

Forum:[Hack Forums](#)

Thread: [Compilation of Java Exercises for Beginners](#)

Mody - April 20, 2015, 6:44 am

Lately people have been asking for Java exercises and assignments, basically something they can work on and test their skills. So, I thought it would be a good idea to make a compilation of Java exercises for beginners.

This compilation covers the following topics:

- Data Types

- Decision Statements

- Loops

- Arrays

- Multi-dimensional Arrays

- Methods

- Recursion

- Classes and Objects

The exercises are for absolute beginners. However, it gradually gets more advanced with each topic.

Sorry about the multiple posts, I couldn't keep everything in one post. It's weird how fast you can run out of characters. You'd think that 8,000 character are more than enough.

Note: All user's inputs are in green.

? Data Types

Exercise #1

[\(Click to View\)](#)

Given a double from user, calculate the square root and the quad root and print the result.

Sample Run 1:

Number: 81

Square Root: 9.00000

Quad Root: 3.00000

Sample Run 2:

Number: 625

Square Root: 25.00000

Quad Root: 5.00000

Exercise #2

[\(Click to View\)](#)

Ask a user for a number then print the ceiling and floor.

Sample Run 1:

Number: 5.6

Ceiling: 6

Floor: 5

Sample Run 2:

Number: 8.4

Ceiling: 9

Floor: 8

Exercise #3

[\(Click to View\)](#)

Write a java program that for a given number of apples, tells the user how many dozens of apples(s) he or she has and how many extra apples are left over.

For Example: If the number of apples = 40, the output should be: 3 dozens and 4 apples.

Sample Run 1:

Apples: 40

Dozens: 3

Left: Over 4

Sample Run 2:

Apples: 1

Dozens: 0

Left Over: 1

Exercise #4

[\(Click to View\)](#)

Write Java program to allow the user to input two integer values and then the program prints the results of adding, subtracting, multiplying, and dividing among the two values.

Sample Run 1:

Enter First Value: 30

Enter Second Value: 10

Adding Result: 40

Subtracting Result: 20

Multiplying Result: 300

Dividing Result: 3.0

Sample Run 2:

Enter First Value: -28

Enter Second Value: 14

Adding Result: -14

Subtracting Result: -42

Multiplying Result: -392

Dividing Result: -2.0

Exercise #5

[\(Click to View\)](#)

Write Java program to allow the user to input temperature in Celsius and convert it into Fahrenheit.

Sample Run 1:

Enter a Temperature in Celsius: 25

The Temperature in Fahrenheit: 77

Sample Run 2:

Enter a Temperature in Celsius: -25

The Temperature in Fahrenheit: -13.0

Exercise #6

[\(Click to View\)](#)

Write a simple program that will compute the amount of interest that is earned on an investment over a period of one year.

The initial amount of the investment and the interest rate are inputs from the user.

Sample Run 1:

Enter the initial investment: 30000

Enter the annual interest rate: 5

The value of the investment after one year is: 31500

Exercise #7

[\(Click to View\)](#)

The equivalent resistance of resistors connected in series is calculated by adding the resistances of individual resistors:

$$R_{eq} = R_1 + R_2$$

The formula for resistors connected in parallel is a little more complex. Given two resistors with resistances R_1 and R_2 connected in parallel the equivalent resistance is given by the inverse of the sum of the inverses:

$$(1/R_{eq}) = (1/R_1) + (1/R_2)$$

Write a program that given two resistances (R_1, R_2) outputs the equivalent resistance (R_{eq}) when they are connected in series and when they are connected in parallel.

Sample Run 1:

Enter R1: 1

Enter R1: 1

Connected in Series: 2.0

Connected in Parallel: 0.5

Sample Run 2:

Enter R1: 0.33

Enter R2: 11.778

Connected in Series: 12.108
Connected in Parallel: 0.321005946481665

Exercise #8 [\(Click to View\)](#)

Write a Java program Count Change to count change. Given the number of quarter, dimes, nickels and cents the program should output the total as a single value in dollars and cents. One dollar corresponds to 100 cents.

One quarter corresponds to 25 cents

One dime corresponds to 10 cents

One nickel corresponds to 5 cents

For example if we have: 3 quarters, 2 dimes, 1 nickel and 6 cents. The total is 1.06 dollars.

Sample Run 1:
Quarters: 3
Dimes: 2
Nickels: 1
Cents: 6
Total Value: 1.06

Sample Run 2:
Quarters: 2
Dimes: 4
Nickels: 55
Cents: 663
Total Value: 10.28

Exercise #9 [\(Click to View\)](#)

Write a single print statement that prints:

"Some where over / the vast ocean \ 'there must' be a land where my thoughts / will meet the distant shore ..." complained a "lonely and longing" singer.

Note: The double quotes are included in the print.

Exercise #10 [\(Click to View\)](#)

Write a program that computes the area and volume of a sphere using its radius as in the following formulas:

$R = \text{Radius (User's input)}$
 $\text{Area} = 4 \text{ PI } r^2$
 $\text{Volume} = (4/3) \text{ PI } r^3 ,$

Sample Run 1:
Enter Radius: 10.5
Area is: 1385.44
Volume is: 4849.59

? Decision Statements

Exercise #1
[\(Click to View\)](#)

A year with 366 days is called a leap year. A year is a leap year if it is divisible by 4 (for example, the year 1980), except it is not a leap year if it is divisible by 100 (for example, the year 1900); however, it is a leap year if it is divisible by 400 (for example, the year 2000).

There were no exceptions before the introduction of the Gregorian calendar on October 15, 1582 (for example, the year 1500 was a leap year).

Write a program that asks the user for a year and computes whether that year is a leap year or not.

Sample Run 1:
Enter the Year: 1500
Leap year.

Sample Run 2:
Enter the Year: 1900
Not a Leap Year!

Sample Run 3:
Enter the Year: 1996
Leap Year.

Sample Run 4:
Enter the Year: 1997
Not a Leap Year!

Sample Run 5:
Enter the Year: 2000
Leap Year.

Exercise #2
[\(Click to View\)](#)

Write a Java program that designs a calculator. The program should read from the user two integer numbers and a character which indicates the type of operation desired.

