SEMESTER ONE EXAMINATIONS SOLUTIONS 2004

MODULE: ***Object-Oriented Programming for Engineers – EE553***

COURSE: ***M.Eng./Grad. Dip./Grad. Cert. in Electronic Systems***
M.Eng./Grad. Dip./Grad. Cert. in Telecommunications Engineering
RAEC – Remote Access to Continuing Eng. Education

Question 1.

(a) Answer the following short questions. Please keep your answers concise.

(i) Why is a *destructor virtual* in C++?

A destructor must be virtual as the destruction of the object must be related to the dynamic type of the object and not the static type. This insures that any dynamic memory or connections open by an object are closed correctly when the object is being destroyed. If the static type destructor was called then the parent destructor would not clean-up the object properly.

(ii) What does the *super* keyword allow in Java?

The super keyword allows the child object to access methods of the parent class. This even includes access to the parent constructor.

(iii) What are *namespaces* in C++?

Namespaces in C++ are regions of scope that are local to a specific section of the application code. They allow the creation of global functions and variables that will not conflict with functions and variables in another namespace.

(iv) When is the *garbage collector* called in Java?

The garbage collector in Java is a daemon thread that counts the references on allocated object, allocating their memory space back to the heap when the object goes out of scope. It is run automatically by the VM when the memory is low, or the CPU is idle. The user can also request that the garbage collector should run.

(v) Compare the *protected* access modifier in C++ to the *protected* access modifier in Java.

The protected keyword in Java and C++ means that the state or method is accessible in the class and any child classes. However, in Java it also means that the state or method is accessible in the same package.

(vi) What does the term *over-riding* mean?

Over-riding means re-writing the same method in a child class, with the exact same set of parameters. Over-riding is a form of polymorphism that allows behaviour to be replaced in a child class.

(vii) What is an *inline method* in C++?

An inline method in C++ is a method that is in effect cut-and-pasted into the code, rather than existing as an indexed block of code. An inline method is efficient as the program counter when executing the code does not need to offset and return, rather the instructions are duplicated. This means that the code is efficient, but also becomes larger.

[2 marks for each point]

(b) What is a *copy constructor*? What operation does the default copy constructor carry out? Using a small source-code example, demonstrate how a user-defined copy constructor can provide useful functionality.

The copy constructor is a special constructor that is automatically a part of every class. It allows a copy to be made of an object. The default copy constructor creates an identical copy of the object, so states have the exact same values. The default copy constructor can be invoked simply by:

```
Account::Account(const Account &sourceAccount):
            accountNumber(sourceAccount.accountNumber + 1),
            balance(0.0f),
            owner (sourceAccount.owner) {}

Account a = Account("Derek Molloy",35.00,34234324);
Account b(a);
```

In this case the copy constructor has been modified to copy the account holder name, but to zero the balance of the new account and to set the new account number to be the last one plus 1 (this is not an ideal implementation, just an example).

[3 marks description – 3 marks source code]

(c) Discuss *exceptions* in Java. Give an example use of exceptions when using arrays in Java. What advantage is there in exception specifications being checked by the Java compiler at compile time?

When an error is detected, an exception is *thrown*. That is, the code that caused the error stops executing immediately, and control is transferred to the *catch clause* for that exception of the first enclosing *try block* that has such a clause. The try block might be in the current method (the one that caused the error), or it might be in some method that called the current method (i.e., if the current method is not prepared to handle the exception, it is "passed up" the call chain). If no currently active method is prepared to catch the exception, an error message is printed and the program stops.

```
try {
   // index the array
} catch (OutOfBoundsException e ) {
   // statements to handle this exception
```

```
}
finally {
   // statements to execute every time this try block executes
}
```

[3 marks for discussion and 2 marks for code outline]

Question 2.

(a) Explain the use of **over-loaded operators** in C++. Why are they a useful feature of the language?

