Question 1.

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

(i) Explain the difference between the terms *declaration* and *definition*?
(ii) Why are *destructors* always virtual in C++?
(iii) Explain how Java avoids the difficulties associated with *multiple inheritance*.
(iv) Explain the use of the *Object class* in Java.
(v) In C++ what is a *static state* and why would it be used?
(vi) What does the term *overloading* mean?
(vii) In C++ the "->" operator allows you to call a method on a pointer to an object. If this operator was not available what would you write to do the same operation?

[14 marks]
[2 marks each part]

1(a)(i)
Definition means "make this variable or method" here, allocating storage for the name. Declaration is the introduction of a name (identifier) to the compiler. It notifies the compiler that a method or variable of that name has a certain form, and exists somewhere.

1(a)(ii)
Destructors are used to tidy up objects on their destruction. For example in the case of a Bank Account object the destructor could print a paper copy of the account object before it is destroyed. Typically destructors are used in C++ to deallocate memory that was allocated by an object. Importantly destructors must be virtual as when the object is dynamically bound to a pointer the destructor most closely associated with the dynamic type must be call otherwise there will be incorrect deallocation when bound to a pointer.

1(a)(iii)
Java uses Interfaces to avoid the use of multiple inheritance. A Java Interface is a way of grouping methods that describe a particular behaviour. They allow developers to reuse design and they capture similarity, independent of the class hierarchy. We can share both methods and constants – no states.

- Methods in an interface are public and abstract.
- Instance variables in an interface are public, static and final.

1(a)(iv)
The Object class in Java is actually the parent class of all classes. It defines methods that are common to all classes. It allows very complex operations to be performed in the Java language such as transporting objects across a network, storage on a common stack or file storage using standard methods. It is referenced as java.lang.Object.

1(a)(v)
Each instance of a class has its own states. However, we also have static states allowing us to share a variable between every instance of a class. You can think of a static state as being a part of the class rather than the objects.

1(a)(vi)
Overloading – Re-using the same method name with different arguments and possibly a different return type is known as overloading. In unrelated class there is no issues when re-using names but in the same class, we can overload methods, provided that those methods have different argument types. A method may not be distinguished by return type alone.

1(a)(vii) (*prtToObject).someMethod();

(b) Discuss the difference between arrays of objects in C++ compared to arrays of objects in Java? In particular, compare
SomeObject[] a = new SomeObject[5]; to:
SomeObject a[5] ;

[5 marks]

In Java, when you declare an array of a class, you could say:

```
        SomeObject[] a = new SomeObject[5];
```

But this creates only an array of references, you must then instantiate each reference, using:

```
        a[1] = new SomeObject ();
```

Even for the default constructor (as shown). In C++, when you declare:

```
        SomeObject  a[5];
```

C++ would call the default constructor for each instance of the array, and there would be no need to explicitly call 'new' again to instantiate each reference of the array.

This form of initialisation only allows you to call the default constructor (or constructor with no parameters). In C++, you can create an array of objects, with each object sitting right next to each other in memory, which lets you move from one object to another, simply by knowing the address of the first object. This is impossible in Java.

In C++ you can also create arrays, of pointer or references to objects. This is more closely related to how Java arrays work. Java arrays are always an array of references to objects (except if you create arrays of the basic data types, int, float, double etc.). If you were to use an array of pointers or references, you would have an array of null references (just like Java). The syntax would be like: SomeObject **f = new SomeObject *[10];

(c) Discuss **constructors** in C++. Can they be overloaded? Why can they not be virtual? In what order are they called when inheritance takes place? What is the copy constructor and how can it provide specific functionality?

[6 marks]

A Constructor can be used for the task of initialising data within a class:
•It is a member function that has the same name as the class name. C++ calls the constructor of the base class and eventually calls the constructor of the derived class.
•A constructor must not have a declared return value (not even void!)
•A constructor <u>cannot be virtual</u>. The constructor is specific to the class that it is declared in. A new constructor must be declared for children classes as it must be specific to that class.

