SEMESTER ONE EXAM SOLUTIONS 2002

Question 1.

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

(i) How is *scope resolution* performed in C++?

- An **automatic variable** is a variable with a scope local to the block of code in which it was declared. Generally automatic variables are declared within functions, destroyed when the function ends.
- An **external variable** is used to declare a variable that is defined in some other module. (e.g.use: **extern float x;**)
- A **static variable** is used to make a variable private to the module in which it occurs, when used with the definition of a global variable.

The scope level can be resolved by the rules of scope, otherwise if required a variable at a different level can be used by using the scope resolution operator :: - for example Car.draw() is a method that could be called from a derived class.

(ii) What is a JIT compiler?

When a Java applet is executed for the first time, the speed at which it executes may seem disappointing. The reason being that the Java applet's bytecode is interpreted by the JVM rather than compiled to native machine instructions. The solution to this problem lies in the Microsoft developed Just-In-Time (JIT) compiler. The JIT compiler reads the bytecode of the Java applet and converts it into native machine code for the intended operating system. Once the Java applet is converted into machine code it runs like a natively compiled program. The JIT compiler can, in certain cases, improve the run-time speed of applets by a factor of 5-10 times.

(iii) Explain the term *event listener*.

An event listener allows an application to listen for events. For example an action listener can be used to listen for button presses or enter presses within a textField. Events are generated when the button is pressed or when the button is pressed. You can use for example myTextField.addActionListener(this) to add a listener to a component.

(iv) Explain the concept of the Class class in Java.

Objects of this class (called class descriptors) are automatically created and associated with objects to which they refer. For example the **getName()** and **toString()** methods return the String containing the name of the class or interface… We could use this to maybe compare the classes of objects…

(v) In C++ what is a *static global variable* and why would it be used?

A global variable has scope throughout the entire application. A **static global variable** is used to make a variable private to the module in which it occurs, when used with the definition of a global variable.

(vi) What is meant by the term *polymorphism*?

A derived class inherits its functions or methods from the base class, often including the associated code. It may be necessary to redefine an inherited method specifically for one of the derived classes, i.e. alter the implementation. So, polymorphism is the term used to describe the situation where the same method name is sent to different objects and each object may respond differently.So polymorphism:- *allows different kinds of objects that share a common behaviour to be used in code that only requires that common behaviour.*

(vii) In the following piece of code explain what occurs. What is the value of x and y after execution (i.e. after the last line)?

```
int x=1,y=5,*p,*q;
p = &y; q=&x;
*p+=5;
*(q+=2);
```

Importantly we are operating on the pointer. P points to y and q points to x . *p+=5 increments the value at y by 5 = 10, but the second operation on q moves the pointer by 2. At the end the value of x is 1 and the value of y is 10.

[14 marks]

(b) What are Java **Interfaces**? Why and when are they used? Give an application example of when you might write your own interface.

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.

```
public interface Demo
{
      void go();
      void stop();
}

public class SomeClass extends XYZ implements Demo
{
      public void go()
      {
            // write code here…
      }

      public void stop()
      {
            // write code here…
      }
}
```

Consider the mouse. We may write many applications that use the mouse, and within this application, the mouse is used in different ways by different components. We can define a common interface for the mouse that each one of these components implements. These components then share a common defined interface.

[6 marks – 4 for desc. 2 for example]

(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?

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 cannot be virtual. 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!

[5 marks – one for each point and 2 for code/other points]

Question 2.

(a) Explain **multiple inheritance** in C++. Why is a useful feature? How does it lead to difficulties in the design process? In particular explain the use of the virtual keyword when it is used in relation to multiple inheritance.

Multiple inheritance is one aspect of C++ that leads to complications, especially when a defined class inherits from two classes.

