Institute of Business Administration (IBA)
CSE:142 Object Oriented Programming Techniques
TASK #2 (10 marks)

Due Date: Monday, 20th December 2010, 08:00 am

Instructions:

- *Create separate project for every question and send each project folder in zip form through email. In any other form, the submission will not be accepted.*
- *In case of plagiarism, zero marks will be given to those who are involved.*
- *There will a quiz from the Task 2 on Sunday 26th December.*

Q1. Define a class called Fraction. This class is used to represent a ratio of two integers. Include mutator functions that allow the user to set the numerator and the denominator. Also include a member function that returns the value of numerator or denominator as a double. Include an additional member function that outputs the value of the fraction reduced to lowest terms (e.g., instead of outputting 20/60 the method should output 1/3). This will require finding the greatest common divisor for the numerator and denominator, then dividing both by that number. Embed your class in a test program.

Q2. Define a class called Odometer that will be used to track fuel and mileage for an automobile. The class should have member variables to track the miles driven and the fuel efficiency of the vehicle in miles per gallon. Include a mutator function to reset the odometer to zero miles, a mutator function to set the fuel efficiency, a mutator function that accepts miles driven for a trip and adds it to the odometer's total, and an accessor method that returns the number of gallons of gasoline that the vehicle has consumed since the odometer was last reset.

Q3 Write a grading program for a class with the following grading policies:
a. There are three quizzes, each graded on the basis of 10 points.
b. There is one midterm exam, graded on the basis of 100 points.
c. There is one final exam, graded on the basis of 100 points.

The final exam counts for 40 percent of the grade. The midterm counts for 35 percent of the grade. The three quizzes together count for a total of 25 percent of the grade. (Do not forget to convert the quiz scores to percentages before they are averaged in.) Any grade of 90 or more is an A, any grade of 80 or more (but less than 90) is a B, any grade of 70 or more (but less than 80) is a C, any grade of 60 or more (but less than 70) is a D, and any grade below 60 is an F. The program should read in the student's scores and output the student's record, which consists of three quiz scores and two exam scores as well as the student's overall numeric score for the entire course and final letter grade.

Define and use a class for the student record. The class should have instance variables for the quizzes, midterm, final, overall numeric score for the course, and final letter grade. The overall numeric score is a number in the range 0 to 100, which represents the weighted average of the student's work. The class should have methods to compute the overall numeric grade and the final letter grade. These last methods should be void methods that set the appropriate instance variables. Your class should have a reasonable set of accessor and mutator methods, an equals method, and a toString method, whether or not your program uses them. You may add other methods if you wish.

Q4. Write a Temperature class that has two instance variables: a temperature value (a floating-point number) and a character for the scale, either C for Celsius or F for Fahrenheit. The class should have four constructor methods: one for each instance variable (assume zero degrees if no value is specified and Celsius if no scale is specified), one with two parameters for the two instance variables, and a no-argument constructor (set to zero degrees Celsius). Include the following:

(1) two accessor methods to return the temperature—one to return the degrees Celsius, the other to return the degrees Fahrenheit— use the following formulas to write the two methods, and round to the nearest tenth of a degree:
degreesC = 5(degreesF - 32)/9
degreesF = (9(degreesC)/5) + 32

(2) three mutator methods: one to set the value, one to set the scale (F or C), and one to set both;
(3) three comparison methods: an equals method to test whether two temperatures are equal, one method to test whether one temperature is greater than another, and one method to test whether one temperature is less than another (note
that a Celsius temperature can be equal to a Fahrenheit temperature as indicated by the above formulas); and
(4) a suitable toString method. Then write a driver program (or programs) that tests all the methods. Be sure to use each of the constructors, to include at least one true and one false case for each of the comparison methods, and to test at least the following temperature equalities: 0.0 degrees C =32.0 degrees F= –40.0 degrees C = –40.0 degrees F, and 100.0 degrees C = 212.0 degrees F.

Q5. Define a class called Fraction. This class is used to represent a ratio of two integers. Include mutator functions that allow the user to set the numerator and the denominator. Also include a method that displays the fraction on the screen as a ratio (e.g., 5/9). This method does not need to reduce the fraction to lowest terms.
Include an additional method, equals, that takes as input another Fraction and returns true if the two fractions are identical and false if they are not. This method should treat the fractions reduced to lowest terms; that is, if one fraction is 20/60 and the other is 1/3, then the method should return true.
Embed your class in a test program that allows the user to create a fraction. Then the program should loop repeatedly until the user decides to quit. Inside the body of the loop, the program should allow the user to enter a target fraction into an anonymous object and learn whether the fractions are identical.

Q6. Define a class named Money whose objects represent amounts of U.S. money. The class should have two instance variables of type int for the dollars and cents in the amount of money. Include a constructor with two parameters of type int for the dollars and cents, one with one constructor of type int for an amount of dollars with zero cents, and a no-argument constructor. Include the methods add for addition and minus for subtraction of amounts of money. These methods should be static methods and should each have two parameters of type Money and return a value of type Money. Include a reasonable set of accessor and mutator methods as well as the methods equals and toString. Write a test program for your class.