Sample Run 1:
First Number: 9
Operator: /
Second Number: 3
Result: 3

Sample Run 2:
First Number: 50
Operator: *
Second Number: 4

Result: 200

Exercise #3

[\(Click to View\)](#)

Complete the following program to determine the raise and new salary for an employee by adding if-else statements to compute the raise. The input to program includes the current annual salary for the employee and a number indicating the performance rating (1 = excellent, 2 = good, and 3 = poor).

An employee with a rating of 1 will receive a 6% raise, an employee with a rating of 2 will receive a 4% raise, and one with a rating of 3 will receive a 1.5% raise.

Sample Run 1:

Annual Salary: 113850

Rating: 3

Raise: 6831

New Salary: 120681

Exercise #4

[\(Click to View\)](#)

Write a program that reads in three floating-point numbers and sorts them.
Use only conditional statements.

Sample Run 1:

Please Enter Three Numbers:

4

9

2.5

The Inputs In Sorted Order Are:

2.5

4

9

Exercise #5

[\(Click to View\)](#)

Find the minimum/maximum of three numbers of your choice using the conditional operator, in just one line of code.

Hint: Use ternary operator.

Exercise #6

[\(Click to View\)](#)

Assume we want to implement a football game. A player can do the following: Press "w" Move up

Press "s" Move down

Press "d" Move right

Press "a" Move left

Press “k” or “j” Kick the ball

Press “p” Pass the ball

Press “o” Over the ball

This should be case insensitive so when user inputs “w” or “W”, it should move up. This should apply for all entered letters; both upper and lower case characters should work.

Write a program that reads an input character and prints the proper movement depending on the input character.

Exercise #7

[\(Click to View\)](#)

Write a program that reads two integer numbers (year, month) and prints the number of days of a particular month in a specific year.

Hint: You need to identify the leap years for this one.

Sample Run 1:

Year: 1700

Month: 2

Days: 28 days.

Sample Run 2:

Year: 1500

Month: 2

Days: 29 days.

Exercise #8

[\(Click to View\)](#)

Zeller’s congruence is an algorithm developed by Christian Zeller to calculate the day of the week. The formula is:

$$H = (\text{Day} + (26 \cdot (\text{Month} + 1) / 10) + K + K/4 + J/4 + 5 \cdot J) \% 7$$

H: The day of the week (0: Saturday, 1: Sunday, 2: Monday, 3: Tuesday, 4: Wednesday, 5: Thursday, 6: Friday).

Day: The day of the month.

Month: The month (3: March, 4: April, ... , 12: December).

J: The century (i.e., year / 100).

K: Year of the century (i.e., year % 100).

Write a program that displays the name of the day of the week.

Hint: January and February are counted as 13 and 14 of the previous year in the formula, so you

need to convert the month 1 to 13 and 2 to 14 and change the year to the previous year.

Sample Run 1:

Enter Day: 22

Enter Month: 5

Enter Year: 1994

Day of the week is Sunday.

Exercise #9

[\(Click to View\)](#)

Write a program that reads user input for a student's grade point average and SAT exam score, and uses these values to decide whether the student is admitted to the university.

A GPA below 1.8 will cause the student to be rejected; and a SAT score below 900 will also cause a rejection. Otherwise the student is accepted.

If both the GPA and the SAT score are too low, print the message about the GPA being too low.

Sample Run 1:

University Admission Program

Enter your GPA out of 5: 1.9

Enter your SAT Score: 900

You are accepted.

Sample Run 2:

University Admission Program

Enter your GPA out of 5: 1.2

Enter your SAT Score: 300

Your GPA is too low.

Sample Run 3:

University Admission Program

Enter your GPA out of 5: 3.5

Enter your SAT Score: 600

Your SAT score is too low.

Exercise #10

[\(Click to View\)](#)

Suppose you want to develop a program to play lottery. The program randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rules: If the user input matches the lottery number in the exact order, the award is \$10,000.

If all digits in the user input match all digits in the lottery number, the award is \$3,000.

If one digit in the user input matches a digit in the lottery number, the award is \$1,000.

Note: The digits of a two-digit number may be 0. If a number is less than 10, we assume the number is preceded by a 0 to form a two-digit number. For example, number 8 is treated as 08 and

number 0 is treated as 00 in the program.

Sample Run 1:

Enter your lottery pick (two digits): 23

The lottery number is: 34

Match one digit. You win \$1,000

Sample Run 2:

Enter your lottery pick (two digits): 45

The lottery number is: 54

Match all digits. You win \$3,000

? Loops

Exercise #1

[\(Click to View\)](#)

Write a program that prints the reverse of any given string (using loops).

Sample Run 1:

Text: CODE

Reversed Text: EDOC

Exercise #2

[\(Click to View\)](#)

Given a string containing words and numbers, return another string after replacing numbers with words. The given string may contain numbers only from 0 to 9.

Note: You should use loops, don't use some existing method to replace string.

Sample Run 1:

String: I have 3 Networking books, 0 Database books, and 8 Programming books.

Result: I have three Networking books, zero Database books, and eight Programming books.

Exercise #3

[\(Click to View\)](#)

Write a program that returns number of decimal digits in the given input string.

Sample Run 1:

Text: 2345abc423bhz

Number of Digits: 7

Exercise #4

[\(Click to View\)](#)

Read a positive integer value, and compute the following sequence. If the number is even, halve it, if it's odd, multiply it by 3 and add 1. Repeat this process until the value is 1, printing out each value. Finally print out how many of these operations you performed. If the input value is less than 1, print a message "Error" and exit the program.

Sample Run 1:

Number: 9

Initial value is 9
Next value is 28
Next value is 14
Next value is 7
Next value is 22
Next value is 11
Next value is 34
Next value is 17
Next value is 52
Next value is 26
Next value is 13
Next value is 40
Next value is 20
Next value is 10
Next value is 5
Next value is 16
Next value is 8
Next value is 4
Next value is 2
Final value 1

Number of steps 19

Exercise #5

[\(Click to View\)](#)

Write a program that checks whether a string is a palindrome. A string is a palindrome if it reads the same forward and backward. The words "mom", "dad", and "noon" for instance, are all palindromes.

One solution is to check whether the first character in the string is the same as the last character. If so, check whether the second character is the same as the second-to-last character. This process continues until a mismatch is found or all the characters in the string are checked, except for the middle character if the string has an odd number of characters.