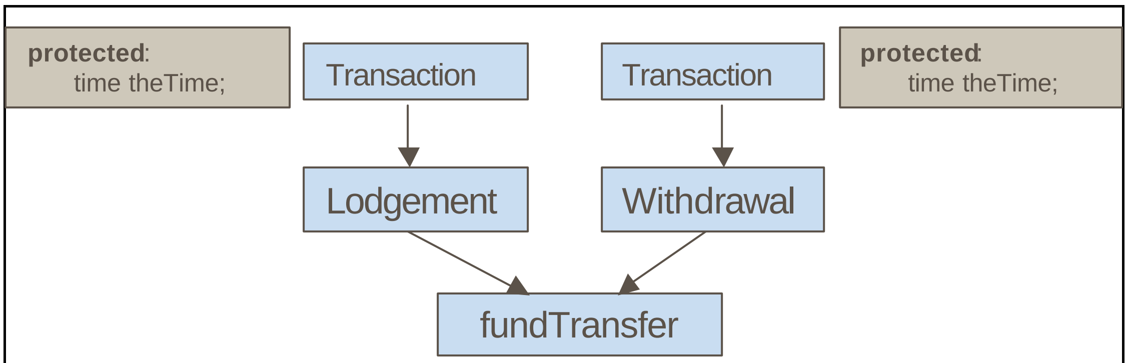


Fig Q2(a)-1 – An example where multiple base class objects are required.

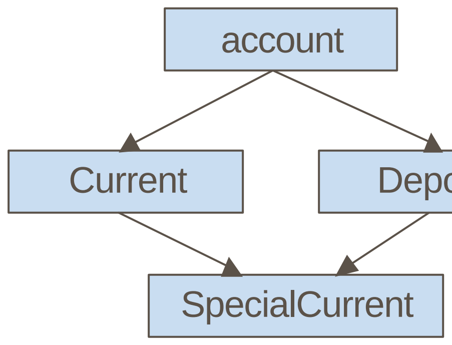


Fig Q2(a)-2 – An example where multiple base class objects are not required.

In Q2(a)-1 A Lodgement ISA Transaction, a Withdrawal ISA Transaction and at different times during execution a fundTransfer is a Lodgement or a Withdrawal.

Two separate transaction objects are being used, and they do not interfere with each other. When referring to a transaction object on the other hand, you must be careful to resolve the ambiguity that results.

```
class fundTransfer: public Lodgement,
                     public Withdrawal
{    public:                  time timeTaken(){
               return abs(Lodgement::theTime
                        - Withdrawal::theTime);
          }
};
```

Any methods that are common to both parents must be referred to it unambiguously. For example if you:

```
fundTransfer *fundPtr;
fundPtr -> display();          // not allowed – ambiguous
fundPtr -> Lodgement::display();     // OK!
fundPtr -> Withdrawal::display();    // OK!
```

This is awkward. It is often better to declare a new display() method in the fundTransfer class to try to avoid this disambiguation.

Virtual Functions and Multiple Inheritance

Here is an example of what goes on when using a virtual function in a multiple inheritance hierarchy. The virtual function selects the definition of that function closest to the most derived class:

In Q2(a)-2 Since account contains the balance of the account, and SpecialCurrent is a type of account, we do not want the balance to be duplicated, i.e. SpecialCurrent should only have one instance of its common indirect base class. We do this by declaring the base class to be virtual.

```
class Current: public virtual account {   // etc..
};
class Deposit: public virtual account {
     // etc..
};
class SpecialCurrent:
          public virtual Current,
          public virtual Deposit{
// etc.. This allows future reuse of SpecialCurrent
};
```

We are declaring that if either Current or Deposit should be used in a multiple inheritance hierarchy, then they should share the same instance of account with any other class that uses virtual inheritance of account. This leads to difficulties in the design process as we must determine at the time that we write deposit and current accounts that we are going to further develop a joint account. If this is not determined at this time then it will be necessary to alter and recompile these classes.

[10 marks – 5 for good example, 5 for other points from question]

(b) Write a section of code in C++ that demonstrates how you would structure multiple inheritance for banking software where the following account Classes exist:

?? **DepositAccount** (which includes an interest rate state),
?? **CurrentAccount** (which includes an overdraft limit state),
?? **Account** (which is generic account with balance and account number states)
?? **Cashsave** (which has an overdraft limit and an interest rate for credit balances)

The code should include the following functionality:

