(xx Marks)

INSTITUTE OF TECHNOLOGY TRALEE

WINTER EXAMINATIONS AY 2013-2014

Object Oriented Programming 2

Module Code PROG61004

CRN 43840

SUGGESTED SOLUTIONS AND MARKING SCHEME

****

Question 1

a) **Classes**, **objects** and **methods** are essential parts of OOP. Define each of these terms as fully as you can.

(6 Marks)

a) Sample solution:
class-container for code(attributes/methods) that can be a blueprint for objects (2 marks) object-instance of a class (2 marks)
method-action an object/class can performed or can be performed on an object (2 marks)

b) Write the class definition for a *Busdriver.* The *Busdriver* should have the following two **attributes**: name and age. The class should have **mutator** and **accessor** methods defined for all attributes as well as a **toString** method that uses the **String format method** to prepare its output. The class should have a **no-argument constructor** that sets up the attributes indirectly via a call to a second **2-argument constructor**.

(11 Marks)

```
import java.io.*;
   public class Busdriver implements Serializable {

   private String name;
```

```
private int age;                                                    (1 marks)

public Busdriver() {
   this("Unknown",0);}                                              (2 marks)

   public Busdriver(String name,int age) {
   setName(name);
   setAge(age);
   }                                                                (2 marks)

public String getName(){return name;}
public int getAge(){return age;}                                    (1 marks)

public void setName(String name){this.name=name;}                   (2 marks)
public void setAge(int age){this.age=age;}

public String toString(){                                           (2 marks)
    return String.format("Name: %10s \nAge:%10d \n",getName(),getAge());
                                                                    }
}
```

c) An additional **class attribute** is required for the *Busdriver* class that tracks the number of bus drivers created using the class. The attribute should have an **appropriate accessor method**. Indicate where appropriate modifications are required to your code written for b) and write the code.

Constructor→numDrivers++; (1 marks)
Attributes→private static int numDrivers; (1 marks)

method→public static int getNumDrivers(){return numDrivers;} (2 marks)

(5 Marks)

d) Draw a **UML** class diagram for the **modified** *Busdriver* class.

(4 Marks)

Solution 1 mark taken off for each major mistake in class diagram

e) Write a minimal **driver** class that creates an **array** of bus drivers. The number of bus drivers in the array, as well as the values for each attribute of a busdriver should be requested from the user using **input dialog boxes**.

(8 Marks)

```
public static void main(String[] args){
String name;
int age;
Busdriver[] busdrivers;                                             (2 Mark)

int numdrivers=Integer.parseInt(JOptionPane.showInputDialog("Enter number of drivers"));
                                                                    (1 mark)
```

```
busdrivers=new Busdriver[numdrivers];                                    (1 mark)

for(int i=0;i<busdrivers.length;i++){                                    (2 marks)
name=JOptionPane.showInputDialog("Enter Name");                          (1 mark)
age=Integer.parseInt(JOptionPane.showInputDialog("Enter age"));
busdrivers[i]=new Busdriver(name,age);}                                  (1 mark)
```

f) One **additional method** is required for the driver class written for e) above. This method should show in a message dialog box the name of the oldest driver. The method should be **invoked** from the main method by passing the array of drivers as an argument.

(7 Marks)

```
private static void findoldestdriver(Busdriver[] busdrivers){
        int oldestdriver=0;
        for(int i=0;i<busdrivers.length;i++){          (2 marks)
            if (busdrivers[i].getAge()>busdrivers[oldestdriver].getAge())
                oldestdriver=i;          (2 marks)
        }

        JOptionPane.showMessageDialog(null,"Oldest Driver: (1 mark)
"+busdrivers[oldestdriver].getName());                                   (2 marks)
}// end findoldestdriver
```

Question 2

A Fraction class is being developed to facilitate arithmetic with fractions. A fractional number has the form: **numerator/denominator** where numerator and denominator are **attributes** of the class.

The following three lines were included in a driver class that tests the functionality of the Fraction class.

fa = f1.multiply(f2);

fb= Fraction.multiply(f1,f2);

fc= Fraction.multiply(f1,f2,f3);

a) Explain the OOP concepts of method **overloading** and method **overriding**. Give an example of each as part of your explanation.

(4 Marks)

overloading-same name methods with method invoked dependent on number/type of argument plus example (2 mark)
overriding different implementation of the same method in a subclass plus example (2 marks)

b) Write the code for a **no argument constructor** for the Fraction class. The code is require to prompt the user for a numerator and denominator and **validate** the numbers entered such that the numerator must be positive and the denominator must be positive and greater than zero. The user should be prompted to re-enter values until a valid entry is made.