Sample Run 1:

Enter a String: noon
noon is a palindrome.

Sample Run 2:

Enter a String: moon
moon is not a Palindrome.

Exercise #6

[\(Click to View\)](#)

Given an integer N, print the following shape using asterisks and numbers. For example, for N = 9 the following shape is printed.

Sample Run 1:

Enter a Number: 9
Shape:

```
*****1
*****22
*****333
*****4444
*****55555
*****666666
*****777777
*****888888
*****999999
```

Exercise #7

[\(Click to View\)](#)

Write a program to read two string and print "Equal" if they are the exact same. It should be case insensitive.

Note: You should use loops. Don't use some existing method to check equality.

Sample Run 1:

First String: CoDE

Second String: Code

Result: Equal.

Sample Run 2:

First String: Wood

Second String: Wolf

Result: Not Equal.

Exercise #8

[\(Click to View\)](#)

Given an integer N, print the following shape using asterisks and spaces. For example, for N = 6 the following shape is printed.

Sample Run 1:

Length: 6

Shape:

```
*
```

Exercise #9

[\(Click to View\)](#)

Given an integer N, print the following diamond shape using asterisks and spaces. For example, for N = 5 the following shape is printed.

Sample Run 1:

Length: 5

Shape:

*

Exercise #10

[\(Click to View\)](#)

Write a program that displays the first fifty prime numbers in five lines, each containing ten numbers. An integer greater than 1 is prime if its only positive divisor is 1 or itself.

For example, 2, 3, 5, and 7 are prime numbers, but 4, 6, 8, and 9 are not.

The problem is to display the first 50 prime numbers in five lines, each of which contains ten numbers.

Sample Run 1:

The first 50 prime numbers are:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

31, 37, 41, 43, 47, 53, 59, 61, 67, 71

73, 79, 83, 89, 97, 101, 103, 107, 109, 113

127, 131, 137, 139, 149, 151, 157, 163, 167, 173

179, 181, 191, 193, 197, 199, 211, 223, 227, 229

Exercise #11

[\(Click to View\)](#)

Write a program called to produce the multiplication table of 1 to 9 as shown using nested loops.

Note: You should add the necessary indents.

Sample Run 1:

Code * | 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

Exercise #12

[\(Click to View\)](#)

Write a program that returns number of occurrences of a string in another string.

Note: You should use only nested loops. Don't use some method like indexOf or substring.

Sample Run 1:

First String: the

Second String: the students are working hard in the faculty of Engineering because they love it

Number of Occurrences: 3

? Arrays

Exercise #1

[\(Click to View\)](#)

Write a Java program that copies an array of int to another array after multiplying each element by 2. And finally print the copied array.

Note: You should store the copied array in a new separate array and print it once at the end of the program.

Sample Run 1:

Enter Array Size: 3

Enter Array Elements:

5

8

10

Result: 10, 16, 20

Exercise #2

[\(Click to View\)](#)

Write a Java program that given an array of integers and a key will print "Found" if (if and only if) the key occurs in the array otherwise it will print "Not Found".

For example, for the Array = {10, 5, 11, 10} and key 11 the method will print "Found" and for Array = {1, 2, 3, 4} and key 11 the method will print "Not Found".

Sample Run 1:

Enter Array Size: 4

Enter Array Elements:

10

5

11

10

Enter Key: 11

Result: Found.

Exercise #3

[\(Click to View\)](#)

A company pays its sales people on a commission basis. The salespeople receive \$200 per week plus 9% of their gross sales for that week. For example, a salesperson who grosses \$3000 in sales in a week receives \$200 plus 9% of \$3000, which is total of \$470.

Write a Java program that uses an array to determine how many salespersons earned salaries in each of the following ranges.

Assume each person's salary is truncated to an integer. \$200 – \$299

\$300 - \$399

\$400 – \$499

\$500 - \$599

\$600 - \$699

\$700 - \$799

\$800 – 899

\$900 - \$1000

\$1000 and above

Note: Use an array to store the counts is a requirement!

Sample Run 1:

Enter number of Employees: 6

Enter Employee gross sale: 3000

Employee salary is \$470

Enter Employee gross sale: 1000

Employee salary is \$290

Enter Employee gross sale: 10000

Employee salary is \$1100

Enter Employee gross sale: 8000

Employee salary is \$920

Enter Employee gross sale: 200

Employee salary is \$218

Enter Employee gross sale: 7000

Employee salary is \$830

Total 6 Employees Reported.

Employees in the range:

200 – 299: 2

300 – 399: 0

400 – 499: 1

500 – 599: 0

600 – 699: 0

700 – 799: 0

800 – 899: 1

900 – 999: 1

Over 1000: 1

Exercise #4

[\(Click to View\)](#)

Write Java program that given two arrays of integers, prints the union array and the intersection array. Assume that the input arrays do not consist of duplicates.

For example:Array 1: { 12, 32, 14, 35, 89, 16, 120 }

Array 2: { 9, 12, 8, 17, 120, 35, 36 }

Union (Array 1, Array 2): { 12, 32, 14, 35, 89, 16, 120, 9, 8, 17, 36 }

Intersection (Array 1, Array 2): { 12, 35, 120 }

Note: You should use loops here.

Sample Run 1:

Enter first array size: 7

Enter first array elements:

12
32
14
35
89
16
120

Enter second array size: 7

Enter second array elements:

9
12
8
17
120
35
36

Union: 12, 32, 14, 35, 89, 16, 120, 9, 8, 17, 36

Intersection: 12, 35, 120

Exercise #5

[\(Click to View\)](#)

Write a code to take integers from the user, store them in an array, find the greater and smaller number and display them along with their position.

Sample Run 1:

Enter the array size: 6

Enter array elements:

76
90
45
69
9
88

The Largest Element is: 90

Its Location: 1

The Smaller Element is: 9

Its Location: 4

Exercise #6

[\(Click to View\)](#)

Write a Java program that given an array of integers, swaps every element with the one next to it. Printing the final array result.

Sample Run 1:

Enter the array size: 5

Enter array elements:

15

29

17

30

10

Swapped Array: 29, 15, 30, 17, 10

Sample Run 2:

Enter the array size: 4

Enter array elements:

50

99

102

77

Swapped Array: 99, 50, 77, 102

Exercise #7

[\(Click to View\)](#)

Giving an array of integers, reorder all the elements twice, ascending and descending. The swapping process should be in place, you shouldn't use a second array for the swapping.