Part Two: Add a second version of the methods for addition and subtraction. These methods should have the same names as the static version but should use a calling object and a single argument. For example, this version of the add method (for addition) has a calling object and one argument. So m1.add(m2) returns the result of adding the Money objects m1 and m2. Note that your class should have all these methods; for example, there should be two methods named add.
Alternate Part Two (If you want to do both Part Two and Alternate Part Two, they must be two classes. You cannot include the methods from both Part Two and Alternate Part Two in a single class. Do you know why?): Add a second version of the methods for addition and subtraction. These methods should have the same names as the static version but should use a calling object and a single argument. The methods should be void methods. The result should be given as the changed value of the calling object. For example, this version of the add method (for addition) has a calling object and one argument. Therefore,

```
m1.add(m2);
```

changes the values of the instance variables of m1 so they represent the result of adding m2 to the original version of m1. Note that your class should have all these methods; for example, there should be two methods named add.

Q7. Write a static method called deleteRepeats that has a partially filled array of characters as a formal parameter and that deletes all repeated letters from the array. Because a partially filled array requires two arguments, the method should actually have two formal parameters: an array parameter and a formal parameter of type int that gives the number of array positions used. When a letter is deleted, the remaining letters are moved one position to fill in the gap. This creates empty positions at the end of the array so that less of the array is used. Because the formal parameter is a partially filled array, a second formal parameter of type int should tell how many array positions are filled. This second formal parameter cannot be changed by a Java method, so have the method return the new value for this parameter. For example, consider the following code:

```
char a[10];
a[0] = 'a';
a[1] = 'b';
a[2] = 'a';
a[3] = 'c';
int size = 4;
size = deleteRepeats(a, size);
```

After this code is executed, the value of a[0] is 'a', the value of a[1] is 'b', the value of a[2] is 'c', and the value of size is 3. (The value of a[3] is no longer of any concern, because the partially filled array no longer uses this indexed variable.) You may assume that the partially filled array contains only lowercase letters. Write a suitable test program for your method.

Q8. Write a program that will allow two users to play tic-tac-toe. The program should ask for moves alternately from player X and player O. The program displays the game positions as follows:

```
1 2 3
4 5 6
7 8 9
```

The players enter their moves by entering the position number they wish to mark. After each move, the program displays the changed board. A sample board configuration is

```
X X O
4 5 6
O 8 9
```

Q9. Write a program to assign passengers seats in an airplane. Assume a small airplane with seat numberings as follows:

```
1 A B C D
2 A B C D
3 A B C D
4 A B C D
5 A B C D
6 A B C D
7 A B C D
```

The program should display the seat pattern, with an 'X' marking the seats already assigned. For example, after seats 1A, 2B, and 4C are taken, the display should look like:

```
1 X B C D
2 A X C D
3 A B C D
4 A B X D
5 A B C D
6 A B C D
7 A B C D
```

After displaying the seats available, the program should prompt for the seat desired, the user can type in a seat, and then the display of available seats should be updated. This continues until all seats are filled or until the user signals that the program should end. If the user types in a seat that is already assigned, the program should say that that seat is occupied and ask for another choice.

Q10. Define a class named Payment that contains a member variable of type double that stores the amount of the payment and appropriate accessor and mutator methods. Also create a method named paymentDetails that outputs an English sentence to describe the amount of the payment.

Next, define a class named CashPayment that is derived from Payment. This class should redefine the paymentDetails method to indicate that the payment is in cash. Include appropriate constructor(s). Define a class named reditCardPayment that is derived from Payment. This class should contain member variables for the name on the card, expiration date, and credit card number. Include appropriate constructor(s). Finally, redefine the paymentDetails method to include all credit card information in the printout.

Create a main method that creates at least two CashPayment and two CreditCardPayment objects with different values and calls paymentDetails for each.

Q11. Define a class named Document that contains a member variable of type String named text that stores any textual content for the document. Create a method namedtoString that returns the text field and also include a method to set this value.

Next, define a class for Email that is derived from Document and includes member variables for the sender, recipient, and title of an email message. Implement appropriate accessor and mutator methods. The body of the email message should be stored in the inherited variable text. Redefine the toString method to concatenate all text fields.

Similarly, define a class for File that is derived from Document and includes a member variable for the pathname. The textual contents of the file should be stored in the inherited variable text. Redefine the toString method to concatenate all text fields.

