Solutions to Semester 1 EE553 Exam Paper 2000 to 2001.
Object Oriented Programming

Dr. Derek Molloy

Answer 4 questions out of 5.

Question 1.

(a) **14 marks total 2 marks per section.**

(i) Friendly in Java relates to the package that the class is defined in. If a state or method has no defined access modifier (public, private, protected) then it is friendly and assessable throughout the package in which it is defined. In C++ a friend is explicitly specified from within the definition of a C++ class. A method or even an entire class may be specified as a friend of a class. Friendship in both cases is granted. There is no concept of the package in C++ so the friend must be explicitly declared.

(ii) Casting is used in C++ and Java to convert a variable type of a wider band to a variable type of a narrower band. It is usually used to in effect 'let the compiler know' that you are aware that there could be a loss of resolution of data. You are not solving this loss of resolution. An example would be where you are converting a floating point number into an integer, e.g. int y = 5.766; would leave y with the truncated value of 5 not the rounded value of 6.

(iii) 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.

(iv) A static method is a method that operates on static data. No other methods may operate on static data. Static methods and static data are related to the class and not to the object (often being called class methods and class states). The storage for the data is defined once for the class (the first time it is met in program execution) and remains for the duration of the application. An example of a static method is the public static void main method.

(v) Encapsulation may simply be defined as data hiding. It allows us to distinguish internal methods from the interface. We can use the public, private and protected modifiers to set the level of encapsulation.

(vi) Heavyweight components are related to the operating system whereas Leightweight components exist independently of the operating system. This allows the development of applications that have similar looks-and-feels across different operating systems. It also allows us to develop our own components. There is of course an overhead related to leightweight components.

(vii) (a) is true as all elements are initialised to 0. (b) is therefore false. (c) is false as there are 25 elements 0 to 24, so 25 is out of bounds. (d) is

false as x[0] = 0. (e) is true as there are 25 elements so 25 would be returned.

Question 1 (b) **6 marks**

Exceptions are generated during program execution if something occurs that is not quite normal from the task in question. For example the network may fail, a file may be corrupt or there may be a bug in the application the causes it to address invalid memory. In Java we may recover from this type of run time error, allowing the user to choose a different file, or check the network before continuing. The syntax we can use is in the form of try{} and catch{}

```
try{
        // Some statement that may generate an exception
} catch (Exception e)
{
        // perform some operation to recover
}
```

We may write our own type of exception for our application. For example if we wrote a method that required a non-negative number and a negative number was passed, we could generate an InvalidNumberArgument exception that was to be handled by the caller, allowing the user to re-enter the number or for that number to be skipped.

Question 1 (c) **5 marks**

Java uses automated garbage collection. C++ suffers from memory leaks, but this is not the case in Java. A threaded system thread (daemon) runs alongside the developed application with the role of checking for unused memory. Under certain conditions this unused memory will be removed and allocated back to the heap. There is an overhead in having this extra thread, but it allows mission critical applications to be developed. There are three ways that the garbage collector may be invoked:

(i) The system is running low on memory
(ii) The system is not using the CPU (e.g. user input time)
(iii) The user may request for the garbage collector to run.

Question 2 (a) **9 marks**

An applet is a specific type of Java application that is designed to be portable and downloadable over the Internet. For this reason an applet may only contain a subset of the total functionality of an application. For example, an applet cannot have access to local files on a machine; otherwise it is theoretically possible for a hostile applet to upload (to some destination) the contents of a local machine. This type of hostile applet also includes applets that may try to use all CPU available, or contact remote sites, without your permission. These too are not allowed. Applications on the other hand are assumed to be trusted, as they reside on the local hard disk. These applications have full access to the local hard-drive and to all network addresses.

An applet would be chosen when an application must be distributed to a number of clients who trust, or don't trust the content. They allow a local version to reside on the server, so only one copy exists, allowing complete version control.

An application on the other hand does not contain the inherent restrictions and so must be completely trusted by the client. An independent version is distributed to each client so there is not the same degree of version control as that associated with applets. There is no overhead in downloading the application before running it. You also do not have to be connected to the Internet to run an application.

The order of execution of an applet is main() and then whatever functionality that is involved in main()

In an applet the order is init() -> start() -> paint() and stop() on the destruction. These calls may happen asynchronously.

An application may be an applet and an application. This is created by extending the Applet class (java.awt.Applet) and then adding a main() method to the Applet class by calling the init()->start()->paint() methods in the order that the appletviewer would.