Sample Run 1:

Enter array size: 5

Enter array elements:

59

21

67

49

37

Ascending Array: 21, 37, 49, 59, 67

Descending Array: 67, 59, 49, 37, 21

Exercise #8

[\(Click to View\)](#)

Consider the left most and right most appearances of some value in an array. The "span" is the number of elements between the two inclusive. A single value has a span of 1.

For example: For {1, 2, 1, 1, 3} Max Span = 4

For {1, 4, 2, 1, 4, 1, 4} Max Span = 6

Write a program which takes an array as an input, and prints the largest span found in the given array.

Sample Run 1:

Enter Array Size: 6

Enter Array Elements:

5

1

5

9

11

3

Max Span: 3

Exercise #9

[\(Click to View\)](#)

Write a program to count the occurrences of each letter in a sentence given by the user as an input. Printing each letter and it's count at the end.

Keep the count of each letter in a separate array, corresponding with the letters array. Check the following sample for more details.

Note: Ignore any spaces in the string.

Sample Run 1:

Enter a Text: LONDON

Letters Count:

L: 1

O: 2

N: 2

D: 1

Exercise #10

[\(Click to View\)](#)

Design a Pokemon game which allows acquiring any number of Pokemon chosen by the user, with the ability to view all current Pokemon at any giving time.

The user should be presented with a list of options after any action, including the choice to exit the game.

If the user picks a Pokemon, it shouldn't be available anymore in the pick list. Check the following example for more details.

Sample Run 1:

Welcome to Pokemon Exchange.

Options:

(1) Pick a Pokemon

(2) View My Pokemon

(0) Exit

User: 2

You have 0 Pokemon. Pick a Pokemon.

Options:

(1) Pick a Pokemon

(2) View My Pokemon

(0) Exit

User: 1

Choose a Pokemon:

(1) Pikachu

(2) Charmeleon

(3) Geodude

(4) Gyarados

(5) Butterfree

User: 3

You have picked Geodude.

Options:

(1) Pick a Pokemon

(2) View My Pokemon

(0) Exit

User: 1

Choose a Pokemon:

(1) Pikachu

(2) Charmeleon

(3) Gyarados

(4) Butterfree

User: 1

You have picked Pikachu.

Options:

(1) Pick a Pokemon

(2) View My Pokemon

(0) Exit

User: 2

You have 2 Pokemon. Geodude, Pikachu.

Options:

(1) Pick a Pokemon

(2) View My Pokemon

(0) Exit

User: 0

Bye!

? Multidimensional Arrays

Exercise #1

[\(Click to View\)](#)

Write a program that given an array of integers, determine the number of repeated integers and their counts. Saving every value and its count in a two-dimensional array in the following format:

```
Code int [][] values = new int[VALUES][COUNT];
```

Check the following sample for more details.

Sample Run 1:

Enter array size: 11

Enter array elements:

13
34
22
4
499
4
22
18
4
1
1

There are 3 repeated numbers.

22: 2 times

4: 3 times

1: 2 times

Exercise #2

[\(Click to View\)](#)

Write a program that given a two-dimensional array, reorders the rows such that the row with the highest row sum is the first row.

For example, if the program will be called with the following array:

```
Code int [][] numbers = new int [] [] { new int [] {1, 3, 5, 9}, new int [] {2, 100}, new int [] {2, 2, 3} }
```

The output should be:

2, 100
1, 3, 5, 9
2, 2, 3

Sample Run 1:

Enter number of rows: 3

Enter row#0 size: 4

Enter row#0 elements:

1
3
5
9

Enter row#1 size: 2

Enter row#1 elements:

2
100
Enter row#2 size: 3
Enter row#2 elements:
2
2
3

Result:
2, 100
1, 3, 5, 9
2, 2, 3

Exercise #3
[\(Click to View\)](#)

Write a program to transpose a square two dimensional array in place without creating a new second array.

Matrix transpose interchanges each row of a square matrix with the corresponding column. It writes the rows of the matrix as columns of the transposed matrix.

For example:

5 30 15
20 10 9
8* 7* 1

Should be transposed to:

5* 20 8
30 10 7
15 9* 1

Hint: You need to use swapping to do the transpose in place.

Note: Asterisks are only added to format the matrix, it has nothing to do with the exercise.

Sample Run 1:

Enter Matrix Size: 3
Enter Row#0 Elements:
5
30
15
Enter Row#1 Elements:
20
10
9
Enter Row#2 Elements:
8
7
1

Result:
5* 20 8
30 10 7

15 9* 1

Exercise #4

[\(Click to View\)](#)

Write a program that grades multiple-choice tests. Assume there are eight students and ten questions, and the answers are stored in a two-dimensional array.

Each row records a student's answers to the questions, as shown in the following array.

Students' Answers to the Questions:

Student 0: A B A C C D E E A D

Student 1: D B A B C A E E A D

Student 2: E D D A C B E E A D

Student 3: C B A E D C E E A D

Student 4: A B D C C D E E A D

Student 5: B B E C C D E E A D

Student 6: B B A C C D E E A D

Student 7: E B E C C D E E A D

The key for the correct answers:

Key: D B D C C D A E A D

Your program grades the test and displays the result. It compares each student's answers with the key, counts the number of correct answers, and displays it.

Sample Run 1:

Student 0's correct answers count is 7

Student 1's correct answers count is 6

Student 2's correct answers count is 5

Student 3's correct answers count is 4

Student 4's correct answers count is 8

Student 5's correct answers count is 7

Student 6's correct answers count is 7

Student 7's correct answers count is 7

? Methods

Exercise #1

[\(Click to View\)](#)

Write a method called `getDayName()` that takes as an argument the day number and returns the day name.

For example, for day number 1 this method should return "Saturday", for day number 2 it should return "Sunday", and so on.

The method header is as follows:

Code `public static String getDayName (int day)`

In your main method, read user's input then invoke the `getDayName()` method and print the day name.

