yet another insignificant programming notes...   |   [HOME](http://www3.ntu.edu.sg/home/ehchua/programming/index.html)

**Java Programming**

**Java Basics - Exercises**

**Exercises on Flow Controls**

**Exercises on Conditional (Decision)**

Exercise (if-else): Write a program called **CheckMark** which prints "PASS" if the int variable "mark" is more than or equal to 50; or prints "FAIL" otherwise.

Hints:

public class **CheckMark** { // saved as "CheckMark.java"

public static void main(String[] args) {

int mark = 49; // set the value of mark here!

System.out.println("The mark is " + mark);

if ( ...... ) {

System.out.println( ...... );

} else {

System.out.println( ...... );

}

}

}

Exercise (if-else): Write a program called **CheckNumber** which prints "Odd Number" if the int variable “number” is odd, or “Even Number” otherwise.

Hints:

public class **CheckNumber** { // saved as "CheckNumber.java"

public static void main(String[] args) {

int number = 49; // set the value of number here!

System.out.println("The number is " + number);

if ( ...... ) {

System.out.println( ...... );

} else {

System.out.println( ...... );

}

}

}

Exercise (nested-if, switch-case): Write a program called **PrintWord** which prints "ONE", "TWO",... , "NINE", "OTHER" if the int variable "number" is 1, 2,... , 9 or other, respectively. Use (a) a "nested-if" statement; (b) a "switch-case" statement.

Hints:

public class **PrintWord** { // saved as "PrintWord.java"

public static void main(String[] args) {

int number = 5;

// nested-if

if (number == 1) {

System.out.println("ONE");

} else if (......) {

......

} else if (......) {

......

} else {

......

}

// switch-case

switch(number) {

case 1: System.out.println("ONE"); break;

case 2: ......

......

default: System.out.println("OTHER");

}

}

}

**Exercises on Loop (Iteration)**

Exercise (Loop): Write a program called **SumAndAverage** to produce the sum of 1, 2, 3, ..., to 100. Also compute and display the average. The output shall look like:

The sum is 5050

The average is 50.5

Hints:

public class **SumAndAverage** { // saved as "SumAndAverage.java"

public static void main (String[] args) {

int sum = 0; // store the accumulated sum

double average; // average in double

int lowerbound = 100; // the lower bound to sum

int upperbound = 100; // the upper bound to sum

for (int number = lowerbound; number <= upperbound; number++) { // for loop

sum += number; // same as "sum = sum + number"

}

// Compute average in double. Beware that int/int produces int.

......

// Print sum and average.

......

}

}

Try:

1. Modify the program to use a "while-do" loop instead of "for" loop.
2. int number = 1;
3. int sum = 0;
4. while (number <= upperbound) {
5. sum += number;
6. number++;

}

1. Modify the program to use a "do-while" loop.
2. int number = 1;
3. int sum = 0;
4. do {
5. sum += number;
6. number++;

} while (number <= upperbound);

1. Modify the program to sum from 111 to 8989, and compute the average. Introduce an int variable called count to count the numbers in the specified range.
2. int count = 0;
3. for (...; ...; ...) {
4. ......
5. count++;

}

1. Modify the program to sum only the odd numbers from 1 to 100, and compute the average.
2. Modify the program to sum those numbers from 1 to 100 that is divisible by 7, and compute the average.
3. Modify the program to find the "sum of the squares" of all the numbers from 1 to 100, i.e. 1\*1 + 2\*2 + 3\*3 + ... + 100\*100.

Exercise (Loop): Write a program called **Product1toN** 10).×...×3×2×to compute the product of integers 1 to 10 (i.e., 1 Try computing the product from 1 to 11, 1 to 12, 1 to 13 and 1 to 14. Write down the product obtained and explain the results.

Hints: Declares an int variable called product (to accumulate the product) and initialize to 1.

Exercise (Loop): Write a program called **HarmonicSum** to compute the sum of a harmonic series, as shown below, where n=50000. The program shall compute the sum from *left-to-right* as well as from the *right-to-left*. Obtain the difference between these two sums and explain the difference. Which sum is more accurate?

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_HarmonicSum.png

Hints:

public class **HarmonicSum** { // saved as "HarmonicSum.java"

public static void main (String[] args) {

int maxDenominator = 50000;

double sumL2R = 0.0; // sum from left-to-right

double sumR2L = 0.0; // sum from right-to-left

// for-loop for summing from left-to-right

for (int denominator = 1; denominator <= maxDenominator; denominator++) {

......

// Beware that int/int gives int.

}

// for-loop for summing from right-to-left

......

// Find the difference and display

......

}

}

Exercise (Loop & Condition): Write a program called **ComputePI** to compute the value of π, using the following series expansion. You have to decide on the termination criterion used in the computation (such as the number of terms used or the magnitude of an additional term). Is this series suitable for computing π?

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_ComputePI.png

JDK maintains the value of π in a double constant called Math.PI. Compare the values obtained, in terms of the ratio between the value computed and the Math.PI, in percents.