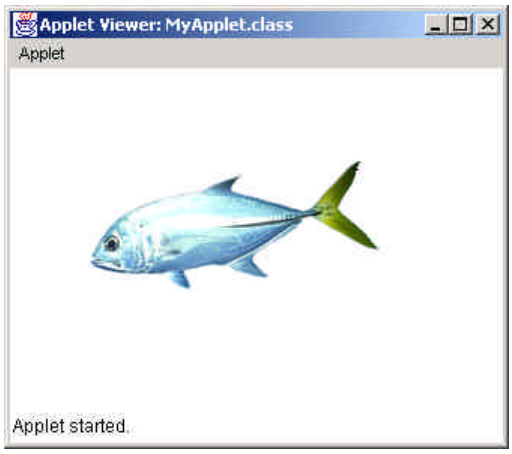
Question 2 (b) **16 marks**

The source code of this is as follows:

```
// The Applet Class - by Derek Molloy Oct 2000.

import java.applet.*;
import java.awt.*;

public class MyApplet extends Applet
{
        MyCanvas myCanvas;

        public void init()
        {
                Image theTempImage = this.getImage(this.getDocumentBase(),"test.jpg");
                while (theTempImage.getHeight(this)!=-1) {}

                myCanvas = new MyCanvas(theTempImage);
```

```
                this.setLayout(new BorderLayout());
                this.add("Center",myCanvas);
        }


}


// The Canvas Class - by Derek Molloy Oct 2000.

import java.awt.*;
import java.awt.event.*;

public class MyCanvas extends Canvas implements MouseListener, MouseMotionListener
{
        private Image theDragImage;
        private int mouseX, mouseY;

        MyCanvas(Image theImage)
        {
                this.theDragImage = theImage;
                this.addMouseListener(this);
                this.addMouseMotionListener(this);
        }

        public void paint(Graphics g)
        {
                g.drawImage(this.theDragImage, this.mouseX, this.mouseY, this);
        }

        public void mousePressed(MouseEvent e) {}
        public void mouseReleased(MouseEvent e) {}
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void mouseClicked(MouseEvent e)
        {
                this.mouseX = e.getX();
                this.mouseY = e.getY();
                this.repaint();
        }
        public void mouseDragged(MouseEvent e)
        {
                this.mouseX = e.getX();
                this.mouseY = e.getY();
                this.repaint();
        }
        public void mouseMoved(MouseEvent e) {}

}
```

The marks are roughly split as 6 marks for the applet and 10 marks for the Canvas class.



The example applet output.

Question 3 (a) **5 marks**

Constructors are used to provide initial values for the creation of objects. If constructors were not available then every object that was created would have the exact same initial states. Constructors may be overloaded so that several different sets of parameter types may be used for the creation of objects. Constructors on the other hand may not be inherited by children classes.

Destructors are called on the destruction of an object. They may be used to unallocate dynamic memory, or to perform some task such as closing network sockets or database connections. In C++ destructors can and should be virtual. In Java this is enforced.

Question 3(b) **11 marks**

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.

Overriding – Re-using the same name with the exact same arguments is known as overriding. This is most commonly used in a derived class, where the method has been defined in the parent, but a version with different functionality is required. This allows the facility to add specialised behaviour to a derived class.

e.g.

```
public void aMethod(String s) {}
public void aMethod(int s) {}
```

is an example of an overloaded method.

We can call an overridden method using the super keyword. For example to call the aMethod from a derived class we can use super.aMethod().

Difficulties occur when we override a method in a derived class, as the other overloaded methods in the base class are no longer available to the derived class.

Access modifiers affect the use of overriding. If for example a method is declared as private then the overriding method may not reduce the level of access to the method.

Question 3 (c) **9 marks**

An example of polymorphism in C++.

```
// code segment 3 - Derek Molloy

class account{
          // state
     public:
          virtual void display()
          // rest of the methods.
};

void account::display() {
     // Normal account display methods
}

class currentAccount:public account{
          // state
     public:
          virtual void display();
          virtual void display(String s);
          // rest of the methods
};

void currentAccount::display() {
     cout << "Current Account";
     // example of overriding in of display in Account
     // scope resolution is:
     account::display();
}

void currentAccount::display(String s) {
     // do something else…
     // example of overloading in currentAccount
}
```

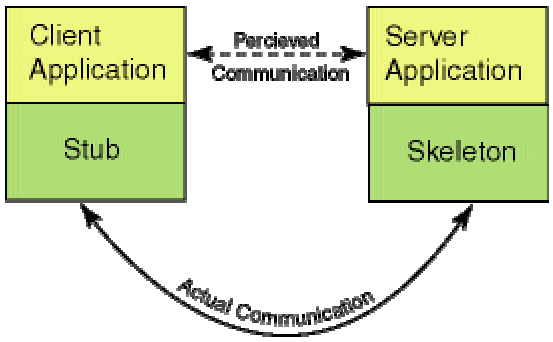
Question 4 (a) **10 marks**

Serialization makes is it easy to send objects across the network, or to write objects to a file, however it does suffer from the problems: Once an object is written to a stream, its state is fixed. In effect a copy of the object is created on the client side and sent to the server side. If this object changes on the client side, while the server is busy processing the copy, then the copy is no longer up-to-date. This is very problematic. One solution is to lock the client object on the client side, so that it cannot be altered while the copied object is in transit to or from the server.

