball()* method so that when the *HB* class is run, the output will be one of the `Strings` in the *answers* `ArrayList`. [3 marks]

(iii) Say whether each of the following is TRUE or FALSE. [3 marks]

(A) `ArrayLists` can be parameterised to object variables, and also to primitive variables.

(B) `Arrays` are much slower than `ArrayLists` because they are not directly mapped to memory.

(C) The enhanced for loop can be used for iterating through what Java calls collections. It can be used with an `ArrayList` because Java considers the class to be a part of the collections framework.

(c) Consider the abstract class, *AbstractRandomWordGame* below:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintStream;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public abstract class AbstractRandomWordGame {
    protected List<String> words;
    protected Random random;
    protected Scanner input;
    protected PrintStream output;
    protected static final Path defaultWordsFilePath =
     Paths.get("gamedictionary.txt");
    protected static final Charset defaultWordsFileCharset =
     Charset.forName("UTF-8");
    protected Path wordsFilePath;
    protected Charset wordsFileCharset;

    public AbstractRandomWordGame(InputStream input, PrintStream
     output, Path wordsFilePath, Charset wordsFileCharset) {
        //some statements missing here
    }

    private void loadWordsOrGetDefaultWords() {
        if (wordsFilePath == null) wordsFilePath =
           defaultWordsFilePath;
        if (wordsFileCharset == null) wordsFileCharset =
           defaultWordsFileCharset;

        try {
            words = Files.readAllLines(wordsFilePath,
               wordsFileCharset);
        } catch (IOException e) {
            reportWordsLoadingError(e);
            words = getDefaultWordsList();
        }
    }

    private void reportWordsLoadingError(IOException e) {
        System.err.println("Error reading file '"+wordsFilePath+"'");
    }
```

```
    private List<String> getDefaultWordsList() {
        return Arrays.asList("compilation","popstar","symphony");
    }

    protected String getRandomWord() {
        int r = random.nextInt(words.size());
        return words.get(r);
    }

    abstract void play();
}
```

The body of the constructor is empty. Complete the constructor such that when an instance of the object is made, every instance variable will be initialised. This includes the `List` variable *words,* which will either be initialised with words taken from a text file, or, if the file cannot be accessed for some reason, will be initialised to a default state. [8 marks]

Question 3

(a)

(i) Say which of the following are TRUE and which are FALSE: [6 marks]

(A) 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.

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

(C) Event sources (such as a `JButton`) can be registered with multiple event handlers.

(D) If a region is not specified then the *add()* method of `BorderLayout` will add the component to the CENTER region.

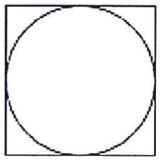
(E) The `BorderLayout` manager has 8 regions.

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

(ii) Consider the following example of a *paintComponent(Graphics)* method:

```
public void paintComponent (Graphics g){
     super.paintComponent(g);
     g.setColor(color);
     g.fillOval(X,Y,diameter,diameter);
}
```

The *paintComponent(Graphics)* method is drawing an oval filled with colour on a `JPanel`. When the oval is drawn on a `JPanel`, it 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 in the *paintComponent(Graphics)* method specify one of the corners of the square that the circle is notionally drawn inside. Which corner? [2marks]



(A) The top right corner.

(B) The top left corner.

(C) The bottom right corner.

(D) The bottom left corner.

(b) Consider the *ClickMeButtonGUI*

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

public class ClickMeButtonGUI{

  JFrame frame;
  JButton button;

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

  public void go(){
/*1*/   frame = new JFrame();
/*2*/   button = new JButton("click me NOW!");
/*3*/   button.addActionListener(new ActionListener(){
/*4*/     public void actionPerformed(ActionEvent e) {
/*5*/       button.setText("YEAH! I'VE BEEN CLICKED!");
/*6*/       frame.repaint();
/*7*/     }
/*8*/   });
/*9*/   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
/*10*/  frame.getContentPane().add(BorderLayout.CENTER,button);
/*11*/  frame.setSize(500,500);
/*12*/  frame.setVisible(true);
  }
}
```