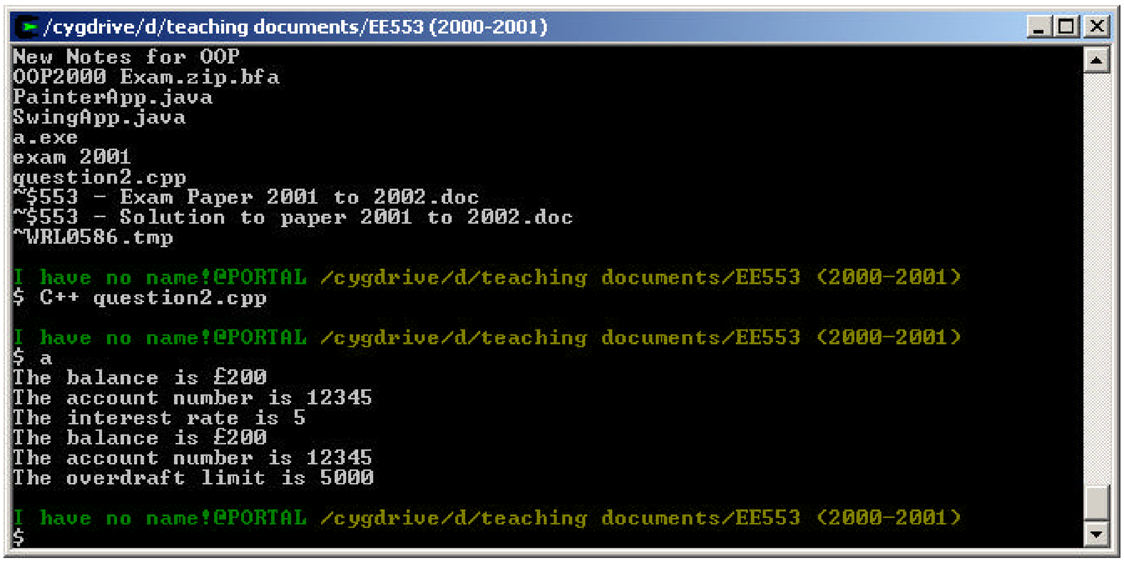
?? **display()** (which displays the details relevant to the object)
?? **makeLodgement()**
?? **makeWithdrawal()**
?? **Relevant constructors**

Do not use separate compilation for this task, i.e. use one C++ file for your code.
Show an example of your class working.

```
#include<iostream.h>

class Account
{
	private:
			float balance;
			int accountNumber;
	public:
			Account(float theBal, int theActNum);
			virtual void display();
			virtual void makeLodgement(float);
			virtual void makeWithdrawal(float);
};


Account::Account(float theBal, int theActNum)
{
	balance = theBal;
	accountNumber = theActNum;
}

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

}

void Account::makeLodgement(float amount)
{
	balance+=amount;
}
```

```
void Account::makeWithdrawal(float amount)
{
	balance-=amount;
}


class DepositAccount: virtual Account
{
	private:
		float interestRate;
	public:
		DepositAccount(float rate, float theBal, int theActNum);
		virtual void display();
};

DepositAccount::DepositAccount(float rate, float theBal, int theActNum):
	Account(theBal, theActNum)
{
	interestRate = rate;
}

void DepositAccount::display()
{
	Account::display();
	cout << "The interest rate is " << interestRate << endl;
}


class CurrentAccount: virtual Account
{
	private:
		float overdraftLimit;
	public:
		CurrentAccount(float overdraft, float theBal, int theActNum);
		virtual void display();
};

CurrentAccount::CurrentAccount(float overdraft, float theBal, int theActNum):
	Account(theBal, theActNum)
{
	overdraftLimit = overdraft;
}

void CurrentAccount::display()
{
	Account::display();
	cout << "The overdraft limit is " << overdraftLimit << endl;
}


class CashSaveAccount: CurrentAccount, DepositAccount
{
	private:

	public:
		CashSaveAccount(float overdraft, float interest, float theBal, int theActNum);
		virtual void display();
};
```

```
CashSaveAccount::CashSaveAccount(float overdraft, float interest, float theBal, int
theActNum):
        Account(theBal, theActNum),
        CurrentAccount(overdraft, theBal, theActNum),
        DepositAccount(interest, theBal, theActNum)
{}

void CashSaveAccount::display()
{
        DepositAccount::display();
        CurrentAccount::display();
}


void main()
{
        CashSaveAccount a(5000,5,200,12345);
        a.display();

}
```



/cygdrive/d/teaching documents/EE553 (2000-2001)

```
New Notes for OOP
OOP2000 Exam.zip.bfa
PainterApp.java
SwingApp.java
a.exe
exam 2001
question2.cpp
~$553 - Exam Paper 2001 to 2002.doc
~$553 - Solution to paper 2001 to 2002.doc
~WRL0586.tmp

I have no name!@PORTAL /cygdrive/d/teaching documents/EE553 (2000-2001)
$ C++ question2.cpp

I have no name!@PORTAL /cygdrive/d/teaching documents/EE553 (2000-2001)
$ a
The balance is £200
The account number is 12345
The interest rate is 5
The balance is £200
The account number is 12345
The overdraft limit is 5000

I have no name!@PORTAL /cygdrive/d/teaching documents/EE553 (2000-2001)
$
```

[15 marks]

Question 3.