Hint: Add to sum if the denominator modulus 4 is 1, and subtract from sum if it is 3.

double sum = 0;

int maxDenom = 10000000;

for (int denom = 1; ..... ; denom = denom + 2) {

if (denom % 4 == 1) {

sum += ......;

} else if (denom % 4 == 3) {

sum -= ......;

} else {

System.out.println("The computer has gone crazy?!");

}

}

Exercise (Loop & Condition): Write a program called **CozaLozaWoza** which prints the number 1 to 110, 11 numbers per line. The program shall print "Coza" in place of the numbers which are multiples of 3, "Loza" for multiples of 5, "Woza" for multiples of 7, "CozaLoza" for multiples of 3 and 5, and so on. The output shall look like:

1 2 Coza 4 Loza Coza Woza 8 Coza Loza 11

Coza 13 Woza CozaLoza 16 17 Coza 19 Loza CozaWoza 22

23 Coza Loza 26 Coza Woza 29 CozaLoza 31 32 Coza

......

Hints:

public class **CozaLozaWoza** { // saved as "CozaLozaWoza.java"

public static void main(String[] args) {

int lowerbound = 1;

int upperbound = 110;

for (int number = lowerbound; number <= upperbound; number++) {

// Print "Coza" if number is divisible by 3

if (......) {

System.out.print("Coza");

}

// Print "Loza" if number is divisible by 5

if (......) {

System.out.print(.....);

}

// Print "Woza" if number is divisible by 7

......

// Print the number if it is not divisible by 3, 5 and 7

if (......) {

......

}

// Print a space

......

// Print a newline if number is divisible by 11

if (......) {

System.out.println();

}

}

}

}

Exercise (Loop): Write a program called **Fibonacci** to display the first 20 Fibonacci numbers F(n), where F(n)=F(n–1)+F(n–2) and F(1)=F(2)=1. Also compute their average. The output shall look like:

The first 20 Fibonacci numbers are:

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

The average is 885.5

Hints:

public class **Fibonacci** {

public static void main (String args[]) {

int n = 3; // the index n for F(n), starting from n=3

int fn; // F(n) to be computed

int fnMinus1 = 1; // F(n-1), init to F(2)

int fnMinus2 = 1; // F(n-2), init to F(1)

int nMax = 20; // maximum n, inclusive

int sum = fnMinus1 + fnMinus2;

double average;

System.out.println("The first " + nMax + " Fibonacci numbers are:");

......

while (n <= nMax) {

// Compute F(n), print it and add to sum

......

// Adjust the index n and shift the numbers

......

}

// Compute and display the average (=sum/nMax)

......

}

}

Tribonacci numbers are a sequence of numbers T(n) similar to Fibonacci numbers, except that a number is formed by adding the three previous numbers, i.e., T(n)=T(n-1)+T(n-2)+T(n-3), T(1)=1, T(2)=1 and T(3)=2. Write a program called **Tribonacci** to produce the first twenty Tribonacci numbers.

**Exercises on Nested-Loop**

Exercise (nested-loop): Write a program called **SquareBoard** that displays the following 5x5 pattern using two nested for-loops.

# # # # #

# # # # #

# # # # #

# # # # #

# # # # #

Your program should use only two output statements, one each of the followings:

System.out.print("# "); // print "# ", without newline

System.out.println(); // print a newline

Hints:

public class **SquareBoard** { // saved as "SquareBoard.java"

public static void main (String[] args) {

int size = 5; // size of the board

for (int row = 1; ......; ......) {

for (int col = 1; ......; ......) {

......

}

......

}

}

}

Exercise (nested-loop): Write a program called **CheckerBoard** that displays the following 7x7 checkerboard pattern using two nested for-loops.

# # # # # # #

# # # # # # #

# # # # # # #

# # # # # # #

# # # # # # #

# # # # # # #

# # # # # # #

Your program should use only three output statements, one each of the followings:

System.out.print("# "); // print "# ", without newline

System.out.print(" "); // print a space, without newline

System.out.println(); // print a newline

Hints:

public class **CheckerBoard** { // saved as "CheckerBoard.java"

public static void main (String[] args) {

int size = 7; // size of the board

for (int row = 1; ......; ......) {

// Use modulus 2 to find alternate lines

if ((row % 2) == 0) { // row 2, 4, 6, 8

......

}

for (int col = 1; ......; ......) {

......

}

......

}

}

}

Exercise (nested-loop): Write a program called TimeTable to produce the multiplication table of 1 to 9 as shown using two nested for-loops:

\* | 1 2 3 4 5 6 7 8 9

-------------------------------

1 | 1 2 3 4 5 6 7 8 9

2 | 2 4 6 8 10 12 14 16 18

3 | 3 6 9 12 15 18 21 24 27

4 | 4 8 12 16 20 24 28 32 36

5 | 5 10 15 20 25 30 35 40 45

6 | 6 12 18 24 30 36 42 48 54

7 | 7 14 21 28 35 42 49 56 63

8 | 8 16 24 32 40 48 56 64 72

9 | 9 18 27 36 45 54 63 72 81

Modify the program to print the multiplication table of 1 to 12.

