SIM UNIVERSITY

ICT131

Examination – January Semester 2017

Introductory Programming and Object Oriented Concepts Using Java

Tuesday, 16 May 2017 **1:00 pm – 3:00 pm**

Time allowed: 2 hours

INSTRUCTIONS TO STUDENTS:

1. This examination contains **FOUR (4)** questions and comprises **TEN (10)** printed pages (including cover page and appendix A).
2. You must answer **ALL** questions.
3. This is a Closed Book examination.
4. All answers must be written in the answer book.
5. Appendix A contains some Java API which you might need.

At the end of the examination
Please ensure that you have written your examination number on each answer book used.

Failure to do so will mean that your work cannot be identified.

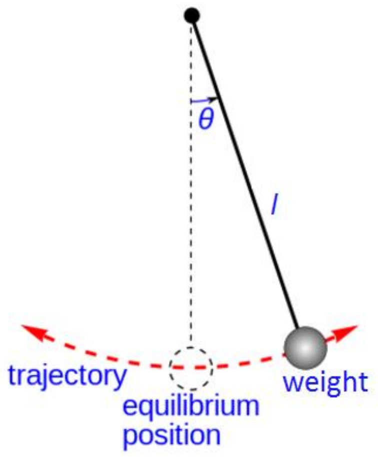
If you have used more than one answer book, please tie them together with the string provided.

THE UNIVERSITY RESERVES THE RIGHT NOT TO MARK YOUR SCRIPT IF YOU FAIL TO FOLLOW THESE INSTRUCTIONS.

Answer all questions. (Total 100 marks)

Question 1

(a) Figure Q1(a) shows a pendulum which is a suspended weight that swings freely.



<u>Figure Q1(a)</u>

When the pendulum swings with a sufficiently small angle θ, then the approximated time taken T to make one complete swing is calculated using this formula:

$$T \approx 2\pi\sqrt{\frac{l}{g}}$$

where l is the length of the pendulum and g is the local acceleration due to gravity.

Use the Java-supplied classes and cite the Java documentation to write a Java program that

- accepts **TWO (2)** input arguments from the command line and converts both to whole numbers, the first input argument is for l and the second is for g,
- uses the Java library value of π and the appropriate method for square root to calculate T, and
- displays T with 2 decimal places.

Shown below is an example output when the inputs of *l* and *g* are `"40"` and `"10"` respectively:

```
Time taken will approximately be 12.57 given length
= 40 and g = 10
```

Note: Do NOT hardcode the values for *l* and *g* in the formula.

(10 marks)

(b) Examine the programs Figure Q1(b)(i) to Figure Q1(b)(iv) and outline the output for each of them.

```
public class Q1bi {
    public static void main (String [] args){
        int [] n = { 2, 3, 4, 5, 6, 7};
        methodX(n[1], n[4]);
        System.out.println("main: " + n[1] + " " + n[4]);
    }

    public static void methodX (int a1, int a2){
        int diff = a1 - a2;
        a1 = a1 - diff;
        a2 = a2 + diff;
        System.out.println("methodX: " + a1 + " " + a2);
    }
}
```

Figure Q1(b)(i)

```
public class Q1bii {
    public static void main (String [] args){
        int [] n = { 2, 3};
        methodX(n);
        for (int i = 0; i < n.length; i++) {
           System.out.println("main: " +n[i]);
        }
    }

    public static void methodX (int [] x){
        for (int i = x.length -1; i > 0; i--) {
           x[i] = x[i-1];
        }

        for (int i = 0; i < x.length; i++) {
           System.out.println("methodX: " +x[i]);
        }
    }
}
```

Figure Q1(b)(ii)

```
public class Q1biii{
    public static void main (String [] args){
        int a = 2, b = 3, c = 5;
        System.out.println(++b / a % c);
        System.out.println(a++ + b * c);
        System.out.println(a <= b || c <= b);
        System.out.println(a < b && !(c > b));
    }
}
```

Figure Q1(b)(iii)

```
public class Q1biv{
    public static void main (String [] args){
        int a = 2, b = 3, c = 5;
        for (a = b; a <= c; a++);
        System.out.println(a + " " + b  + " " + c);
    }
}
```

Figure Q1(b)(iv)

(15 marks)

Question 2

Outline the control structures of structured programming in **EACH** of the scenarios described in Table Q2(a), section (b) and section (c).