The program uses an anonymous class to implement the `ActionListener` interface. Rewrite the class so that it uses an inner class called *ClickButtonListener* to implement the `ActionListener` interface. The output of the class should remain the same. [9 marks]

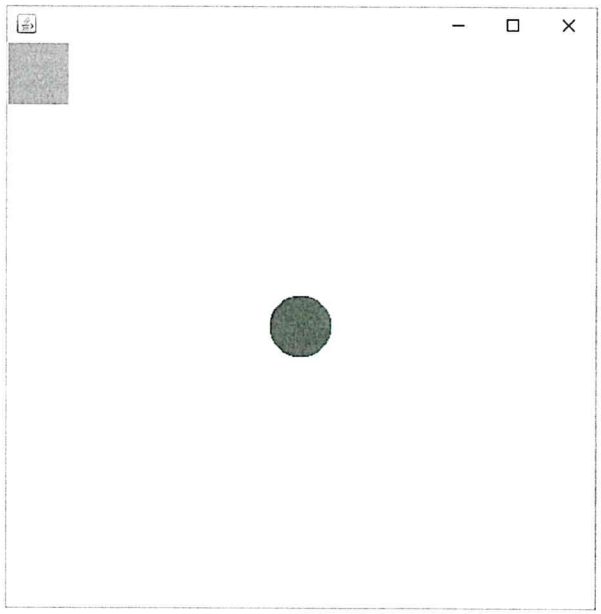
NOTE: The lines comprising the body of the *go()* method have been numbered for ease of reference. This means that in your answer you need only write down (1) your new inner class; (2) the number(s) of the line or lines from the *go()* method that you are deleting; and (3) the new statement or statements that you are adding to the *go()* method.

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

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Graphics.*;

public class AnimatedGraphicsGUI{
     private JFrame frame;
     private int circleX;
     private int squareX;
     private int diameter;
     private boolean moveSquare;
     private boolean growCircle;
     private ShapesDrawPanel drawPanel;


     public AnimatedGraphicsGUI(){
       circleX = 1;
       squareX = 1;
       diameter=50;
       moveSquare = true;
       growCircle = true;
       drawPanel = new ShapesDrawPanel();
     }

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

     public void go(){
       frame = new JFrame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.getContentPane().add(drawPanel,BorderLayout.CENTER);
       frame.setSize(500,500);
       frame.setVisible(true);
     }
```

```
      class ShapesDrawPanel extends JPanel{
        public void paintComponent (Graphics g){

/*1*/     Graphics2D g2=(Graphics2D)g;
/*2*/     g2.setColor(Color.yellow);
/*3*/     g2.fillRect(0,0,this.getWidth(), this.getHeight());
/*4*/     g2.setColor(Color.blue);
/*5*/     g2.fillOval((this.getWidth()-(diameter+circleX))/2,
            (this.getHeight()-(diameter+circleX))/2,
               diameter+circleX, diameter+circleX);
/*6*/     g2.setColor(Color.red);
/*7*/     g2.fillRect(squareX,0,50,50);

        }
      }
}
```

When the *AnimatedGraphicsGUI* class is run the user will see a `JFrame` with a square in the top left corner, and a circle in the middle, as in the image below.



Make changes to the *ShapesDrawPanel* inner class such that when the class is run: [8 marks]

- The square moves across the *drawPanel* until its leading edge hits the other side. It then rebounds and moves back to its starting position. Once it reaches the top left corner again, it starts moving back across the *drawPanel*. This back and forth movement will continue indefinitely.
- The circle grows until part of its border touches an edge of the *drawPanel*. Once the circle has touched the edge of the *drawPanel* it starts to shrink until it has reached its original size and position. Once it has reached its original size, the circle starts to grow again. This growing and shrinking continues indefinitely.
- Make sure to slow the animation so that it is at a sensible speed for the user to watch.

NOTE in your answer book you need only write your amended *ShapesDrawPanel* class. The lines in the body of the *paintComponent(Graphics)* method in the in *ShapesDrawPanel* class have been numbered for your convenience. This means that you may, if you wish, include in your answer only the line numbers of those statements from the method that you are including with no change.

Part B

Question 4

(a)

(i) Consider the following class:

```
public class Counts{
     private int howMany;
     static int index = 0;

     public Counts(int howMany){
          this.howMany = howMany;
     }

     public static void main(String[] args){
          Counts count1 = new Counts(5);
          Counts count2 = new Counts(11);
          count1.index = -19;
          count2.index = 10234;
          System.out.println(count1.index);
          System.out.println(count2.index);
     }
}
```

What will be the output of the class? [3 marks]

(A)
```
10234
-19
```

(B)
```
-19
10234
```

(C)
```
10234
10234
```

(D) None of the above.

(ii) Consider the *Athletes* class, below:

```
public class Athletes{
     private String event;

     public Athletes(String event){
          this.event = event;
     }

     static void equipment(int i){
          if (i == 1) System.out.println("trainers");
          else System.out.println("pole");
     }

     public static void main(String[] args){
          equipment(1);
          Athletes athlete = new Athletes("high jump");
          System.out.println(athlete.event);
     }
}
```

What will be the output of the *Athletes* class? [3 marks]

(A) There will be a compilation error because the *equipment(int)* method is invoked before an instance of the class has been made.

(B) There will be a run-time error because the *equipment(int)* method is invoked before an instance of the class has been made.

(C)
```
trainers
high jump
```

(D) None of the above.

(iii) Is there a way of preventing a variable from being changed (reassigned) after it has been instantiated? [2 marks]

(b)

(i) Give the output of the class *StringThing1* [3 marks]

```
class StringThing1{
  public static void main(String[] args){
    String eg = "15/20=greetings to all";
    String[] result1 = eg.split("=");
    String[] result2 = result1[0].split("/");
    for(String token:result2) System.out.println(token);
    String[] result3 = result1[1].split(" ");
    for(String token:result3) System.out.println(token);
  }
}
```

(ii) Consider the class *SFA*

```
import java.time.LocalDate;

public class SFA{

     public static String formatter(){
          String a = "Today";
          LocalDate date = LocalDate.now();
          //STATEMENT DELETED
          return s;
     }

     public static void main(String[] args){
          System.out.println(formatter());
     }
}
```

Given that the output of the *SFA* class is:

```
2019-05-02          Today
```

which one of the following statements has been deleted from the *formatter()* method of the *SFA* class? [3 marks]

(A) `String s = String.format("%-20s %-20s", a, date);`

(B) `String s = String.format("%-20s %-20s", date, a);`

(C) `String s = String.format("%-7s %-13s", a, date);`

(D) `String s = String.format("%-11s %-7s", date, a);`

(iii) Consider the *StatsA* class. Since it is a utility class with only static methods, it would not be appropriate to make an instance of the class. Write an appropriate constructor for the class that will prevent instantiation. [3 marks]

```
public class StatsA{

  public static double mean (double[] a){
    double sum = 0.0;
    for(double d: a) sum = sum + d;
    return sum / a.length;
  }

  public static double variance(double[] a,double mean){
    double diff = 0;
    double sq = 0;
    double sum = 0;
    double variance = 0;
    for (double i:a){
      diff = i - mean;
      sq = diff * diff;
      sum = sum + sq;
    }
    return variance = sum/a.length;
  }
}
```

(c) Consider the *Person* class below:

```
import java.io.Serializable;

public class Person implements Serializable {

     private String name;
     private String jobTitle;
     private String teamName;
     private int age;

     public Person(String name, String jobTitle,
     String teamName, int age) {
          this.name = name;
          this.jobTitle = jobTitle;
          this.teamName = teamName;
          this.age = age;
     }

     public String toString() {
          return name+", "+jobTitle+", "+teamName+",
     "+age;
     }
}
```

A text file called "people.csv" has three lines of content, as follows:

```
A. Dude, Systems Analyst, Logistics, 20
A.N. Otherdude, Software Developer, Logistics, 25
A Dudess, Editor, Web Team, 27
```

The following class, *PersonFromFile*, provides static utility methods to read *Person* objects from a text file, and store them in an `ArrayList`. The method *parsePerson(String),* called by the *readPeople()* method, has not been written.

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.IOException;
import java.io.FileReader;

public class PersonFromFile{

     private static String Filename = "people.csv";
     private static ArrayList<Person> people;

     private static ArrayList<Person> readPeople() {
          Scanner in = null;
          ArrayList<Person> people = new
     ArrayList<Person>();
          try {
               in = new Scanner(new
     FileReader(Filename));
          }
          catch (IOException e) {
               System.err.println("Error reading from
     file.");
          }

          while(in.hasNextLine()) {
               String line = in.nextLine();
               Person person = parsePerson(line);
               people.add(person);
          }
          in.close();
          return people;
     }
}
```

Write the *parsePerson(String)* method. [8 marks]

Question 5

(a)

(i) What does a `File` object refer to? [2 marks]

(A) The contents of the file.

(B) The name of a file.

(C) The path to a file.

(D) None of the above.

(ii) A connection stream: [2 marks]

(A) Reads and writes bytes.

(B) Cannot be used to read or write binary data as it only reads or writes text.

(C) Cannot directly read input or write output but must be connected to another stream that can.

(D) None of the above.

(iii) Say which of the following are TRUE and which are FALSE: [4 marks]

(A) `BufferedReader` and `BufferedWriter` are connection streams.

(B) `BufferedReader` is efficient as it reads into a buffer and does not read from the file again until all the data in its buffer has been processed.

(C) `BufferedWriter` can be made to write to a file before its buffer is full by using its *flush()* method.

(D) `BufferedReader` can be chained to an `InputStreamReader` in order to read data from any source (*e.g.* the terminal, the internet, *etc*).

(b)

(i) Consider the class *SourceSaver*

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

public class SourceSaver{

  private static void writeToFile(String fileName,
     String text) {

    try {
      FileWriter writer = new FileWriter(fileName);
      writer.write(text);
      writer.close();
    } catch (IOException e) {
      System.out.print(e);
    }
  }

  public static void main(String args[]) {
    String webpage = "www.gold.ac.uk";

    StringBuilder builder = new StringBuilder();
    try {
      URL u = new URL("https://" + webpage);
      Reader r = new BufferedReader(new
          InputStreamReader(/*missing fragment*/));

      int c = 0;
      while ((c = r.read()) != -1) {
        builder.append((char) c);
      }
      String file = webpage + ".html";
      writeToFile(file, builder.toString());
    }

    catch (MalformedURLException e) {
      System.err.println(e);
    } catch (IOException e) {
      System.err.println(e);
    }
  }
}
```

Part of the statement starting `Reader r` is missing. Give the missing fragment. [3 marks]

(ii) Consider the class *MA*, below:

```
import java.net.*;

