CMPUT 102: Structural Programming and Data Structures

Final Exam : Tuesday December 19th, 2000
Instructor : Osmar Zaïane
Section : A1

Copy #: ___________

Student Name :____

Student ID :____

Seat Number : ____

- The duration of the exam is 3 hours.
- There are 5 sections worth a total of 100 points.
- This final exam will count for 35% of the overall course grade.
- Read all questions carefully.
- Use a pen and not a pencil.
- For full grades, answer all parts of all questions.
- Be concise and give clear and legible answers in the provided spaces.
- Non-legible answers will not be marked.
- Use the back of the pages for your rough notes and calculations.
- Cheating is a serious offence in the code of student behaviour.
- No books, notes or other aids are permitted during the exam.
- Good luck!

Section 1	Section 2	Section 3	Section 4	Section 5	**Total**
Out of 10	Out of 30	Out of 10	Out of 35	Out of 15	**Out of 100**

Section 1: Simple Java programs [10 points]

1- (2 points) Use the Keyboard class and write a java program that reads an integer from the keyboard and displays the double (i.e. twice the value) of that integer on the screen.

```
public class question1 {
   /* Reads an integer and prints its double */
      public static void main (String args[]) {

             int myInt;

                 myInt=Keyboard.in.readInteger().intValue();

                 System.out.print(2*myInt);


      }
}
```

2- (3 points) Use the Keyboard class and write a java program that reads an integer representing the age of a person and displays the message "too young to drive" if the age is less than 16, the message "check with your doctor" if the age is greater than 75, and "drive safely" otherwise. No input validation is necessary.

```
public class question2 {
   /* Reads a driver's age and advises on drivability */
      public static void main (String args[]) {

             Integer myInteger;

                 myInteger=Keyboard.in.readInteger();

                 if (myInteger.intValue()<16)
                      System.out.print("too young to drive");
                 else if (myInteger.intValue()>75)
                      System.out.print("check with your doctor");
                 else
                      System.out.print("drive safely");


      }
}
```

3- (5 points) Use the Keyboard class and write a java program that reads one digit and displays the number in full letters otherwise displays “this is not a number”. The program doesn’t validate the input. You can use readString() method of the Keyboard class to read the input and the charAt(0) method of the String Class to get the first character of the input. The rest of the characters are ignored. Only digits are considered and you should output “zero” for 0, “one” for 1, “two” for 2, etc. and “nine” for 9.

```
public class question3 {
   /* outputs the one digit number in full letters */

      public static void main (String args[]) {

            String mySTring;
            char   myChar;

                myString=Keyboard.in.readString();
                myChar=myString.charAt(0);

                switch(myChar) {

                     case ‘0’: System.out.print(“zero”); break;

                     case ‘1’: System.out.print(“one”); break;

                     case ‘2’: System.out.print(“two”); break;

                     case ‘3’: System.out.print(“three”); break;

                     case ‘4’: System.out.print(“four”); break;

                     case ‘5’: System.out.print(“five”); break;

                     case ‘6’: System.out.print(“six”); break;

                     case ‘7’: System.out.print(“seven”); break;

                     case ‘8’: System.out.print(“eight”); break;

                     case ‘9’: System.out.print(“nine”); break;

                     default:
                        System.out.print(“this is not a number”);
                }


      }
}
```

Section 2: Multiple choice [30 points] (2 points each)

Circle exactly ONE choice as the best answer to each question.

1- Which one of these statements is true?

a) elementAt() changes the state of a vector;

b) A vector is bigger than an array;

c) addElement() changes the state of a vector;

d) A vector is a container that could contain objects and literals.

2- A, B, and C are variables of type *int*. Which of the following program segments would NOT have the effect of interchanging the values of the variables A and B?

a) C=A; A=B; B=C;

b) C=A; B=A; A=B;

c) A=A+B; B=A-B; A=A-B;

d) C=B; B=A; A=C;

3- Syntax errors are found by

a) The compiler

b) Blackbox testing

c) White box testing

d) The Java virtual machine;

e) The editor;

4- Suppose myVector is created and initialized, the following statement has an error. What error is it?

x=myVector.elementAt(i);

a) The name x is too short

b) There is no casting

c) It should be addElement()

d) We don't need the assignment

e) i should be an object

5- How many stars (*) are printed when the following Java code is executed?

```
for(i=1;i<=5;i++)
    for(j=i;j>0;j--)
         System.out.print("*");
```

a) 25

b) 15

c) 5

d) 14

e) different from the above numbers.

