School of Computer Science and Information Technology, RMIT University

COSC1094/COSC1095: Programming Principles 2J

Semester 1, 2003 Examination Paper

Student Number: _____

Student Name: _____

Student Signature: _____

Date: 25 June 2003

Time Allowed: 3 hours

Time Of Examination: 1:45 pm – 5:00 pm

Reading Time: 15 minutes (Starting at 1:45 am)

Number of Pages (including this page):

INSTRUCTIONS - PLEASE READ CAREFULLY

- Contribution of this examination to assessment: 50%.
- Total marks for this examination is 100.
- You have to achieve at least 50 marks on this examination to pass the course.
- This is a closed book examination. It consists of:

  Part A: 20 multiple-choice questions: 10 x 1 = 10 marks

  Part B: 4 short programming questions, 4 x 10= 40 marks

  Part C: Question 1: 20 marks and Question 2: 30 marks, Q1 + Q2= 50 marks

- Use blue or black ink pen to answer all questions.
- All answers should be written on this examination paper. You can use the back of the sheets if you need more spaces for your answer. Clearly state the question number. Cross out any mistakes made.
- Circle the appropriate answer in the question when answering the multiple-choice questions. If you make a mistake or change your mind, clearly cross out your choice and circle the new choice.
- No books or reference materials are allowed.
- Electronic devices are not allowed.
- Selected AWT components and methods are in the Appendix of this examination paper (page ------)

Part A: Multiple Choice Questions (10 marks: 1 mark per question)

1. Which one of the following statements about a constructor is false?

A. It must return type void
B. It may be overloaded
C. It has the same name as the class
D. It initializes the instance variables

Ans: A

2. Which one or more of the statements in the *main* method below will report syntax error(s)?

```
interface Printable {
   public void print();
}

class PrintableSquare1 {
   public void print(){System.out.println(" I am a
Square");}
}

class PrintableSquare2 implements Printable {
   public void print(){System.out.println(" I am a
Square");}
}

interface PrintableMoveable extends Printable{
   public void move(int x, int y);
}

public class TestInterface{
   public static void main (String[] args)throws
IOException {
     Printable p;
     p = new Printable();                // statement 1
     p = new PrintableMoveable();        // statement 2
     p = new PrintableSquare1();         // statement 3
     p = new PrintableSquare2();         // statement 4
   }
}
```

A. statement 1
B. statement 2
C. statement 3
D. statement 4

ANS: A.B C

3. In the program below class **A** has two methods *meth1()* and *meth2()*. The method *meth2()* is overridden in the subclass **B**. The subclass **B** has another method *meth3()*. Which one of the statements below in the *main* method of the **Test** class will result in compilation error?

```
public class Test  {
    public static void main(String[] args) {
          B b = new B();
          b.meth1();
          b.meth2();
          b.meth3();
    }
}
class A {
    public void meth1() {    }
    public void meth2() {    }
}
class B extends A{
    public void meth2() {    }
    public void meth3() {    }
}
```

A. a.meth1();
B. a.meth2();
C. a.meth3();
D. None of the above

Ans: D

4. Which one of the following classes can be used to convert a Java I/O byte stream to a Java I/O character stream?

A. BufferedReader
B. PrintStream
C. InputStreamReader
D. PrintWriter

Ans: C

5. Which one of the following statements about the methods in the **Applet** class is true?

A. *Repaint()* calls *update()*
B. *init()* calls *start()*
C. *stop()* calls *destroy()*
D. *paint()* calls *update()*

Ans: A

6. For the method *increment()* in the **SomeOtherClass**, which one of the statements below are valid?

```
class A {
     …
   private int a;
   int b;
   protected int c;
   public int d;

}
class SomeOtherClass { // not in same package
  void increment() {
     A refA = new A(...);
     // statements to be inserted here
  }
}
```

```
   A. refA.a++
   B. refA.b++
   C. refA.c++
   D. refA.d++
```

ANS: D

7. In the following code **C** is a subclass of **B** and **B** is a subclass of **A**. Both **A** and **B** are abstract classes. To make **C** a concrete class (as opposed to abstract) which one or more methods below must be implemented in that class?

```
abstract class A {
     public abstract void printOne();
     public void printTwo(){
          System.out.println("2");
     }
}
abstract class B extends A{
   public abstract void printThree();
   public void printFour(){
      System.out.println("4");
   }
}
class C extends B {
   // ...
}
```

```
   A.  printOne()
   B.  printTwo()
   C.  printThree()
   D.  printFour()
```