C++ allows the programmer to associate specific functionality of a class with the standard predefined operators, such as '+','-','=','==' etc. One class which demonsrates the use of these over-loaded operators is the String class, which associates the '+' with concatenation, '=' to assignment etc. We can over-load the predefined opertors to suit our own classes

For example, in the Account class we can add a + operator, allowing Account objects to be summed in an appropriate way. For example:

```
class Account{

  protected:

     int accountNumber;
     float balance;
     string owner;

  public:

     Account operator + (Account);
     ...
  };

Account Account::operator + (Account a)
{
  return Account(owner, balance + a.balance, accountNumber);
}
```

[5 marks for description 3 marks for code sample]

(b) Write a **Colour class** in C++ that has the following functionality:

- The class should contain three states for red, green and blue, each with a range of 0-255.
- Add appropriate constructors.
- Is the default copy constructor appropriate?
- Write overloaded operators for +, -, =, ==, < and >.
- Write code that checks the bounds on all assignments, i.e. values cannot be <0 or >255.
- Write a sort segment of code to demonstrate your Colour class in action.

```
// Colour Example

#include<iostream.h>
#include<string.h>

using namespace std;

class Colour{

  private:

    int red;
    int green;
    int blue;

    void checkBounds();

  public:

    Colour();
    Colour(int, int, int);

    virtual int toGrayScale();

    virtual int getRed()          { return red; }
    virtual int getGreen()        { return green; }
    virtual int getBlue()         { return blue; }

    virtual void operator + (Colour);
    virtual void operator - (Colour);
    virtual void operator = (Colour);
    virtual bool operator == (Colour);
    virtual bool operator < (Colour);
    virtual bool operator > (Colour);

    virtual void display();
};

Colour::Colour():
            red(0), green(0), blue(0) {}

Colour::Colour(int r, int g, int b):
            red(r), green(g), blue(b)
{
    checkBounds();
}

void Colour::checkBounds()
{
    if (red<0) red=0;
    if (green<0) green=0;
    if (blue<0) blue=0;

    if (red>255) red=255;
    if (blue>255) blue=255;
```

```
    if (green>255) green=255;
}

int Colour::toGrayScale()
{
    return (int) ( red + green + blue / 3 );
}

void Colour::operator + (Colour a)
{
    red+=a.red;
    green+=a.green;
    blue+=a.blue;
    checkBounds();
}

void Colour::operator - (Colour a)
{
    red-=a.red;
    green-=a.green;
    blue-=a.blue;
    checkBounds();
}

void Colour::operator = (Colour a)
{
    red   = a.red;
    green = a.green;
    blue  = a.blue;
    checkBounds();
}

bool Colour::operator == (Colour a)
{
    if ((a.red == red) && (a.green == green) && (a.blue == blue)) return true;
    else return false;
}

bool Colour::operator < (Colour a)
{
    if (toGrayScale() < a.toGrayScale()) return true;
    else return false;
}

bool Colour::operator > (Colour a)
{
    if (toGrayScale() > a.toGrayScale()) return true;
    else return false;
}

void Colour::display()
{
    cout << "The colour is (r,g,b) = (" << red << "," << green << "," << blue << ")" << endl;
}

void main()
```

```
{
  Colour c(220,200,204);
  c.display();

  Colour d(290,100,100);
  d.display();

  if ( c > d ) cout << "C is greater than D!\n";

  c.display();
}
```

The copy constructor is appropriate.

[12 marks]
[6 marks for operators, 1 mark per operator]
[2 marks for example use]
[1 mark for no copy constructor required]
[3 marks for constructors and related code]

(c) In C++ one class can be **a friend** of another class, why? Show the coding syntax used to create this relationship, and discuss the advantages and disadvantages of this approach.

A class can declare a function or class to be a friend, allowing that function to access private and protected members (friendship is granted! Not acquired.).

```
class A
  {
    int x;
      friend class B;
    public:
      //methods here
  };
```

Friend functions are useful as:

- Friend functions avoid adding unnecessary public methods to the interface.
- Prevent states from having to be made public.