Finally, create several sample objects of type Email and File in your main method. Test your objects by passing them to the following subroutine that returns true if the object contains the specified keyword in the text property.

```
public static boolean ContainsKeyword(Document docObject, String keyword)
{
if (docObject.toString().indexOf(keyword,0) >= 0)
return true;
return false;
}
```

Q12. Create a class called Vehicle that has the manufacturer's name (type String), number of cylinders in the engine (type int), and owner (type Person given below). Then, create a class called Truck that is derived from Vehicle and has the following additional properties: the load capacity in tons (type double since it may contain a fractional part) and towing capacity in pounds (type int). Be sure your class has a reasonable complement of constructors, accessor and mutator methods, and suitably defined equals and toString methods. Write a program to test all your methods. The definition of the class Person is below. Completing the definitions of the methods is part of this programming project.

```
public class Person
{
private String name;
public Person()
{...}
public Person(String theName)
{...}
public Person(Person theObject)
{...}
public String getName()
{...}
public void setName(String theName)
{...}
public String toString()
{...}
public boolean equals(Object other)
{...}
}
```

Q13. Create a class named Movie that can be used with your video rental business. The Movie class should track the Motion Picture Association of America (MPAA) rating (e.g., Rated G, PG-13, R), ID Number, and movie title with appropriate accessor and mutator methods. Also create an equals() method that overrides Object's equals() method, where two movies are equal if their ID number is identical. Next, create three additional classes named Action, Comedy, and Drama that are derived from Movie. Finally, create an overridden method named calcLateFees that takes as input the number of days a movie is late and returns the late fee for that movie. The default late fee is $2/day. Action movies have a late fee of $3/day, comedies are $2.50/day, and dramas are $2/day. Test your classes from a main method.

Q14. Write a program that calculates the average of *N* integers. The program should prompt the user to enter the value for *N* and then afterward must enter all *N* numbers. If the user enters a non-positive value for *N,* then an exception should be thrown (and caught) with the message "*N* must be positive." If there is any exception as the user is entering the *N* numbers, an error message should be displayed and the user prompted to enter the number again.

Q15. Here is a snippet of code that inputs two integers and divides them:

```
Scanner scan = new Scanner(System.in);
int n1, n2;
double r;
n1 = scan.nextInt();
n2 = scan.nextInt();
r = (double) n1 / n2;
```

place the snippet of code inside a method. The method should be named ReturnRatio and read the input from the keyboard and throw different exceptions for divide by zero or input mismatch between text and an integer. Create your own exception class for divide by zero. Invoke ReturnRatio from
your main method and catch the exceptions in main. The main method should invoke the ReturnRatio method again if any exception occurs.

Q16. Write a program that can serve as a simple calculator. This calculator keeps track of a single number (of type double) that is called result and that starts out as 0.0. Each cycle allows the user to repeatedly add, subtract, multiply, or divide by a second number.

The result of one of these operations becomes the new value of result. The calculation ends when the user enters the letter R for "result" (either in uppercase or lowercase). The user is allowed to do another calculation from the beginning as often as desired.

The input format is shown in the following sample dialogue. If the user enters any operator symbol other than +, ↕, *, or /, then an UnknownOperatorException is thrown and the user is asked to reenter that line of input. Defining the class UnknownOperatorException is part of this project.

```
Calculator is on.
result = 0.0
+5
result + 5.0 = 5.0
new result = 5.0
*2.2
result * 2.2 = 11.0
updated result = 11.0
% 10
% is an unknown operation.
Reenter, your last line:
* 0.1
result * 0.1 = 1.1
updated result = 1.1
r
Final result = 1.1
Again? (y/n)
```

```
yes
result = 0.0
+10
result + 10.0 = 10.0
new result = 10.0
/2
result / 2.0 = 5.0
updated result = 5.0
r
Final result = 5.0
Again? (y/n)
N
End of Program'
```

Q17. Write a program that takes its input from a text file of strings representing numbers of type double and outputs the average of the numbers in the file to the screen. The file contains nothing but strings representing numbers of type double, one per line.

Q18. Listed below is the skeleton for a class named InventoryItem. Each inventory item has a name and a unique ID number:

```
class InventoryItem
{
private String name;
private int uniqueItemID;
}
```

Flesh out the class with appropriate accessors, constructors, and mutatators. The uniqueItemID's are assigned by your store and can be set from outside the InventoryItem class—your code does not have to ensure that they are unique. Next, modify the class so that it implements the Comparable interface. The compareTo() method should compare the uniqueItemID's; e.g., the InventoryItem with item ID 5 is less than the InventoryItem with ID 10. Test your class by creating an array of sample InventoryItem's and sort them using a sorting method that takes as input an array of type Comparable.

Q19. In the sport of diving, seven judges award a score between 0 and 10, where each score may be a floating-point value. The highest and lowest scores are thrown out and the remaining scores are added together. The sum is then multiplied by the degree of difficulty for that dive. The degree of difficulty ranges from 1.2 to 3.8 points. The total is then multiplied by 0.6 to determine the diver's score. Write a computer program that inputs a degree of difficulty and seven judges' scores and outputs the overall score for that dive. The program should use an ArrayList of type Double to store the scores.

Q20. Write a generic class, MyMathClass, with a type parameter T where T is a numeric object type (e.g,. Integer, Double, or any class that extends java.lang.Number). Add a method named standardDeviation that takes an ArrayList of type T and returns as a double the standard deviation of the values in the ArrayList. Use the doubleValue() method in the Number class to retrieve the value of each number as a double. Refer to Programming Project 6.2 for a definition of computing the standard deviation. Test your method with suitable data. Your program should generate a compile-time error if your standard deviation method is invoked on an ArrayList that is defined for non-numeric elements (e.g., Strings).