Homework 2. Exercises on Java Basics

Writing Good Programs

The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.
It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these good programs. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.
Pay particular attention to:

1. **Coding style**:
   - Read Java code convention: "Java Style and Commenting Guide".
   - Follow the Java Naming Conventions for variables, methods, and classes STRICTLY. Use CamelCase for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., radius) and class names (e.g., Circle). Use verbs for methods (e.g., getArea(), isEmpty()).
   - **Use Meaningful Names**: Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like i, j, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and i for array index. Use meaningful names like row and col (instead of x and y, i and j, x1 and x2), numStudents (not n), maxGrade, size (not n), and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use books for an array of books, and book for each item).
   - Use consistent indentation and coding style. Many IDEs (such as Eclipse / NetBeans) can re-format your source codes with a single click.
2. **Program Documentation**: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.
3. The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)).

1 Exercises on Decision and Loop

Write a Java program called DecisionAndLoopsHomework that implements the following methods.

1.1 Number Guess

Write a program to play the number guessing game. The program shall generate a random number between 0 and 99. The player inputs his/her guess, and the program shall response with "Try higher", "Try lower" or "You got it in n trials" accordingly. For example:

Command window

```
Key in your guess:
50
Try higher
70
Try lower
65
Try lower
61
You got it in 4 trials!
```

The program is designed using 2 functions with prototypes as follows:

```java
public static void guessNumber();
public static void guessNumber(int number, Scanner in);
```

- void guessNumber(): Generate a random number between 0 and 99, then call the method void guessNumber(int number, Scanner in) to guess the generated number.
- void guessNumber(int number, Scanner in): The player inputs his/her guess, and the program shall response with "Try higher", "Try lower" or "You got it in n trials" accordingly.

Hints
Use Math.random() to produce a random number in double between 0.0 (inclusive) and 1.0 (exclusive). To produce an int between 0 and 99, use:

```java
final int SECRET_NUMBER = (int)(Math.random()*100);  // truncate to int
```

1.2 Word Guess

Write a program to guess a word by trying to guess the individual characters. The word to be guessed shall be provided by player. Your program shall look like:

Command window
```
Key in one character or your guess word: t
Trial 1: t__t___
Key in one character or your guess word: g
Trial 2: t__t__g
Key in one character or your guess word: e
Trial 3: te_t__g
Key in one character or your guess word: testing
Congratulation!
You got in 4 trials
```

The program is designed using 2 functions with prototypes as follows:

```java
public static void guessWord();
public static void guessWord(String guessedString, Scanner in);
```

- void guessWord(): Prompt player for a word, then call the method void guessWord(String guessedString, Scanner in) to guess the word.
- void guessWord(String guessedString, Scanner in): The player inputs his/her guess.

Hints

1. Set up a *boolean* array (of the length of the word to be guessed) to indicate the positions of the word that have been guessed correctly.
2. Check the length of the input string to determine whether the player enters a single character or a guessed word. If the player enters a single character, check it against the word to be guessed, and update the boolean array that keeping the result so far.

Try
Try retrieving the word to be guessed from a text file (or a dictionary) randomly.

2 Exercises on Nested-Loops

Write a Java program called NestedLoopsExercise that implements the following methods.

2.1 SquarePattern

Write a method called squarePattern() (with prototype void squarePattern(int n)) that prints the following square pattern using two nested for-loops. Write a method called testSquarePattern() prompts user for the *size* (a non-negative integer in int); and prints the square pattern.