However, the overuse of friend functions can make the code very complex and hard to follow.

[1 marks explain]
[2 marks code]
[2 mark for adv/disadv]

Question 3.

(a) Explain using an example why you would need to synchronize a segment of code when using Java threads? (Your answer should show a line-by-line step through of a segment of code, explaining why it would not work correctly if the segment of code was not synchronized). If synchronization is a solution to making an application thread safe, then why should we not just synchronize all our code?

It can be difficult to control threads when they need to share data. It is difficult to predict when data will be passed, often being different each time the application is run. We add synchronization to the methods that we use to share this data like:
public synchronized void theSynchronizedMethod()
or we can select a block of code to synchronize:

```
synchronized(anObject)
{
}
```

This works like a lock on objects. When two threads execute code on the same object, only one of them acquires the lock and proceeds. The second thread waits until the lock is released on the object. This allows the first thread to operate on the object, without any interruption by the second thread.

First Thread	Second Thread
call theSynchronizedMethod()	
acquires the lock on the theObject	
executes theSynchronizedMethod on theObject	
	calls theSynchronizedMethod on theObject
	some other thread has a lock on theObject
returns from theSynchronizedMethod()	
	acquires the lock on the theObject
	executes theSynchronizedMethod on theObject

Again, synchronization is based on objects:

- two threads call synchronized methods on different objects, they proceed concurrently.
- two threads call different synchronized methods on the same object, they are synchronized.
- two threads call synchronized and non-synchronized methods on the same object, they proceed concurrently.

Static methods are synchronized per class. The standard classes are multithread safe.

It may seem that an obvious solution would be to synchronize everything!! However this is not that good an idea as when we write an application, we wish to make it:

- Safe - we get the correct results
- Lively - It performs efficiently, using threads to achieve this liveliness

These are conflicting goals, as too much synchronization causes the program to execute sequentially, but synchronization is required for safety when sharing objects. Always remove synchronization if you know it is safe, but if you're not sure then synchronize.

[9 marks overall]
[4 marks for correct example]
[3 marks for intro explain]
[2 marks for safety vs. lively]

(b) Write a Java egg timer that looks like the application below.

- Once the start button is pressed the timer will count to two minutes.
- The timer can be stopped and reset at any time, by pressing stop.
- When the timer is finished, it should pop-up an appropriate dialog box that states the time is up.


```
// Egg Timer Solution  - Derek Molloy

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;
import java.applet.*;

public class EggTimerApplication extends JFrame implements ActionListener
{
        private JProgressBar progressBar;
        private JButton startButton, stopButton;
        private JLabel textLabel;
        private EggTimer eggTimer;

        public EggTimerApplication()
        {
                super("Egg Timer");

                JPanel p = new JPanel();
                p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
                p.setBorder(new TitledBorder("Egg Timer:"));

                progressBar = new JProgressBar(JProgressBar.HORIZONTAL,0,120);

                JPanel controls = new JPanel();
                textLabel = new JLabel("  0 Seconds ");
                startButton = new JButton("Start");
                startButton.addActionListener(this);
                stopButton = new JButton("Stop");
                stopButton.addActionListener(this);
```

```
                p.add(progressBar);
                p.add(textLabel);
                controls.add(startButton);
                controls.add(stopButton);

                this.getContentPane().add("Center",p);
                this.getContentPane().add("South",controls);

                this.setSize(300,130);
                this.show();

                this.eggTimer = new EggTimer(this.progressBar, this.textLabel);
        }

        public void actionPerformed(ActionEvent e)
        {
          if (e.getSource().equals(startButton))
          {
                this.eggTimer.start();
          }
          else if (e.getSource().equals(stopButton))
          {
               this.eggTimer.stopTimer();
          }
        }

        public static void main(String[] args)
        {
           new EggTimerApplication();
        }
}


class EggTimer extends Thread
{
        private int numberSeconds = 0;
        private boolean running = true;
        private JProgressBar progressReference;
        private JLabel textLabel;


        public EggTimer(JProgressBar progressReference, JLabel textLabel)
        {
          this.progressReference = progressReference;
          this.textLabel = textLabel;
        }

        public void run()
        {
                while(running)
                {
                  if (this.updateProgress()==false) running = false;

                  try{
                        Thread.currentThread().sleep(100);
                  }
```

```
                    catch (InterruptedException e)
                    {
                      System.out.println(e.toString());
                    }
                  }
        }

        private boolean updateProgress()
        {
                  int current = progressReference.getValue();
                  int updateTo = ++current;
                  if (updateTo <= 120)
                  {
                    this.progressReference.setValue(updateTo);
                    this.textLabel.setText(" " + updateTo + " Seconds ");
                    return true;
                  }
                  else
                  {
                    this.displayDialog();
                    return false;
                  }
        }

        public void stopTimer()
        {
          this.running = false;
        }

        private void displayDialog()
        {
          JOptionPane.showMessageDialog(null, "Your Egg is Ready!",
                  "Egg Timer", JOptionPane.INFORMATION_MESSAGE);

        }
}
```