ANS: A and C
8. Which is the output of the program below?

```
public class Test2DArray {
   public static void main (String[] args){
     int[][] m = new int[3][3];
       for (int i=0; i<3; i++)
         for (int j=0; j<3; j++)
            if (i == j)
               m[i][j] = 1;
            else m[i][j] = 0;
     for (int i=0; i<3; i++) {
       for (int j=0; j<3; j++)
          System.out.print("   "+m[i][j]);
       System.out.println();
     }
   }
}
```

```
A.
0  0  0
1  1  1
0  0  0

B.
0  1  0
0  1  0
0  1  0

C.
1  0  0
0  1  0
0  0  1

D.
0  1  0
1  0  1
0  1  0
```

ANS: C
9. Any exceptions that are thrown in a method and NOT caught, they:-

A. must be ignored
B. must be declared as part of the signature of the main method
C. must never occur
D. must be declared as part of the method's signature
E. none of the above

Ans: D

10. Consider the following method:

```
public void recursion (int x) {
   System.out.print (x + " ");
   if (x < 10) {
       recursion (x + 3);
   }
   System.out.println (x + " ");
}
```

What is the output when *recursion(1)* is executed?

A. 1 1
B. 1 4 7 10
C. 1 4 7 10 10
D. 1 4 7 10 10 7 4 1

Ans: D

Part B: Short programming questions (40 marks: 4 questions each worth 10 marks)

1. Write program that will read line(s) of text from the keyboard and is terminated when the word "DONE" is entered. The program then output the number of words and the number of non-space characters as shown in the sample output below.

Please enter text, type DONE to quit:
This is a test
Here is another test
DONE
Number of words: **8**
Number of characters: **28**

Notes:

(a) You must use the **StringTokenizer** class. You can make use of the **ConsoleReader** class.
(b) The output in bolds are the user inputs.

```
import java.util.StringTokenizer;
// Count the number of words and characters ignore spaces
public class CountWords
{
   public static void main (String [] args)
   {
      int wordCount=0, characterCount =0;
      String line, word;
      StringTokenizer tokenizer;

      ConsoleReader console= new ConsoleReader (System.in);
      System.out.println("Please enter text, type DONE to quit:");
      line =console.readLine();

      while (!line.equals("DONE"))
      {
         tokenizer=new StringTokenizer(line);
         while (tokenizer.hasMoreTokens())
         {
            word=tokenizer.nextToken();
            wordCount++;
            characterCount +=word.length();
         }
         line =console.readLine();
      }
      System.out.println("Number of words: " + wordCount);
      System.out.println("Nmuber of characters: "+ characterCount);
   }
}
```

2. ~~The following to be completed program involves making use of linked list technique~~ to store the title of a list of books. It then scans through the list and then prints the title of the books in the list. The classes in this program are **Book**, **BookList** and **Library**.

The **Book** class below set up a list of book objects that are to be stored as a linked list in the **BookList** class.

```
// Stores title and a reference to another book
class Book
{
     private String title;
     private Book next;//refers to the next book in the
list

     Book (String newTitle)
     {
        title=newTitle;
        next=null;
     }
     public Book getNext()
     {
         return next;
     }
     public void setNext(Book nextBook)
     {
        next=nextBook;
     }
     public void print()
     {
       System.out.println(title);
     }
}// end Book class
```

Your task is to complete the codes in the **BookList** class below. This class contains:

(a) an *add(String newTitle)* method to form the books in the **Book** class into a linked list and

(b) a *print()* method to print out the list of book title.

```
// This class creates a linked list of books
class BookList
{   // to be completed

    private Book list;
    BookList()
    {
        list=null;
    }
    public void add (String newTitle)
    {
       Book newbook =new Book (newTitle);
       Book current;

       if (list==null)
          list=newbook;
       else
       {
           current=list;
           while(current.getNext() !=null)
              current=current.getNext();
           current.set_next(newbook);
       }
    }

    public void print()
    {
       Book current=list;
       while (current!=null)
       {
           current.print();
           current=current.getNext();
       }
    }
}

// The driver class to populate the list of books and then
prints them out.

public class Library
{
   public static void main (String [] args)
   {
      BookList books=new BookList();
      books.add( "Animal farm");
      books.add("Burmese days");
      books.add("Coming up for air");

      books.print();
   }
}// end Library class
```

3. Write a Java program to read a text file named, **input.txt** and output the content of the input file to another file named, **output.txt**. The output file must have the same content and format as in the input file except that each line of text is numbered incrementally as shown in the sample printout below. You must use exception handling technique by using try, catch, and finally blocks to produce the following messages (a) "File not found." when the file to be read is not available or (b) "Error in reading the file." when the file is corrupted and (c) "Bye for now." no matter what happens.