An example constructor code for the class Account:

```
// A constructor code example
class account{
                int myAccountNumber;
                float myBalance;
        public:
                // the constructor definition
                account(float aBalance, int anAccNumber);
};
// the constructor code implementation
account::account(int anAccNumber, float aBalance)
{
        myAccountNumber = anAccNumber;
        myBalance = aBalance;
}
// now call the constructor!
void main()
{
        account anAccount = account(35.00,34234324); //OK
        account testAccount(0.0, 34234325); //OK
        account myAccount;  //Wrong!
}
```

•If a class has a constructor then the arguments must be supplied. The constructor can be overloaded allowing other constructors to be added to the same class.
•in main() above the first two object creation calls (for anAccount and testAccount) are correct and it is up to the users style of programming. The third object construction (for myAccount) is incorrect, as the parameters are not provided to match the declared constructor!

The copy constructor is a default constructor associated with all Classes. It can be used by simply creating a new object of a class, passing the object that you wish to copy.

e.g.
Account a(1234, 10);
Account b(a);

Creates a b object of the Account class using a for that states. We can override the behaviour of this copy constructor to provide specific behaviour. For example it does not make sense to copy account numbers to new objects. Simply write a method of the form Account::Account(Account &a) and add specific behaviour.

Question 2.

(a) Discuss STL Containers, Iterators and Algorithms; in particular discuss how they work together and also list different types of containers, iterators and algorithms.

[12 marks]

STL can be described and categorised as follows:

- Containers: data structures with memory managment capability.
- Iterators: logic that binds containers and algorithms together.
- Algorithms: provide a mechanism for manipulating data through the iterator interface.

Containers are data structures that contain memory managment capabilities. There are two categories of containers, sequential containers and associative containers. Some examples of sequential containers are: vector: a flexible array of elements. deque: a dynamic array that can grow at both ends. list: a doubly linked list. Vectors are easy to declare as an array and are automatically dynamically resized during program execution.

Associative containers associate a value with a name and some examples are:
- map: a collection of name-value pairs, sorted by the keys value.
- set: a collection of elements sorted by their own value.
- multimap: same as a map, only duplicates are allowed.
- multiset: same as a set, only duplicates are allowed.

STL Iterators
The container "contains" the elements together, and the iterator provides a mechanism for traversing the different types of those containers. Even though each container may have a completely different data structure, the iterator can provide the level of abstraction necessary to build a generic interface to traverse any type of container.
- Input iterators
- Output iterators
- Forward iterators
- Bidirectional iterators
- Random iterators

Iterators are abstract, requiring a class that implements the methods for use as the iterator. Input and Ouput iterators are the simplest form of iterators (being simple iterator interfaces to the

iostream classes, with the random access iterator being the most complex, and by definition, it has implemented the other iterator categories.
STL Algorithms. Algorithms provide a means of accessing and manipulating the data in containers using the iterator interface. The algorithms even implement a for_each loop.

- The algorithms can be classified as:
- Nonmodifying algorithms: e.g. for_each, count, search ...
- Modifying algorithms: for_each, copy, transform ...
- Removing algorithms: remove, remove_if, ...
- Mutating algorithms: reverse, rotate, ...
- Sorting algorithms: sort, partial_sort, ...
- Sorted range algorithms: binary_search, merge ...
- Numeric algorithms: accumulate, partial_sum ...

(b) The for_each loop condenses the for loop into a single line. This single line iterates through the vector and executes a function at every element within the vector. The function outputFunction receives the element type the vector was specialized with at compile time. The function can do anything, but in this case, it simply prints to the standard output.

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// declare a simple function that outputs an int
void outputFunction(int x)
{
cout << x << endl;
}

void main(void)
{
    int x;
    vector<int> vect;  // declare a vector container of ints

    cout << "Please enter a list of numbers:" << endl;
    while(cin >> x)
    {
    vect.push_back(x);
    }

    sort(vect.begin(), vect.end());   // sort the array

    cout << endl << "Sorted output of numbers:" << endl;

    // loop through vector with for_each loop and execute
    // outputFunction() for each element

    for_each(vect.begin(), vect.end(), outputFunction);
}
```

[8 marks]

(c) What are namespaces in C++? Explain the difference between the lines:

```
#include<iostream.h>   and
#include<iostream>
```

The namespace concept was introduced during the standardisation of C++, to allow the programmer to define a namespace that is limited in scope. When writing C++ applications we can make it explicit that we are using the standard namespace by a "using directive":

```
using namespace std;  // i.e. I want to use the declarations and
// definitions in this namespace
```

Writing:

```
#include<iostream.h>
```

Is the same as writing:

```
#include<iostream>
using namespace std;
```

Question 3

(a) 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.

(b)
```
// The Colour Chooser Applet Exercise

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


  public class ColorChooserApplication extends JFrame implements ActionListener,
AdjustmentListener
  {

      private JScrollBar rBar, gBar, bBar;
      private JTextField rField, gField, bField;
      private ColorCanvas cs;

      public ColorChooserApplication()
      {
        super("Colour Chooser Application");
        JPanel rightPanel = new JPanel(new GridLayout(3,3));

        rightPanel.add(new JLabel("Red:"));
        rBar = new JScrollBar(JScrollBar.HORIZONTAL, 128, 10, 0, 265);
        rBar.addAdjustmentListener(this);
        rightPanel.add(rBar);
        rField = new JTextField("128",10);
        rField.addActionListener(this);
```

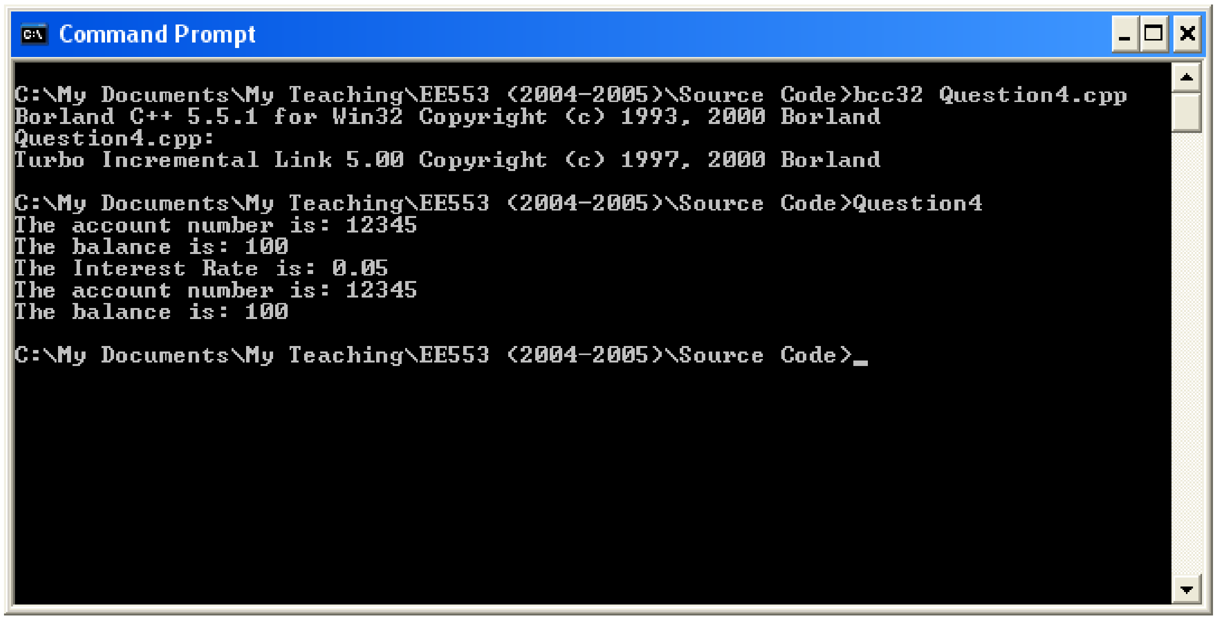
```
          rightPanel.add(rField);

          rightPanel.add(new JLabel("Green:"));
          gBar = new JScrollBar(JScrollBar.HORIZONTAL, 128, 10, 0, 265);
          gBar.addAdjustmentListener(this);
          rightPanel.add(gBar);
          gField = new JTextField("128");
          gField.addActionListener(this);
          rightPanel.add(gField);

          rightPanel.add(new JLabel("Blue:"));
          bBar = new JScrollBar(JScrollBar.HORIZONTAL, 128, 10, 0, 265);
          bBar.addAdjustmentListener(this);
          rightPanel.add(bBar);
          bField = new JTextField("128");
          bField.addActionListener(this);
          rightPanel.add(bField);

          cs = new ColorCanvas();
          cs.setSize(100,100);
          this.getContentPane().setLayout(new FlowLayout());
          this.getContentPane().add(cs);
          this.getContentPane().add(rightPanel);
          this.updateColor();
          this.setSize(500,150);
          this.setVisible(true);
        }

