



<b>DUBLIN CITY UNIVERSITY</b>
-------------------------------

**SEMESTER ONE EXAMINATIONS 2009/2010**

**[SOLUTIONS]**

**MODULE:** EE553 Object-oriented Programming for Engineers

**COURSE:** MTC – M.Eng. in Telecommunications Engineering  
MEN – M.Eng. in Electronic Systems  
IPME - Individual Postgrad. Modules-Electronics  
MEQ - Masters Engineering Qualifier Course

**YEAR:** C

**EXAMINERS:** Dr. Derek Molloy, Ext no. 5355

**TIME ALLOWED:** 3 Hours

**INSTRUCTIONS:** Answer Four Questions

**Question 1(a)**

(i) The `super()` call must be the first line of the constructor. It must be there to call the parent's constructor before the child's constructor can complete. If this was not the case, the internal parent object may not construct correctly, or modifications to the parent could be called before the parent is created. This compares directly to the member initialisation list in C++ constructors, which have the exact same ordering requirements.

(ii) The `Math` class has no constructor, which makes sense as there is no requirement to have an object of `Math`. The methods are called statically, e.g. `Math.sqrt(25.0)` which compares directly to a global function in C++. In Java it is not possible to have a global function.

(iii) Pointers require a reference type – e.g. `int* x;` means that `x` has a reference type of `int`. This is required as a pointer needs to know the size in order to increment its value – e.g. `x++`, which needs to know how many bytes to move in memory. Also operations like `cout << *x;` needs to know how to treat the value that has been dereferenced – in this case the output stream operator is being passed an `int` value.

(iv) There is no difference, except that the default access specifier in a class is private, but public in the case of a struct.

(v) Abstract classes in Java do not need to have an abstract method. This allows a class to be set as abstract as a placeholder in case that it needs to be set abstract in the future, or if there is a requirement that no objects be created of the class.

(vi) break causes a looping statement to quit and continue causes the statement to move to the next iteration.

(vii) It would not make any sense – an abstract method is implemented in the child class, but a static method is tied to the class itself. It would not be possible provide the implementation for the static abstract method and therefore it would not make any sense.

(viii) A modal dialog box must be closed or exited before an application can continue, but a modeless dialog box can 'run in parallel' with the application. Modal dialog boxes are much easier to control in feeding information to the application, but a modeless dialog can return values at any time, which is more difficult to structure.

[1(a) 2 marks each part]

[16 marks total]

### Question 1(b)

When dealing with assignments in Java we have a very important difference between code written in C++ and code written in Java that on initial inspection seems exactly the same. Looking at the C++ version:

```
/* In C++ */
Account a();
CurrentAccount b(500,5000); //bal = 500, overdraft = 5000
a = b;
a.display(); // results in "I am an account" being displayed
           // with no mention of overdraft and a balance
           // of 500. The compiler would have prevented b = a;
```

Looking at the equivalent Java version:

```
/* In Java */
Account a;
CurrentAccount b = new CurrentAccount(500,5000);
a = b;
a.display(); // results in "I am a current account"
           // with an overdraft of 5000 and a balance of 500
```

In C++ when we assign a=b; the CurrentAccount object is simply sliced into an Account object - i.e. "fitting into the box for an Account object". In Java we simply have a reference to the original object so that object never changes - there is no slicing performed.

[4 marks]

### Question 1(c)

In C++ methods are non-virtual by default, so to replace the behaviour (allow over-riding) of a method in the derived class you have to explicitly use the virtual keyword in the parent class. You are able to replace the behaviour of a non-virtual method but this causes serious difficulties when the behaviour of the object depends on the static rather than the dynamic type! In Java, methods are virtual by default and we always operate on the dynamic type of the object. You cannot specify a method as non-virtual, but there is a final keyword. Once you use the final keyword on a method you cannot replace it - so there are no issues with static and dynamic types.

[5 marks]

### Question 2(a)

If you state that a pointer is void in C++ it means that it can point to the address of any type of variable. This feature of the language should probably be avoided unless it is completely necessary as it allows one type to be treated as another type.

```
1
2 // void pointer example
3 int main()
4 {
5     int a = 5;
6     void* p = &a;
7
8     // *p = 6; would be a compile time error. We must cast back
9     // to an int pointer before dereferencing, e.g.
10    *((int *) p) = 6;
11 }
12
```

The void pointer type cannot be dereferenced. In this example (int \*) p is the statement that casts p into an int pointer. However, we could just as easily have cast it to any other pointer type, e.g. a float pointer, in which case modifying such a pointer could easily crash the program.