input.txt

This is a test.
Here is another test.
This is fun.

output.txt

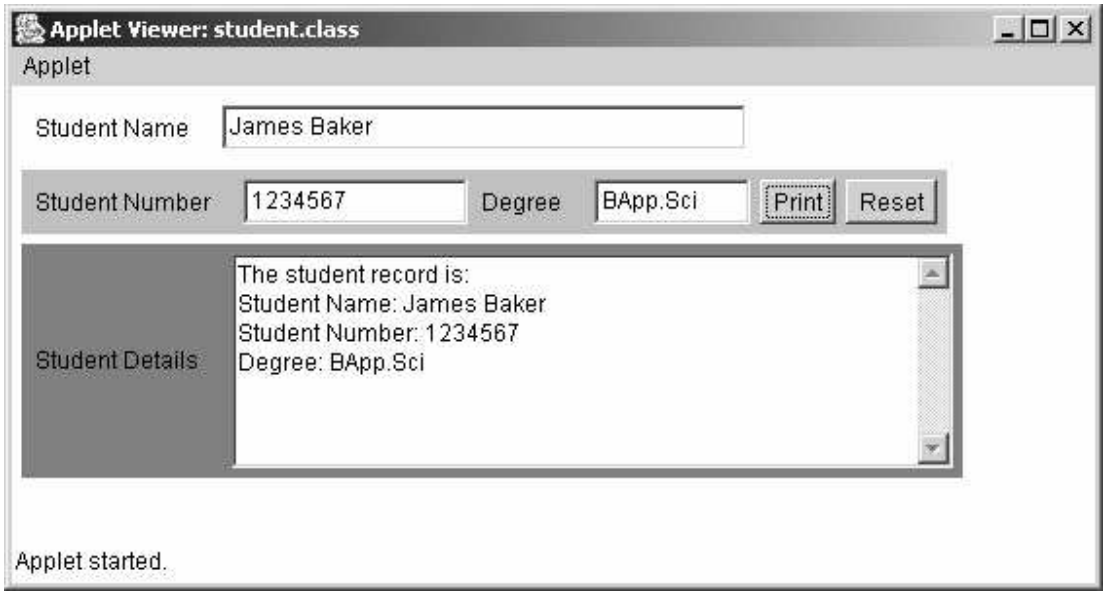
1 This is a test.
2 Here is another test.
3 This is fun.

```
import java.io.*;

public class TextEOFDemo
{
   public static void main(String[] args)
   {
      try
      {
         BufferedReader inputStream =
             new BufferedReader(new FileReader("input.txt"));
         PrintWriter outputStream =
             new PrintWriter(new FileOutputStream("output.txt"));

         int count = 0;
         String line = inputStream.readLine();
         while (line != null)
         {
            count++;
            outputStream.println(count + " " + line);
            line = inputStream.readLine();
         }
         inputStream.close();
         outputStream.close();
      }
      catch(FileNotFoundException e)
      {
         System.out.println("File not found.");
      }
      catch(IOException e)
      {
         System.out.println("Error in reading the file.");
      }
```

```
      finally
      {
         System.out.ptintln("Bye for now.");
      }
   }
}
```

~~Part C: Extended programming questions (50 marks)~~

1. Graphical User Interface and events handling (20 marks)



Implement a Graphical User Interface as shown above. The following properties are needs to be accounted:

(a) Use Panels to create "Student name", "Student Number", and "Student Details" labels together with the text boxes or text area. Make these panels white, lighGray and gray colour respectively. They are required to be adjusted to the left-hand side of the window
(b) The text boxes have size of 35, 13 and 8 in the order shown in the Applet above. The text area has the size of (6,50).
(c) Once information is entered into the text boxes and the "Print" button is clicked, the information is to be printed in the text area as shown. If the "Reset" button is clicked the text area will be cleared.
(d) You may like to use either Applet (AWT) or Java Swing. Selected AWT components methods are in the Appendix of this paper.