6- How many stars (*) are printed when the following Java code is executed?

```
i=0;
while (i<5) {
  j=i;
  while (j>=1)
   System.out.print("*");
}
```

a) 25

b) 15

c) 5

d) 14

e) there is an infinite loop.

7- How many stars (*) are printed when the following Java code is executed?

```
i=0; while (i<5) {
  j=i;  i+=1; while (j>=1) {
   System.out.print("*"); j+=1;}
}
```

a) 25

b) 15

c) 5

d) 14

e) more than the above numbers.

8- Where is the expression “throws Exception” used?

a) In the method declaration of a method that uses input/output streams;
b) In the beginning of a program that uses files;
c) Wherever in the code an error may occur;
d) When we add elements in a vector.

9- If a message is sent to an object variable that was declared but not bound to an object yet, what would happen?

a) The message is ignored;
b) A null pointer exception occurs during execution time;
c) The compiler would fix the problem;
d) The program continues and gives wrong results;
e) A syntax errors is identified by the compiler.

10- After sorting a collection of strings in alphabetical order, the string “memory” would be placed before the string “memorization” because:

a) The string “memory” is shorter than the string “memorization”;
b) The order is ascending order;
c) You need memory to memorize;
d) The statement is wrong. The string “memorization” is actually placed before the string “memory”.

11- What is not inherited in a subclass?

a) Static variables;
b) Instance variables;
c) Constructors;
d) Instance methods;
e) Static methods.

12- An object from a subclass can receive messages that an object from the superclass receives because:

a) Messages can always be received;
b) A subclass inherits from its superclass;
c) A superclass has more methods than its subclass.
d) The classes are chained;
e) The receiver decides.

13- Black box testing is a process to:

a) Eliminate syntax errors from Java programs;
b) Validate Java programs;
c) Make sure methods of a class behave as described in the specifications;
d) Test the black box class.
e) Describe hypothetical Java classes.
f) Make sure that all paths of the Java code are visited.

14- Given the collection: 5, 11, 23, 36, 41, 55, 62, 65, 72, 75. How many comparisons are done in a binary search that is looking for 41 then 95?

a) 1 and 4 respectively;
b) 2 and 8 respectively;
c) 5 and 10 respectively;
d) 1 and 7 respectively;
e) 2 and 10 respectively;
f) 5 and 6 respectively.

15- Which of the following statements is NOT true?

a) We can’t send a message to an array, even to determine its size.
b) Binary search can be used only if the collection is sorted.
c) Merge-sort works only with arrays.
d) Recursive methods often use more memory than non-recursive ones.

Section 3: Arrays and Vectors [10 points]

1- [10 points] Given an array A initialized with some integers, and B, a Vector, you are asked to write a Java program to copy the 5 last integers and the first 5 integers from A into B, the copy them back to A starting from the first index and print them on the screen. Fill in the missing Java statements. Remember, vectors contain objects. The integers given here are arbitrary. The values could be different. Also the number of integers in A is arbitrary and could be bigger.

```
import java.util.*;

public class copyA {
   /* Copy 5 integers from the right and 5 integers from the left
of an array of ints into a vector then copy them back to the
array and print the array*/

      public static void main (String args[]) {

            int A[]={23,34,12,5,64,7,91,45,65,35,70,4,93};
            Vector B;
            int k; Integer myInteger;


            B= new Vector();

            for (k=0; k<5; k++) {

                    myInteger=new Integer(A[k]);

                    B.addElement(myInteger);
            }

            for (k=A.length-5; k<A.length; k++) {

                    myInteger= new Integer(A[k]);

                    B.addElement(myInteger);
            }



            for (k=0; k<B.size(); k++) {

                    myInteger = (Integer) B.elementAt(k);

                    A[k] = myInteger.intValue();
            }



            for(k=0;k<A.length;k++) System.out.print(A[k]+" ");
            System.out.println();
      }
}
```

Section 4: File input/output and vectors [35 points]