        // 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);
        }

        public static void main(String args[])
        {
              System.out.println("Application Started.");
              new ColorChooserApplication();
        }
    }


    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);
        }
    }
```

[20 marks total]
[2 marks for using Swing]
[5 marks for correct Canvas (or JLabel) use]
[5 marks for correct events]
[5 marks for functionality]
[3 marks for JFrame and application aspects]
INDICATIVE - SUBJECT TO SOLUTION PRESENTED.

Question 4.

(a)



Command Prompt

```
C:\My Documents\My Teaching\EE553 (2004-2005)\Source Code>bcc32 Question4.cpp
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
Question4.cpp:
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:\My Documents\My Teaching\EE553 (2004-2005)\Source Code>Question4
The account number is: 12345
The balance is: 100
The Interest Rate is: 0.05
The account number is: 12345
The balance is: 100

C:\My Documents\My Teaching\EE553 (2004-2005)\Source Code>_
```

```
#include<string>
#include<iostream>

using namespace std;

class Account
{
	int actNum;
	float balance;
public:
	Account(int, float);
	virtual void display();
	void nvDisplay();
};

class Deposit: public Account
{
	float iRate;
public:
	Deposit(int, float, float);
	virtual void display();
	void nvDisplay();
};

Account::Account(int num, float theBalance):
	actNum(num), balance(theBalance) {}

void Account::display()
{
	cout << "The account number is: " << actNum << endl;
	cout << "The balance is: " << balance << endl;
}

void Account::nvDisplay()
{
	cout << "The account number is: " << actNum << endl;
	cout << "The balance is: " << balance << endl;
}

Deposit::Deposit(int num, float theBalance, float rate):
        Account(num, theBalance), iRate(rate) {}

void Deposit::display()
{
        Account::display();
        cout << "The Interest Rate is: " << iRate << endl;
}

void Deposit::nvDisplay()
{
        Account::nvDisplay();
        cout << "The Interest Rate is: " << iRate << endl;
}

int main(void)
{
        Account *ptr = new Deposit(12345, 100.00f, 0.05f);
        ptr->display();
        ptr->nvDisplay();
}
```

[10 marks total]
[5 marks for virtual example]
[5 marks for non-virtual example]
INDICATIVE - SUBJECT TO SOLUTION PRESENTED.

(b) Counting Application

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

public class CounterApplication extends Frame implements ActionListener
{
        private Button start, stop, pause;
        private TextField countField1;
        private Counter theCounter1;

        public CounterApplication()
        {
          super("Counter Application");
          this.setLayout(new FlowLayout());
          countField1 = new TextField(10);
          this.add(countField1);

          start = new Button("Start");
          start.addActionListener(this);
          this.add(start);

          stop = new Button("Stop");
          stop.addActionListener(this);
          this.add(stop);

          pause = new Button("Pause");
          pause.addActionListener(this);
          this.add(pause);

          this.theCounter1 = new Counter(countField1, 100, 0);

          this.setSize(300,100);
          this.setVisible(true);
        }

        public void actionPerformed(ActionEvent e)
        {
          if (e.getSource().equals(start))
          {
            this.theCounter1.start();
          }
          else if (e.getSource().equals(stop))
          {
            this.theCounter1.stopCounter();
          }
          else if (e.getSource().equals(pause))
          {
            this.theCounter1.toggleCounter();
          }
        }

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

  class Counter extends Thread
  {
        private int count, delay;
        private boolean running = true, paused = false;
        private TextField theField;


        public Counter(TextField t, int delay, int startValue)
        {
          this.theField = t;
          this.count = startValue;
          this.delay = delay;
        }

        public void run()
        {
          while (running)
          {
            theField.setText("Count: "+this.count++);
            try
            {
              Thread.currentThread().sleep(this.delay);

              synchronized(this) {
                      while (paused)
                          wait();
                }
            }
            catch (InterruptedException e)
            {
              theField.setText("Interrupted!");
              this.stopCounter();
            }
          }
        }

        public int getCount() { return count; }

        public void stopCounter() { this.running = false; }
        public synchronized void toggleCounter()
        {
          this.paused = !this.paused;
          if (!this.paused) notify();
        }
}
```

15 marks total]
[2 marks for knowing to use threads]
[5 marks for writing code correctly]
[5 marks for events and program structure]
[3 marks for Frame and application aspects]
INDICATIVE - SUBJECT TO SOLUTION PRESENTED.