[4 marks]

### Question 2(b)

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

class Person
{
    private:
        string name;
        friend void clearName(Person &);
    public:
        Person(string);
        Person(const Person &);
        virtual bool operator == (Person);
        virtual void display();
};

Person::Person(string nm): name(nm){}
Person::Person(const Person &p) {}

bool Person::operator == (Person p)
{
    if (p.name == name) return true;
    else return false;
}
```

```

}

void Person::display()
{
    cout << "A Person with name: " << name << endl;
}

void clearName(Person &p) { p.name = ""; }

```

[8 marks - 2 for operator overloading and friend method, 2 for constructors, 2 for display]

### Question 2(c)

```

class Student:public Person
{
    private:
        int idNumber;
    public:
        Student(string, int);
        virtual void display();
};

void Student::display()
{
    cout << "A student:" << endl;
    Person::display();
    cout << "and student number: " << idNumber << endl;
}

Student::Student(string nm, int id): Person(nm), idNumber(id) {}

int main(string args[]){
    Person *p = new Person("John");
    Student *s = new Student("Joe", 123456);
    p->display();
    s->display();
    clearName(*p);
    p->display();
}

```

[8 marks – 4 for student class and 4 for main method]

### Question 2(d)

C++ distinguishes between a static type and a dynamic type of an object. The static type is determined at compile time. It is the type specified in the declaration. For example, the static type of both \*a in the example below is Person. However, the dynamic types of the pointer are determined by the type of object to which it points, which is Student. When we call the member function display(), C++ resolves the dynamic type of a and ensures that the appropriate version of display() is called. Notice how powerful dynamic binding is: You can derive additional classes from Person that override display() even after the main function has been compiled.

```

// dynamic binding
Person *a = new Student("Derek", 121212);
a->display();

```

[5 marks – 3 for desc., 2 for code]

### Question 3(a)

STL can be described and categorised as follows:

Containers: Data structures with memory management capability.

Iterators: Logic that binds containers and algorithms together.

Algorithms: Provide a mechanism for manipulating data through the iterator interface.

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.
- 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 permitted.
- multiset: Same as a set, only duplicates are permitted.

Some Iterators:

- Input iterators
- Output iterators
- Forward iterators
- Bidirectional iterators
- Random Iterators

The algorithms can be classified as:

- Non-modifying 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 ...

[ 7 marks – not all text required, only 4 examples of each]

[3 marks for general description, 4 marks for examples]

### Question 3(b)

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

using namespace std;

template <class T>
void outputFunction(T x)
{
    cout << x << endl;
}

int 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);
}

```

[7 marks, 2 for the template function, 2 for vector, 1 for sort, 2 for for\_each]

### Question 3(c)

```

import java.util.Vector;

public class VectorDemo {

    public static void main(String[] args) {

        String names[] = {"Derek", "John", "Sarah", "Jane", "Joe"};

        Vector<String> v = new Vector<String>(5);
        for(int i=0; i<names.length; i++)
            v.add(names[i]);

        // Demonstration of capacity()
        System.out.println("The capacity of the vector is: " + v.capacity());
        v.add("Adam");
        System.out.println("The capacity of the vector is: " + v.capacity());

        // use of contains
        if (v.contains(new String("John")))
        {
            System.out.println("Contains John!");
        }

        // Removal of the last element
        System.out.println("The size of the vector is: " + v.size());
        v.removeElementAt(v.size()-1);
        System.out.println("The size of the vector is now: " + v.size());

        // Use of setElementAt:
        v.setElementAt(new String("Joe"), 1);

        // last index of:
        int index = v.lastIndexOf(new String("Joe"));
        System.out.println("The last index of Joe is: " + index);

        // Display contents:
        for(int i=0; i<v.size(); i++)
            System.out.println("The Vector v["+i+"] contains: " + v.elementAt(i));
    }
}

```

#### Output:

The capacity of the vector is: 5  
 The capacity of the vector is: 10  
 Contains John!  
 The size of the vector is: 6  
 The size of the vector is now: 5

The last index of Joe is:4  
The Vector v[0] contains: Derek  
The Vector v[1] contains: Joe  
The Vector v[2] contains: Sarah  
The Vector v[3] contains: Jane  
The Vector v[4] contains: Joe

[11 marks]  
[~2 marks each point]

#### Question 4(a)

A Window object is a top-level window with no borders and no menu bar. A Frame is a top-level window with a title and a border and is a child of the Window class. A Dialog is a top-level window with a title and a border that is typically used to take some form of input from the user. A Dialog is also a child of the Window class. A Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.

Java GUIs Event Driven: user-interface objects (such as buttons, list boxes, menus, etc.) keep an internal list of "listeners". These listeners are notified (that is, the listener's methods are called) when the user-interface object generates an event. To add listeners to the list you make a call like yellowButton.addActionListener(...). What you put in the place of the ... must be an object implementing the ActionListener interface, so it will have methods capable of processing the events generated.