(a) What is **Remote Method Invocation** (RMI) and how is it used in Java? Explain the terms *skeletons* and *stubs*. What are the limitations of RMI?

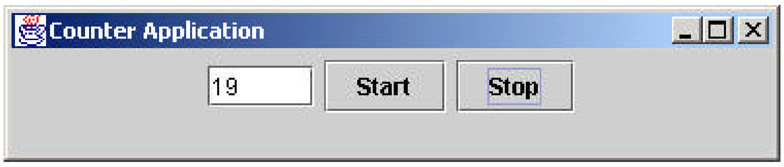
Remote method invocation (RMI) is a full-grown architecture for distributed computing, scalable to very complex tasks. Serialization handles the details of writing object level protocols, so the programmer can send objects to a stream without worrying about their structure. RMI provides a way of distributing objects as services so that a remote service request looks like a local request. The object is stationed in one place and the VM is responsible for serving the object declared exportable and puts it where an RMI object server can call on it when a request comes in. Once the object is exported, RMI takes care of the rest:

?? Serialization
?? Object transport
?? Exception handling
?? Security management.

Advance knowledge of the remote calls is essential. This involved the use of a pair of classes called **skeletons** and **stubs**. These two classes are derived directly from the class file of the remote object. When the remote client calls on the object, it does so by a lookup. This lookup is coded into the client, as are the calls to the object's methods. If the lookup succeeds the RMI server returns the remote object's stub, which is a stand-in for the remote object's class and methods. On a call to any of one of these methods, the stub sends the request to the skeleton reference, which resides on the server side. The skeleton retrieves the operations of the method to co-ordinate routing the object's response back through the stub.

[9 marks- 2 each for stubs and skeletons – 5 for remaining description]

(b) Write a Java application that uses the Java **Swing** set to create the following **application**. It is a stopwatch that starts when the "start" button is pressed and stops when the "stop" button is pressed.



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

public class SwingApp extends JFrame implements ActionListener, WindowListener, Runnable
{
        JButton start, stop;
        JTextField count;
        private int countValue = 0;
        private boolean running = false;
        Thread t;

        SwingApp()
        {
                super("Counter Application");
                start = new JButton("Start");
                start.addActionListener(this);
```

```
		stop = new JButton("Stop");
		stop.addActionListener(this);
		count = new JTextField(5);
		this.getContentPane().setLayout(new FlowLayout());
		this.getContentPane().add(count);
		this.getContentPane().add(start);
		this.getContentPane().add(stop);
		this.addWindowListener(this);
		this.setSize(400,80);
		this.show();
	}

	public void actionPerformed(ActionEvent e)
	{
		if(e.getSource().equals(start))
		{
			t = new Thread(this);
			this.running = true;
			this.countValue = 0;
			t.start();
		}
		else
		{
			this.running = false;
		}
	}

	public void windowOpened(WindowEvent e) {}
	public void windowClosed(WindowEvent e) {}
	public void windowClosing(WindowEvent e) {
		this.dispose();
		System.exit(0);
	}
	public void windowIconified(WindowEvent e) {}
	public void windowDeiconified(WindowEvent e) {}
	public void windowActivated(WindowEvent e) {}
	public void windowDeactivated(WindowEvent e) {}

	public void run()
	{
		while(this.running)
		{
			this.countValue++;
			count.setText(""+countValue);
			try{
				t.sleep(100);
			}
			catch(InterruptedException e) {}
		}
	}

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

[16 marks – half marks for structure – rest for innovation]

Question 4.

(a) What is JDBC? Give an example of how it may be used. Explain the key steps and terms related to JDBC.