Sample Run 1:

Enter Day Number: 5

Day Name: Wednesday

Sample Run 2:

Enter Day Number: 1

Day Name: Saturday

Exercise #2

[\(Click to View\)](#)

Create a class named Triangle that contains the following two methods, in addition to the main method:

Code public static boolean isValid (double side1, double side2, double side3)

public static double perimeter (double side1, double side2, double side3)

? isValid() This method checks whether the given sides are valid to form a triangle by checking that the sum of any two sides is larger than the third side. It returns true if the given sides form a valid triangle and false otherwise.

? perimeter() This method calculates the perimeter of a triangle which is the summation of all its sides. The perimeter method should check first whether the given sides form a valid triangle by invoking the isValid() method. If the given sides do not form a valid triangle, it should print a message to the user, and return 0. If they form a valid triangle, it returns the perimeter value.

Implement this methods by reading three double values from the user and invoking the perimeter method. If a non-zero value is returned by the perimeter method, print the perimeter value.

Sample Run 1:

Side One: 3

Side Two: 4.5

Side Three: 5.3

Perimeter: 12.8

Sample Run 2:

Side One: 1

Side Two: 1

Side Three: 2

1.0, 1.0, and 2.0 does not form a valid triangle.

Exercise #3

[\(Click to View\)](#)

Write a program that creates a two-dimensional array initialized with test data. Use any primitive data type you wish. The program should have the following methods:

? getTotal() This method should accept a two-dimensional array as its argument and return the total of all the values in the array.

? getAverage() This method should accept a two-dimensional array as its argument and return the average of all the values in the array.

? getRowTotal() This method should accept a two-dimensional array as its first argument and an integer as its second argument. The second argument should be the subscript of a row in the array. The method should return the total of the values in the specified row.

? getColumnTotal() This method should accept a two-dimensional array as its first argument and an integer as its second argument. The second argument should be the subscript of a column in the array. The method should return the total of the values in the specified column.

? getHighestInRow() This method should accept a two-dimensional array as its First argument and an integer as its second argument. The second argument should be the subscript of a row in the array. The method should return the Highest values in the specified row in the array.

? getLowestInRow: This method should accept a two-dimensional array as its First argument and an integer as its second argument. The second argument should be the subscript of a row in the array. The method should return the lowest values in the specified row in the array.

Demonstrate each of the methods in your program.

Note: Reading arrays from the user is up to you.

Sample Run 1:

Total : 364.70000005

Average : 45.58750000006

Total of Row 0 : 242.97

Total of Row 1 : 121.73

Total of Column 0 : 98.9

Total of Column 1 : 76.98

Highest in Row 0 : 99.5

Highest in Row 1 : 54.66

Lowest in Row 0 : 23.54

Lowest in Row 1 : 11.12

Exercise #4

[\(Click to View\)](#)

You have to create a program that displays this following menu:

Add Keychains to Order.

Remove Keychains from Order.

View Current Order.

Checkout.

You have to use four different functions, one each to addKeychains(), removeKeychains(), viewOrder() and checkout().

The four functions respond to the user's entered choice 1, 2, 3, 4. Following are their functionality:

? addKeychain() It will have one int arguments and int return type. It will ask the user the no of key chains to be added, add them and then return the new value.

? removeKeychain() It will have one int argument and int return type. it will ask the user no. of key chains to be removed and then return the new value.

? viewOrder() It will have two int arguments and void return type. It will display the no. of key chains, there individual prices and the total.

? checkout() It will have two int arguments and void return type. It will ask the user's name. Display the order details and thank the user.

Sample Run 1:

Welcome To The Keychain Shop!

1. Add Key Chains to Order.
2. Remove Key Chains from Order.
3. View Current Order.
4. Checkout.

Please Enter Your Choice: 1

You have 0 key chains. How many do you want to add: 2

1. Add Key Chains to Order.
2. Remove Key Chains from Order.
3. View Current Order.
4. Checkout.

Please Enter Your Choice: 2

You have 2 Key Chains. How many do you want to remove: 1

1. Add Key Chains to Order.
2. Remove Key Chains from Order.
3. View Current Order.
4. Checkout.

Please Enter Your Choice: 3

Number of Key Chains: 1

Cost Per Key Chain: 10

Total Cost: 10

1. Add Key Chains to Order.
2. Remove Key Chains from Order.
3. View Current Order.
4. Checkout.

Please Enter Your Choice: 4

What is your name: John

No. of Key Chains: 1

Cost Per Key Chain: 10

Total Cost: 10

Thank you for your order John.

Exercise #5

[\(Click to View\)](#)

Given a string and an int (a key), write two functions for encrypting and decrypting the given string using the given key.

The encryption is done by adding the key to ASCII of each character of the string, and the decryption is done by subtracting the key from ASCII of each character.

For example if the input string is "abcz" and the key is 1 the encryption function should return "bcda".

You have to keep the characters in the string readable (between a - z) after encrypting or decrypting, so you should encrypt or decrypt in a cyclic way.

Consider these examples: Encrypt character 'z' with key 1, should be 'a'

Encrypt character 'z' with key 3, should be 'c'

Decrypt character 'b' with key 3, should be 'y'

Hint: You need to use mod operation.

Sample Run 1:

Enter Text: abcz

Enter Key: 2

Encrypt or Decrypt (e or d): e

Encrypted Text: cdeb

Sample Run 2:

Enter Text: abcz

Enter Key: 1

Encrypt or Decrypt (e or d): d

Dycrypted Text: zaby

Exercise #6

[\(Click to View\)](#)

The purpose of this exercise is to create a working text-mode version of the popular game, Connect 4.

Connect 4 is a turn based 2 player game played on a 6 rows by 7 columns board. Each player on his turn chooses a column to put a piece at. A piece played in a column falls till it reaches the bottom of the column, or it gets stacked above another piece. The winner is the player who forms a line of 4 adjacent pieces of his color (horizontal, vertical or diagonal).

The text mode version displays the board using characters (for example "X" for one player, "O" for the other player, and a space for an empty slot).

The program asks the player for the column to play, and plays the piece for him at this location.

If the column is full, or is invalid, the program warns the user about the error, and asks him again to re-enter a column number.

When the game is finished, the program displays the winner. If the board is full and no player win, the program displays "Withdraw".

Hint: You should use two-dimensional arrays.

Sample Run 1:

Welcome to Connect 4.

Starting The Game...

Player One (X): 4

Code | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

			X				

Player Two (O): 5

Code | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

			X	O			

Player One (X): 5

Code | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

				X			
			X	O			

Player Two (O): 9

Invalid Column. Play Again.