[5 marks, 2 for first part, 3 for event driven]

#### Question 4(b)

Write the Java code to create the Java Swing Application that is displayed in Figure 4.1. The application should allow a user to enter text in a text field that will be displayed in the text area when the send button is pressed. If the clear button is pressed the text that was being entered in the text field is cleared. This must be written as a Swing application and the application should terminate when the X button is pressed.

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Question4 extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    private JTextArea display;
    private JTextField write;
    private JButton send, clear;

    public Question4()
    {
        super("Question 4");
        this.setLayout(new BorderLayout(this.getContentPane(), BorderLayout.Y_AXIS));
        display = new JTextArea(10,40);
        display.setEditable(false);
        JScrollPane topPanel = new JScrollPane(display);
        topPanel.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        this.getContentPane().add(topPanel);
```

```

        write = new JTextField(40);
        this.getContentPane().add(write);

        JPanel bottomPanel = new JPanel();
        bottomPanel.setLayout(new FlowLayout());
        send = new JButton("Send");
        send.addActionListener(this);
        clear = new JButton("Clear");
        clear.addActionListener(this);
        bottomPanel.add(send);
        bottomPanel.add(clear);
        this.getContentPane().add(bottomPanel);

        this.pack();
        this.setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        if(e.getSource().equals(clear)){
            write.setText("");
        }
        else if(e.getSource().equals(send)){
            String text = this.write.getText() + "\n";
            display.append(text);
            write.setText("");
        }
    }

    public static void main(String[] args) {
        new Question4();
    }
}

```

[10 marks – 5 for user interface, 3 for events, 2 for general class code]

#### Question 4(c)

Modify the Java Swing application written in part (a) to allow any number of windows to be opened at the same time. When a message is typed in any window's text field and send is pressed, the message will be displayed in the text area of all windows. The number of windows could be passed from the command line interface. See Figure 4.2 below. Note: This is not a client/server question as the windows will all be on the same JVM.

```

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.*;

public class Question4 extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    private JTextArea display;
    private JTextField write;
    private JButton send, clear;
    Vector<Question4> clients;

    public Question4(String title)
    {
        super(title);
        this.setLayout(new BorderLayout(this.getContentPane(), BorderLayout.Y_AXIS));
        display = new JTextArea(10,40);
        display.setEditable(false);
        JScrollPane topPanel = new JScrollPane(display);
        topPanel.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        this.getContentPane().add(topPanel);

        write = new JTextField(40);
        this.getContentPane().add(write);

        JPanel bottomPanel = new JPanel();

```

```

        bottomPanel.setLayout(new FlowLayout());
        send = new JButton("Send");
        send.addActionListener(this);
        clear = new JButton("Clear");
        clear.addActionListener(this);
        bottomPanel.add(send);
        bottomPanel.add(clear);
        this.getContentPane().add(bottomPanel);

        this.pack();
        this.setVisible(true);
    }

    public void setClients(Vector<Question4> clients){
        this.clients = clients;
    }

    public void sendMessage(String msg){
        display.append(msg);
    }

    public void actionPerformed(ActionEvent e) {
        if(e.getSource().equals(clear)){
            write.setText("");
        }
        else if(e.getSource().equals(send)){
            String text = this.write.getText() + "\n";
            for(int i=0; i<clients.size(); i++){
                clients.elementAt(i).sendMessage(text);
            }
            write.setText("");
        }
    }

    public static void main(String[] args) {
        Vector<Question4> clients = new Vector<Question4>();
        Question4[] client = new Question4[3];
        for (int i=0; i<3; i++)
        {
            client[i] = new Question4("Q4 Window "+i);
            clients.addElement(client[i]);
        }
        for (int i=0; i<3; i++)
        {
            client[i].setClients(clients);
        }
    }
}

```

[10 marks – 5 for adapting main method, 5 for adapting events and setClients/sendMessage]

## Question 5(a)

Client.java

```

import java.net.*;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import javax.swing.*;

public class Client extends JFrame implements ActionListener
{
    private Socket socket = null;
    private ObjectOutputStream os = null;
    private ObjectInputStream is = null;
    private static final long serialVersionUID = 1L;
    private JTextArea display;
    private JTextField write;
    private JButton send, clear;

    public Client(String title, String serverIP)

```

```

{
    super(title);
    this.setLayout(new BorderLayout(this.getContentPane(), BorderLayout.Y_AXIS));
    display = new JTextArea(10,40);
    display.setEditable(false);
    JScrollPane topPanel = new JScrollPane(display);
    topPanel.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    this.getContentPane().add(topPanel);

    write = new JTextField(40);
    this.getContentPane().add(write);