A large amount of software development involves the development of client/server applications. Java provides huge advantages when developing software for client/server applications, but also provides advantages when developing client/server database applications. The facilities to do this in Java are provided by Java DataBase Connectivity (JDBC). JDBC allows the development of platform and database independent data handling solutions. The JDBC API has Java classes to represent database connections, SQL statements, result sets, etc... It allows a Java programmer to send SQL queries to a database and process the returned results.

The JDBC API is implemented using a driver manager that can support multiple drivers connecting to different databases. These drivers can either be:

Written completely in Java, so that they may be downloaded within an applet, or

written using native methods to bridge existing database access libraries.

The Connection Object:

The Connection object represents a connection with a database. A connection session includes the SQL statements that are executed and the results that are returned over the connection. One application may have more than one connection with the same database, or multiple different databases.

The Statement Object:

The Statement object is used to send SQL statements to a database. There are three kinds of Statement object that acts as containers for executing SQL statements on a connection.

A **Statement** object - used to execute simple SQL statements with no parameters.

A **PreparedStatement** object - inherits from Statement and is used to execute a precompiled SQL statement, with or without parameters.

A **CallableStatement** inherits from PreparedStatement and is used to execute a call to a database stored procedure.

So the Statement object is the generic form and the PreparedStatement/CallableStatement are specialised forms for sending particular SQL statements.

The ResultSet Object

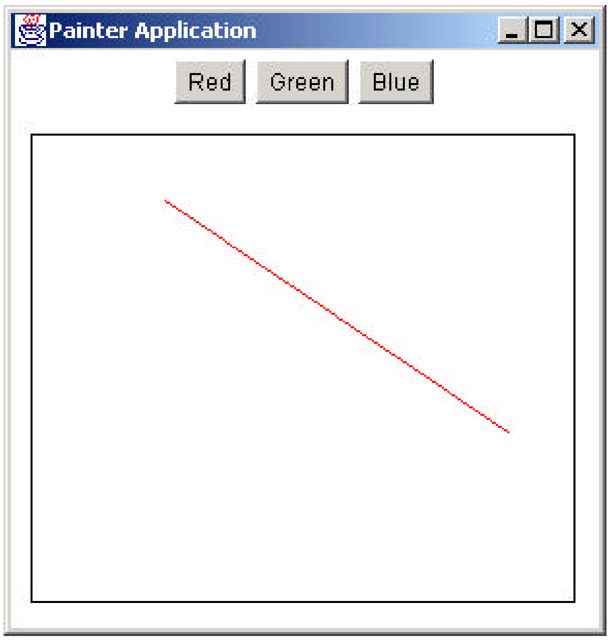
The ResultSet contains all the rows that satisfy the conditions in the SQL statement. It has a set of methods, getString(), getFloat(), getDate() etc.. to provide access to the data in these rows. The resultSet.next() method is used to move to the next row of the ResultSet.

The Driver Manager

The driver manager dynamically maintains all the driver objects that the application needs for performing database queries. If we have two different databases that we wish to connect to, we need to load in two different driver objects. The driver objects register themselves with the driver manager at the time of loading, which we can request using the Class.forName() method. The driver manager also takes care of maintenance tasks, such as driver login time limits, log tracking etc..

[8 marks]

(b) Write a Java application that uses the AWT package to write an application as shown in the figure. The colour can be chosen using the options on the left and the mouse may be used to draw wherever the button is pressed. Remember that the picture must stay visible after the application is minimized and then maximized. A code sample is provided to get you started.



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

public class PainterApp extends Frame implements ActionListener, WindowListener
{
        Color currentColor;
        Button red, green, blue;
        MyCanvas c;

        PainterApp()
        {
                super("Painter Application");

                Panel topPanel = new Panel();
                topPanel.setLayout(new FlowLayout());
                red = new Button("Red");
                red.addActionListener(this);
                green = new Button("Green");
                green.addActionListener(this);
                blue = new Button("Blue");
                blue.addActionListener(this);

                topPanel.add(red);
                topPanel.add(green);
                topPanel.add(blue);

                this.add("North", topPanel);

                c = new MyCanvas();
                this.add("Center", c);

                this.addWindowListener(this);
                this.setSize(300,300);
                this.show();
        }

        public void actionPerformed(ActionEvent e)
        {
                if (e.getSource().equals(red)) c.setColor(Color.red);
                else if (e.getSource().equals(green)) c.setColor(new Color(0,255,0));
                else c.setColor(new Color(0,0,255));
        }

        public void windowOpened(WindowEvent e) {}
        public void windowClosed(WindowEvent e) {}
        public void windowClosing(WindowEvent e) {
                this.dispose();
                System.exit(0);
        }
        public void windowIconified(WindowEvent e) {}
        public void windowDeiconified(WindowEvent e) {}
        public void windowActivated(WindowEvent e) {}
        public void windowDeactivated(WindowEvent e) {}