  public class MA{

    public static void main (String[] args){
      try {
         InetAddress inetAddress=InetAddress.getLocalHost();
         System.out.println(inetAddress.getHostAddress());
      }catch (UnknownHostException e){
         System.out.println(e);
      }
    }
}
```

What will the *MA* class do when run? [3 marks]

(A) It will print to standard output the IP address of the machine it is run on.

(B) If given a host name such as "www.bbc.co.uk" it will print the corresponding IP address to standard output.

(C) It will search for a web page on the machine it is run on, if it finds one it will print its address to standard output.

(D) None of the above.

(iii) If a developer wishes to keep open the option of later changing a class definition (which could cause deserialization issues with earlier versions of the same class), what should the developer include in the class so that the JVM can assess if the class is compatible with the serialized object? [3 marks]

(A) Serial case number.

(B) Serialization ID.

(C) Serial version ID.

(D) None of the above.

(c) Consider the *SerializationUtil* class.

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

public class SerializationUtil {
  public static void serialize(ArrayList<String> results, File
     file) throws FileNotFoundException, IOException {
          //body of method missing
  }

  public static ArrayList<String> deserialize(File file) throws
     Exception {
       ArrayList<String> results = null;
       ObjectInputStream in = new ObjectInputStream(new
          FileInputStream(file));
       results = (ArrayList<String>) in.readObject();
       in.close();
       return results;
  }
}
```

(i) List the subclasses of `Exception` that could be thrown by the *deserialize(File)* method. [4 marks]

(ii) Write the body of the *serialize(ArrayList<String>, File)* method. The method is making the file that will be deserialized by the *deserialize(File)* method. [4 marks]

Question 6

(a)

(i) Answer TRUE or FALSE to each of the following statements: [5 marks]

(A) All exceptions can be caught with a single supertype catch.

(B) Java has checked and unchecked exceptions. Unchecked means that exceptions cannot be handled with try/catch.

(C) Checked exceptions are for run-time errors that are outside of the control of the developer.

(D) All unchecked exceptions subclass `java.lang.RuntimeException`.

(E) `ArrayIndexOutOfBoundsException` is an unchecked exception.

(ii) Consider the *Q6EX* class, below:

```
import java.io.*;
class Q6EX
{
  public static void main(String[] args){
    try{
      FileInputStream fis=new FileInputStream("test.txt");
      InputStreamReader in=new InputStreamReader(fis, "UTF-8");
      int c;
      while ((c=fis.read())!=-1) System.out.print((char)c);
      fis.close();
}

    catch (UnsupportedEncodingException e){
      System.out.println("file encoding not found or not
         supported");
      e.printStackTrace();
    }

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

What would happen if the `catch` block for the `IOException` was placed first, and the catch block for the `UnsupportedEncodingException` was placed second, and the class was re-compiled? [3 marks]

(A) The compiler would report the following error:
```
error: exception
UnsupportedEncodingException has
already been caught
```

(B) The class would compile because the compiler does not check the order of catch blocks.

(C) The class would compile but when run would stop immediately with a run-time error.

(D) None of the above.

(b)

(i) A `Thread` can be made by implementing the `Runnable` interface. State another way of making a `Thread`. [3 marks]

(ii) Why should synchronization be used sparingly? [3 marks]

(A) Because method access is slowed down.

(B) Because concurrency is restricted because other threads are forced to wait in line.

(C) Because of the hazards of potential thread deadlock.

(D) All of the above.

(iii) Give the missing text from the following sentences: [3 marks]

A `Runnable` object must have a _____ method because this is defined by the `Runnable` interface. This method is placed on the bottom of the _____ _____.

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

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

public class ListStringClient {

  private int port = 6006;

  public void go(String host){

    try {
      System.out.println("Contacting "+host+" on port "+port);
      Socket socket = /*1*/
      ObjectInputStream ois = new ObjectInputStream(/*2*/);
      ArrayList<String> results =
           (ArrayList<String>)ois.readObject();
      ois./*3*/
      socket./*4*/
      for (String s : results) System.out.println(s);

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

  public static void main(final String[] args) {
    new ListStringClient().go("results host");
  }
}
```

The *ListStringClient* connects to a server, accepts a serialized `ArrayList` of `Strings`, deserializes the `ArrayList`, and then prints the contents of the `ArrayList` to standard output.

There are some code fragments missing from the *ListStringClient* class, each missing fragment has been numbered. Give the fragment corresponding to each number. [8 marks]

END OF PAPER