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new FlowLayout());
    send = new JButton("Send");
    send.addActionListener(this);
    clear = new JButton("Clear");
    clear.addActionListener(this);
    bottomPanel.add(send);
    bottomPanel.add(clear);
    this.getContentPane().add(bottomPanel);
    this.pack();
    this.setVisible(true);
    if (!connectToServer(serverIP))
    {
        System.out.println("Cannot open socket connection...");
    }
}

public void actionPerformed(ActionEvent e) {
    if(e.getSource().equals(clear)){
        write.setText("");
    }
    else if(e.getSource().equals(send)){
        String text = this.write.getText() + "\n";
        try
        {
            display.append(text);
            send(text);
            receive();
        }
        catch (Exception ex)
        {
            System.out.println(ex.toString());
        }
        write.setText("");
    }
}

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;
    }
}

// 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();
        display.append(o.toString());
    }
    catch (Exception ex)
    {
        System.out.println(ex.toString());
    }
    return o;
}

static public void main(String args[])
{
    if(args.length>0)
    {
        String title = "Client Title";

        try{
            InetAddress ip = InetAddress.getLocalHost();
            title = ip.toString();
        }
        catch (Exception e)
        {
            title = "Failed IP";
        }
        Client theApp = new Client(title, args[0]);
    }
    else
    {
        System.out.println("Error: you must provide the IP of the server");
        System.exit(1);
    }
}
}

```

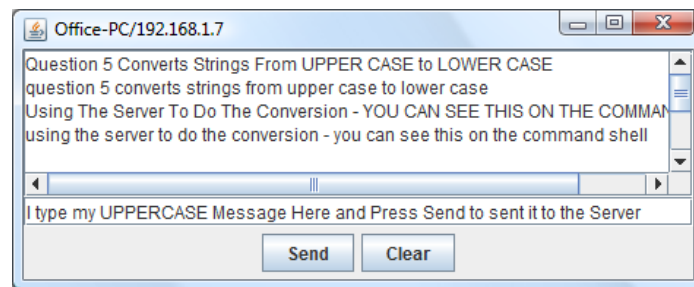


Figure 5.1 Client

Server.java

```

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

public class Server
{
    public static void main(String args[])
    {
        ConnectionHandler client;

        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

        client = new ConnectionHandler(clientSocket);
        client.start();
    }
    try
    {
        System.out.println("Closing server socket.");
        serverSocket.close();
    }
    catch (IOException e)
    {
        System.err.println("Could not close server socket. " + e.getMessage());
    }
}

```

ConnectionHandler.java

```

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

public class ConnectionHandler extends Thread
{
    private Socket clientSocket;           // Client socket object
    private ObjectInputStream is;           // Input stream
    private ObjectOutputStream os;         // Output stream

    // The constructor for the connection 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 run()
    {
        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()
{
    String s = null;
    try
    {
        s = (String)is.readObject();
    }
    catch (Exception e)
    {
        s = null;
    }

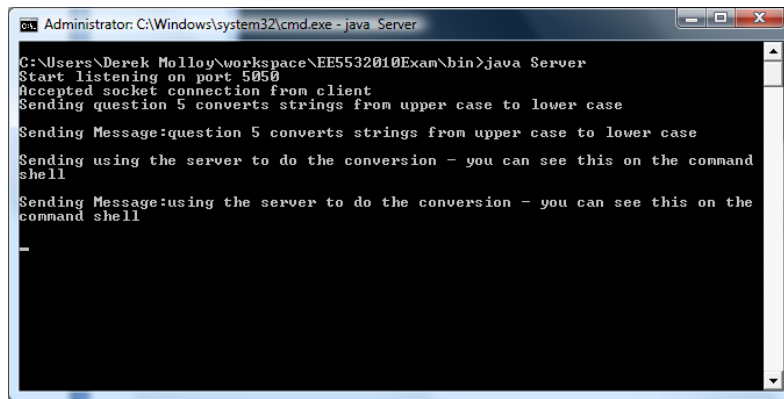
    if (s!=null)
    {
        //Modify s
        s = s.toLowerCase();
        this.send(s);
        System.out.println("Sending Message:" + s);
    }
    else
    {
        System.out.println("String is null");
    }
    if (s == null)
    {
        this.closeSocket();
        return false;
    }
    return 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());
    }
}
}

```



```
Administrator: C:\Windows\system32\cmd.exe - java Server
C:\Users\Derek Molloy\workspace\EE5532010Exam\bin>java Server
Start listening on port 5050
Accepted socket connection from client
Sending question 5 converts strings from upper case to lower case
Sending Message:question 5 converts strings from upper case to lower case
Sending using the server to do the conversion - you can see this on the command
shell
Sending Message:using the server to do the conversion - you can see this on the
command shell
```

Figure 5.2 Server

NOTE: A significant amount of the code is made available in advance. Modified code has been highlighted in bold.

**[25 Marks]**

- [5 marks for making the server fully threaded]
- [5 marks for a suitable GUI on the client and linking to client]
- [5 marks for sending and displaying from the client]
- [5 marks for receiving and modifying the message]
- [5 marks for a fully working implementation and overall structure/code]