Command window

```
Enter the size: 5
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

Hints

The code pattern for printing 2D patterns using nested loops is:

```java
// Outer loop to print each of the rows
for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
  // Inner loop to print each of the columns of a particular row
  for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
    System.out.print( ...... );  // Use print() without newline inside the inner loop
    ......
  }

  // Print a newline after printing all the columns
  System.out.println();
}
```

Notes

1. You should name the loop indexes *row* and *col*, NOT i and j, or x and y, or a and b, which are meaningless.
2. The *row* and *col* could start at 1 (and upto *size*), or start at 0 (and upto $size - 1$). As computer counts from 0, it is probably more efficient to start from 0. However, since humans counts from 1, it is easier to read if you start from 1.

Try

1. Rewrite the above program using nested while-do loops.

2.2 CheckerPattern

Write a method called checkerPattern() (with prototype void checkerPattern(int n)) that prints the following checkerboard pattern. Write a method called testCheckerPattern() that prompts user for the *size* (a non-negative integer in int); and prints the checkerboard pattern.

Command window

```
Enter the size: 7
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

Hints

```java
// Outer loop to print each of the rows
for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
   // Inner loop to print each of the columns of a particular row
   for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
      if ((row % 2) == 0) {   // row 2, 4, 6, ...
      ......
   }

   System.out.print( ...... );    // Use print() without newline inside the inner loop
   ......
   }

   // Print a newline after printing all the columns
   System.out.println();
}
```

2.3 TimeTable

Write a method called timeTable() (with prototype void timeTable(int n)) that prints the multiplication table as the following table. Write a method called testTimeTable() that prompts user for the *size* (a positive integer in int); and prints the multiplication table.

Command window

```
Enter the size: 10
  * |   1   2   3   4   5   6   7   8   9  10
---------------------------------------------
  1 |   1   2   3   4   5   6   7   8   9  10
  2 |   2   4   6   8  10  12  14  16  18  20
  3 |   3   6   9  12  15  18  21  24  27  30
  4 |   4   8  12  16  20  24  28  32  36  40
  5 |   5  10  15  20  25  30  35  40  45  50
  6 |   6  12  18  24  30  36  42  48  54  60
  7 |   7  14  21  28  35  42  49  56  63  70
  8 |   8  16  24  32  40  48  56  64  72  80
  9 |   9  18  27  36  45  54  63  72  81  90
 10 |  10  20  30  40  50  60  70  80  90 100
```

Hints

1. Use printf() to format the output, e.g., each cell is %4d.

2.4 TriangularPattern

Write 4 methods called triangularPatternX() (with prototype void triangularPatternX(int n), X = A, B, C, D) that prints each of the patterns as shown. Write a method called testTriangularPattern() prompts user for the *size* (a non-negative integer in int); and prints each of the patterns.

Command window

```
Enter the size: 8

#                 # # # # # # # #   # # # # # # # #                 #
# #               # # # # # # #       # # # # # # #               # #
# # #             # # # # # #           # # # # # #             # # #
# # # #           # # # # #               # # # # #           # # # #
# # # # #         # # # #                   # # # #         # # # # #
# # # # # #       # # #                       # # #       # # # # # #
# # # # # # #     # #                           # #     # # # # # # #
# # # # # # # #   #                               #   # # # # # # # #
      (a)                (b)              (c)               (d)
```

Hints

1. On the main diagonal, $row = col$. On the opposite diagonal, $row + col = size + 1$, where row and col begin from 1.

2. You need to print the leading blanks, in order to push the # to the right. The trailing blanks are optional, which does not affect the pattern.

3. For pattern (a), if ($row \geq col$) print #. Trailing blanks are optional.
4. For pattern (b), if ($row + col \leq size + 1$) print #. Trailing blanks are optional.
5. For pattern (c), if ($row \geq col$) print #; else print blank. Need to print the leading blanks.
6. For pattern (d), if ($row + col \geq size + 1$) print #; else print blank. Need to print the leading blanks.
7. The coding pattern is:

```java
// Outer loop to print each of the rows
for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
  // Inner loop to print each of the columns of a particular row
  for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
    if (......) {
      System.out.print("# ");
    } else {
      System.out.print("  ");  // Need to print the "leading" blanks
    }
  }

  // Print a newline after printing all the columns
  System.out.println();
}
```