**Exercises on Input**

Exercise (Keyboard Input): Write a program called KeyboardInput to prompt user for an int, a double, and a String. The output shall look like (the inputs are shown in bold):

Enter an integer: **12**

Enter a floating point number: **33.44**

Enter your name: **Peter**

Hi! Peter, the sum of 12 and 33.44 is 45.44

Hints:

import java.util.Scanner; // needed to use Scanner for input

public class KeyboardInput {

public static void main(String[] args) {

int num1;

double num2;

String name;

double sum;

// Setup a Scanner called in to scan the keyboard (System.in)

Scanner in = new Scanner(System.in);

System.out.print("Enter an integer: ");

num1 = in.nextInt(); // use nextInt() to read int

System.out.print("Enter a floating point number: ");

num2 = in.nextDouble(); // use nextDouble() to read double

System.out.print("Enter your name: ");

name = in.next(); // use next() to read String

// Display

......

}

}

Exercise (File Input): Write a program called **FileInput** to read an int, a double, and a String form a text file called "in.txt", and produce the following output:

The integer read is 12

The floating point number read is 33.44

The String read is "Peter"

Hi! Peter, the sum of 12 and 33.44 is 45.44

You need to create a text file called "in.txt" (in Eclipse, right-click on the "project" ⇒ "New" ⇒ "File") with the following contents:

12

33.44

Peter

import java.util.Scanner; // Needed to use Scanner for input

import java.io.File; // Needed to use File

import java.io.FileNotFoundException; // Needed for file operation

public class FileInput {

public static void main(String[] args)

throws FileNotFoundException { // Needed for file operation

int num1;

double num2;

String name;

double sum;

// Setup a Scanner to read from a text file

Scanner in = new Scanner(new File("in.txt"));

num1 = in.nextInt(); // use nextInt() to read int

num2 = in.nextDouble(); // use nextDouble() to read double

name = in.next(); // use next() to read String

// Display

......

}

}

Exercise (User Input): Write a program called **CircleComputation**, which prompts user for a radius and compute the area and perimeter of a circle. The output shall look like:

Enter the radius: **1.2**

The area is 4.5239

The perimeter is 7.5398223686155035

Hints: π is kept in a constant called Math.PI.

Exercise (User Input & String Operations): Write a program called ReverseString, which prompts user for a String, and prints the *reverse* of the String. The output shall look like:

Enter a String: **abcdef**

The reverse of String "abcdef" is "fedcba".

Hints:

import java.util.Scanner;

public class ReverseString {

public static void main(String[] args) {

String inStr; // input String

int inStrLen; // length of the input String

Scanner in = new Scanner(System.in);

System.out.print("Enter a String: ");

inStr = in.next(); // use next() to read String

inStrLen = inStr.length();

// Use inStr.charAt(index) to extract character at 'index' from inStr

......

}

}

For a String called inStr, you can use inStr.length() to retrieve the *length* of the String; and inStr.charAt(index) to retrieve the char at the index position, where index begins with 0.

Exercise (User Input & String Operations): 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 digits. Use a nested-if in this exercise. Modify your program to use an *array* for table look-up later.

Hints: You can use in.next().toLowerCase() to read a string and convert it to lowercase to reduce your cases.

Exercise (Palindrome): 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".

Hints: Read in a word and convert to lowercase via in.next().toLowercase().

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 test palindromic phrase.

Hints: Read in the lowercase phrase via in.nextLine().toLowercase(). Maintain two indexes, forwardIndex and backwardIndex, used to scan the phrase forward and backward.

Exercise (Bin2Dec): Write a program called Bin2Dec to convert a binary string into its equivalent decimal number. Your output shall look like:

Enter a Binary string: 1011

The equivalent decimal number for binary "1011" is 11

Enter a Binary string: 1234

Error: Invalid Binary String "1234"

Hints: For a n-bit binary number bn-1bn-2...b1b0, bi∈{0,1}, the equivalent decimal number is bn-1×2n-1+bn-2×2n-2+ ...+b1×21+b0×20.

import java.util.Scanner;

public class Bin2Dec {

public static void main(String[] args) {

String binStr; // input binary string

int binStrLen; // length of the input string

int dec = 0; // equivalent decimal number

char binChar; // each individual char in the binary string

Scanner in = new Scanner(System.in);

// Read input binary string

......

// Convert binary string into Decimal

......

}

}

You can use JDK method Math.pow(x, y) to compute the x raises to the power of y. This method takes two doubles as argument and returns a double. You may have to cast the result back to int.

To convert a char (of digit '0' to '9') to int, simply subtract by '0', e.g., '5'-'0'→5.