You are expected to:

- **select ONE (1) selection** and **ONE (1) repetition** control structures from the following lists:
  - selection: `if` or `switch` or `if-else`
  - repetition: `for` or `while` or `do-while`

  Note that if a listed control structure has been chosen for a scenario, it can NOT be reused in the other scenarios.
- **develop a program segment** using the chosen control structures for the scenario.
- **If there is any need to read user input,** assume that a `Scanner` object has been created and is referenced with the variable `sc`. You may **include any other variables you need.**

(a) Write a short program to display the total cost of a pizza order that consists of one or more personal size pizzas from **THREE (3)** different categories as follows:

Category	Pizza (number)	Cost per Personal Pizza
Classic Favourite	Hawaiian(1) Mighty Meat-T(5) Vege Delight (6)	$9.95
House Speciality	Seafood Ahoy(2) Cheesy Awesome (3)	$12.95
Exquisite Taste	Wild About Mushroom (4) Garden Herbs (7)	$14.95

Table Q2(a)

Your program segment should allow a customer to order his pizzas by repeatedly entering the numbers 1 to 7 representing the different types of pizzas as shown in the middle column of Table Q2(a). The customer enters 0 to end the order. Note that the order can be ended only if at least one pizza has been ordered. Thus, if the customer enters a 0 before ordering any pizza, simply allow the user to re-enter a pizza number.

Any other pizza number (except for 1-7 and 0 to end) entered is an invalid entry and hence display the error message `Invalid Pizza Number`. After the customer has completed his order, your program segment will display the number of pizzas ordered and the total cost of all the pizzas in the order.

Assume that the variables `pizza, count` and `cost` have been declared.

- `pizza` stores the pizza number that the customer enters.
- `count` stores the number of pizzas ordered thus far and it has been initialised to 0.
- `cost` stores the cost of the order thus far and it has been initialised to 0.

(9 marks)

(b) Write a program segment that allows a school administrator to repeatedly enter a student number, each belonging to an absent student.

A student number is a positive number. If the school administrator enters a negative number (an invalid student number), display the error message `Invalid Student Number!`

When the school administrator has completed her data entry, she ends by entering `0` for a student number. The program segment then displays how many students are absent.

Assume that the variables `absentCount` and `studentNumber` have been declared.

- `absentCount` stores how many students are absent and it has been initialised to 0.
- `studentNumber` stores the student number that the administrator enters.

(8 marks)

(c) Write a program segment that allows a user to guess a number (1-10) that has been randomly generated.

The user is given a maximum of 3 attempts. If the user has guessed the number correctly before the 3rd attempt, do not make the user guess again.

After the user has guessed correctly or has exhausted all 3 attempts, the program segment displays whether the user has managed to guess correctly and the number of attempts made.

Assume that the variables `computerNumber`, `guess` and `correct` have been declared.

- `computerNumber` stores the generated random number.
- `guess` stores the user guess.
- `correct` stores `true` if the user has guessed correctly, and stores `false` if the user has not been able to guess the number in 3 attempts. `correct` has been initialised to `false`.

(8 marks)

Question 3

Develop a Java program using the control structures of structured programming on an array data structure for inventory application that manages products and their quantity currently available (quantity-on-hand or qoh). The inventory contains a maximum of 100 products and is implemented using

- one array `product` to record the name of each product.
- one array `qoh` to record the quantity-on-hand which is a whole number.

Write the following Java methods:

(a) A method `searchAProduct` to search for a product in the inventory.
- The method accepts the two arrays, the name of the product to search, and an integer `count` which is the number of products already in the inventory.
- The method returns -1 if the product is not located or if the inventory is empty.
- Otherwise, the method returns the array index of where the product is located.

(7 marks)

(b) A method `addAProduct` to add a product into the inventory.

- The method accepts the two arrays and an integer `count` which is the number of products already in the inventory.
- If the count of products is already the maximum number of products that can be stored in the inventory, the method prints the error message: `No more products can be added!`
- Otherwise, the method prompts the user for a name and quantity of the product.
- The method also calls the method `searchAProduct` method to search for the given product name.
  - ➢ If the name is located, the method prints the error message: `This product already exists in the inventory!`
  - ➢ If the name is not located, the method stores the new product details in the appropriate arrays and increments `count` as one more product has been added.
- This method returns the value of `count`.

