Cape Peninsula University of Technology

creating futures

ASSESSMENT COVER PAGE [Memo]

FACULTY:

QUALIFICATION (S)	**Bachelor of Engineering Technology in Computer Engineering**			**CODE (S)**	**BPETCP**
SUBJECT (S)	**Software Design 2**			**CODE (S)**	**SDN260S**
NO OF PAGES (including cover page)	**8**	**DATE**	**November 10, 2022**	**TIME**	**14h00-17h00**
ANNEXURE (S) (Y/N)	**N**			**DURATION**	**3 Hours**
COLOUR IMAGES (Y/N)	**N**				

EXAMINER NAME	**Mr. H. Mataifa**
INTERNAL MODERATOR	**Mr. V. Moyo**
EXTERNAL MODERATOR	**N/A**

INSTRUCTIONS

1. Answer **ALL** questions
2. Write your name and student number on each page of everything you submit
3. Document all your work as thoroughly as possible; include comments in your code to explain the logic behind your implementation
4. You **may not** collaborate with anyone, neither asking for help from anyone nor giving help to anyone
5. Create a personal working folder on the desktop where all your work will be saved; name the folder with your surname and student number
6. Zip your folder; you will need to upload the zipped file onto Blackboard at the end

REQUIREMENTS

None

DO NOT turn the page over before the starting time

QUESTION 1
[20]

Compare and contrast each of the following (i.e. main difference and/or similarity):

1.1 A String and a StringBuilder [2]

Answer:

*A String is **immutable** in Java (meaning it cannot be modified once initialized), whereas a StringBuilder is **mutable** (i.e. its value can be changed after being initialized).*

1.2 String methods equals and compareTo [2]

Answer:

*Both **equals** and **compareTo** String methods check the equality of the contents of two strings; the main difference is that **equals** returns a **boolean** type (true when contents are the same, false otherwise), whereas **compareTo** returns an **integer** type (which may be less than zero, equal to zero or greater than zero, depending on how the contents of the two string compare)*

1.3 A recursive method and a standard (non-recursive) method [2]

Answer:

*A **recursive method** is a method that **calls itself** within the definition of the method. It differs from standard method in its structure, because it always has two cases: the base case and the recursive method call. The base case must eventually be reached for the recursive method to terminate successfully. A **standard method does not refer to itself** within the definition of the method*

1.4 Recursion and iteration [2]

Answer:

Recursion and iteration are similar in the sense that they both make use of a control statement, repetition, and a termination test, although the way in which the control, repetition and termination test is implemented is different in the two paradigms

1.5 A Class and an Object [2]

Answer:

A Class in OOP can be thought of as a "blueprint" that specifies the attributes or characteristics and operations or behaviours of related program modules (referred to as objects). Every object belongs to a Class, and is referred to as an instance of the Class. So a Class must first be defined before Objects of the Class can be created

1.6 A static method and a non-static method [2]

Answer:

*A **static method** is also referred to as a **class method**. It can be invoked without need for instantiating an object of the class. In contrast, a **non-static method** is always invoked in conjunction with an instance of a class (i.e. an object).*

1.7 A private method and a protected method [2]

Answer:

A private method can only be invoked within the class to which it belongs (i.e. it is hidden outside the class, even from subclasses). A protected method, on the other hand, can be invoked *either within the class in which it's been defined, or within classes that inherit from the class to which it belongs, or within the package to which the class in which it's been defined belongs.*

1.8 A Set and a List [2]

Answer:

*A **set** is **unordered** and **cannot contain duplicate elements**, whereas a **list** is **ordered** and **can contain duplicate elements***

1.9 A standard Array and an ArrayList [2]

Answer:

*An **Array** has a **fixed size** (which needs to be specified upon initialization), whereas an **ArrayList**'s **size can change dynamically***

1.10 A Stack and a Queue [2]

Answer:

*A **Stack** is referred to as a **Last-In-First-Out** (**LIFO**) data structure (i.e. the last element to be added to the Stack is also the first one to be removed from it), whereas a **Queue** is referred to as a **First-In-Fist-Out** (**FIFO**) data structure (i.e. the first element to be added to a standard Queue is also the first one to be removed from it)*

QUESTION 2 [30]

Write a Java application that will validate a telephone number entered by a user using a regular expression. The application should check the validity of the telephone number entered against the following requirements:

- The format should be (XXX) XXX-XXXX (i.e. ten digits, where each X represents a digit)
- The first three digits, enclosed in round brackets, are the area code
- The next three digits should be separated from the area code by single white space (or alternatively a hyphen), and by a hyphen (or alternatively a single white space) from the last four digits
- The area code should always begin with 0
- The second digit of the area code should lie between 1 and 7
- The third digit of the area code should lie between 0 and 3
- The remaining (seven) digits can lie anywhere between 0 and 9
- If the user enters an invalid telephone number, the application should inform the user and grant them two more attempts to enter a telephone number in the correct format
- After three unsuccessful attempts, the application should output an appropriate message to the user and terminate the program
- Once the user has entered a telephone number in the required format, the application should thank the user and print the entered telephone number to the screen in the following format (on separate lines):

    Area code: XXX

Phone number XXX-XXXX

The application needs to *show the user the correct format in which the telephone number needs to be entered* (although it need not inform the user about the limits on the digits as specified above.

Solution program code:

```
// Instantiate a (character-based) input stream object (Scanner):
        Scanner input = new Scanner(System.in);

        // Request user to enter telephone number:
        System.out.println("Please enter telephone number in the format: (XXX) XXX-XXXX\n"
                + "3-digit code must be enclosed in parentheses\n"
                + "with space between area code and next 3 digits\n"
                + "and hyphen between next 3 digits and last 4 digits: ");
        String telNumber=input.nextLine();

        // Formulate regular expression:
        String regexp="\\(0[1-7][0-3]\\)[ -][0-9]{3}[ -][0-9]{4}";

        boolean telNumberMatches=telNumber.matches(regexp);

        int i=3;
        // prompt user up to three times to enter correct telephone number:
        while (!telNumberMatches && i>1){
            System.out.println("\nEntry is not valid.");
            System.out.println("\nPlease enter telephone number in the format: (XXX) XXX-XXXX\n"
                + "3-digit code must be enclosed in parentheses\n"
                + "with space between area code and next 3 digits\n"
                + "and hyphen between next 3 digits and last 4 digits: ");
            telNumber=input.nextLine();
            telNumberMatches=telNumber.matches(regexp);
            i-=1;
        }

        // Process entered telephone number and print to screen:
        if (!telNumberMatches)
            System.out.println("\nMaximum number of attempts reached. Thank you and goodbye.");
        else {
            String[] telNumberParts;
            if (telNumber.charAt(5)==' ')
                telNumberParts=telNumber.split(" ");
            else
                telNumberParts=telNumber.split("-");
            String areaCode=telNumberParts[0].substring(1, 4);

            if (telNumberParts.length>2){
            System.out.printf("\nThank you. Following is the information you entered:\n"
                    + "Area code: %s\nPhone number: %s",
                    areaCode, telNumberParts[1]+telNumberParts[2]);
            }
            else {
                System.out.printf("\nThank you. Following is the information you entered:\n"
                    + "Area code: %s\nPhone number: %s",
                    areaCode, telNumberParts[1]);
            }
        }
```

QUESTION 3: **[30]**

Write a Java application that will use a recursive method to *add all the **odd** numbers between 0 and an* (integer) *value entered by the user*. The application will do the following:

- Request the user to enter the (integer) number which is the upper (or lower) limit for the computation.
- If the user enters a *positive integer* (for example *n=11*), the application will add all odd numbers between **0** and **11**
- If the user enters a *negative integer* (for example *n = -11*), the application will add all odd numbers between **-11** and **0**
- Repeatedly prompt the user until they enter a valid (*positive or negative*) **integer** (in case the user initially enters anything other than an integer)
- Use the recursive method for computing the sum of odd numbers
- If the user entered a *positive integer*, print the result to the screen as "Sum of odd numbers between 0 and n = result", where n is the number entered by the user, and result is the result obtained by applying the recursive method
- If the user entered a *negative integer*, print the result to the screen as "Sum of odd numbers between n and 0 = result", where n is the number entered by the user, and result is the result obtained by applying the recursive method