Exercise (Hex2Dec): Write a program called HexToDec to convert a hexadecimal string into its equivalent decimal number. Your output shall look like:

Enter a Hexadecimal string: 1a

The equivalent decimal number for hexadecimal "1a" is 26

Enter a Hexadecimal string: 1y3

Error: Invalid Hexadecimal String "1y3"

Hints: For a n-digit hexadecimal number hn-1hn-2...h1h0, hi∈{0,…,9,A,…,F}, the equivalent decimal number is hn-1×16n-1+hn-2×16n-2+ ...+h1×161+h0×160.

You do not need a big nested-if statement of 16 cases (or 22 with upper and lower letters). Extract the individual character from the hexadecimal string, says c. If char c is between '0' to '9', you can get the integer offset via c-'0'. If c is between 'a' to 'f' or 'A' to 'F', the integer offset is c-'a'+10 or c-'A'+10.

String hexStr;

char hexChar;

......

hexChar = hexStr.charAt(i);

......

if (hexChar >= '0' && hexChar <= '9') {

... (hexChar-'0') ...

...

} else if (hexChar >= 'a' && hexChar <= 'f') { // lowercase

... (hexChar-'a'+10) ...

...

} else if (hexChar >= 'A' && hexChar <= 'F') { // uppercase

... (hexChar-'A'+10) ...

...

} else {

System.out.println("Error: Invalid hexadecimal string");

System.exit(1); // quit the program

}

**Exercises on Array**

Exercise (Array): Write a program called GradesAverage, which reads in n grades (of int between 0 and 100) and displays the average. You should keep the grades in an int[] (an array of int). Your output shall look like:

Enter the number of students: **3**

Enter the grade for student 1: **55**

Enter the grade for student 2: **108**

Invalid grade, try again...

Enter the grade for student 2: **56**

Enter the grade for student 3: **57**

The average is 56.0

Exercise (Array): Write a program called Hex2Bin to convert a hexadecimal string into its equivalent binary string. The output shall look like:

Enter a Hexadecimal string: **1abc**

The equivalent binary for hexadecimal "1abc" is 0001 1010 1011 1100

Hints: Use an array of 16 binary Strings corresponding to hexadecimal number '0' to 'F', as follows:

String[] hexBits = {"0000", "0001", "0010", "0011",

"0100", "0101", "0110", "0111",

"1000", "1001", "1010", "1011",

"1100", "1101", "1110", "1111"}

**Exercises on Command-line Arguments**

Exercise (Command-line arguments): Write a Java program called **Arithmetic** that takes three command-line arguments: two integers followed by an arithmetic operator (+, -, \* or /). The program shall perform the corresponding operation on the two integers and print the result. For example:

> java Arithmetic **3 2 +**

3+2=5

> java Arithmetic **3 2 -**

3-2=1

> java Arithmetic **3 2 /**

3/2=1

Hints:

The method main(String[] args) has a parameter: "an array of String", which is often (but not necessary) named args. This parameter captures the command-line arguments supplied by the user when the program is invoked. For example, if a user invokes:

> java Arithmetic **12345 4567 +**

The three command-line arguments "12345", "4567" and "+" will be captured in a String array and passed into the main() method as parameter args. That is,

args is {"12345", "4567", "+"}; // args is a String array

args.length is 3; // length of the array

args[0] is "12345"; // 1st element of the String array

args[1] is "4567"; // 2nd element of the String array

args[2] is "+"; // 3rd element of the String array

args[0].length() is 5; // length of 1st String element

args[1].length() is 4; // length of the 2nd String element

args[2].length() is 1; // length of the 3rd String element

public class **Arithmetic** {

public static void main (String[] args) {

int operand1, operand2;

char theOperator;

// Check if there are 3 command-line arguments in the

// String array args[] by using length variable of array.

if (args.length != 3) {

System.err.println("Usage: java Arithmetic int1 int2 op");

return;

}

// Convert the 3 Strings args[0], args[1], args[2] to int and char.

// Use the Integer.parseInt(aStr) to convert a String to an int.

operand1 = Integer.parseInt(args[0]);

operand2 = ......

// Get the operator, assumed to be the first character of

// the 3rd string. Use method charAt() of String.

theOperator = args[2].charAt(0);

System.out.print(args[0] + args[2] + args[1] + "=");

switch(theOperator) {

case ('-'):

System.out.println(operand1 – operand2); break;

case ('+'): ......

case ('\*'): ......

case ('/'): ......

default:

System.err.println("Error: invalid operator!");

}

}

}

Notes:

* To provide command-line arguments, use the "cmd" shell to run your program in the form "java *ClassName* *arg1* *arg2* ....".
* To provide command-line arguments in Eclipse, right click the source code ⇒ "Run As" ⇒ "Run Configurations..." ⇒ Select "Main" and choose the proper main class ⇒ Select "Arguments" ⇒ Enter the command-line arguments, e.g., "3 2 +" in "Program Arguments".
* To provide command-line arguments in Netbeans, right click the "Project" name ⇒ "Set Configuration" ⇒ "Customize..." ⇒ Select categories "Run" ⇒ Enter the command-line arguments, e.g., "3 2 +" in the "Arguments" box (but make sure you select the proper Main class).