Question 5.

(a) Explain using an example when 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 as programmers, why should we not just synchronize all code?
[7 marks]

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:

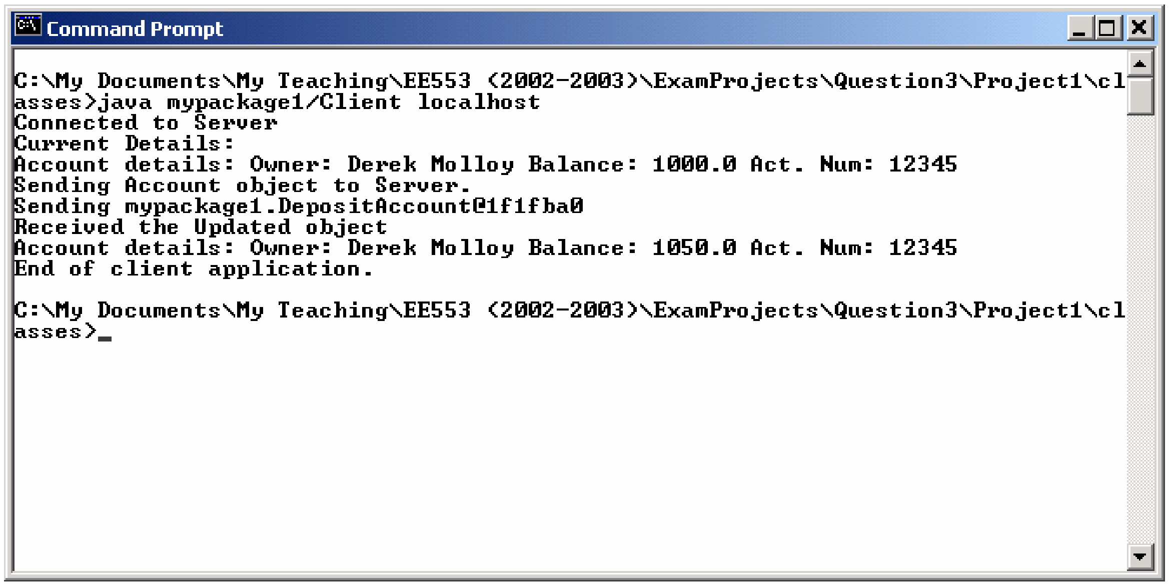
- Safe - we get the correct results
- Lively - It performs efficiently, using threads to achieve this liveliness

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.

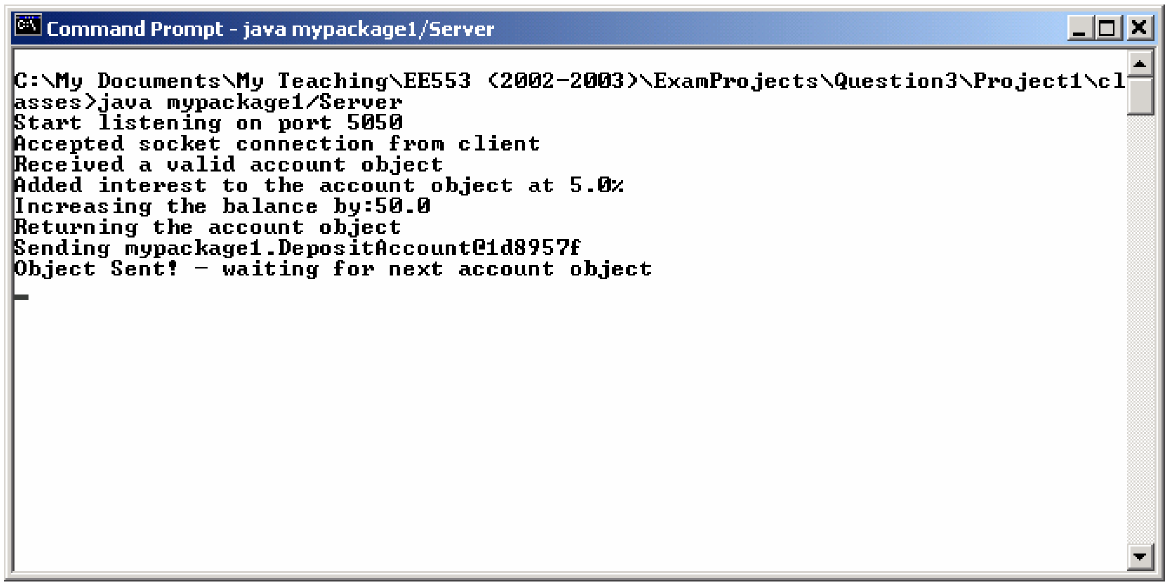
(b) Write a Java client/server application pair, where the client passes a DepositAccount object to the server and the server calculates the interest on the account using the current interest rate available on the server. The account object is then passed back to the client, where the client displays the details including the updated balance.
You have been supplied with three sets of code to handle the basic aspects of this application. These are called:

- **Client.java**,
- **Server.java** and
- **ConnectionHandler.java**

These files are in the directory **question5**.



```
C:\My Documents\My Teaching\EE553 (2002-2003)\ExamProjects\Question3\Project1\cl
asses>java mypackage1/Client localhost
Connected to Server
Current Details:
Account details: Owner: Derek Molloy Balance: 1000.0 Act. Num: 12345
Sending Account object to Server.
Sending mypackage1.DepositAccount@1f1fba0
Received the Updated object
Account details: Owner: Derek Molloy Balance: 1050.0 Act. Num: 12345
End of client application.

C:\My Documents\My Teaching\EE553 (2002-2003)\ExamProjects\Question3\Project1\cl
asses>_
```



```
C:\My Documents\My Teaching\EE553 (2002-2003)\ExamProjects\Question3\Project1\cl
asses>java mypackage1/Server
Start listening on port 5050
Accepted socket connection from client
Received a valid account object
Added interest to the account object at 5.0%
Increasing the balance by:50.0
Returning the account object
Sending mypackage1.DepositAccount@1d8957f
Object Sent! - waiting for next account object
_
```

Classes Required

Deposit Account class – must implement serializable and must add in a calculate interest method - **[approx 8 marks]**
Client Class – must be modified to send account objects **[approx 5 marks]**
Connection Handler class – must be modified to call methods of deposit account [**approx 5 marks]**
Server Class – does not have to change from server class given.

[18 marks total]

Solution Code:

DepositAccount Class

```
package mypackage1;

import java.io.Serializable;

public class DepositAccount implements Serializable
{
  private String owner;
  private float balance;
  private int number;

  public DepositAccount(String owner, float balance, int number)
  {
    this.owner = owner;
    this.balance = balance;
    this.number = number;
  }

  public void makeLodgement(float amount)
  {
    this.balance+= amount;
  }

  public boolean makeWithdrawal(float amount)
  {
    if (amount > this.balance) return false;
    balance-=amount;
    return true;
  }

  public void display()
  {
    System.out.println("Account details: Owner: " + owner + " Balance: " + balance + " Act. Num: " + number);
  }

  public float addInterest(float rate)
  {
    float interestAmountAdded = this.balance*rate/100;
    this.balance+=interestAmountAdded;
    return interestAmountAdded;
  }
}
```

