SEMESTER ONE EXAMINATIONS 2008/2009
SOLUTIONS

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

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

1(a i) this - allows us to refer to "this object", i.e. to access states or methods for the class that the code is currently within. It allows us to pass a reference to the current object and it also allows us to access the states of the class that are currently out of scope.

1(a ii) The new keyword in C++ is used to dynamically allocate memory and it returns a pointer to the address of the space that was allocated to the object/data. In Java the new keyword works in the same way but returns a reference to the object in memory that was dynamically allocated.

1(a iii) Translator programs called *compilers* then convert the high-level languages into machine code. C, C++ and Java are all examples of high-level languages. Large programs can take significant time to compile from the high-level language form into the low-level machine code form. An alternative to this is to use *interpreters*; programs that execute high-level code directly by translating instructions on demand. These programs do not require compilation time, but the interpreted programs execute much more slowly.

1(a iv) C++ has a structure that allows us to have different types of data within the same variable. A union is particularly memory efficient as it calculates the minimum amount of memory to store the largest element in the structure. Therefore, contained data overlaps in memory.

1(a v) Java abstract classes use the abstract keyword instead of "=0" to identify an abstract method. One distinction is that a class can be abstract in Java even if it is complete. This can be used as a future design tool – to prevent objects from being created of an object so that it could be defined as abstract in the future.

1(a vi) Conditional operator ?. It has the form conditionalExpression ? trueExpression1 : falseExpression2, for example:
cout << (s.length() < t.length() ? s : t); // Display the shorter of s and t

1(a vii) This segment of code creates a function template for the function add() that will accept any type of object or primitive type T. It will accept two object of the same type call the + function on them and then return the resulting object of the same type. If the + is undefined for that type then a compiler error will occur.

1(a) **[14 marks total – 2 for each part i to vii]**

1(b) In this segment of code class A is a friend of class B. This means that when an object of the class A is passed to the method x() within class B that class B has full access to the private x state of A and Point 1 is therefore valid. However, friendship is not inherited and class C is therefore not a friend of class A, meaning that it has no access to the private state x of class A.

Friend Methods are useful as: Friend methods avoid adding unnecessary public methods to the interface; Prevent states from having to be made public. However, the overuse of friend methods can make the code very complex and hard to follow.

[6 mark]

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.

2(a)

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

class Vehicle
{
      string color;
      string brandName;
    public:
      Vehicle(string, string);
      virtual void display();
};

class Car: public Vehicle
{
      int numberSeats, numberDoors, numberWheels;

    public:
      Car(string, string, int, int, int);
      Car(Vehicle, int, int, int);
      virtual void display();
};

class Motorbike: public Vehicle
{
      int numberWheels;
      bool isOffRoad;
    public:
      Motorbike(string, string, int, bool);
      Motorbike(Vehicle, int, bool);
      virtual void display();
};


Vehicle::Vehicle(string brand, string col):
   color(col), brandName(brand) {}

void Vehicle::display()
{
     cout << "The Vehicle is a " << color << " " << brandName << endl;
}

Car::Car(string brand, string col,  int numSeats, int numDoors, int numWheels):
  Vehicle(brand, col), numberSeats(numSeats), numberDoors(numDoors),
  numberWheels(numWheels) {}

Car::Car(Vehicle v, int numSeats, int numDoors, int numWheels):
  Vehicle(v), numberSeats(numSeats), numberDoors(numDoors),
  numberWheels(numWheels){}
```

```
void Car::display()
{
   Vehicle::display();
   cout << " It is a Car with " << numberSeats << " seats, " << numberWheels <<
        " wheels and " << numberDoors << " doors\n";
}

Motorbike::Motorbike(string brand, string col, int numWheels, bool offRoad):
  Vehicle(brand, col), numberWheels(numWheels), isOffRoad(offRoad) {}

Motorbike::Motorbike(Vehicle v, int numWheels, bool offRoad):
  Vehicle(v), numberWheels(numWheels), isOffRoad(offRoad) {}

void Motorbike::display()
{
   Vehicle::display();
   cout << " It is a Motorbile with " << numberWheels << " wheels and ";
   cout << (isOffRoad ? "is off road\n" : "is not off road\n");
}

int main(void)
{
  Car c("Ferrari", "red", 2, 2, 4);
  Car d(Vehicle("Ford", "blue"), 4, 5, 4);
  Motorbike m("Ducatti", "red", 2, false);
  c.display();
  d.display();
  m.display();
  system("pause");
  return 0;
}
```

[1-2 mark for each method]
[3 marks for display]
[4 marks for motorbike]
[2 marks for main with both constructors]
[16 marks total]

2(b) Corrected code is:

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

class Calculator
{
    float total;

   public:
      Calculator(float);
      virtual float add(float);
      virtual float multiply(float);
      virtual float subtract(float);
      virtual float divide(float);
      virtual float getTotal();
      virtual void  setTotal(float);
 };

 Calculator::Calculator(float a): total(a) {}
```

```
18  float Calculator::add(float a)  {return total+=a;}
19  float Calculator::multiply(float a)  {return total*=a;}
20  float Calculator::subtract(float a)  {return total-=a;}
21  float Calculator::divide(float a)  {return total/=a;}
22  float Calculator::getTotal()  {return total;}
23  void  Calculator::setTotal(float a)  {total = a;}
24
25  int main()
26  {
27      Calculator c(100), d(30);
28      c.add(50.0f);
29      c.divide(2.0f);
30      d.add(20.0f);
31      cout << "The value of c is: " << c.getTotal() << endl;
32      cout << "The value of d is: " << d.getTotal() << endl;
33      return 0;
34  }
```

Line 8 constructor cannot be virtual
Line 5 a state cannot be virtual
Line 25 main is not a method of Calculator
Line 27 d must take an argument
Line 14 setTotal is missing the float argment
Line 00/01 missing use of standard namespace.
Line 25 main should return int and not void
Line 29 only one argument should be passed to the method.
Line 32 d cannot call d.total as total is private to the class.

[9 marks]

3(a) ***(Code in italics was provided)***

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

class Person{
        string name, id;
  public:
        Person(string, string);
        virtual void display();
        virtual string getRole() = 0;
};

Person::Person(string nm, string anid): name(nm), id(anid) {}

void Person::display()
{
        cout << endl << "Role is: " << getRole() << endl;
        cout << "Name is: " << name << " and id is: " << id << endl;
}

class Student: public Person
{
        string programme;
        int year;
  public:
        Student(string, string, string, int);
        virtual void display();
        virtual string getRole();
};

Student::Student(string name, string id, string prog, int yr):
  Person(name, id), programme(prog), year(yr) {}

string Student::getRole() { return "student"; }

void Student::display()
{
        Person::display();
        cout << "Programme is: " <<  programme << " and year is: " << year << endl;
}

class Lecturer: public Person
{
        string office;
        int phoneNum;
  public:
        Lecturer(string, string, string, int);
        virtual void display();
        virtual string getRole();
};

Lecturer::Lecturer(string name, string id, string off, int ph):
  Person(name, id), office(off), phoneNum(ph) {}

string Lecturer::getRole() { return "lecturer"; }

void Lecturer::display()
{
        Person::display();
        cout << "Office is: " << " and phone number is: " << phoneNum << endl;
}

int main(void)
{
   // test of Student and Lecturer
   Student s("John","1234", "MENG", 1);
   s.display();
   Lecturer l("Derek", "5123", "S356", 5355);
   l.display();
}
```

[10 marks]

~1 for each of the 8 methods, 1 for main, and 1 for correct abstract use

3 (b)

```
template<class T, int size>
class Storage
{
        T values[size];
        int next;
  public:
        T& operator [](int index)
        {
          return values[index];
        }
        bool addElement(T value)
        {
          if (next>size) return false;
          values[next++]=value;
          return true;
        }

        int getSize() { return next-1; }
};
```

[7 marks]

2 for template syntax, 2 for operator and 2 for add, 1 for size

3(c)

```
  Storage<Person*, 10> store;
  store.addElement(new Lecturer("Derek Molloy", "5123", "S356", 5355));
  store.addElement(new Student("John Doe","1234", "MENG", 1));
  store.addElement(new Student("James Doe","1235", "MENG", 1));
  for (int i=0; i<=store.getSize(); i++)
        store[i]->display();
```

[3 marks]

3(d)

```
  vector<Person*> vStore;
  vStore.push_back(new Lecturer("Derek Molloy", "5123", "S356", 5355));
  vStore.push_back(new Student("John Doe","1234", "MENG", 1));
  vStore.push_back(new Student("James Doe","1235", "MENG", 1));
  for (int i=0; i<vStore.size(); i++)
     vStore[i]->display();
```

[5 marks]

4(a)

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

synchronized(anObject)
{
}

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

First Thread	Second Thread
call theSynchronizedMethod()	
acquires the lock on the theObject	
executes theSynchronizedMethod on theObject	

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

Again, synchronization is based on objects:

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

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

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

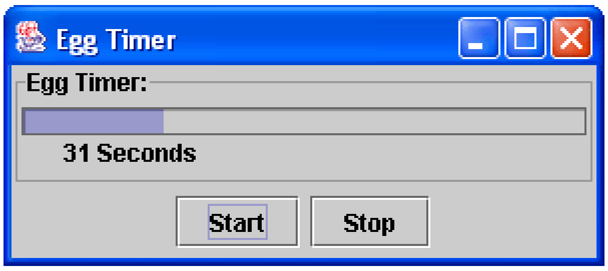
- Safe - we get the correct results
- Lively - It performs efficiently, using threads to achieve this liveliness

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

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

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

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



```
// Egg Timer Solution  - Derek Molloy

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

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

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

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

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

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

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

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

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

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

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

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


class EggTimer extends Thread
```

```
{
          private int numberSeconds = 0;
          private boolean running = true;
          private JProgressBar progressReference;
          private JLabel textLabel;


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

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

                      try{
                                Thread.currentThread().sleep(100);
                      }
                      catch (InterruptedException e)
                      {
                        System.out.println(e.toString());
                      }
                    }
          }

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

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

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

          }
}
```

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

5(a)

```
// The Transaction class - by Derek Molloy [13 marks]
import java.net.*;
import java.io.*;

public class Transaction implements Serializable
{
	public static final int LODGEMENT = 1;
	public static final int WITHDRAWAL = 2;
	public static final int BALANCEENQ = 3;
	public static final int CLOSEACT = 4;
	private int accountNumber;
	private int transactionType;
	private float amount;
	private String returnMessage = "None";
	private boolean successfulTransaction = false;

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

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

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

	public boolean isForAccount(Account a)
	{
		if ( a.getAccountNumber() == accountNumber) return true;
		else return false;
	}

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

Modifications to Server.java are: [2 marks]

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

Modifications to HandleConection.java are: [5 marks]

```
	private Account[] theAccounts;
	// The constructor for the connecton handler

	public ConnectionHandler(Socket clientSocket, Account[] theAccounts)
	{
		this.clientSocket = clientSocket;
		this.theAccounts = theAccounts;
	}

	/** Receive and process incoming command from client socket */
	private boolean readCommand()
	{
		Transaction t = null;
		try
		{
			t = (Transaction)is.readObject();
		}
		catch (Exception e)
		{
			t = null;
		}
		if (t == null)
		{
			this.closeSocket();
			return false;
		}
		for (int i=0; i<=2; i++)
		{
			if (t.isForAccount(theAccounts[i]))
			{
				System.out.println("Applying a transaction");
				t.applyTo(theAccounts[i]);
			}
		}
		return true;
	}
```

Modifications to Client.java are: [5 marks]

```
private void doTransactions()
{
	Transaction[] trans = { new Transaction(Transaction.LODGEMENT, 100001, 1000f),
		new Transaction(Transaction.BALANCEENQ, 100001),
		new Transaction(Transaction.WITHDRAWAL, 100001, 50f),
		new Transaction(Transaction.BALANCEENQ, 100001),
		new Transaction(Transaction.CLOSEACT, 100001),
		new Transaction(Transaction.BALANCEENQ, 100001) };
	for (int i=0; i<=5; i++)
		this.send(trans[i]);
}
```

[25 marks total]