Question: Try "java Arithmetic 2 4 \*" (in CMD shell and Eclipse/Netbeans) and explain the result obtained. How to resolve this problem?

Exercise (Command-line arguments): Write a Java program called SumDigits to sum up the individual digits of a positive integer, given in the command line. The output shall look like:

> **java SumDigits 12345**

The sum of digits = 1 + 2 + 3 + 4 + 5 = 15

**Exercises on Method**

Exercise (Method): Write a program called **GradesStatistics**, which reads in n grades (of int between 0 and 100, inclusive) and displays the *average*, *minimum*, *maximum*, and *standard deviation*. Your program shall check for valid input. You should keep the grades in an int[] and use a method for each of the computations. Your output shall look like:

Enter the number of students: **4**

Enter the grade for student 1: **50**

Enter the grade for student 2: **51**

Enter the grade for student 3: **56**

Enter the grade for student 4: **53**

The average is 52.5

The minimum is 50

The maximum is 56

The standard deviation is 2.29128784747792

Hints: The formula for calculating standard deviation is:

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_GradesAverage.png

public class **GradesStatistics** {

public static int[] grades; // Declare an int[], to be allocated later

// main() method

public static void main(String[] args) {

readGrades();

System.out.println("The average is " + average());

System.out.println("The minimum is " + min());

System.out.println("The maximum is " + max());

System.out.println("The standard deviation is " + stdDev());

}

// Prompt user for the number of students and allocate the "grades" array.

// Then, prompt user for grade, check for valid grade, and store in "grades".

public static void readGrades() { ....... }

// Return the average value of int[] grades

public static double average() { ...... }

// Return the maximum value of int[] grades

public static int max() { ...... }

// Return the minimum value of int[] grades

public static int min() { ....... }

// Return the standard deviation of the int[] grades

public static double stdDev() { ....... }

}

Exercise (Method): Write a program called **GradesHistogram**, which reads in n grades (of int between 0 and 100, inclusive) from a text file called "grades.in" and displays the histogram. The file has the following format:

numStduents:int

grade1:int grade2:int .... gradeN:int

For example:

15

49 50 51 59 0 5 9 10 15 19 50 55 89 99 100

The output shall consist of a horizontal histogram and a vertical histogram as follows:

0 - 9: \*\*\*

10 - 19: \*\*\*

20 - 29:

30 - 39:

40 - 49: \*

50 - 59: \*\*\*\*\*

60 - 69:

70 - 79:

80 - 89: \*

90 -100: \*\*

\*

\*

\* \* \*

\* \* \* \*

\* \* \* \* \* \*

0-9 10-19 20-29 30-39 40-49 50-59 60-69 70-79 80-89 90-100

Hints:

public class **GradesHistogram** {

public static int[] grades;

// Declare an int array of grades, to be allocated later

public static int[] bins = new int[10];

// Declare and allocate an int array for histogram bins.

// 10 bins for 0-9, 10-19,...., 90-100

public static void main(String[] args) {

readGrades("grades.in");

computeHistogram();

printHistogramHorizontal();

printHistogramVertical();

}

// Read the grades from "filename", store in "grades" array.

// Assume that the inputs are valid.

public static void readGrades(String filename) { ...... }

// Based on "grades" array, populate the "bins" array.

public static void computeHistogram() { ....... }

// Print histogram based on the "bins" array.

public static void printHistogramHorizontal() { ...... }

// Print histogram based on the "bins" array.

public static void printHistogramVertical() { ...... }

}

Exercise (Method): Write a method called reverseArray() with the following signature:

public static void reverseArray(int[] intArray)

The method accepts an int array, and reverses its orders. For example, if the input array is {12, 56, 34, 79, 26}, the reversal is {26, 79, 34, 56, 12}. You MUST NOT use another array in your method (but you need a temporary variable to do the swap). Also write a test class called ReverseArrayTest to test this method. Take note that the array passed into the method can be modified by the method (this is called "pass by reference"). On the other hand, primitives passed into a method cannot be modified. This is because a clone is created and passed into the method instead of the original copy (this is called "pass by value").

**More Basic Java Exercises**

Exercise (JDK Source Code): Extract the source code of the class Math from the JDK source code ("$JAVA\_HOME" ⇒ "src.zip" ⇒ "Math.java" under folder "java.lang"). Study how constants such as E and PI are defined. Also study how methods such as abs(), max(), min(), toDegree(), etc, are written.

Exercise (Matrix Operation): Similar to Math class, write a Matrix library that supports matrix operations (such as addition, subtraction, multiplication) via 2D arrays. The operations shall support both doubles and ints. Also write a test class to exercise all the operations programmed.