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


class MyCanvas extends Canvas implements MouseListener
{
        Color curColour;
        int x1,y1,x2,y2;
        boolean pressed = false;
        Vector v;

        MyCanvas()
        {
                this.curColour = new Color(0,0,0);
                this.addMouseListener(this);
                v = new Vector(100);
        }


        public void paint(Graphics g)
        {
                g.drawRect(10,10,280,240);
                g.setColor(curColour);
                g.drawLine(x1,y1,x2,y2);

        }

        public void setColor(Color theColour) { this.curColour = theColour; }

        public void mousePressed(MouseEvent e) {
                x1=e.getX();
                y1=e.getY();
                pressed = true;
        }
```

```
		public void mouseReleased(MouseEvent e) {
			x2=e.getX();
			y2=e.getY();
			pressed = false;
			repaint();
		}
		public void mouseClicked(MouseEvent e) {}
		public void mouseEntered(MouseEvent e) {}
		public void mouseExited(MouseEvent e) {}
	}
```

[17 marks based on structure and initiative]

Question 5.

(a) What does object serialization mean?

Before the advent of object serialization, if the programmer wished to send objects over a network, it was necessary to break up the objects into its constituent parts, ints, arrays etc. Object serialization makes this a relatively simple task. Once an object is committed to a stream its state is fixed. The usual means of communicating through streams is a procedural one, the client sends a request, and the server sends a response. Serialization alone requires the user to maintain the protocol for communicating.

[3 marks]

(b) A Java Vector (java.util.vector) is a useful storage mechanism class. Compare it to an array and outline its advantages and disadvantages. Give an example of how you might use it.

A Vector class implements a growable array of objects. Like an array it contains components that can be accessed using an integer index. However, the size of a vector can grow or shrink as needed. It has no serious disadvantages other than the overhead associated with a class/object.