You were hired by the financial department of the Banana Republic government to write a Java program that would calculate the income tax of all the inhabitants of the island. It was explained to you that the financial department has a file that contains the necessary information about the citizens of the island. Your program is supposed to read this file and first produce another file that contains on each line the name of a citizen followed by the income tax to be paid by that person; second, print on the screen the average gross income of all tax payers, the number of the tax payers, as well as the total income tax that the government is supposed to collect. The output file should be sorted by income tax from the highest to the lowest. It was also explained to you that the income tax is calculated as follows: there is a personal deductible which is a standard amount of $15,000 that is not taxable, and for each dependant (child for example) there is an extra $2000 that is not taxable. The tax rate is 20% of the net income (income after personal and dependant deductions) if the net income is less than $100,000 and 25% otherwise. For example if someone has $80,000 as gross income and has 2 dependants, the net income is $80,000 –$15,000-(2*2,000) = $61,000 and the income tax to be collected is 20% of 61,000 which is $12,200. When checking the input file, you discover that there are no identification numbers and that taxpayers in the Banana Republic are identified by name. The file has three lines per taxpayer. The first line contains the name, the second line contains the gross income, and the third line contains the number of dependants of that particular taxpayer. The gross income is always rounded to the dollar and doesn't have decimals. You also discover that a previous consultant already started writing the program. However, he didn't know Java well enough to finish the program. Now your easy task is just to fill in the missing Java statements. Note that all the local variables are already declared. You don't need to add any other variable.

Here is a sample output of the main program.

```
Average gross income per citizen: $45677.50
Number of taxpayers: 243576
Total income tax to be collected is $1218050503.20
```

Remember, the input file has three lines per taxpayer, and the output file which needs to be sorted by income tax, has one line per taxpayer. The names of both the input and output files are coded in the program and do not need to be read from the keyboard.

1- [2 points] Fill in the missing Java statements to finish the constructor of the BananaTax class.

```
Import java.util.*;
import java.io.*;
public class BananaTax {
   /* Each instance of this class represents a list of citizens
and the income tax they need to pay for the Banana Republic */

/* Instance variables */

Vector peopleNames;
Vector incomeTax;

/*static constants */

static final String citizensFile = "BananaPeople.txt";
static final String incometaxFile = "incomeTax.txt";
static final int personalDeductible = 15000;
static final int dependantDeductible = 2000;
static final double rangeLimit = 100000.0d;
static final double taxRate1 = 0.20d;
static final double taxRate2 = 0.25d;

/* constructor */

     public BananaTax() {
     /* initializes the vectors of names and tax to be empty */


          this.peopleNames=new Vector();

          this.incomeTax=new Vector();


     }
```

2- [6 points] Fill in the missing Java statements to finish the two static methods: *calculateTax* and *convert*. *calculateTax* receives an income amount and a number of dependants then calculates and returns the income tax based on the procedure described above. The method returns a Double object with the calculated income tax as value. Use the constants (static final variables) defined in the beginning of the class file to do your calculations.

convert receives a String of digits (example "12345") and converts it into an int (example 12345). This can be done by taking digit by digit from the left to the right, checking its value, and adding this value to the previously converted digits multiplied by 10. Remember, the string does not contain decimals and contains only digits (no need to validate). You can get a character from a String using the charAt(int position) method of the String class.

```
      /*static methods */

static public Double calculateTax(int income, int dependants) {
      /* Calculates the income tax given an income and a number
         of dependants */

          double myTax;
          Double taxToPay;
          int myIncome;

          myIncome = income – personalDeductible –
                    (dependants*dependantDeductible);


            if (myIncome >= rangeLimit)

                 myTax=taxRate2*myIncome;

            else

                 myTax=taxRate1*myIncome;


          taxToPay=new Double(myTax);
          return taxToPay;
      }

static public int convert (String aString) {
      /* Converts a String into an int by checking character by
         character.
         The input String is assumed to be only digits ‘0’..‘9’*/

         int i;      int result=0;
         int digit;  char ch;


         for (i=0;i<aString.length();i++) {

            ch=aString.charAt(i);
                 switch (ch) {
                      case '1': digit=1;break;
                      case '2': digit=2;break;
                      case '3': digit=3;break;
                      case '4': digit=4;break;
                      case '5': digit=5;break;
                      case '6': digit=6;break;
                      case '7': digit=7;break;
                      case '8': digit=8;break;
                      case '9': digit=9;break;
                      default : digit=0;
                 }
                 result=(result*10)+digit;
         }

         return result;

      }
```