Hints:

public class **Matrix** {

public static void printMatrix(int[][] m) { ...... }

public static void printMatrix(double[][] m) { ...... }

public static boolean haveSameDimension(int[][] m1, int[][] m2) { ...... }

public static boolean haveSameDimension(double[][] m1, double[][] m2) { ...... }

public static int[][] add(int[][] m1, int[][] m2) { ...... }

public static double[][] add(double[][] m1, double[][] m2) { ...... }

......

}

Exercise (Special Characters and Escape Sequences): Write a program called **PrintAnimalPattern**, which uses println() to produce this pattern:

'\_\_'

(©©)

/========\/

/ || %% ||

\* ||----||

¥¥ ¥¥

"" ""

Hints:

* Use escape sequence \uhhhh where hhhh are four hex digits to display Unicode characters such as ¥ and ©. ¥ is 165 (00A5H) and © is 169 (00A9H) in both ISO-8859-1 (Latin-1) and Unicode character sets.
* Double-quote (") and black-slash (\) require escape sign inside a String. Single quote (') does not require escape sign.

Print the same pattern using printf(). Hints: Need to use %% to print a % in printf() because % is the suffix for format specifier.

Exercise (Print Pattern using nested-loop): Write a method to print each of the followings patterns using nested loops in a class called **PrintPattern**. The signatures of the methods are:

public static void printPatternX(int size) // 'X' from 'A' to ..., size is a positive integer.

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # #

(a) (b) (c) (d)

Hints: On the diagonal, row = col. On the opposite diagonal, row + col = size + 1.

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

# # # # # # # # # #

# # # # # # # # # #

# # # # # # # #

# # # # # # # # # #

# # # # # # # # # #

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

(e) (f) (g) (h) (i)

# # # # # # # # # # # # #

# # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # # # # #

# # # # # # # # # # # # # # # # # # # # # # #

(j) (k) # # # # # # # # #

# # # # # # #

# # # # #

# # #

#

(l)

1 1 2 3 4 5 6 7 8 1 8 7 6 5 4 3 2 1

1 2 1 2 3 4 5 6 7 2 1 7 6 5 4 3 2 1

1 2 3 1 2 3 4 5 6 3 2 1 6 5 4 3 2 1

1 2 3 4 1 2 3 4 5 4 3 2 1 5 4 3 2 1

1 2 3 4 5 1 2 3 4 5 4 3 2 1 4 3 2 1

1 2 3 4 5 6 1 2 3 6 5 4 3 2 1 3 2 1

1 2 3 4 5 6 7 1 2 7 6 5 4 3 2 1 2 1

1 2 3 4 5 6 7 8 1 8 7 6 5 4 3 2 1 1

(m) (n) (o) (p)

1 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1

1 2 1 1 2 3 4 5 6 7 6 5 4 3 2 1

1 2 3 2 1 1 2 3 4 5 6 5 4 3 2 1

1 2 3 4 3 2 1 1 2 3 4 5 4 3 2 1

1 2 3 4 5 4 3 2 1 1 2 3 4 3 2 1

1 2 3 4 5 6 5 4 3 2 1 1 2 3 2 1

1 2 3 4 5 6 7 6 5 4 3 2 1 1 2 1

1 2 3 4 5 6 7 8 7 6 5 4 3 2 1 1

(q) (r)

1 1 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1

1 2 2 1 1 2 3 4 5 6 7 7 6 5 4 3 2 1

1 2 3 3 2 1 1 2 3 4 5 6 6 5 4 3 2 1

1 2 3 4 4 3 2 1 1 2 3 4 5 5 4 3 2 1

1 2 3 4 5 5 4 3 2 1 1 2 3 4 4 3 2 1

1 2 3 4 5 6 6 5 4 3 2 1 1 2 3 3 2 1

1 2 3 4 5 6 7 7 6 5 4 3 2 1 1 2 2 1

1 2 3 4 5 6 7 8 7 6 5 4 3 2 1 1 1

(s) (t)

1

2 3 2

3 4 5 4 3

4 5 6 7 6 5 4

5 6 7 8 9 8 7 6 5

6 7 8 9 0 1 0 9 8 7 6

7 8 9 0 1 2 3 2 1 0 9 8 7

8 9 0 1 2 3 4 5 4 3 2 1 0 9 8

(u)

Exercise (Print Pattern using nested-loop): Write a method to print each of the following patterns using nested-loops in a class called **PrintTriangle**. The signatures of the methods are:

public static void printXxx(int numRows) // Xxx is the pattern's name

Write the main() which prompts the user for the numRows and prints all the patterns.