```
Vector v = new vector(4); // space for 4 objects initially
v.add(new String("test"));
for (int i=0; i<v.size(); i++)
        String s = (String) v.elementAt(i);
```

[6 marks – 3 for the description and 3 for code example]

(c) Write a Java client/server pair, where the client sends a Vector (java.util.vector) object containing a number of words to the server and the server sorts the vector alphabetically and sends back a vector with the sorted words. The client should then display the sorted words.

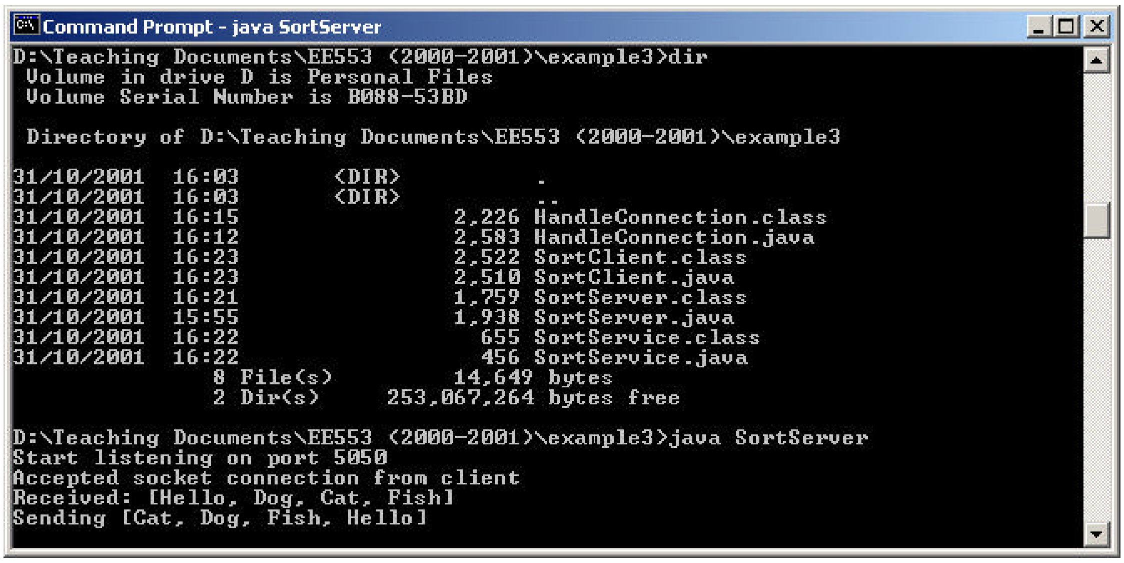
e.g. Send – ["Hello" "World" "Dog" "Cat" "House"] as a Vector and receive back ["Cat" "Dog" "Hello" "House" "World"].

You have been supplied with three sets of code to handle the basic aspects of this application. These are called

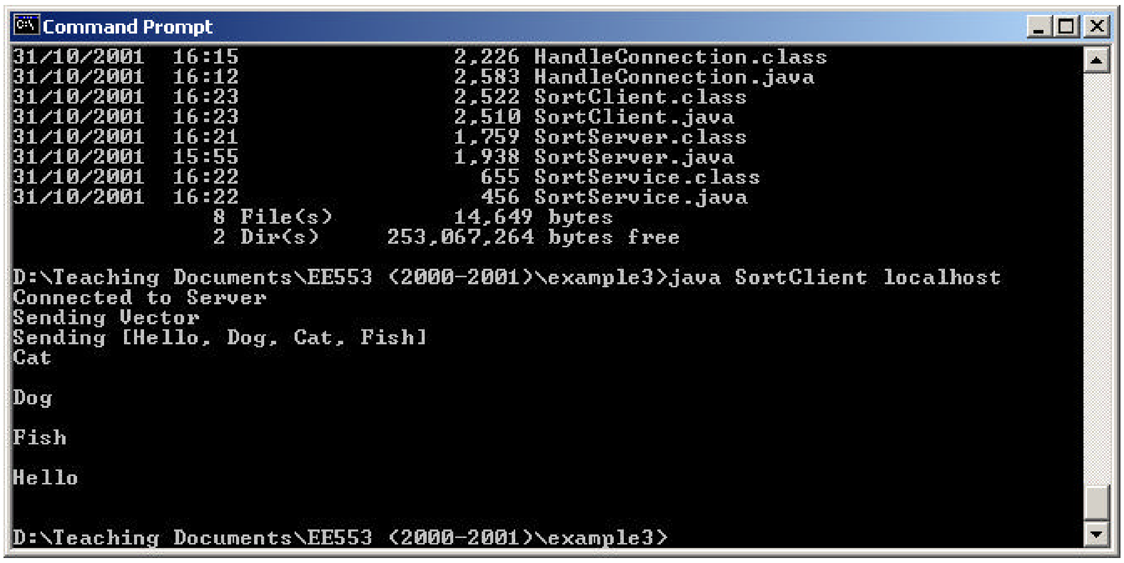
?? **Client.java**,
?? **Server.java** and
?? **ConnectionHandler.java**

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

Solution:



```
D:\Teaching Documents\EE553 (2000-2001)\example3>dir
 Volume in drive D is Personal Files
 Volume Serial Number is B088-53BD

 Directory of D:\Teaching Documents\EE553 (2000-2001)\example3

31/10/2001  16:03       <DIR>          .
31/10/2001  16:03       <DIR>          ..
31/10/2001  16:15                2,226 HandleConnection.class
31/10/2001  16:12                2,583 HandleConnection.java
31/10/2001  16:23                2,522 SortClient.class
31/10/2001  16:23                2,510 SortClient.java
31/10/2001  16:21                1,759 SortServer.class
31/10/2001  15:55                1,938 SortServer.java
31/10/2001  16:22                  655 SortService.class
31/10/2001  16:22                  456 SortService.java
               8 File(s)         14,649 bytes
               2 Dir(s)     253,067,264 bytes free

D:\Teaching Documents\EE553 (2000-2001)\example3>java SortServer
Start listening on port 5050
Accepted socket connection from client
Received: [Hello, Dog, Cat, Fish]
Sending [Cat, Dog, Fish, Hello]
```