As the communication becomes more complex, the task becomes even more difficult. Serialization, although providing a huge advantage over manually streaming data, still requires the programmer to maintain the protocol for communication, distributing the protocols and making sure that both the client and server are aware of the serialized object Class allowing the casting to take place.

RMI is a full architecture for distributed computing. It provides a way for distributing objects as services, where a remote service request looks similar to a local one. The object is not passed to and from the client/server, rather it is fixed in the one place. The Virtual Machine responsible for the object declares it exportable and makes it available to an **object server** that can call on it when a request is received.

The remote client obtains a reference, so it must know the **name of the object, where it is** and also **what methods it has**. Once we know this information, RMI takes care of the other parts, such as: object serialization, object transport, exception handling (Exceptions may occur when network connections are broken, one of the client/server pair may crash etc..) and security management.



To use a remote object, we first must have a reference to it. The only concise way to do this is to have lookup information available (such as the method names) to the client, generally prior to run-time. This is facilitated through the use of a pair of classes, called **skeletons** and **stubs** that are derived directly from the remote object.

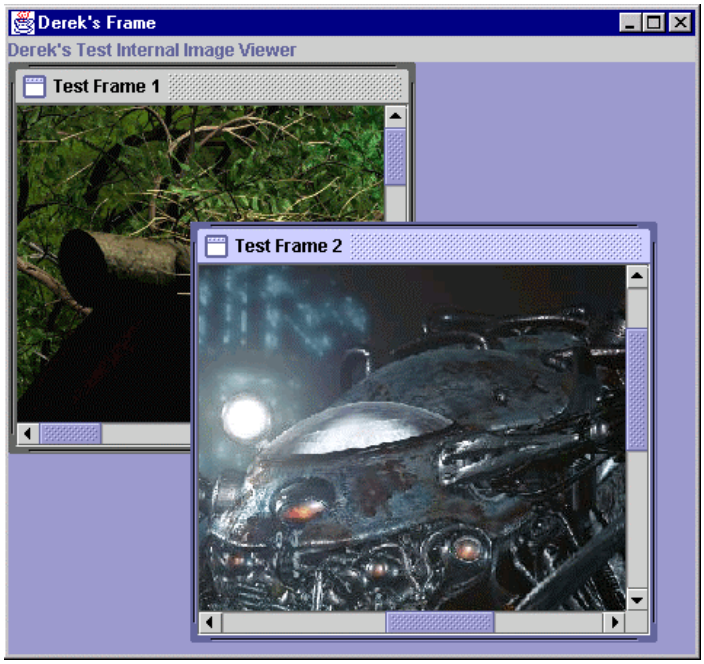
So now, the client calls the remote object by using this lookup information, which is coded into the client. If the lookup succeeds then the **RMI Server** returns the remote object's stub, which acts as a stand-in for the remote object's class and methods. A call to any of the stub's methods is sent from the stub to the skeleton reference, which resides on the server side. The skeleton in effect calls the method on the server side and routes the remote objects response back through the stub.

This communication takes place over the RMI remote reference layer (which relies on the TCP/IP transport layer). The main() method creates a **registry** on port 1099 (this is the default port for the registry services) and binds a new instance of the Request object (obj) to it. So, createRegistry(int port) creates and exports a Registry on a local host that accepts requests on a specified port (It returns a static Registry, i.e. The registry.). Placing an object in a registry makes it available to clients on other virtual machines, once those clients have access to the machine, they can obtain a reference to the remote object by specifying the machine name, port number and the name of the exported object.