[16 marks Total – Approximate breakdown]
[6 for working thread]
[4 for user interface]
[3 for dialog box]
[3 for event structure]

Question 4.

(a) Compare the C++ language to the Java language under the following headings only:

- Virtual and non-virtual methods.
- Access specifiers.
- Multiple inheritance.

(1) In C++ methods are non-virtual by default, so to replace the behaviour (allow over-riding) of a method in the derived class you have to explicitly use the `virtual` keyword in the parent class. You are able to replace the behaviour of a non-virtual method but this causes serious difficulties when the behaviour of the object depends on

the static rather than the dynamic type! In Java, methods are virtual by default and we always operate on the dynamic type of the object. You cannot specify a method as non-virtual, but there is a `final` keyword. Once you use the `final` keyword on a method you cannot replace it - so there are no issues with static and dynamic types.

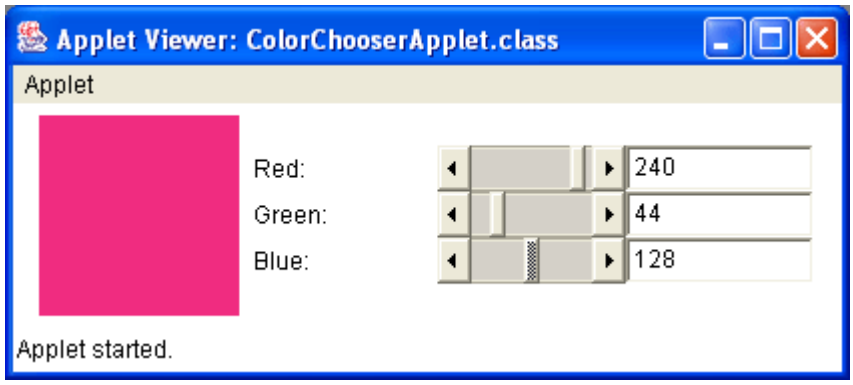
(2) Like C++, in Java we have public, private and protected access specifiers, but we also have another access specifier "package". This is the default access specifier and means that all states and methods are accessible to all classes within the same package. There is no package access specifier keyword, it is simply the default if public, private or protected are not used. Access specifiers in Java are:

- public - accessible everywhere, an interface method (same as C++)
- private - accessible only in the class where it was defined (same as C++)
- protected - accessible in the class where it is defined, the derived classes and in the same package (almost the same as C++ with the exception of the package)
- "package" - default (there is no package specifier keyword) and means that it is accessible by any class in the same package. You can equate this to the C++ friendly condition by saying that all of the classes in the same package (directory) are friendly.