2.5 BoxPattern

Write 4 methods called boxPatternX() (with prototype void boxPatternX(int n), X = A, B, C, D) that prints the pattern as shown. Write a method called testBoxPattern() that prompts user for the *size* (a non-negative integer in int); and prints the patterns.

```
Command window

Enter the size: 8

# # # # # # #     # # # # # # #     # # # # # # #     # # # # # # #
#           #       #                         #         #       #
#           #         #                     #             #   #
#           #           #                 #                 #
#           #             #             #                 #   #
#           #               #         #                 #       #
# # # # # # #     # # # # # # #     # # # # # # #     # # # # # # #
     (a)               (b)               (c)               (d)
```

Hints

1. On the main diagonal, $row = col$. On the opposite diagonal, $row + col = size + 1$, where row and col begin from 1.
2. For pattern (a), if ($row == 1 \,||\, row == size \,||\, col == 1 \,||\, col == size$) print #; else print blank. Need to print the intermediate blanks.
3. For pattern (b), if ($row == 1 \,||\, row == size \,||\, row == col$) print #; else print blank.

2.6 HillPattern

Write 4 methods called hillPatternX() (with prototype void hillPatternX(int n) , X = A, B, C, D) that prints the pattern as shown. Write a method called testHillPatternX() that prompts user for the size (a non-negative integer in int); and prints the patterns.

Command window

```
Enter the rows: 5

        #         # # # # # # # # #         #         # # # # # # # # #
      # # #         # # # # # # #         # # #       # # # #   # # # #
    # # # # #         # # # # #         # # # # #     # # #       # # #
  # # # # # # #         # # #         # # # # # # #   # #           # #
# # # # # # # # #         #         # # # # # # # # # #               #
       (a)               (b)          # # # # # # #   # #           # #
                                        # # # # #     # # #       # # #
                                          # # #       # # # #   # # # #
                                            #         # # # # # # # # #
                                           (c)               (d)
```

Hints

1. For pattern (a):

```java
for (int row = 1; ......) {
  // numCol = 2*numRows - 1
  for (int col = 1; ......) {
    if ((row + col >= numRows + 1) && (row >= col - numRows + 1)) {
      ......;
    } else {
      ......;
    }
  }
  ......;
}
```

or, use 2 sequential inner loops to print the columns:

```java
for (int row = 1; row <= rows; row++) {
  for (int col = 1; col <= rows; col++) {
    if ((row + col >= rows + 1)) {
      ......
    } else {
      ......
    }
  }

  for (int col = 2; col <= rows; col++) {  // skip col = 1
    if (row >= col) {
      ......
    } else {
      ......
    }
  }
  ......
}
```

3 Exercises on String and char Operations

Write a Java program called StringAndCharacterHomework that implements the following methods.

3.1 Exchange Cipher

This simple cipher exchanges 'A' and 'Z', 'B' and 'Y', 'C' and 'X', and so on.
Write a method called exchangeCipher() (with prototype String exchangeCipher(String inStr)) that computes the ciphertext; and return the ciphertext in uppercase. Write a method called testExchangeCipher() that prompts user for a plaintext string consisting of mix-case letters only, compute the ciphertext; and print the ciphertext in uppercase. For examples,

Command window

```
Enter a plaintext string: abcXYZ
The ciphertext string is: ZYXCBA
```

Hints

1. Use in.next().toUpperCase() to read an input string and convert it into uppercase to reduce the number of cases.

2. You can use a big nested-if with 26 cases ('A' - 'Z'), or use the following relationship:

'A' + 'Z' == 'B' + 'Y' == 'C' + 'X' == ... == plainTextChar + cipherTextChar
Hence, cipherTextChar = 'A' + 'Z' - plainTextChar

3.2 TestPalindromicWord and TestPalindromicPhrase

A word that reads the same backward as forward is called a palindrome, e.g., "mom", "dad", "racecar", "madam", and "Radar" (case-insensitive). Write a method called **isPalindromicWord()** (with prototype boolean isPalindromicWord(String inStr)) that checks for palindromic work. Write a method called testPalindromicWord() that prompts user for a word and prints ""xxx" is | is not a palindrome".