Player Two (O): 3

Code | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

				X			
		O	X	O			

Exercise #7

[\(Click to View\)](#)

For this task you will create a class containing a number of static methods for processing an array of marks, which are scores in a test. Each mark is an integer in the range 0 to 100 inclusive.

You can use the following method to generate grades from 1 to 100:

Code private static int[] getMarks() {

Random random = new Random();

```

    for (int i = 0; i < marks.length ; i++) {
        marks[i] = random.nextInt(100) + 1;
    }

    return marks;
}

```

The class ProcessMarks that you create will have the methods specified below. Most will accept an array of marks as an argument, one will accept an array of characters. The return type should be appropriate for the returned value.

- max, min and range() Will return the maximum mark, the minimum mark and the difference between the maximum and minimum marks respectively.
- mean() Will return the mean of the set of marks.
- median() Will return the median value of the set of marks. The median value is the middle one when the values are placed in order. To obtain an ordered version of the marks you may use an appropriate sort method of the Java API's Arrays class. Be careful not to destroy the original array of marks. If there is an even number of marks, the middle value is taken as the average of the two values that are nearest to the middle.
- mode() Will return the mode of the set of marks, which is the most commonly occurring mark. To find the mode, use an ordered version of the set of marks, as used for finding the median. If there is more than one value that is most common, any one of the most common values will do for the mode.
- grades() Will return an array of characters, which are the grades corresponding to the integer marks in the array of marks.
The grades are to be assigned using the following lower boundaries for the corresponding marks:
A: Lower boundary is 90
B: Lower boundary is 75
C: Lower boundary is 60
D: Lower boundary is 50
E: Lower boundary is 45
F: The grade for all other marks

- gradeDistribution() method will accept an array of characters, which are the grades assigned for the array of marks, such as returned by the grades method. The gradeDistribution() method will return an array of integer values containing the distribution of grades, which is the number of occurrences of each grade in the assigned grades. The characters used for grades are fixed. The returned array should provide the distribution in order from grade A to grade F, in this following format:
A: 10

B: 30

C: 105

D: 75

E: 35

F: 10

Notes: None of your code should change the contents of the original array of marks.

You should not make any assumption that your methods will be called in particular order.

Exercise #8

[\(Click to View\)](#)

Write a program that asks the user to enter a distance in meters. The program will then present the following menu of selections:

Convert to kilometers

Convert to inches

Convert to feet

Quit the program

The program will convert the distance to kilometers, inches, or feet, depending on the user's selection. Here are the required methods:

? calculateKilometers() Accepts the number of meters as an argument. The method should return the argument converted to kilometers. Convert the meters to kilometers using the following formula: $\text{kilometers} = \text{meters} \times 0.001$

? calculateInches() Accepts the number of meters as an argument. The method should return the argument converted to inches. Convert the meters to inches using the following formula: $\text{inches} = \text{meters} \times 39.37$

? calculateFeet() Accepts the number of meters as an argument. The method should return the argument converted to feet. Convert the meters to feet using the following formula: $\text{feet} = \text{meters} \times 3.281$

? menu() Returns a list of menu of selections. This method should not accept any arguments.

The program should continue to display the menu until the user enters 4 to quit the program. If the user selects an invalid choice from the menu, the program should display an error message.

Sample Run 1:

Enter a distance in meters: 100

1. Convert to Kilometers

2. Convert to Inches
3. Convert to Feet
4. Quit the Program

Enter your choice: 1
100.0 meters is 0.1 kilometers.

1. Convert to Kilometers
2. Convert to Inches
3. Convert to Feet
4. Quit the Program

Enter your choice: 2
100.0 meters is 3936.9999995 inches.

1. Convert to Kilometers
2. Convert to Inches
3. Convert to Feet
4. Quit the Program

Enter your choice: 4
Bye!

Exercise #9
[\(Click to View\)](#)

Write a program that lets the user play the game of Rock, Paper, Scissors against the computer. The program should work as follows:

? When the program begins, a random integer number in the range of 1 through 3 is generated. If the number is 1, then the computer has chosen rock. If the number is 2, then the computer has chosen paper. If the number is 3, then the computer has chosen scissors. (Don't display the computer's choice yet).

? The user enters his or her choice of "rock", "paper", or "scissors" at the keyboard. You can use a menu if you prefer.

? The computer's choice is displayed.

? A winner is selected according to the following rules: If one player chooses rock and the other player chooses scissors, then rock wins.

If one player chooses scissors and the other player chooses paper, then scissors wins.

If one player chooses paper and the other player chooses rock, then paper wins.

If both players make the same choice, the game must be played again to determine the winner.

? After a winner is selected, the program asks if the user wants to repeat the game. If the answer is yes, restart the game.

Hint: Divide the program into functions that perform each major task.

Sample Run 1:

1. Rock
2. Paper
3. Scissors

Your choice: 3

The computer's choice was paper.
The user's choice was scissors.

Scissors beats paper.
The user wins!

Exercise #10
[\(Click to View\)](#)

Write a static method named `anglePairs()` that accepts three angles (integers), measured in degrees, as parameters and returns whether or not there exists both complementary and supplementary angles among the three angles passed.

Two angles are complementary if their sum is exactly 90 degrees; two angles are supplementary if their sum is exactly 180 degrees.

Therefore, the method should return `true` if any two of the three angles add up to 90 degrees and also any two of the three angles add up to 180 degrees, otherwise the method should return `false`. You may assume that each angle passed is non-negative.

Sample Run 1:
Enter Three Angles:
45
135
45
Has complementary and supplementary angles: True

? Recursion

Exercise #1
[\(Click to View\)](#)

Write a recursive function `power(int x, int y)` that calculates x to the power y .

Sample Run 1:
Enter Base: 10
Enter Exponent: 3
Result: 1000

Exercise #2
[\(Click to View\)](#)

Write a recursive function `int sum(int [] x)` that calculates the sum of integers in array `x`.

Sample Run 1:
Enter Array Size: 5
Enter Array Elements:
6
10
5
8
1
Sum: 30

Exercise #3
[\(Click to View\)](#)

Write a recursive method `String eliminates(String sentence, char letter)` that takes a `String` and a `char` and return a new string and deletes each occurrence of this character from the string.