(3) C++ allows multiple inheritance - Java does not! As discussed previously in the C++ section of the notes, multiple inheritance is complex for the programmer, in that they must make complex decisions about the way that the data of the base class is to be inherited. In Java, we do not have multiple inheritance but we do have the use of Interfaces. Interfaces are a special kind of abstract class that have no data or implemented code. Interfaces might not sound too useful, but they allow a form of multiple inheritance, without having the associated difficulties of dealing with data. This replacement behaviour works very well and does not impact on development, once you get used to the change.

[9 marks]
[3 marks for each point]

(b) Use the Java AWT to build the Color Chooser applet below. The applet should allow the user to choose a colour in the range (red,green,blue) of (0-255, 0-255, 0-255) using either the scrollbars or the textfields.




```
// The Colour Chooser Applet Exercise

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class ColorChooserApplet extends Applet implements ActionListener, AdjustmentListener
{

        private Scrollbar rBar, gBar, bBar;
        private TextField rField, gField, bField;
        private ColorCanvas cs;

        public void init()
        {
          //For the layout I used a 3x3 grid with the canvas and grid added to FlowLayout
          Panel rightPanel = new Panel(new GridLayout(3,3));

          rightPanel.add(new Label("Red:"));
          rBar = new Scrollbar(Scrollbar.HORIZONTAL, 128, 10, 0, 265);
          rBar.addAdjustmentListener(this);
          rightPanel.add(rBar);
          rField = new TextField("128",10);
          rField.addActionListener(this);
          rightPanel.add(rField);

          rightPanel.add(new Label("Green:"));
          gBar = new Scrollbar(Scrollbar.HORIZONTAL, 128, 10, 0, 265);
          gBar.addAdjustmentListener(this);
          rightPanel.add(gBar);
          gField = new TextField("128");
          gField.addActionListener(this);
          rightPanel.add(gField);

          rightPanel.add(new Label("Blue:"));
          bBar = new Scrollbar(Scrollbar.HORIZONTAL, 128, 10, 0, 265);
          bBar.addAdjustmentListener(this);
          rightPanel.add(bBar);
          bField = new TextField("128");
          bField.addActionListener(this);
          rightPanel.add(bField);

          cs = new ColorCanvas();
          cs.setSize(100,100);
          this.add(cs);
          this.add(rightPanel);
          this.updateColor();
        }

        // Since no invalid value is possible with the scrollbar error checking is not required.
        public void adjustmentValueChanged(AdjustmentEvent e)
        {
          rField.setText(new Integer(rBar.getValue()).toString());
          gField.setText(new Integer(gBar.getValue()).toString());
          bField.setText(new Integer(bBar.getValue()).toString());
          this.updateColor();
        }

        //actionPerformed is a little complex as the user could type in values
        //>255 <0 or even enter an invalid string
        public void actionPerformed(ActionEvent e)
        {
```

```
            try
            {
              Integer rVal = new Integer(rField.getText());
              Integer gVal = new Integer(gField.getText());
              Integer bVal = new Integer(bField.getText());
              int rValInt = rVal.intValue();
              int gValInt = gVal.intValue();
              int bValInt = bVal.intValue();

              if(rValInt>=0 && rValInt<=255) rBar.setValue(rValInt);
                      else throw(new NumberFormatException());
              if(gValInt>=0 && gValInt<=255) gBar.setValue(gValInt);
                      else throw(new NumberFormatException());
              if(bValInt>=0 && bValInt<=255) bBar.setValue(bValInt);
                      else throw(new NumberFormatException());
            }
            catch (NumberFormatException nfe)
            {
              rField.setText(new Integer(rBar.getValue()).toString());
              gField.setText(new Integer(gBar.getValue()).toString());
              bField.setText(new Integer(bBar.getValue()).toString());
            }
            updateColor();
          }

          // I use the scrollbars as storage rather than states
          private void updateColor()
          {
             Color c = new Color(rBar.getValue(), gBar.getValue(), bBar.getValue());
             cs.updateColor(c);
          }
}


class ColorCanvas extends Canvas
{
          private Color col;

          public void updateColor(Color c)
          {
            this.col = c;
            repaint();
          }

          public void paint(Graphics g)
          {
            g.setColor(col);
            g.fillRect(0,0,100,100);
          }
}
```

[16 marks in Total – approximate breakdown]
[5 marks for the user interface]
[3 marks for the canvas class]
[6 marks for the event structure]

[2 marks for data/range checking]

Question 5.