A phrase that reads the same backward as forward is also called a palindrome, e.g., "Madam, I'm Adam", "A man, a plan, a canal - Panama!" (ignoring punctuation and capitalization). Modify your methods (called isPalindromicPhrase() and **testPalindromicPhrase()**) to check for palindromic phrase. Use in.nextLine() to read a line of input.

Hints

1. Maintain two indexes, forwardIndex (*fIdx*) and backwardIndex (*bIdx*), to scan the phrase forward and backward.

```
int fIdx = 0;
int bIdx = strLen - 1;
while (fIdx < bIdx) {
    ......
    ++fIdx;
    --bIdx;
}

// or
for (int fIdx = 0, bIdx = strLen - 1; fIdx < bIdx; ++fIdx, --bIdx)
    ↪ {
    ......
}
```

2. You can check if a char c is a letter either using built-in boolean function Character.isLetter(c); or boolean expression (c ≥ 'a' && c ≤ 'z'). Skip the index if it does not contain a letter.

4 Exercises on Array

4.1 PrintArrayInStars

Write a Java program called ArrayHomework that implements the following methods. Write a method called printArrayInStars() (with prototype void printArrayStars(Scanner in)) which prompts user for the number of items in an array (a non-negative integer), and saves it in an int variable called NUM_ITEMS. It then prompts user for the values of all the items (non-negative integers) and saves them in an int array called items. The program shall then print the contents of the array in a graphical form, with the array index and values represented by number of stars. For examples,

Command window

```
Enter the number of items: 5
Enter the value of all items (separated by space): 7 4 3 0 7
0: *******(7)
1: ****(4)
2: ***(3)
3: (0)
4: *******(7)
```

Hints

```java
// Declare variables
final int NUM_ITEMS;
int[] items;   // Declare array name, to be allocated after NUM_ITEMS is known
......
......

// Print array in "index: number of stars" using a nested-loop
// Take note that rows are the array indexes and columns are the value in that index
for (int idx = 0; idx < items.length; ++idx) {   // row
  System.out.print(idx + ": ");
  // Print value as the number of stars
  for (int starNo = 1; starNo <= items[idx]; ++starNo) {   // column
    System.out.print("*");
  }
  ......
}
......
```

4.2 Matrices (2D Arrays)

Write a Matrix library that supports matrix operations (such as addition, subtraction, multiplication) via 2D arrays. The operations shall support both double and int. Also write methods to test all the operations.

Hints

```java
public class Matrix {
  // Method signatures
  public static void print(int[][] matrix);
  public static void print(double[][] matrix);

  // Used in add(), subtract()
  public static boolean haveSameDimension(int[][] matrix1,
                                          int[][] matrix2);
  public static boolean haveSameDimension(double[][] matrix1,
                                          double[][] matrix2);

  public static int[][] add(int[][] matrix1, int[][] matrix2);
  public static double[][] add(double[][] matrix1,
                               double[][] matrix2);

  public static int[][] subtract(int[][] matrix1, int[][] matrix2);
  public static double[][] subtract(double[][] matrix1,
                                    double[][] matrix2);

  public static int[][] multiply(int[][] matrix1, int[][] matrix2);
  public static double[][] multiply(double[][] matrix1,
                                    double[][] matrix2);

  ......
}
```

5 Exercises on Method

Write a Java program called MathLibraryHomework that implements the following methods.

5.1 Trigonometric Series

Write methods to compute $\sin(x)$ and $\cos(x)$ using the following series expansion. The prototypes of the methods are:

```java
// x in radians, NOT degrees
public static double sin(double x, int numTerms);
public static double cos(double x, int numTerms);
```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$
$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Write a method called testTrigonometric() that prompts user for x value (in double) and $numTerms$ (in int) value, then compute the sin and cosine values of x. Compare the values computed using the series with the JDK methods *Math.sin()*, *Math.cos()* at $x = 0, \pi/6, \pi/4, \pi/3, \pi/2$ using various numbers of terms.

Hints
Do not use int to compute the factorial; as factorial of 13 is outside the int range. Avoid generating large numerator and denominator. Use double to compute the terms as:

$$\frac{x^n}{n!} = \left(\frac{x}{n}\right)\left(\frac{x}{n-1}\right)\dots\left(\frac{x}{1}\right).$$

5.2 Exponential Series

Write a method to compute the sum of the series. The signature of the method is:

```java
public static double specialSeries(double x, int numTerms);
```

$$x + \frac{1}{2} \times \frac{x^3}{3} + \frac{1 \times 3}{2 \times 4} \times \frac{x^5}{5} + \frac{1 \times 3 \times 5}{2 \times 4 \times 6} \times \frac{x^7}{7} + \frac{1 \times 3 \times 5 \times 7}{2 \times 4 \times 6 \times 8} \times \frac{x^9}{9} + \cdots ; -1 \le x \le 1.$$

Also write a method to test the sum of the series called testSpecialSeries(), which prompts user for x value (in double) and $numTerms$ (in int) values, then computes the sum of the series.

5.3 FactorialInt (Handling Overflow)

Write a method called factorialInt() to list all the factorials that can be expressed as an int (i.e., 32-bit signed integer in the range of $[-2147483648,\ 2147483647]$). Your output shall look like:

Command window
```
The factorial of 1 is 1
The factorial of 2 is 2
...
The factorial of 12 is 479001600
The factorial of 13 is out of range
```

Hints
The maximum and minimum values of a 32-bit int are kept in constants *Integer.MAX_VALUE* and *Integer.MIN_VALUE*, respectively. Try these statements:

```java
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use $F(n) * (n+1) > Integer.MAX\_VALUE$ to check for overflow. Instead, overflow occurs for $F(n+1)$ if ($Integer.MAX\_VALUE$ / Factorial(n)) $< (n+1)$, i.e., no more room for the next number.

Try
Modify your program called factorialLong() to list all the factorial that can be expressed as a long (64-bit signed integer). The maximum value for long is kept in a constant called *Long.MAX_VALUE*.

5.4 FibonacciInt (Handling Overflow)

Write a method called fibonacciInt() to list all the Fibonacci numbers, which can be expressed as an int (i.e., 32-bit signed integer in the range of $[-2147483648, 2147483647]$). The output shall look like:

Command window

```
F(0) = 1
F(1) = 1
F(2) = 2
...
F(45) = 1836311903
F(46) is out of the range of int
```

Hints
The maximum and minimum values of a 32-bit int are kept in constants *Integer.MAX_VALUE* and *Integer.MIN_VALUE*, respectively. Try these statements:

```java
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use $F(n) = F(n-1) + F(n-2) > Integer.MAX\_VALUE$ to check for overflow. Instead, overflow occurs for $F(n)$ if *Inte-*

ger.MAX_VALUE $-F(n-1) < F(n-2)$ (i.e., no more room for the next Fibonacci number).

Try
Write a similar method called tribonacciInt() for Tribonacci numbers.

5.5 Number System Conversion

Write a method call toRadix() which converts a positive integer from one radix into another. The method has the following header:

```java
// The input and output are treated as String.
public static String toRadix(String in, int inRadix, int outRadix)
```

Write a method called testNumberConversion(), which prompts the user for an input string, an input radix, and an output radix, and display the converted number. The output shall look like:

Command window

```
Enter a number and radix: A1B2
Enter the input radix: 16
Enter the output radix: 2
"A1B2" in radix 16 is "1010000110110010" in radix 2.
```