Hint: You can use `String.substring(index)`.

Sample Run 1:
Enter String: Welcome to Recursion World
Enter Char: o
Modified String: Welcme t Recursin Wrld

Exercise #4
[\(Click to View\)](#)

Write a "String Inverter" recursive function. It takes a sentence and returns the sentence in reverse order.

For example, given "Welcome to Recursion World" as input, your program should print out "World Recursion to Welcome".

Hint: You can use `String.split(" ")` to split the input string around the spaces into an array.

Sample Run 1:
Enter Sentence: Welcome to Recursion World
Reversed Sentence: World Recursion to Welcome

Exercise #5
[\(Click to View\)](#)

We will build a recursive method to compute numbers in the Fibonacci sequence. This infinite sequence starts with 0 and 1, which we'll think of as the zeroth and first Fibonacci numbers, and each succeeding number is the sum of the two preceding Fibonacci numbers. Thus, the second number is $0 + 1 = 1$. And to get the third Fibonacci number, we'd sum the first (1) and the second (1) to get 2. And the fourth is the sum of the second (1) and the third (2), which is 3. And so on.

Write a method `fib()` that takes some integer `N` as a parameter and returns the Fibonacci number. For example: `fib(6)` should return 8, and `fib(7)` should return 13.

Sample Run 1:
Enter Number: 10
Fibonacci: 55

Exercise #6

[\(Click to View\)](#)

Write a method `printSquares()` that has an integer parameter `N`, and prints the squares of the integers from 1 to `N`, separated by commas. It should print the squares of the odd integers in descending order first and then following with the squares of the even integers in ascending order. It does not print a newline character. It should throw an `IllegalArgumentException` if the specified integer is less than 1.

For example: `printSquares(4)` should print 9, 1, 4, 16

`printSquares(1)` should print 1

`printSquares(7)` should print 49, 25, 9, 1, 4, 16, 36

Note: You may not use helper methods to solve this problem, write a single method.

Sample Run 1:

Enter a Number: 4

Squares: 9, 1, 4, 16

Exercise #7

[\(Click to View\)](#)

Consider climbing stairs. You can take a single stair up, take two stairs up as a single step, or take three stairs up as a single step.

How many different ways can you climb the steps taking a single stair, two stairs, or three stairs on each step.

For example, if there are three stairs, then you can climb them in four different ways: 3 single steps for each stair, 1 single stair followed by a pair, and a pair of stairs followed by single stair, or all three stairs in a single step.

Four stairs can be climbed in seven different ways.

Write a recursive method `findNumWays()` that has an integer parameter that specifies the number of stairs to climb, and returns the number of different ways to climb the stairs using steps that take one, two, or three stairs at a time.

Note: You may assume that the number of stairs specified is non-negative.

Sample Run 1:

Number of Stairs: 4

Numbers of Ways: 7

Exercise #8

[\(Click to View\)](#)

Write a recursive method `loanLength()` which take three parameters, `loanAmount`, `annualInterest`, and `monthlyPayment`. Returns an `Integer` with the number of months until the loan is paid off, assuming that the interest is compounded monthly.

For Example: `loanLength(1000, 0.10, 200)` is 6 months

loanLength(1000, 0.10, 1050) is 1 month

loanLength(0, 0.90, 50) is 0 months

Sample Run 1:

Enter Loan Amount: 1000

Enter Annual Interest: 0.10

Enter Monthly Payments: 200

Last Payment In: 6 Month(s).

? Classes and Objects

Exercise #1

[\(Click to View\)](#)

Write a Circle class that has the following fields:radius: A double value.

PI: a final double initialized with the value 3.14159

The class should have the following methods:Constructor() A no argument constructor that sets the radius field to 0.0

Constructor() Accepts the radius of the circle as an argument.

setRadius() A setter method for the radius field.

getRadius() A getter method for the radius field.

getArea() Returns the area of the circle which is calculated as $\text{Area} = \text{PI} \times \text{radius}^2$

getDiameter(): Returns the diameter of the circle which is calculated as $\text{Diameter} = 2 \times \text{radius}$

getCircumference(): Returns the circumference of the circle, which is calculated as $\text{circumference} = 2 \times \text{PI} \times \text{radius}$

Write a program that demonstrates the Circle class by asking the user for the circle's radius, creating a Circle object, and then reporting the circle's area, diameter, and circumference.

Sample Run 1:

Enter Circle Radius: 10.5

Area is 346.3602975

Diameter is 21.0

Circumference is 65.97339

Exercise #2

[\(Click to View\)](#)

For this exercise, you will design a set of classes that work together to simulate a car's fuel gauge and odometer. The classes you will design are the following:

The FuelGauge Class. This class will simulate a fuel gauge. Its responsibilities are as follows: To know the car's current amount of fuel, in gallons.

To report the car's current amount of fuel, in gallons.

To be able to increment the amount of fuel by 1 gallon. This simulates putting fuel in the car. (The car can hold a maximum of 15 gallons.)

To be able to decrement the amount of fuel by 1 gallon, if the amount of fuel is greater than 0 gallons. This simulates burning fuel as the car runs.

The Odometer Class. This class will simulate the car's odometer. Its responsibilities are as follows: To know the car's current mileage.

To report the car's current mileage.

To be able to increment the current mileage by 1 mile. The maximum mileage the odometer can store is 999,999 miles. When this amount is exceeded, the odometer resets the current mileage to 0.

To be able to work with a FuelGauge object. It should decrease the FuelGauge object's current amount of fuel by 1 gallon for every 24 miles traveled. (The car's fuel economy is 24 miles per gallon.)

Demonstrate the classes by creating instances of each. Simulate filling the car up with fuel, and then run a loop that increments the odometer until the car runs out of fuel. During each loop iteration, print the car's current mileage and amount of fuel.

Sample Run 1:

Mileage: 1
Fuel level: 15 gallons

Mileage: 2
Fuel level: 15 gallons

Mileage: 3
Fuel level: 15 gallons

Mileage: 359
Fuel level: 1 gallons

Mileage: 360
Fuel level: 0 gallons

Exercise #3
[\(Click to View\)](#)

Write a program that simulates a fishing game. In this game, a six-sided dice is rolled to determine what the user has caught. Each possible item is worth a certain number of fishing points. The points will remain hidden until the user is finished fishing, and then a message is displayed congratulating the user, depending on the number of fishing points gained.