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

public class student extends Applet implements ActionListener {

   TextField nameField=new TextField(35);
   TextField numberField=new TextField(13);
   TextField gpaField=new TextField(8);
   TextArea studentArea=new TextArea(6,50);
```

```
   public void init() {
      setLayout(new FlowLayout(FlowLayout.LEFT));

      Panel panel1=new Panel();
      Panel panel2=new Panel();
      Panel panel3=new Panel();

      panel1.setLayout(new FlowLayout(FlowLayout.LEFT));
      panel2.setLayout(new FlowLayout(FlowLayout.LEFT));
      panel3.setLayout(new FlowLayout(FlowLayout.LEFT));

      panel2.setBackground(Color.lightGray);
      panel3.setBackground(Color.gray);

      Label nameLabel=new Label ("Student Name");
      Label numberLabel=new Label ("Student Number");
      Label gpaLabel=new Label ("Degree");
      Label summaryLabel=new Label ("Student Details");
      Button storeButton=new Button("Store");
      Button reset=new Button("Reset");

      panel1.add(nameLabel);
      panel1.add(nameField);

      panel2.add(numberLabel);
      panel2.add(numberField);
      panel2.add(gpaLabel);
      panel2.add(gpaField);
      panel2.add(storeButton);
      panel2.add(reset);

      panel3.add(summaryLabel);
      panel3.add(studentArea);

      add(panel1);
      add(panel2);
      add(panel3);

      storeButton.addActionListener(this);
      reset.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e) {
      //if (e.getSource() instanceof Button) {
      if (e.getActionCommand().equals("Store")) {

         studentArea.setText("The student record is:");
         studentArea.append("\nStudent Name: "+ nameField.getText());
         studentArea.append("\nStudent Number: "+
numberField.getText());
         studentArea.append("\nDegree: "+ gpaField.getText());
      }
      else if (e.getActionCommand().equals("Reset"))
         studentArea.setText("");
   }
}
```

2. Employees Payment System (30 marks)

The following is an implementation of a simple company salary payment system to pay its workers. It demonstrates the use of inheritance and polymorphism in Java programming.

There are 7 classes in this system. These classes are **Company**, **Staff**, **StaffMember**, **Volunteer**, **Employee**, **WageWorker** and **HourlyWorker**.

Following is the design of the system with descriptions of each mentioned classes. Your task is to complete the code in the classes using the information given. Note that, a part from the methods mentioned in the instructions, no extra methods are to be added.

(a) **StaffMember** class
This is an abstract class. It does not represent a particular type of employee but rather it serves as the ancestor (superclass) of all employee classes (including the **Volunteer** class shown below) and contains information that applies to all employees. Each staff has a name, address, and phone number, which are inherited by all descendants (subclasses). This class contains a *getDetails()* method to return the information managed by it. It also contains an abstract method, *public abstract double pay()* to output the pay (if any).

```
abstract public class StaffMember  {     // to be completed

{
   protected String name;
   protected String address;
   protected String phone;

   //  Sets up a staff member using the specified information.
   public StaffMember (String eName, String eAddress, String ePhone)
   {
      name = eName;
      address = eAddress;
      phone = ePhone;
   }

   //  Returns a string including the basic employee information.
   public String getDetails()
   {
      String result = "Name: " + name + "\n";

      result += "Address: " + address + "\n";
      result += "Phone: " + phone;

      return result;
   }

   //  Derived classes must define the pay method for each type of
   //  employee.
   public abstract double pay();
}
```

(b) **Volunteer** class
This class is inherited from the **StaffMember** class and represents a person that does not get any pay for the work. It has a constructor that pass on the person's personal details (name, address, and phone number) to the superclass. It has the *pay()* method which returns 0.0 amount.

```
public class Volunteer extends StaffMember  {  // to be completed

   //  Sets up a volunteer using the specified information.
   public Volunteer (String eName, String eAddress, String ePhone)
   {
      super (eName, eAddress, ePhone);
   }

   //  Returns a zero pay value for this volunteer.
   public double pay()
   {
      return 0.0;
   }
}
```

(c) **Employee** class
This class is inherited from the **StaffMember** class and represents an employee that gets paid at a particular rate for each pay period. The pay rate as well as the employee's tax file number, is passed on to the superclass along with personal details of the employee. The *getDetails()* method is overridden here to concatenate this additional information. The *pay()* method in this class simply returns the rate for that employee.

```
public class Employee extends StaffMember  {  // to be completed

   protected String taxFileNumber;
   protected double payRate;

   //  Sets up an employee with the specified information.
   public Employee (String eName, String eAddress, String ePhone,
                    String taxNumber, double rate)
   {
      super (eName, eAddress, ePhone);

      TaxFileNumber = taxNumber;
      payRate = rate;
   }

   //  Returns information about an employee as a string.
   public String getDetails()
   {
      String result = super. getDetails();

      result += "\nTax File Number: " + TaxFileNumber;

      return result;
   }

   //  Returns the pay rate for this employee.
   public double pay()
   {
      return payRate;
   }
}
```

(d) **WageWorker** class

This class is inherited from the **Employee** class and represents an employee who may earn a bonus in addition to his/her normal pay rate. Personal details of the wage type worker are passed on to the superclass and the bonus is initially set to 0. A bonus is awarded to an wage worker using the *bonus(double aBonus)* method. The *pay()* method is overridden here (though inheritance) to include the bonus. After the bonus is awarded it is reset to 0.