ClientClass

```
package mypackage1;
// The Date Client - written by Derek Molloy

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

public class Client
{

    private Socket socket = null;
    private ObjectOutputStream os = null;
    private ObjectInputStream is = null;

            // the constructor expects the IP address of the server - the port is fixed
    public Client(String serverIP)
    {
                    if (!connectToServer(serverIP))
        {
                            System.out.println("Cannot open socket connection...");
                    }
            }

            private boolean connectToServer(String serverIP)
    {
                    try                 // open a new socket to port: 5050
        {
                            this.socket = new Socket(serverIP,5050);
                            this.os = new ObjectOutputStream(this.socket.getOutputStream());
                            this.is = new ObjectInputStream(this.socket.getInputStream());
                            System.out.print("Connected to Server\n");
                }
        catch (Exception ex)
        {
                            System.out.print("Failed to Connect to Server\n" + ex.toString());
                            System.out.println(ex.toString());
                            return false;
                }
                    return true;
            }

            private void getInterest()
    {
                    DepositAccount d = new DepositAccount("Derek Molloy", 1000, 12345);
                    System.out.println("Current Details:");
    d.display();
    System.out.println("Sending Account object to Server.");
                this.send(d);
                d = (DepositAccount)receive();
                if (d != null)
      {
        System.out.println("Received the Updated object");
        d.display();
      }
    System.out.println("End of client application.");
        }

        // method to send a generic object.
        private void send(Object o) {
                try
      {
                        System.out.println("Sending " + o);
                        os.writeObject(o);
                        os.flush();
                }
      catch (Exception ex)
      {
                        System.out.println(ex.toString());
                }
        }

        // method to receive a generic object.
        private Object receive()
    {
                Object o = null;
                try
      {
                        o = is.readObject();
                }
      catch (Exception ex)
      {
                        System.out.println(ex.toString());
                }
                return o;
        }

        static public void main(String args[])
    {
                if(args.length>0)
                {
                        Client theApp = new Client(args[0]);
                        try
        {
          theApp.getInterest();
        }
        catch (Exception ex)
        {
                                System.out.println(ex.toString());
                        }
                }
                else
                {
                        System.out.println("Error: you must provide the IP of the server");
                        System.exit(1);
                }
                System.exit(0);
        }
  }
```

ConnectionHandler Class

```
package mypackage1;
// The connection handler class - by Derek Molloy

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


public class ConnectionHandler
{

   private Socket clientSocket;                    // Client socket object
   private ObjectInputStream is;                   // Input stream
   private ObjectOutputStream os;                          // Output stream

        // The constructor for the connecton handler
   public ConnectionHandler(Socket clientSocket)
   {
      this.clientSocket = clientSocket;
      //Set up a service object to get the current date and time
      //theDateService = new DateTimeService();
   }

   /** Thread execution method */
   public void init()
   {
      String inputLine;

      try
      {
         this.is = new ObjectInputStream(clientSocket.getInputStream());
         this.os = new ObjectOutputStream(clientSocket.getOutputStream());
         while (this.readAccount()) {}

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

   /** Receive and process incoming command from client socket */
   private boolean readAccount()
   {
      DepositAccount tempAct = null;
      float rate = 5;

      try
      {
         tempAct = (DepositAccount)is.readObject();
         System.out.println("Received a valid account object");
      }
      catch (Exception e)
      {
         tempAct = null;
      }
      if (tempAct == null)
      {
         this.closeSocket();
         return false;
      }
```

```
        // invoke the appropriate function based on the command
        if (tempAct!=null)
        {
           float amount = tempAct.addInterest(rate);
           System.out.println("Added interest to the account object at "+rate+"%");
           System.out.println("Increasing the balance by:"+amount);
           System.out.println("Returning the account object");
           this.send(tempAct);
           System.out.println("Object Sent! - waiting for next account object");
        }
        else
        {
           this.sendError("Invalid Account Object");
        }
        return true;
     }

     // Send a message back through the client socket
     private void send(Object o)
     {
        try
        {
           System.out.println("Sending " + o);
           this.os.writeObject(o);
           this.os.flush();
        }
        catch (Exception ex)
        {
           ex.printStackTrace();
        }
     }

     // Send a pre-formatted error message to the client
     public void sendError(String msg)
     {
        this.send("error:" + msg);    //remember a string IS-A object!
     }

     // Close the client socket
     public void closeSocket()                    //close the socket connection
     {
        try
        {
           this.os.close();
           this.is.close();
           this.clientSocket.close();
        }
        catch (Exception ex)
        {
           System.err.println(ex.toString());
        }
     }
  }
```

Exam Details (General) 2004-2005 Dublin City University, Ireland.

MODULE: **Object Oriented Programming – EE553**

COURSE: M.Eng./Grad. Dip. in Electronic Systems

M.Eng./Grad. Dip. in Telecommunications Engineering
RAEC – Remote Access to Continuing Eng. Education

YEAR: Postgraduate (Year 5)
EXAMINER: Dr. Derek Molloy (DCU extension 5355)
TIME ALLOWED: 3 hours
INSTRUCTIONS: Answer FOUR questions. All questions carry equal marks
.