1

1 2 1

1 2 4 2 1

1 2 4 8 4 2 1

1 2 4 8 16 8 4 2 1

1 2 4 8 16 32 16 8 4 2 1

1 2 4 8 16 32 64 32 16 8 4 2 1

1 2 4 8 16 32 64 128 64 32 16 8 4 2 1

(a) PowerOf2Triangle

1 1

1 1 1 1

1 2 1 1 2 1

1 3 3 1 1 3 3 1

1 4 6 4 1 1 4 6 4 1

1 5 10 10 5 1 1 5 10 10 5 1

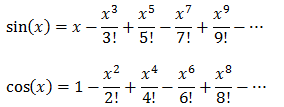
1 6 15 20 15 6 1 1 6 15 20 15 6 1

(b) PascalTriangle1 (c) PascalTriangle2

Exercise (Series): Write a method to compute sin(x) and cos(x) using the following series expansion, in a class called **TrigonometricSeries**. The headers of the methods are:

public static double sin(double x, int numTerms) // x in radians

public static double cos(double x, int numTerms)



Compare the values computed using the series with the JDK methods Math.sin(), Math.cos() at x=0, π/6, π/4, π/3, π/2 using various numbers of terms.

Hints: Avoid generating large numerator and denominator (which may cause arithmetic overflow, e.g., 13! is out of int range). Compute the terms as:

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_TrigonometricSeriesHint.png

Exercise (Series): Write a method to compute the sum of the series in a class called Series. The signature of the method is:

public static double sumOfSeries(double x, int numTerms)

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_Series.png

Exercise (Fibonacci Number & Overflow) : Write a program 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:

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:

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-1) + F(n-2) > Integer.MAX\_VALUE to check for overflow. Instead, overflow occurs for F(n) if (Integer.MAX\_VALUE – F(n-1)) < F(n-2) (i.e., no room for the next Fibonacci number).

Write a similar program for Tribonacci numbers.

Exercise (Factorial & Overflow): Write a program called **Factorial1to10**, to compute the factorial of n, for 1≤n≤10. Your output shall look like:

The factorial of 1 is 1

The factorial of 2 is 2

...

The factorial of 10 is 3628800

Modify your program (called **FactorialInt**), to list all the factorials, that can be expressed as an int (i.e., 32-bit signed integer). Your output shall look like:

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: Overflow occurs for Factorial(n+1) if (Integer.MAX\_VALUE / Factorial(n)) < (n+1).

Modify your program again (called **FactorialLong**) to list all the factorials 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.

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

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

Write a program called NumberConvert, which prompts the user for an input number, an input radix, and an output radix, and display the converted number. The output shall look like:

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.

Exercise (Guess a Number): Write a Java program called **NumberGuess** 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:

> **java NumberGuess**

Key in your guess:

**50**

Try higher

**70**

Try lower

**65**

Try lower

"

You got it in 4 trials!

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

int secretNumber = (int)(Math.random()\*100);

Exercise (Guess a Word): Write a program called WordGuess to guess a word by trying to guess the individual characters. The word to be guessed shall be provided using the command-line argument. Your program shall look like:

**java WordGuess testing**

Key in one character or your guess word: **t**

Trail 1: t\_\_t\_\_\_

Key in one character or your guess word: **g**

Trail 2: t\_\_t\_\_g

Key in one character or your guess word: **e**

Trail 3: te\_t\_\_g

Key in one character or your guess word: **testing**

Trail 4: Congratulation!

You got in 4 trials

Hints:

* Set up a boolean array to indicate the positions of the word that have been guessed correctly.
* 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 retrieving the word to be guessed from a text file (or a dictionary) randomly.

Exercise (Day of the Week): Write a program called DayOfWeek, which takes a date (in year, month and day), and returns the day of the week.

There is an interesting algorithm for finding the day of week given year, month and day (e.g., 26-9-2010), as follows:

1. Take the last two digit of the year, and add a quarter (divide by 4 and discard the remainder). In our example, 10 + 10/4 = 12
2. Add a value according to the month as follow: Jan: 1, Feb: 4, Mar: 4, Apr: 0, May: 2, Jun: 5, Jul: 0, Aug: 3, Sep: 6, Oct: 1, Nov: 4, Dec: 6. For our example, 12 + 6 (Sep) = 18.
3. Add the day. For our example, 18 + 26 = 44
4. Add a century offset according to century value as follows: 18xx:2, 19xx: 0, 20xx: 6, 21xx: 4. For our example, 44 + 6 (20xx) = 50. For years outside this range, add or subtract 400 to bring the year into this range. This is based on the fact that the calendar repeats every 400 years.
5. For leap year (a leap year is a year that is divisible by 4 and not divisible by 100, or divisible by 400), if month is Jan or Feb, subtract 1. For our example, 2010 is not a leap year.
6. Take modulus 7, and retrieve the day of the week from the array {Sat, Sun, Mon, Tues, Wed, Thurs, Fir}. For our example, 50 % 7 = 1, which is a Sunday.