```
public class WageWorker extends Employee  {   // to be completed

   private double bonus;

   //  Sets up a wage worker with the specified information.
   public WageWorker (String eName, String eAddress, String ePhone,
                      String taxNumber, double rate)
   {
      super (eName, eAddress, ePhone, taxNumber, rate);

      bonus = 0;  // bonus has yet to be awarded
   }

   //  Awards the specified bonus to this worker.
   public void bonus (double aBonus)
   {
      bonus = aBonus;
   }

   //  Computes and returns the pay for an wage worker, which is the
   //  regular employee payment plus a one-time bonus.
   public double pay()
   {
      double payment = super.pay() + bonus;

      bonus = 0;

      return payment;
   }
}
```

(e) **HourlyWorker** class

This class is also inherited from the Employee class and represents an employee whose pay rate is applied on an hourly basis. Personal details of the hourly worker are passed on to the superclass and the hours worked is initially set to 0. The number of hours worked is set by the *hours(int NoOfHours)* method. The pay method is overridden here (though inheritance) to calculate the pay for this worker. Once the pay is calculated the number of hours worked is then reset to zero. The *getDetails()* method is also overridden here to show the number of hours worked.

```
public class HourlyWorker extends Employee  { // to be completed

   private int hoursWorked;

   //  Sets up this hourly employee using the specified information.
   public HourlyWorker (String eName, String eAddress, String ePhone,
                  String taxNumber, double rate)
   {
      super (eName, eAddress, ePhone, taxNumber, rate);

      hoursWorked = 0;
   }

   //  Adds the specified number of hours to this employee's
   //  accumulated hours.
   public void hours (int noOfHours)
   {
      hoursWorked += noOfHours;
   }

   //  Computes and returns the pay for this hourly employee.
   public double pay()
   {
      double payment = payRate * hoursWorked;

      hoursWorked = 0;

      return payment;
   }

   //  Returns information about this hourly employee as a string.
   public String getDetails()
   {
      String result = super.getDetails();

      result += "\nHours worked: " + hoursWorked;

      return result;
   }
}
```

(f) **Staff** class
This class maintains an array of objects that represent individual employees of various kinds. This is shown in the codes as follows:

```
public class Staff  {

   private StaffMember[] staffList;

   public Staff () {

      staffList = new StaffMember[4];

      staffList[0] = new WageWorker ("Jeff", "123 Main st",
         "11111111", "123456789", 1200.50);

      staffList[1] = new Employee ("Helen", "456 High St",
         "22222222", "234567890", 1000.00);

      staffList[2] = new HourlyWorker ("Jill", "678 Fifth Ave.",
         "33333333", "345678900", 10.00);

      staffList[3] = new Volunteer ("Bill", "987 Bay Crt.",
         "44444444");

      ((WageWorker)staffList[0]).bonus (500.00);

      ((HourlyWorker)staffList[2]).hours (40);

    }// end Staff constructor
```

The **Staff** class contains a *payday()* method that scans though the list of employees, printing their information and invoking their *pay()* methods (abstract method in the **StaffMember** class above) to determine how much each employee should be paid. This method call is through polymorphism. When the pay for a particular staff is 0.0 (ie. a volunteer) a "Thanks!" message is then output.

```
   public void payday () {   // to be completed

      double amount;

      for (int count=0; count < staffList.length; count++)
      {
         System.out.println (staffList[count]);

         amount = staffList[count].pay();  // polymorphic

         if (amount == 0.0)
            System.out.println ("Thanks!");
```

```
         else
            System.out.println ("Paid: " + amount);

         System.out.println ("-----------------------------------");
      }
   }
}
```

The output due to above is as follows:

Name: Jeff
Address: 123 Main St
Phone: 11111111
Tax File Number: 123456789
Paid: 1700.50

Name: Helen
Address: 456 High St
Phone: 2222222
Tax File Number: 234567890
Paid: 1000.00

Name: Jill
Address: 678 Fifth Ave
Phone: 33333333
Tax File Number: 345678900
Hours worked: 40
Paid: 400.00

Name: Bill
Address: 987 Bay Crt
Phone: 44444444
Thanks!

(g) Company class
This is the driver class that creates a staff of employees and invokes the *payday()* method to pay them.

```
public class Company  {
   public static void main (String [] args) {   // to be completed

      Staff personnel = new Staff();

      personnel.payday();
   }
}
```