Here are some suggestions for the game's design:

? Each round of the game is performed as an iteration of a loop that repeats as long as the player wants to fish for more items.

? At the beginning of each round, the program will ask the user whether or not he or she wants to continue fishing.

? The program simulates the rolling of a six-sided dice.

? Each item that can be caught is represented by a number generated from the dice; for example, 1 for "huge fish", 2 for "an old shoe", 3 for a "little fish", and so on.

? Each item the user catches is worth different amount of points.

? The loop keeps a running total of the user's fishing points.

? After the loop has finished the total number of fishing points is displayed, along with a message that varies depending on the points earned.

Sample Run 1:

Lets go fishing!

You rolled: 4

You caught: 30 inch walleye

Points: 40

Lets go fishing! Enter Y to play: y

You rolled: 4

You caught: 30 inch walleye

Points: 40

Lets go fishing! Enter Y to play: y

You rolled: 3

You caught: little fish

Points: 30

Lets go fishing! Enter Y to play: y

You rolled: 1

You caught: huge fish

Points: 10

Lets go fishing! Enter Y to play: n

Thanks for playing, total points: 120

Exercise #4

[\(Click to View\)](#)

Freezing and boiling points of several substances. Ethyl Alcohol. Freezing Point: -173. Boiling Point: 172.

Oxygen. Freezing Point: -362. Boiling Point: -306.

Water. Freezing Point: 32. Boiling Point: 212.

Design a class that stores a temperature in a temperature field and has the appropriate methods for the field. In addition to appropriate constructors, the class should have the following methods:

? isEthylFreezing() This method should return the boolean value true if the temperature stored in the temperature field is at or below the freezing point of ethyl alcohol. Otherwise, the method should return false.

? isEthylBoiling() This method should return the boolean value true if the temperature stored in the temperature field is at or above the boiling point of ethyl alcohol. Otherwise, the method should return false.

? isOxygenFreezing() This method should return the boolean value true if the temperature stored in the temperature field is at or below the freezing point of oxygen. Otherwise, the method should return false.

? isOxygenBoiling() This method should return the boolean value true if the temperature stored in temperature field is at or above the boiling point of oxygen. Otherwise, the method should return false.

? isWaterFreezing() This method should return the boolean value true if the temperature stored in the temperature field is at or below the freezing point of water. Otherwise, the method should return false.

? isWaterBoiling() This method should return the boolean value true if the temperature stored in the temperature field is at or above the boiling point of water. Otherwise, the method should return false.

Write a program that demonstrate the class. The program should ask the user to enter a temperature, and then display a list of the substance that will freeze at that temperature and those that will boil at that temperature. For example, if the temperature is -20 the class should report that water will freeze and oxygen will boil at that temperature.

Sample Run 1:
Enter a Temperature: 48
Oxygen Will Boil.

There is more, but I can't make any more posts for now. Anyway, I really hope this helps. If you're facing any problems post here or [PM](#). Good luck.

United States of America - April 20, 2015, 8:59 am

Hey thanks man, I believe this is one of the most needed post in the section.

Just Ice - April 21, 2015, 7:57 am

This is great for anyone looking for some exercises to boost their java knowledge, great contribution to the community :) thankyou!

Sommmen - April 21, 2015, 8:07 am

my god if this only existed when i started. just code and see for ourselves.

MrOrang - April 21, 2015, 10:40 am

This is a great thread man, keep up the quality!

I am sure this is useful for beginners and might even be useful for the other members here to refresh their knowledge.

M0dy - April 22, 2015, 10:02 am

Thanks everyone. Yea true, doesn't have to be just for beginner. Some of these exercises are actually challenging..well, at least for me.

Infernumi - April 22, 2015, 12:00 pm

Thanks for this M0dy! I will definitely try some of these myself with my new IDE. ;)

BeNT- - April 22, 2015, 12:01 pm

Very HQ post man, props and Great job!!

+Haze - April 22, 2015, 12:08 pm

I think this is a great compilation, suited for all types of programmers. Keep up the HQ posts.

madcrazydrumma - April 22, 2015, 12:31 pm

For the first section: "Data Types", the volume answer is wrong just a heads up

It should be - Volume: 4849.048260815845 or 4849.05 (rounded)

Aside from this, amazing stuff! Great revision for me ?

nOrr - April 22, 2015, 3:37 pm

+1 (or rather +3) - nice thread!

I might be tempted to use some of the recursion tasks for my work's tech-screenings during the recruitment process. Some of them are not trivial (at least not for me at 00:30 in the night after a long day), but could easily be done within reasonable time (10-20 minutes). If that's OK with you :P

For your next handful, you could specify tasks that are more technical, but of same size, such as implementing a searching algorithm (binary is easy), or implementing BFS and DFS.

Maxiim - April 23, 2015, 12:40 am

I wonder if I may borrow some of these problems when I will be 'teaching' Java in september/october :)

madcrazydrumma - April 23, 2015, 5:09 am

For the decision statements exercise 3:

Quote:Sample Run 1:
Annual Salary: 113850
Rating: 3
Raise: 6831
New Salary: 120681

The rating should be 1, not 3 :)

M0dy - April 23, 2015, 8:12 am

(Apr 23, 2015 05:09 AM)madcrazydrumma Wrote: [\[CITE\]](#) For the decision statements exercise 3:

The rating should be 1, not 3 :)

Yea true, my bad. I checked multiple times but apparently I missed few things, and I can't edit this anymore.

(Apr 22, 2015 03:37 PM)nOrr Wrote: [\[CITE\]](#) +1 (or rather +3) - nice thread!

I might be tempted to use some of the recursion tasks for my work's tech-screenings during the recruitment process. Some of them are not trivial (at least not for me at 00:30 in the night after a long day), but could easily be done within reasonable time (10-20 minutes). If that's OK with you :P

For your next handful, you could specify tasks that are more technical, but of same size, such as implementing a searching algorithm (binary is easy), or implementing BFS and DFS.

(Apr 23, 2015 12:40 AM)Maxiim Wrote: [\[CITE\]](#) I wonder if I may borrow some of these problems when I will be 'teaching' Java in september/october :)

You guys can use whatever you want. I simply collected those in one topic for easier access. :) and will sure try nOrr.