Question 4 (b) **15 marks**

```
// Example6 - Swing Example 6 - Derek Molloy

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

public class Example6 extends JFrame
{
      JDesktopPane desktop;

      public Example6()
      {
            super("Derek's Frame");
            JLabel aLabel = new JLabel("Derek's Test Internal Image
Viewer");
            this.getContentPane().add("North",aLabel);

            desktop = new JDesktopPane();
            this.getContentPane().add("Center",desktop);

            // First Internal Frame...
            JInternalFrame jif1 = new JInternalFrame();
            jif1.setTitle("Test Frame 1");
            jif1.setSize(150,150);
            jif1.show();
            Image image1 = this.getToolkit().getImage("test.gif");
            ImageIcon i1 = new ImageIcon(image1);
            JLabel l1 = new JLabel(i1);
            JScrollPane p1 = new JScrollPane(l1);
            jif1.getContentPane().add(p1);
            jif1.setResizable(true);

            // Second Internal Frame...
            JInternalFrame jif2 = new JInternalFrame();
            jif2.setTitle("Test Frame 2");
            jif2.setSize(150,150);
            jif2.show();
            Image image2 = this.getToolkit().getImage("test2.jpg");
            ImageIcon i2 = new ImageIcon(image2);
            JLabel l2 = new JLabel(i2);
            JScrollPane p2 = new JScrollPane(l2);
            jif2.getContentPane().add(p2);
            jif2.setResizable(true);
```

```
            jif2.setLocation(100,100);

            desktop.add(jif1);
            desktop.add(jif2);

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

      public static void main(String[] args)
      {
            Example6 example = new Example6();
      }
}
```



Question 5(a) **10 marks**

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:
JDBC consists of three main steps:

(i) Create a connection to a database
(ii) Send SQL statements
(iii) Process the results

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.

(i) A **Statement** object - used to execute simple SQL statements with no parameters.
(ii) A **PreparedStatement** object - inherits from Statement and is used to execute a precompiled SQL statement, with or without parameters.
(iii) 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..

Question 5 (b) **15 marks**

A Java client/server pair, where the client sends a string to server and returns an encrypted version. Here are the modifications to the code required. First off, write the encryption service:

```
public class EncryptService
{

	public String encryptString(String s)
	{
		String back = "";
		char[] theChars = s.toCharArray();
		for (int i=0; i<s.length(); i++)
		{
			theChars[i] = (char)((int)theChars[i] + 1);
		}
		back = new String(theChars);
		return back;
	}

	public String decryptString(String s)
	{
		String back = "";
```

```
                char[] theChars = s.toCharArray();
                for (int i=0; i<s.length(); i++)
                {
                        theChars[i] = (char)((int)theChars[i] - 1);
                }
                back = new String(theChars);
                return back;
        }
}
```

Then modify the client.java to send the string as a parameter:

```
public void getEncrypt(String toEncrypt)
{
        String theEncrypt = "";
        System.out.println("Encrypt the string: "+toEncrypt);
        send("Encrypt"+toEncrypt);
        theEncrypt = (String)receive();
        if (theEncrypt != null)
        {
                System.out.println("The Encrypted string is " + theEncrypt);
         }
}

public void getDecrypt(String toDecrypt)
{
        String theDecrypt = "";
        System.out.println("Decrypt the string: "+toDecrypt);
        send("Decrypt"+toDecrypt);
        theDecrypt = (String)receive();
        if (theDecrypt != null)
        {
                System.out.println("The Decrypted string is " + theDecrypt);
         }
}
```

and modifying the main method()

```
        EncryptClient myApp = new EncryptClient(args[0]);
        try
        {
                myApp.getEncrypt(args[1]);
        }
        catch (Exception ex)
        {
                System.out.println(ex.toString());
        }

        try
        {
                myApp.getDecrypt(args[1]);
```

```
        }
        catch (Exception ex)
        {
                System.out.println(ex.toString());
        }
```

Then modify the ConnectionHandler.java code:

```
/** Receive and process incoming command from client socket */
private boolean readCommand()
 {
           String s = null;

           try
           {
              s = (String)is.readObject();
           }
           catch (Exception e)
           {
              s = null;
           }

           if (s == null)
           {
              closeSocket();
              return false;
           }

           // invoke the appropriate function based on the command
                 String startx = s.substring(0,7);
           if (startx.equals("Encrypt"))
           {
              getEncrypt(s.substring(7,s.length()));
           }
          else if (startx.equals("Decrypt"))
           {
              getDecrypt(s.substring(7,s.length()));
           }
           else
           {
              sendError("Invalid command -> " + s);
           }

           return true;
        }

        private void getEncrypt(String theString)
        {
           String encrypt = theService.encryptString(theString);
           send(encrypt);
```

```
}

private void getDecrypt(String theString)
{
    String decrypt = theService.decryptString(theString);
    send(decrypt);
}
```

The Server.java class remains mainly the same. This is shorter than it seems to write as sections of the code can be cut and paste without much difficulty.