```
31/10/2001  16:15                2,226 HandleConnection.class
31/10/2001  16:12                2,583 HandleConnection.java
31/10/2001  16:23                2,522 SortClient.class
31/10/2001  16:23                2,510 SortClient.java
31/10/2001  16:21                1,759 SortServer.class
31/10/2001  15:55                1,938 SortServer.java
31/10/2001  16:22                  655 SortService.class
31/10/2001  16:22                  456 SortService.java
               8 File(s)         14,649 bytes
               2 Dir(s)     253,067,264 bytes free

D:\Teaching Documents\EE553 (2000-2001)\example3>java SortClient localhost
Connected to Server
Sending Vector
Sending [Hello, Dog, Cat, Fish]
Cat

Dog

Fish

Hello

D:\Teaching Documents\EE553 (2000-2001)\example3>
```

```
// The Sort Client - written by Derek Molloy

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

public class SortClient
{

    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 SortClient(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 sortVector(Vector v)
  {

            System.out.println("Sending Vector");
            this.send(v);
            Vector theReturn = (Vector)receive();
            if (theReturn != null)
      {
                for (int i=0; i<v.size(); i++)
                {
                        String s = (String) theReturn.elementAt(i);
                        System.out.println(s+"\n");
                }

      }
    }

    // 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[])
  {
          Vector testVector = new Vector();
          testVector.add("Hello");
          testVector.add("Dog");
          testVector.add("Cat");
          testVector.add("Fish");


          if(args.length>0)
          {
                    SortClient theApp = new SortClient(args[0]);
                    try
       {
         theApp.sortVector(testVector);
                    }
       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);
    }
}

// The DateServer - by Derek Molloy

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

public class SortServer
{
   public static void main(String args[])
   {
      ServerSocket serverSocket = null;
      try
      {
         serverSocket = new ServerSocket(5050);
         System.out.println("Start listening on port 5050");
      }
      catch (IOException e)
      {
         System.out.println("Cannot listen on port: " + 5050 + ", " + e);
         System.exit(1);
      }
      while (true) // infinite loop - wait for a client request
      {
         Socket clientSocket = null;
         try
         {
            clientSocket = serverSocket.accept();
            System.out.println("Accepted socket connection from client");
         }
         catch (IOException e)
         {
            System.out.println("Accept failed: 5050 " + e);
            break;
         }    // create a new thread for the client
         HandleConnection con = new HandleConnection(clientSocket);
         if (con == null)
         {
            try
            {
               ObjectOutputStream os = new
    ObjectOutputStream(clientSocket.getOutputStream());
               os.writeObject("error: Cannot open socket thread");
               os.flush();
               os.close();
            }
            catch (Exception ex)
            {
               System.out.println("Cannot send error back to client:  5050 " + ex);
            }
         }
         else { con.init(); }
      }
      try
      {
         System.out.println("Closing server socket.");
         serverSocket.close();
      }
      catch (IOException e)
      {
         System.err.println("Could not close server socket. " + e.getMessage());
      }
   }
}


// The connection handler class - by Derek Molloy

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


public class HandleConnection
{

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

```
  // The constructor for the connecton handler
public HandleConnection(Socket clientSocket)
{
    this.clientSocket = clientSocket;
    theSortService = new SortService();
}

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

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

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

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

    try
    {
        v = (Vector)is.readObject();
    }
    catch (Exception e)
    {
        v = null;
    }
    if (v == null)
    {
        this.closeSocket();
        return false;
    }

  System.out.println("Received: "+v.toString());

  theSortService.setData(v);
  this.getSorted();

    return true;
}

private void getSorted()
{
    Vector returnData = theSortService.sortData();
    this.send(returnData);
}
```

```
    // 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());
        }
    }
}



// The DateTimeService that provides the current date
// by Derek Molloy.

import java.util.*;

public class SortService
{
      Vector v;

      public SortService()
      {
                  v = new Vector();
      }

      public Vector sortData()
      {
                  Object o[] = v.toArray();

                  Arrays.sort(o);
            Vector x = new Vector();
            for (int i=0; i<o.length; i++)
            {
                        x.add(o[i]);
            }

            return x;
      }

      public void setData(Vector v)
      {
            this.v = v;
      }
}
```

[16 marks – 8 for structure and 8 for coding that shows initiative]