BACS2023 Object-Oriented Programming

Practical 2: Methods & Exception Handling

Q1. Write a class named **`UnitConverter`** that contains the following two methods:

```
/** Converts from inches to centimeters */
public static double inchToCentimeter(double in)

/** Converts from centimeters to inches */
public static double centimeterToInch(double cm)
```

The formula for the conversion is:

```
centimeters = inches * 2.54
```

Write a **`main`** method that invokes these methods to display the following tables:

```
Inches      Centimeters          Centimeters      Inches

1.0         2.54                 5.0              1.97
2.0         5.08                 10.0             3.94
3.0         7.62                 15.0             5.91
4.0         10.16                20.0             7.87
5.0         12.70                25.0             9.84
6.0         15.24                30.0             11.81
7.0         17.78                35.0             13.78
8.0         20.32                40.0             15.75
9.0         22.86                45.0             17.72
10.0        25.40                50.0             19.69
```

Q2. Create a class named Average that contains 4 overloaded methods named **`calculateAverage`** that will calculate and return the average of its parameters. Details of the 4 methods are provided below:

- **`calculateAverage`** method with 2 integer parameters
- **`calculateAverage`** method with 3 integer parameters
- **`calculateAverage`** method with 2 double parameters
- **`calculateAverage`** method with 3 double parameters

Demonstrate the use of the 4 methods in a separate driver program.

Q3. Write a program that prints the following table using the **`sqrt`** method in the **`Math`** class.

```
Number   SquareRoot
0        0.0000
2        1.4142
…        …
18       4.2426
20       4.4721
```

Q4. Instead of using the **`sqrt`** method in the **`Math`** class, you have been asked to implement the **`sqrt`** method using the approximation approach described below:

For numbers greater than one, the square root method must initially set a lower limit to one and an upper limit to the number (since the square root of the number always lies between one and the number). It must then determine the midpoint between the lower and upper limits and evaluate the square of the midpoint. If the square of the midpoint is greater than the number, the square root method must

move the upper limit to the midpoint and similarly if the square of the midpoint is less than the number, it must move the lower limit to the midpoint. After moving the appropriate limit, the square root method must evaluate a new midpoint and repeat the process until the desired precision is obtained.

For numbers less than one, the algorithm is essentially the same, except that the initial lower limit is the number and the initial upper limit is one.

The required precision for double precision floating point numbers is 8 significant digits. The precision at any iteration can be determined by dividing the difference between the limits by the lower limit. When this is less than 1/108 any number between the limits will be an estimate of the square root of the number to the required precision. To minimize the error, the square root method should return the midpoint between the final limits that satisfy the precision requirement.

The square root method must return exact values for the special cases of zero and one.

If an application attempts to calculate the square root of a negative number, the square root method should display an appropriate message and terminate the program.

(Extracted from eGenting Programming Competition 2011)