(9 marks)

(c) A method `increaseProductQOH` to increase the quantity-on-hand of one product.

- The method accepts the two arrays and an integer count which is the number of products already in the inventory.
- The method returns false if the inventory is empty.
- Otherwise, the method prompts the user for the product name and quantity to increase by.
- If the product does not exist, the method returns false.
- Otherwise, the method adds the quantity to the quantity-on-hand of the product and returns true.

(9 marks)

Question 4

You are tasked to write programs to develop a prototype using the object-oriented programming approach for LessThanTwoDollars, an online store that sells packed items for less than $2. Details of each item sold include: name, price, pieces per pack and availability. Two examples of items sold are shown in Table Q4.

Name	Price	Pieces per Pack	Availability
DIY Cardboard Finger Puppets	$1.50	4	true
Neon Colour Skipping Rope	$1.00	1	true

Table Q4

Write a Java class that models an item. This class should have the following:

(a) Suitable instance variables.

(2 marks)

(b) A constructor with the appropriate actions on the instance variables. The default value for availability is true.

(5 marks)

(c) The get and set methods for the instance variables for price, and only the get method for availability.

(6 marks)

(d) A `changeAvailability ()` method which changes the item's availability. If currently it is not available, then change it to available, and vice versa.

(2 marks)

(e) A `pricePerPiece ()` method which returns the price of one piece in a pack. Note that a pack may contain one or more pieces.

(2 marks)

(f) A `toString()` method that returns a string containing the values of the instance variables with descriptions.

(2 marks)

(g) Write a test program to exercise the class written in parts (a) to (e). The test program should perform the following:

- Create **TWO (2)** items as shown in the Table Q4(a) and **store the objects in an array**.

Use **ONE (1) for** loop and
- apply the `changeAvailability ()` method to both items.
- decrease the prices of both items by 10 cents.
- print the details of both items after the changes. The expected output is shown here:

```
DIY Cardboard Finger Puppets   Price: $ 1.40 Pieces Per Pack: 4 Available: false
Neon Colour Skipping Rope      Price: $ 0.90 Pieces Per Pack: 1 Available: false
```

(6 marks)

Appendix A

Class Double

Modifier and Type	Method and Description
`double`	`doubleValue()` Returns the `double` value of this `Double` object.
`static double`	`parseDouble(String s)` Returns a new `double` initialized to the value represented by the specified `String`, as performed by the `valueOf` method of class `Double`.

Class Integer

Modifier and Type	Method and Description
`int`	`intValue()` Returns the value of this `Integer` as an `int`.
`static int`	`parseInt(String s)` Parses the string argument as a signed decimal integer.
`static Integer`	`valueOf(int i)` Returns an `Integer` instance representing the specified `int` value.

Class Math

Modifier and Type	Field and Description
`static double`	**`PI`** The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.
Modifier and Type	**Method and Description**
`static double`	`sqrt(double a)` Returns the correctly rounded positive square root of a double value.

Class Scanner

Modifier and Type	Method and Description
`double`	`nextDouble()` Scans the next token of the input as a`double`.
`int`	`nextInt()` Scans the next token of the input as an `int`.
`String`	`nextLine()` Advances this scanner past the current line and returns the input that was skipped.

Class String

Modifier and Type	Method and Description
`char`	`charAt(int index)` Returns the `char` value at the specified index.
`int`	`compareTo(String anotherString)` Compares two strings lexicographically.
`int`	`compareToIgnoreCase(String str)` Compares two strings lexicographically, ignoring case differences.
`String`	`concat(String str)` Concatenates the specified string to the end of this string.
`int`	`indexOf(String str)` Returns the index within this string of the first occurrence of the specified substring.
`boolean`	`equals(Object anObject)` Compares this string to the specified object.
`boolean`	`equalsIgnoreCase (String anotherString)` Compares this String to another String, ignoring case considerations.
`int`	`indexOf(String str, int fromIndex)` Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
`int`	`lastIndexOf(String str)` Returns the index within this string of the last occurrence of the specified substring
`int`	`length()` Returns the length of this string.

----- END OF PAPER -----