Homework 1. 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.

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 Nested-Loops

1.1 SquarePattern

Write a program called **SquarePattern** that prompts user for the *size* (a non-negative integer in *int*); and prints the following square pattern using two nested for-loops.

```
Command window

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

Hints

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

```
// 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.

1.2 CheckerPattern

Write a program called **CheckerPattern** that prompts user for the size (a non-negative integer in *int*); and prints the following checkerboard pattern.

Command window

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

Hints

```
// 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();
}
```

1.3 TimeTable

Write a program called **TimeTable** that prompts user for the *size* (a positive integer in *int*); and prints the multiplication table as shown:

```
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.

1.4 TriangularPattern

Write 4 programs called **TriangularPatternX** (X = A, B, C, D) that prompts user for the *size* (a non-negative integer in *int*); and prints each of the patterns as shown:

```
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:

```
// 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();
}
```

1.5 BoxPattern

Write 4 programs called **BoxPatternX** (X = A, B, C, D) that prompts user for the *size* (a non-negative integer in *int*); and prints the pattern as shown:

```
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.

1.6 HillPattern

Write 3 programs called **HillPatternX** (X = A, B, C, D) that prompts user for the *size* (a non-negative integer in *int*); and prints the pattern as shown:

```
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:

```
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 {
      ......
    }
  }
  ......
}
```

2 Debugging/Tracing Programs using a Graphic Debugger

2.1 Factorial

The following program computes and prints the factorial of n ($= 1 * 2 * 3 * \cdots * n$). The program, however, has a logical error and produce a wrong answer for $n = 20$ ("The Factorial of 20 is -2102132736" – negative?!).
Use the graphic debugger (of Eclipse/NetBeans) to debug the program by single-step through the program and tabulating the values of i and factorial at the statement marked by (*).
You should try out debugging features such as "Breakpoint", "Step Over", "Watch variables", "Run-to-Line", "Resume", "Terminate", among others. (Read "Eclipse for Java" or "NetBeans for Java" for details).

```
// Print factorial of n
public class Factorial {
  public static void main(String[] args) {  // Set an initial
     ↪ breakpoint at this statement
    int n = 20;
    int factorial = 1;

    // n! = 1*2*3...*n
    for (int i = 1; i <= n; i++) {  // i = 1, 2, 3, ..., n
      factorial = factorial * i;    // *
    }
    System.out.println("The Factorial of " + n + " is " + factorial);
```

```
    }
}
```

3 Exercises on String and char Operations

3.1 ReverseString

Write a program called **ReverseString**, which prompts user for a String, and prints the reverse of the String by extracting and processing each character. The output shall look like:

Command window

```
Enter a String: abcdef
The reverse of the String "abcdef" is "fedcba".
```

Hints

For a String called inStr, you can use inStr.length() to get the length of the String; and inStr.charAt(idx) to retrieve the char at the idx position, where idx begins at 0, up to instr.length() - 1.

```
// Define variables
String inStr;        // input String
int inStrLen;        // length of the input String
......

// Prompt and read input as "String"
System.out.print("Enter a String: ");
inStr = in.next();   // use next() to read a String
inStrLen = inStr.length();

// Use inStr.charAt(index) in a loop to extract each character
// The String's index begins at 0 from the left.
// Process the String from the right
for (int charIdx = inStrLen - 1; charIdx >= 0; --charIdx) {
  // charIdx = inStrLen-1, inStrLen-2, ... ,0
   ......
}
```

3.2 CountVowelsDigits

Write a program called **CountVowelsDigits**, which prompts the user for a String, counts the number of vowels (a, e, i, o, u, A, E, I, O, U) and digits $(0 - 9)$ contained in the string,

and prints the counts and the percentages (rounded to 2 decimal places). For example,

```
Command window

Enter a String: testing12345
Number of vowels: 2 (16.67%)
Number of digits: 5 (41.67%)
```

Hints

1. To check if a char c is a digit, you can use boolean expression ($c \geq '0'$ && $c \leq '9'$); or use built-in boolean function Character.isDigit(c).
2. You could use in.next().toLowerCase() to convert the input String to lowercase to reduce the number of cases.
3. To print a % using printf(), you need to use %%. This is because % is a prefix for format specifier in printf(), e.g., %d and %f.

3.3 PhoneKeyPad

On your phone keypad, the alphabets are mapped to digits as follows: ABC(2), DEF(3), GHI(4), JKL(5), MNO(6), PQRS(7), TUV(8), WXYZ(9). Write a program called **PhoneKeyPad**, which prompts user for a String (case insensitive), and converts to a sequence of keypad digits. Use (a) a nested-if, (b) a switch-case-default.

Hints

1. You can use *in.next().toLowerCase()* to read a *String* and convert it to lowercase to reduce your cases.
2. In switch-case, you can handle multiple cases by omitting the break statement, e.g.,

```
switch (inChar) {
  case 'a':
  case 'b':
  case 'c':  // No break for 'a' and 'b', fall thru 'c'
    System.out.print(2);
    break;
  case 'd':
  case 'e':
  case 'f':
    ......
  default:
    ......
}
```

3.4 Caesar's Code

Caesar's Code is one of the simplest encryption techniques. Each letter in the plaintext is replaced by a letter some fixed number of position (n) down the alphabet cyclically. In this exercise, we shall pick $n = 3$. That is, 'A' is replaced by 'D', 'B' by 'E', 'C' by 'F', ..., 'X' by 'A', ..., 'Z' by 'C'.
Write a program called **CaesarCode** to cipher the Caesar's code. The program shall prompt user for a plaintext string consisting of mix-case letters only; compute the ciphertext; and print the ciphertext in uppercase. For example,

```
Command window

Enter a plaintext string: Testing
The ciphertext string is: WHVWLQJ
```

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'). But it is much better to consider 'A' to 'W' as one case; 'X', 'Y' and 'Z' as 3 separate cases.
3. Take note that char 'A' is represented as Unicode number 65 and char 'D' as 68. However, 'A' + 3 gives 68. This is because char + *int* is implicitly casted to *int* + *int* which returns an int value. To obtain a char value, you need to perform explicit type casting using (char)('A' + 3). Try printing ('A' + 3) with and without type casting.

3.5 Decipher Caesar's Code

Write a program called **DecipherCaesarCode** to decipher the Caesar's code described in the previous exercise. The program shall prompts user for a ciphertext string consisting of mix-case letters only; compute the plaintext; and print the plaintext in uppercase. For example,

```
Command window

Enter a ciphertext string: wHVwLQJ
The plaintext string is: TESTING
```

3.6 Exchange Cipher

This simple cipher exchanges 'A' and 'Z', 'B' and 'Y', 'C' and 'X', and so on.
Write a program called **ExchangeCipher** that prompts user for a plaintext string consisting of mix-case letters only. You program shall 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

3.7 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 program 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 program (called **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.

3.8 CheckHexStr

The hexadecimal (hex) number system uses 16 symbols, $0 - 9$ and A - F (or a - f). Write a program to verify a hex string. The program shall prompt user for a hex string; and decide if the input string is a valid hex string. For examples,

```
Command window

Enter a hex string: 123aBc
"123aBc" is a hex string

Enter a hex string: 123aBcx
"123aBcx" is NOT a hex string
```

Hints

```
if (!((inChar >= '0' && inChar <= '9')
        || (inChar >= 'A' && inChar <= 'F')
        || (inChar >= 'a' && inChar <= 'f'))) {  // Use positive logic
           ↪ and then reverse
   ......
}
```

3.9 Bin2Dec

Write a program called **Bin2Dec** to convert an input binary string into its equivalent decimal number. Your output shall look like:

```
Command window

Enter a Binary string: 1011
The equivalent decimal number for binary "1011" is: 11

Enter a Binary string: 1234
error: invalid binary string "1234"
```

3.10 Hex2Dec

Write a program called **Hex2Dec** to convert an input hexadecimal string into its equivalent decimal number. Your output shall look like:

```
Command window

Enter a Hexadecimal string: 1a
The equivalent decimal number for hexadecimal "1a" is: 26

Enter a Hexadecimal string: 1y3
error: invalid hexadecimal string "1y3"
```

3.11 Oct2Dec

Write a program called **Oct2Dec** to convert an input Octal string into its equivalent decimal number. For example,

```
Command window

Enter an Octal string: 147
The equivalent decimal number "147" is: 103
```

3.12 RadixN2Dec

Write a program called **RadixN2Dec** to convert an input string of any radix (≤ 16) into its equivalent decimal number.

```
Command window

Enter the radix: 16
Enter the string: 1a
The equivalent decimal number "1a" is: 26
```