3- [6 points] Fill in the missing Java statements to finish the readFileCompute() method that reads citizens' information from the input file, computes the income tax, and adds the names and income tax in the appropriate vectors. Use the static method *convert* defined above to convert the strings containing the incomes and dependants into *int* integers. Use the *readLine()* method of the BufferedReader class. You know you have reached the end of file when readline() returns null. Remember that you have three lines per citizen. Use the static method *calculateTax()* when needed.

```
     public double readFileCompute()    throws Exception  {

     /* Reads the info from input file, calculates income tax
and fills my vectors */

          File aFile;
          FileInputStream      inputStream;
          InputStreamReader    aReader;
          BufferedReader       aBufferedReader;

          Double myTax;      int myIncome;
          int myDependants;  double myTotal=0.0d;

          aFile= new File(citizensFile);
          inputStream = new FileInputStream(aFile);
          aReader = new InputStreamReader(inputStream);
          aBufferedReader = new BufferedReader(aReader);

          String aString;


          aString = aBufferedReader.readLine();

          while (aString != null) {
               this.peopleNames.addElement(aString);
               aString = aBufferedReader.readLine();
               myIncome=BananaTax.convert(aString);
               aString = aBufferedReader.readLine();
               myDependants=BananaTax.convert(aString);
               myTax=
                  BananaTax.calculateTax(myIncome,myDependants);
               this.incomeTax.addElement(myTax);
               myTotal = myTotal + myIncome;
               aString = aBufferedReader.readLine();
          }








          inputStream.close();
          return myTotal;
     }
```

4- [6 points] Fill in the missing Java statements to finish the *getLargest()* and *exchange()* methods that are used in the already written *sort()* method. The sort() method uses a selection similar to the one you saw in class. In order to change an object in a vector, use the method *setElementAt(Object myObject, int index)* from the Vector class that changes the object at position index by the given object myObject.

```
public void sort() {
     /* Sorts my vectors using a selection sort. The order is
        from the largest income tax to the smallest */

     int        index;
     int        largestIndex;

     for (index = 0; index < (this.incomeTax.size()-1); index++){
                largestIndex = this.getLargest(index);
                this.exchange(index, largestIndex);
     }
}




private int getLargest(int start) {
     /* Returns the index of the largest element in my incomeTax
        vector the index of which is greater than or equal to the
        given start index. */

          int        largestIndex;
          int        index;
          Double current, largest;



     largestIndex = start;

     for(index=start+1; index<this.incomeTax.size();index++){
          current=(Double)this.incomeTax.elementAt(index);
          largest=
              (Double)this.incomeTax.elementAt(largestIndex);

          if (current.doubleValue() > largest.doubleValue())
                    largestIndex = index;
     }

     return largestIndex;







}
```

```
private void exchange(int i, int j) {
        /*Exchanges the elements of my vectors with the given two indexes*/

                String tempString;
                Double tempDouble;


        tempString = (String)this.peopleNames.elementAt(i);

        this.peopleNames.setElementAt((String)this.peopleNames.elem
entAt(j),i);

        this.peopleNames.setElementAt(tempString,j);

        tempDouble = (Double)this.incomeTax.elementAt(i);

        this.incomeTax.setElementAt((Double)this.incomeTax.elementA
t(j),i);

        this.incomeTax.setElementAt(tempDouble,j);


}
```

5- [3 points] Fill in the missing Java statements to finish the *getTotal*() that calculates and returns the sum of all income taxes to be paid to the government. Remember, they are stored in the Vector *incomeTax*.

```
public int getPopulation() {
    /*Return the number of citizens after they were read from
          the input file and saved in my state*/

         return this.peopleNames.size();
}

public double getTotal(){
        /*Return the sum of all income tax saved in my state*/

        int i;
        double total=0.0d;
        Double current;



        for (i=0;i<this.incomeTax.size();i++){
              current=(Double)this.incomeTax.elementAt(i);
                total+=current.doubleValue();
        }

        return total;





}
```

6- [6 points] Fill in the missing Java statements to finish the writeFile() method that writes the names and the income tax due to the output file. Each line in the file contains a name and the corresponding income tax, separated by a space.

```
    public void writeFile()   throws Exception {

    /* Writes names and income taxes from vectors into output
       file.*/

        File aFile;
        FileOutputStream outputStream;
        aFile= new File(incometaxFile)

        int i;
        String aString;
        Double aDouble;


        outputStream = new FileOutputStream(aFile);
        PrintStream aPrintStream;
        aPrintStream = new PrintStream(outputStream);



        for(i=0; i<this.peopleNames.size();i++) {
             aString=(String)this.peopleNames.elementAt(i);
             aDouble=(Double)this.incomeTax.elementAt(i);
             aPrintStream.print(aString);
             aPrintStream.print(" ");
             aPrintStream.println(aDouble);
        }









        outputStream.close();
    }
```