You can compare the day obtain with the Java's Calendar output as follows:

// Construct a Calendar with the given year, month and day

Calendar cal = new GregorianCalendar(year, month - 1, day); // month is 0-based

// Get the day of the week number: 1 (Sunday) to 7 (Saturday)

int dayNumber = cal.get(Calendar.DAY\_OF\_WEEK);

String[] calendarDays = { "Sunday", "Monday", "Tuesday", "Wednesday",

"Thursday", "Friday", "Saturday" };

// Print result

System.out.println("It is " + calendarDays[dayNumber - 1]);

This above algorithm work for Gregorian dates only. The calendar we used today is known as *Gregorian calendar*, which came into effect in October 15, 1582 in some countries and later in other countries. It replaces the *Julian calendar*. 10 days were removed from the calendar, i.e., October 4, 1582 (Julian) was followed by October 15, 1582 (Gregorian). The only difference between the Gregorian and the Julian calendar is the "leap-year rule". In Julian calendar, every four years is a leap year. In Gregorian calendar, a leap year is a year that is divisible by 4 but not divisible by 100, or it is divisible by 400, i.e., the Gregorian calendar omits century years which are not divisible by 400. Furthermore, Julian calendar considers the first day of the year as march 25th, instead of January 1st.

It is difficult to modify the above algorithm to handle pre-Gregorian dates. A better algorithm is to find the number of days from a known date.

**Exercises on Number Theory**

Exercise (Perfect and Deficient Numbers): A positive integer is called a *perfect number* if the sum of all its factors (excluding the number itself, i.e., proper divisor) is equal to its value. For example, the number 6 is perfect because its proper divisors are 1, 2, and 3, and 6=1+2+3; but the number 10 is not perfect because its proper divisors are 1, 2, and 5, and 10≠1+2+5.

A positive integer is called a *deficient number* if the sum of all its proper divisors is less than its value. For example, 10 is a deficient number because 1+2+5<10; while 12 is not because 1+2+3+4+6>12.

Write a Java method called isPerfect(int posInt) that takes a positive integer, and return true if the number is perfect. Similarly, write a Java method called isDeficient(int posInt) to check for deficient numbers.

Using the methods, write a program called PerfectNumberList that prompts user for an upper bound (a positive integer), and lists all the perfect numbers less than or equal to this upper bound. It shall also list all the numbers that are neither deficient nor perfect. The output shall look like:

Enter the upper bound: **1000**

These numbers are perfect:

6 28 496

[3 perfect numbers found (0.30%)]

These numbers are neither deficient nor perfect:

12 18 20 24 30 36 40 42 48 54 56 60 66 70 72 78 80 ......

[246 numbers found (24.60%)]

Exercise (Prime): A positive integer is a *prime* if it is divisible by 1 and itself only. Write a Java method called isPrime(int posInt) that takes a positive integer and returns true if the number is a prime. Write a Java program called PrimeList that prompts the user for an upper bound (a positive integer), and lists all the primes less than or equal to it. Also display the percentage of prime (up to 2 decimal places). The output shall look like:

Please enter the upper bound: **10000**

1

2

3

......

......

9967

9973

[1230 primes found (12.30%)]

Hints: To check if a number n is a prime, the simplest way is try dividing n by 2 to √n.

Exercise: Write a method isProductOfPrimeFactors(int posInt) that takes a positive integer, and return true if the product of all its prime factors (excluding 1 and the number itself) is equal to its value. For example, the method returns true for 30 (30=2×3×5) and false for 20 (20≠2×5). You may need to use the isPrime() method in the previous exercise.

Write a program called PerfectPrimeFactorList that prompts user for an upper bound. The program shall display all the numbers (less than or equal to the upper bound) that meets the above criteria. The output shall look like:

Enter the upper bound: **100**

These numbers are equal to the product of prime factors:

1 6 10 14 15 21 22 26 30 33 34 35 38 39 42 46 51 55 57 58 62 65 66 69 70 74 77 78 82 85 86 87 91 93 94 95

[36 numbers found (36.00%)]

Exercise (GCD): One of the earlier known algorithms is the Euclid algorithm to find the GCD of two integers (developed by the Greek Mathematician Euclid around 300BC). By definition, GCD(a, b) is the greatest factor that divides both a and b. Assume that a and b are positive integers, and a≥b, the Euclid algorithm is based on these two properties:

GCD(a, 0) = a

GCD(a, b) = GCD(b, a mod b), where (a mod b) denotes the remainder of a divides by b.

For example,

GCD(15, 5) = GCD(5, 0) = 5

GCD(99,88) = GCD(88,11) = GCD(11,0) = 11

GCD(3456,1233) = GCD(1233,990) = GCD(990,243) = GCD(243,18) = GCD(18,9) = GCD(9,0) = 9

The pseudocode for the Euclid algorithm is as follows:

GCD(a, b) // assume that a ≥ b

while (b != 0) {

// Change the value of a and b: a ← b, b ← a mod b, and repeat until b is 0

temp ← b

b ← a mod b

a ← temp

}

// after the loop completes, i.e., b is 0, we have GCD(a, 0)

GCD is a

Write a method called gcd() with the following signature:

public static int gcd(int a, int b)

Your methods shall handle arbitrary values of a and b, and check for validity.

TRY: Write a *recursive* version called gcdRecursive() to find the GCD.

Latest version tested: JDK 1.6  
Last modified: September, 2010

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg)   |   [HOME](http://www3.ntu.edu.sg/home/ehchua/programming/index.html)