Solution program code:

```
// Instantiate a Scanner object to retrieve user input from keyboard:
Scanner input = new Scanner(System.in);

// Prompt user for input:
System.out.println("Please enter an integer (positive or negative): ");

// Loop until user enters valid input (i.e. integer, positive or negative):
while (!input.hasNextInt()){
   System.out.println("Number entered is not a valid integer."
         + "Please enter a valid integer (positive or negative): ");
   input.nextLine();
}
// Once valid input is entered, assign it to integer variable n:
int n=input.nextInt();

// Compute the sum of even numbers, and print to screen:
if (n>=0)
System.out.printf("Sum of even numbers between 0 and %d is %s.\n",
       n, recursiveSumEvenNumbers(n));
else
   System.out.printf("Sum of even numbers between %d and 0 is %s.\n",
            n, recursiveSumEvenNumbers(n));
}

// Define recursive method to compute sum of even numbers between 0 and n:
    public static int recursiveSumEvenNumbers(int n){
        if (n<0){ /* check whether user entered a negative number */
            if (n%2==-1) /* check whether user entered an odd number. If so, increment */
                n+=1;

        if (n==0)    /* base case for negative n */
            return n;
        else
            return n+recursiveSumEvenNumbers(n+2); /* recursive call for negative n */

        }
        else {
        if (n%2==1) /* check whether user entered an odd number. If so, decrement */
            n-=1;

        if (n==0)    /* base case for positive n */
            return n;
        else
            return n+recursiveSumEvenNumbers(n-2); /* recursive call for positive n */
    }
```

QUESTION 4: [30]

Write a Java application that will perform *randomized arithmetic operations* on *two arrays* of *type int*. The application should do the following:

- Generate two *random-number arrays* of *type int*, each of *size* ***10***
- The random numbers in array1 should range from 0 to 49
- The random numbers in array2 should range from 1 to 49
- The two generated arrays will be printed onto the screen on separate lines
- The application will then perform *element-wise* ***randomized*** *arithmetic operations* on the *array element pairs* as follows:
    - Picking the first element from each array, the application will randomly choose an arithmetic operation from (add, subtract, multiply divide) and apply it to the two operands picked from the two arrays
    - It will then perform the computation and print the result onto the screen
    - This will be done for each pair of array elements, from the first to the last, the result of the arithmetic operation being printed on a new line each time
- The application will also *write all the results to a text file*, as they are printed on the screen. The results should be appended to the text file, without overwriting the previously written information
- The screenshots below show a sample of running the specified application, which are the results your application is expected to produce

Solution program code:

```
 // Instantiate a random number generator:
        Random generator = new Random();
        //  Instantiate two arrays of type double and size 10:
        double[] operand1 = new double[10];
        double[] operand2 = new double[10];
        // Instantiate a String array containing the arithmetic operations
        // to be performed:
        String[] operations={"Add","Subtract","Multiply","Divide"};

        // Instantiate a character-based output stream object:
        Formatter output = new Formatter (new BufferedWriter
        (new FileWriter("arithmetic_operations.txt", true)));

        // Populate the arrays with random numbers in the specified ranges:
        for (int i=0; i<operand1.length; i++){
            operand1[i]=20.0*Math.round((generator.nextDouble())*1000)/1000;
            operand2[i]=1.0+19.0*Math.round((generator.nextDouble())*1000)/1000;

        }
        // Output the generated arrays:
        System.out.println("\nGenerated random double arrays:");
        System.out.printf("Array 1 content: %s\n", Arrays.toString(operand1));
        System.out.printf("Array 2 content: %s\n", Arrays.toString(operand2));

        System.out.println("\nFollowing operations have been carried out:");
        // Write the same information to a text file:
        output.format("\nGenerated random double arrays:"
                + "\nArray 1 content: %s\nArray 2 content: %s\n"
                + "\nFollowing operations have been carried out:\n",
                Arrays.toString(operand1),Arrays.toString(operand2));

        // Instantiate some variables to be used in the arithmetic operations:
        String operation=null;
        double result=0.0;
        char opChar=' ';

        // Loop through the array elements and perform the randomized
        // arithmetic operations:
        for (int i=0; i<10; i++){
            // Randomly select the arithmetic operation to perform:
            operation=operations[generator.nextInt(4)];

            // Use the selected operation in a switch..case statement
            // to execute the right operation:
            switch (operation){
                case "Add":
                    result = operand1[i]+operand2[i];
                    opChar='+';
                    break;
                case "Subtract":
                    result = operand1[i]-operand2[i];
                    opChar='-';
                    break;
                case "Multiply":
                    result = operand1[i]*operand2[i];
                    opChar='*';
                    break;
                case "Divide":
                    result = operand1[i]/operand2[i];
                    opChar='/';
                    break;
                default:
                    break;
            }
            // Print the result to screen and write to file:
            System.out.printf("%.2f %c %.2f = %.2f\n", operand1[i],
                    opChar, operand2[i], result);
            output.format("%.2f %c %.2f = %.2f\n", operand1[i],
                    opChar, operand2[i], result);
        }
        // close the output file stream:
        output.close();
```

End of Assessment