7- [6 points] Write the main program that uses the BananaTax class to read the citizens file, compute the income taxes, sort the citizens from the highest income tax to pay to the lowest, and outputs the income taxes sorted in a file. The program should also print on the screen the necessary information as described in the sample of output given above. For the sake of simplicity, do not consider the number of digits after the decimal point. Just print a double as it is. Note that all necessary local variables for the main method were declared. You do not necessarily need other variables.

```
public class myProgram {


     public static void main(String args[]) throws Exception {
     /* Program statements go here. */

       BananaTax citizens;
       double myTotalIncome;
       double myAverage;
        int taxPayers;

       citizens = new BananaTax();

       myTotalIncome=citizens.readFileCompute();
       taxPayers=citizens.getPopulation();
       myAverage=myTotalIncome/taxPayers;



        citizens.sort();
        citizens.writeFile();

        System.out.print("Average gross income per citizen: ");
        System.out.println(myAverage);
        System.out.print("Number of taxpayers: ");
        System.out.println(taxPayers);
        System.out.print("Total income tax to be collected is ");

        System.out.println(citizens.getTotal());








     }
}
```

Section 5: Tracing code [15 points]

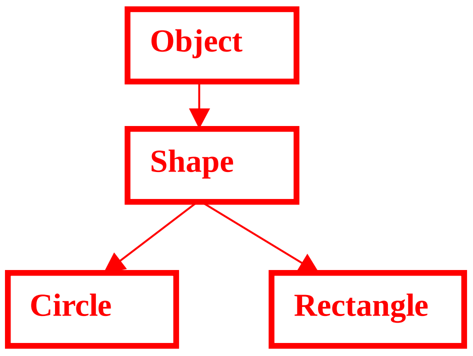
Consider the following Java classes:

```
public class Shape extends Object {

/* instance variables */
      private int xPos;
      private int yPos;

      public Shape() {
            this.xPos = 0;
            this.yPos = 0;
      }
      public Shape(int x, int y) {
            this.xPos = x;
            this.yPos = y;
      }
      public int getX() {
            return this.xPos;
      }
      public int getY() {
            return this.yPos;
      }
      public void move(int a, int b) {
            this.xPos += a;
            this.yPos += b;
      }
      public String toString() {
            return “(” + this.xPos + “,” + this.yPos + “)”;
      }
}

public class Circle extends Shape {

/* instance variable */
      protected int radius;

      public Circle() {
            this.radius = 2;
      }

      public Circle(int x, int y, int z) {
            super(y,z);
            this.radius = x;
      }

      public Circle(int x, int y) {
            this();
            super(x,y);
      }

      public double area() {
            return 3.14d * this.radius * this.radius;
      }

      public String toString() {
            return “Circle:” + super.toString() + “ R=”+this.radius;
      }
}
```

```
public class Rectangle extends Shape {

/* instance Variables */
      protected int height;
      protected int length;

      public Rectangle() {
             this.height=this.length=2;
      }

      public Rectangle(int x, int y, int ht, int len){
             super(x,y);
             this.height=ht;
             this.length=len;
      }

      public Rectangle(int x, int y){
             this();
             super(x,y);
      }

      public void stretch (int horizontal, int vertical) {
             this.length += horizontal;
             this.height += vertical;
      }

      public int area() {
             return this.length * this.height;
      }

      public String toString() {
            return "Rectangle: (" + this.getX() + "," +
                                    this.getY() + "," +
                                    (this.getX()+this.length) + "," +
                                    (this.getY()+this.height) + ")";
      }

}
```

Note that this example has a non-linear chaining of constructors, which is normally not allowed in Java. It is used in this question for the sake of the tracing exercise. The program above would not compile with CodeWarrior because of the crossed chaining.

a) [2 points] Draw the inheritance tree of these classes.



b) [2 points] What is the output of the program segment:

Shape s; s=new Shape(10,15); s.move(5,6);
System.out.println(s.toString());

Output

```
(15, 21)
```

c) [3 points] What is the output of the program segment:

Circle c; c=new Circle(); c.move(5,6);
System.out.println(c.toString());

Output

```
Circle: (5,6) R=2
```

d) [4 points] What is the output of the program segment:

Circle c1; Circle c2; c1=new Circle(2,2,4); c2=new Circle(2,2);
System.out.println(c1.toString()); System.out.println(c2.toString());

Output

```
Circle: (2,4) R=2
Circle: (2,2) R=2
```

e) [4 points] What is the output of the program segment:

Rectangle r; r=new Rectangle(10,15);
System.out.println(r.toString());
r.move(5,6); System.out.println(r.toString());
r.stretch(20,40); System.out.println(r.toString());

Output

```
Rectangle: (10,15,12,17)
Rectangle: (15,21,17,23)
Rectangle: (15,21,37,63)
```