(a) Write a Java client/server banking application pair, where the client passes a Transaction object to the server, the transaction operates on an Account object on the server side, and then returns an appropriate response to the client. The basic transactions to be handled are lodgment, withdrawal, balance enquiry and close account.

```
// The Transaction class - by Derek Molloy   [13 marks]

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

public class Transaction implements Serializable
{
    public static final int LODGEMENT = 1;
    public static final int WITHDRAWAL = 2;
    public static final int BALANCEENQ = 3;
    public static final int CLOSEACT = 4;

    private int accountNumber;
    private int transactionType;
    private float amount;
    private String returnMessage = "None";
    private boolean successfulTransaction = false;

    public void display()
    {
            System.out.println("A transaction object");
            System.out.println("With Transaction Type: " + transactionType);
            System.out.println("With account number: " + accountNumber);
            System.out.println("With amount: " + amount);
    }

    public Transaction(int type, int actNum, float amount)
    {
            this.transactionType = type;
            this.accountNumber = actNum;
            this.amount = amount;
    }
    public Transaction(int type, int actNum)
    {
            this(type, actNum, 0.0f);
    }

    public void setReturnMessage(String s) { returnMessage = s; }
    public String getReturnMessage() {return returnMessage; }
    public void setReturnSuccess(boolean b) { successfulTransaction = b; }
    public boolean getReturnSuccess() { return successfulTransaction;}

    public boolean isForAccount(Account a)
```

```
    {
        if ( a.getAccountNumber() == accountNumber) return true;
        else return false;
    }

    public boolean applyTo(Account a)
    {
        if (this.transactionType == Transaction.LODGEMENT)
            a.makeLodgement(amount);
        else if (this.transactionType == Transaction.WITHDRAWAL)
            return a.makeWithdrawal(amount);
        else if (this.transactionType == Transaction.BALANCEENQ)
            a.display();
        else if (this.transactionType == Transaction.CLOSEACT)
            a.close();
        else return false;
        return true;
    }
}
```

Modifications to Server.java are: [2 marks]

```
    break;
        } // create a new thread for the client
        ConnectionHandler con = new ConnectionHandler(clientSocket, theAccounts);
```

Modifications to HandleConection.java are: [5 marks]

```
  private Account[] theAccounts;

            // The constructor for the connecton handler
    public ConnectionHandler(Socket clientSocket, Account[] theAccounts)
    {
        this.clientSocket = clientSocket;
        this.theAccounts = theAccounts;
    }


    /** Receive and process incoming command from client socket */
    private boolean readCommand()
    {
        Transaction t = null;

        try
        {
            t = (Transaction)is.readObject();
        }
        catch (Exception e)
        {
            t = null;
        }
        if (t == null)
        {
            this.closeSocket();
```

```
        return false;
    }


        for (int i=0; i<=2; i++)
        {
            if (t.isForAccount(theAccounts[i]))
            {
                    System.out.println("Applying a transaction");
                    t.applyTo(theAccounts[i]);
            }
        }

    return true;
}
```

Modifications to Client.java are: [5 marks]

```
private void doTransactions()
{
        Transaction[] trans = { new Transaction(Transaction.LODGEMENT, 100001, 1000f),
                                new Transaction(Transaction.BALANCEENQ, 100001),
                                new Transaction(Transaction.WITHDRAWAL, 100001, 50f),
                                new Transaction(Transaction.BALANCEENQ, 100001),
                                new Transaction(Transaction.CLOSEACT, 100001),
                                new Transaction(Transaction.BALANCEENQ, 100001) };

        for (int i=0; i<=5; i++)
                this.send(trans[i]);
}
```

[25 marks total]