(6 Marks)

```
public Fraction()
{
boolean valid=false;
while (!valid ){
 numerator = Integer.parseInt(JOptionPane.showInputDialog("Enter numerator"));
 denominator= Integer.parseInt(JOptionPane.showInputDialog("Enterdenominator"));
if (numerator>=0 && denominator>0){
setNumerator(numerator);
setDenominator(denominator);
valid=true;}
}
}
```

c) Write the code for the Fraction class that will allow the three statements shown for the driver class to compile properly. You are only required to write code for the multiply method(s). The multi-argument methods are required to invoke the single argument method.

(14 Marks)

```
public Fraction multiply(Fraction f)
{
int num = f.getNumerator()*this.getNumerator();
int denom = this.getDenominator()*f.getDenominator();
Fraction fans = new Fraction(num,denom);
return fans;
}

public static Fraction multiply(Fraction f1, Fraction f2)
{
return f1.multiply(f2);
}
public static Fraction multiply(Fraction f1, Fraction f2,Fraction f3)
{
return f1.multiply(f2).multiply(f3);
```

d) }

Java provides a powerful tool called **javadoc** for generating web pages to describe classes. Write javadoc style comments, including a general **description**, **@param** and **@return** tags for the two argument multiply method written for part b) above.

(6 Marks)

```
/** 2 argument multiply method
@param f1 a fraction
@param f2 a fraction
@return the answer as a fraction */
```

Question 3

The class BusDriverSystem.java (**Appendix 1**) is a **JFrame** based **GUI** program that allows the user to set up and maintain a **collection** of bus drivers. Each bus driver is an object of the Busdriver class developed in Question1. The menu system has **JMenu** objects, one called 'fileMenu' that allows for creating, opening and saving a file as well as quitting the program, the other called 'busDriverMenu' that allows for adding a new Busdriver and displaying the entire **LinkedList** of bus drivers. Examine the code and answer the following questions. See **Appendix 2** for some useful information.

a) Explain what changes are required to the code if an **ArrayList** was required instead of a **LinkedList**.

(3 Marks)

Use ArrayList instead of LinkedList in the declaration and creation of the collection

b) Write code for the 'addDrivers' method suitable for inclusion in BusDriverSystem.java. The method is required to add multiple bus drivers to the collection. The method should use a **loop** structure to prompt the user for driver details and a **JOptionPane.YES_NO_Option confirm dialog** to check if the user is finished adding new drivers.

(10 Marks)

```
public void addBusdrivers(){
int response=JOptionPane.YES_OPTION;                                    (1 mark)
do{                                                                     (loop 2)
Busdriver driver = new Busdriver();          (create and pop bus draiver (3 marks)
driver.setName(JOptionPane.showInputDialog("Name of BusDriver"));
driver.setAge(Integer.parseInt(JOptionPane.showInputDialog("Age of BusDriver")));
busdrivers.add(driver);                                                 (2 marks)
response=JOptionPane.showConfirmDialog(null,"Do you wish to continue?","Add
Driver",JOptionPane.YES_NO_OPTION);                                     (2 MARKS)
} while (response==JOptionPane.YES_OPTION);
} // end addBusdriver
```

c) Write code for the 'display' method suitable for inclusion in BusDriverSystem.java. The method should use a **JTextArea** object to display the details of the bus drivers and use an **iterator** to scan through the **LinkedList** to retrieve the appropriate details for each BusDriver object.

(10 Marks)

```
int numBusdrivers = busdrivers.size();
if (numBusdrivers <1)
    showMessage("No busdrivers in the system");
  else {
  JTextArea area = new JTextArea();                                     (1 mark)
Iterator <Busdriver> iterator = busdrivers.iterator( );                 (3 marks)
```

```
while ( iterator.hasNext( ) )                                    (2 marks)
    area.append(iterator.next( ) + "\n");                        (3 marks)
  showMessage(area);                                             (2 marks)
}

public void showMessage (JTextArea s){
    JOptionPane.showMessageDialog(null,s);
}
```

d) When the program was first run and the save option was selected from the menu an **IOException** occurred because the BusDriver class did not implement **Serializable**. Explain precisely how this exception was handled by the program.

(7 Marks)

IOException is passed up to the calling method by the save method (throws IOException in save method) (2 marks)
There is a try catch block in actionPerformed method that catches the exception (2 marks)
A message dialog display a message "Not able to save file…….) (1 marks)
Printstack method of IOException gives details of problem indicating class does not implement Serializable interface (2 marks)

Question 4

The class BusDriverSystem.java (**Appendix 1**) is a **JFrame** based **GUI** program that allows the user to set up and maintain a **collection** of bus drivers. Each bus driver is an object of the Busdriver class developed in Question1. The menu system has **JMenu** objects, one called 'fileMenu' that allows for creating, opening and saving a file as well as quitting the program, the other called 'busDriverMenu' that allows for adding a new BusDriver and displaying the entire **LinkedList** of bus drivers. Examine the code and answer the following questions. See **Appendix 2** for some useful information.

a) The event-handling model of Java is based on the concept known as the **delegation-based event model**. Explain how this model works. Include a diagram as part of your explanation.

(6 Marks)

Diagram (2 marks), Explanation include references to source of events, handler, registering event handler with source, event fired, passed to handler for processing (4 marks)

b) Write partial code, including the **header**, for the **actionPerformed** method that **invokes** the 'addBusdrivers' and 'display' methods when the appropriate event is detected by the program. There is no requirement to write code to handle any of the items in the 'File' menu.

(10 Marks)

```
public void actionPerformed (ActionEvent e) {  (2 marks)
    if  (e.getActionCommand() .equals ("Add")){  (if 2 marks,  condition 3 marks
    addBusdrivers();                                                    (1 mark)
    }
    else if (e.getActionCommand() .equals ("Display")){                 (1 mark)
       display ();                                                      (1 marks)
                                                                        }
```

c) What modifications are required to allow for an **inner class** to take care of event handling in the program. You are required to write the **header** code for the inner class and its method(s) only and indicate any additional changes that are required elsewhere in the program.

(8 Marks)

```
    private class myHandler implements ActionListener{ (3 marks)

    public void actionPerformed (ActionEvent e) {                       (1 marks)
```

change class header by taking out implements ActionListener (2 marks)
replace this in all addActionListener methods to new MyHandler() (2 marks)

d) Write code for the 'save' method, suitable for inclusion in BusDriverSystem.java, that will **save** an **entire LinkedList** of busdrivers to a binary file called 'busDrivers.dat'. See Appendix 2 for some useful information for this part.

(6 Marks)

```
public void save() throws IOException {
    ObjectOutputStream os;                                  (1 mark create file))
    os = new ObjectOutputStream(new FileOutputStream ("busDrivers.dat")); (2 marks)
    os.writeObject(busdrivers);                                         (2 marks)
    os.close();                                                          (1 mark)
   } //end save
```

Appendix 1 BusdriverSystem.java

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

// manages a collection of bus drivers, holding the info in a LinkedList
public class BusDriverSystem extends JFrame implements ActionListener{

    JMenu fileMenu,busdriverMenu;
    LinkedList <Busdriver> busdrivers;

   public static void main( String[] args ) {
      BusDriverSystem frame = new BusDriverSystem();
      frame.setVisible(true);
   }

   public BusDriverSystem( ) {
      newSystem();
      setTitle    ( "Busdriver system" );
      setSize     ( 400,200 );
      setLocation  ( 100,100 );
      Container pane = getContentPane();
      setDefaultCloseOperation( EXIT_ON_CLOSE );
      createFileMenu();
      createBusDriverMenu();
      JMenuBar menuBar = new JMenuBar();
      setJMenuBar(menuBar);
      menuBar.add(fileMenu);
      menuBar.add(busdriverMenu);
   } // end BusDriverSystem

  public void newSystem() {
          busdrivers = new LinkedList<Busdriver>();
   } // end new system

   public void save() throws IOException {

// code required here

   } //end save

   public void open() {
          try{

          // code not given
          }
          catch(Exception e){
                  JOptionPane.showMessageDialog(null,"open didn't work");
                  e.printStackTrace();
          }

    } // end open()

   public void addBusdrivers(){

// code required here
```

```
    } // end addBusdrivers


    // Version of display which uses an iterator
    public void display (){

//code required here

    } // end display

*** actionPerformed method***

//  code required here
// partial code for the save part in the actionPerformed method given below
            try{
             save();
             showMessage("Data saved successfully");
            } // try
             catch (IOException f){
                        showMessage("Not able to save the file:\n"+
                        "Check the console printout for clues to why ");
                        f.printStackTrace();

    } // actionPerformed

     private void createFileMenu(){
            fileMenu = new JMenu("File");
            JMenuItem item;
            item = new JMenuItem("Save");
            item.addActionListener(this);
            fileMenu.add(item);
            item = new JMenuItem("Open");
            item.addActionListener(this);
            fileMenu.add(item);
            item = new JMenuItem("New File");
            item.addActionListener(this);
            fileMenu.add(item);
            fileMenu.addSeparator();
            item = new JMenuItem("Quit");
            item.addActionListener(this);
            fileMenu.add(item); }

    private void createBusDriverMenu(){
            busdriverMenu = new JMenu("BusDriver");
            JMenuItem item;
            item = new JMenuItem("Add");
            item.addActionListener(this);
            busdriverMenu.add(item);
            item = new JMenuItem("Display");
            item.addActionListener(this);
            busdriverMenu.add(item);}

     /** utility methods to make the code simpler */
    public void showMessage (String s){
            JOptionPane.showMessageDialog(null,s); }

    public void showMessage (JTextArea s){
            JOptionPane.showMessageDialog(null,s); }
}
```

Q1. Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Q2. xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx