# Homework: Loops, Methods, Classes

This document defines homework assignments from the [“Java Basics“ Course @ Software University](https://softuni.bg/courses/java-basics/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Symmetric Numbers in Range

Write a program to generate and **print all symmetric numbers in given range** [**start**…**end**] (0 ≤ **start** ≤ **end** ≤ 999). A number is symmetric if its digits are symmetric toward its middle. For example, the numbers 101, 33, 989 and 5 are symmetric, but 102, 34 and 997 are not symmetric. Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 11 | 5 6 7 8 9 11 |
| 101 110 | 101 |
| 555 777 | 555 565 575 585 595 606 616 626 636 646 656 666 676 686 696 707 717 727 737 747 757 767 777 |

## Generate 3-Letter Words

Write a program to generate and **print all 3-letter words consisting of given set of characters**. For example if we have the characters '**a**' and '**b**', all possible words will be "**aaa**", "**aab**", "**aba**", "**abb**", "**baa**", "**bab**", "**bba**" and "**bbb**". The input characters are given as string at the first line of the input. Input characters are **unique** (there are no repeating characters). Examples:

|  |  |
| --- | --- |
| **Input** | **Output** |
| x | xxx |
| ab | aaa aab aba abb baa bab bba bbb |
| abx | aaa aab aax aba abb abx axa axb axx baa bab bax bba bbb bbx bxa bxb bxx xaa xab xax xba xbb xbx xxa xxb xxx |

## Full House

In most Poker games, the "[**full house**](http://en.wikipedia.org/wiki/List_of_poker_hands#Full_house)" hand is defined as three cards of the same face + two cards of the same face, other than the first, regardless of the card's suits. The cards faces are "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "**10**", "**J**", "**Q**", "**K**" and "**A**". The card suits are "♣", "♦", "♥" and "♠". Write a program to generate and print all full houses and print their number. Example:

|  |
| --- |
| **Output** |
| (2♣ 2♦ 2♥ 3♣ 3♦) … (2♣ 2♦ 2♥ 3♣ 3♦) … (2♣ 2♦ 2♥ 3♣ 3♥) … (A♠ A♦ A♥ K♠ K♦) …  3744 full houses |

## \*\* Full House with Jokers

In most Poker games, the "**full house**" hand is defined as three cards of the same face + two cards of the same face, other than the first, regardless of the card's suits. The cards faces are "**2**", "**3**", "**4**", "**5**", "**6**", "**7**", "**8**", "**9**", "**10**", "**J**", "**Q**", "**K**" and "**A**". The card suits are "♣", "♦", "♥" and "♠". A special card "**Joker**" (denoted as "**\***") is used as **wildcard** and can replace any other card. Jokers do not have a suite. Jokes can be used several times in a hand. Write a program to generate and print all full houses and print their number. Example:

|  |
| --- |
| **Output** |
| (2♣ 2♦ 2♥ 3♣ 3♦) … (2♣ 2♦ 2♥ 3♣ 3♦) … (2♣ 2♦ 2♥ 3♣ 3♥) … (2♣ 2♦ 2♥ 3♣ \*) …  (2♣ \* \* 3♣ \*) … (A♠ A♥ A♦ K♠ K♣) … (A♦ A♥ A♠ \* \*) … (\* \* \* \* \*)  119808 full houses |

## Angle Unit Converter (Degrees ↔ Radians)

Write a **method** to **convert from degrees to radians**. Write a **method** to **convert from radians to degrees**. You are given a number **n** and **n** queries for conversion. Each conversion **query** will consist of a **number** + space + **measure**. Measures are "**deg**" and "**rad**". Convert all radians to degrees and all degrees to radians. Print the results as **n** lines, each holding a number + space + measure. Format all numbers with 6 digit after the decimal point. Examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  180 deg  90 deg  3 rad | 3.141593 rad  1.570796rad  171.887339 deg | 2  3.141592 rad  5.5 rad | 179.999963  315.126787 | 4  0 rad  120 deg  1.55 rad  2.1 rad | 0.000000  2.094395  88.808458  120.321137 |

## Random Hands of 5 Cards

Write a program to generate **n** **random hands of 5 different cards** form a standard suit of 52 cards. Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 5 | Q♣ J♦ 6♠ 6♣ A♥  4♦ 7♣ 8♦ 9♣ 3♦  10♣ 8♥ 10♥ A♣ Q♥  2♥ 2♠ 2♣ 8♠ J♦  J♣ 10♦ J♠ A♠ K♥ | 3 | 10♠ 7♣ A♥ 3♣ A♦  2♦ 6♦ 10♣ 5♦ 5♣  J♥ A♣ 6♥ 6♦ J♣ |

## Days between Two Dates

Write a program to calculate the **difference between two dates** in number of days. The dates are entered at two consecutive lines in format **day-month-year**. Days are in range [1…31]. Months are in range [1…12]. Years are in range [1900…2100]. Examples:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 1-01-2014  2-01-2014 | 1 | 28-02-2000  8-03-2000 | 9 | 30-11-2014  27-03-2015 | 117 | 14-05-2014  14-06-1980 | -12387 |

## Sum Numbers from a Text File

Write a program to read a text file "**Input.txt**" holding a sequence of integer numbers, each at a separate line. Print the **sum of the numbers** at the console. Ensure you close correctly the file in case of exception or in case of normal execution. In case of exception (e.g. the file is missing), print "**Error**" as a result. Examples:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input.txt** | **Output** |  | **Input.txt** | **Output** |  | **Input.txt** | **Output** |
| 3  5  -1 | 7 | 100  200 | 300 | (missing file) | Error |

## List of Products

Create a class **Product** to hold products, which have **name** (string) and **price** (decimal number). Read from a text file named "**Input.txt**" a **list of products**. Each product will be in format **name** + space + **price**. You should hold the products in objects of class **Product**. **Sort** the products **by price** and write them in format **price** +space+ **name** in a file named "**Output.txt**". Ensure you close correctly all used resources. Examples:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input.txt** | **Output.txt** |  | **Input.txt** | **Output.txt** |
| milk 2.80  apple 1.20  coffee 8.50 | 1.20 apple  2.80 milk  8.50 coffee | juice 2.50  water 1.20  vodka 18.70  beer 1.12 | 1.12 beer  1.20 water  2.50 juice  18.70 vodka |

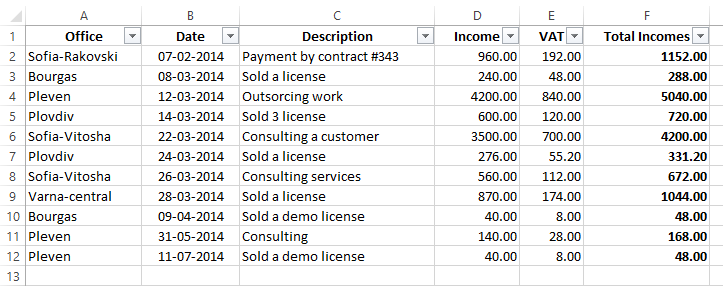
## \* Order of Products

Create a class **Product** to hold products, which have **name** (string) and **price** (decimal number). Read from a text file named "**Products.txt**" a **list of product with prices** and keep them in a list of products. Each product will be in format **name** + **space** + **price**. You should hold the products in objects of class **Product**. Read from a text file named "**Order.txt**" an order of products with quantities. Each order line will be in format **product** + space + **quantity**. Calculate and print in a text file "**Output.txt**" the **total price** of the order. Ensure you close correctly all used resources. Examples:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Products.txt** | **Order.txt** | **Output.txt** |  | **Input.txt** |  | **Output.txt** |
| milk 1.80  apple 3.20  coffee 8.50 | 12 milk  3.2 coffee  2 coffee  1.5 apple | 70.5 | juice 2.50  water 1.20  vodka 18.70  beer 1.12 | 15 water  2 vodka  3 juice  1 water | 64.1 |

## \*\*\* Excel

You are given an Excel file **Incomes-Report.xlsx** holding an incomes report in the following format:



Each office puts in this Excel file all their incomes (office, date, description, income, 20% VAT, total income). Your task is to **read the report** and to calculate the **incomes sub-totals for each office** (with VAT). Order the offices alphabetically. Print the result at the console in format **town** Total -> **incomes**. Finally calculate and print the grand total (the sum of all incomes in all offices). Sample output (for the above report):

|  |
| --- |
| **Output** |
| Bourgas Total -> 336.00  Pleven Total -> 5256.00  Plovdiv Total -> 1051.20  Sofia-Rakovski Total -> 1152.00  Sofia-Vitosha Total -> 4872.00  Varna-central Total -> 1044.00  Grand Total -> 13711.20 |

You are free to use a Java library of choice to open and read Excel spreadsheets (**.xlsx** files).

# Exam problems.\*\* - Java Basics Exam 22nd June 2014

All of the problems below are given from the previous C# Basics exams. **You are not obligated** to submit any of them in your homework. We highly recommend you to try solving some or all of them so you can be well prepared for the upcoming exam. You need to learn how to use conditional statements, loops, arrays and other things (learn in internet how or read those chapters in the book “[Fundamentals of computer programming with C#](http://www.introprogramming.info/intro-csharp-book/read-online/)”). If you still find those problems too hard for solving it’s very useful to **check** and **understand** the solutions. You can download all solutions and tests for this variant [here](https://softuni.bg/downloads/svn/csharp-basics/Feb-2014/9.%20CSharp-Basics-Exam-April-2014-Variant-1.zip) or check all [previous exams](https://softuni.bg/trainings/coursesinstances/details/2) (scroll down to the bottom of the page). You can also test your solutions in our automated [judge system](http://judge.softuni.bg/Contests/2/CSharp-Basics-Exam-10-April-2014-Morning) to see if you pass all tests.

# Problem 12\* – Cognate Words

You are a given **string** in a single line. Assume “words” are all sequences of **Latin letters**. For example in the input string "**java..?|basics/\*-+=javabasics**" we have 3 words: "**java**", "**basics**" and "**javabasics**".

Write a program to find in the input string all **unique** sets of 3 “words” {**a**, **b**, **c**}, such that **a**|**b** = **c**. Assume that "**a**|**b**" means to concatenate the “word” **b** after **a**. We call these “words” {**a**, **b**, **c**} **cognate words**.

For example in the input string "**java..?|basics/\*-+=javabasics**" we have one cognate: **java|basics=javabasics**.

*Notes*: All “words” must be **case sensitive**! Don't repeat the cognate words in the output.

### Input

The input comes from the console. It hold a single text line – the input string.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console all **cognate words** {**a**, **b**, **c**} found in the input sequence in format "**a**|**b**=**c**" (without any spaces), each at a separate line. The **order** of the output lines **is not important**. Print "**No**" in case no cognate words exist among the input sequence of characters.

### Constraints

* The characters in the input string will be no more than: **1000.**
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| java..?|basics/\*-+=javabasics | java|basics=javabasics |
| Hi, Hi, Hihi | No |
| Uni(lo,.ve=I love SoftUni (Soft) | Soft|Uni=SoftUni  lo|ve=love |
| a a aa a | a|a=aa |
| x a ab b aba a hello+java a b aaaaa | a|b=ab  ab|a=aba |
| aa bb bbaa | bb|aa=bbaa |
| ho hoho | No |

# Problem 13\*\* – Durts

Nakov and SoftUni team were bored and they decided to make a special game, called “Durts”. The rules of the game were easy: all players throw one dart and if the dart get into the figure, the player takes a point. The shape of the figure represents a cross like the picture on the right. Your task is to write a program that calculates if the dart is in the figure.

Each game starts with given **coordinates of the center (CX, CY)** of the figure, the **size r,** the **count n**, and **n coordinates (X, Y)** of the darts. See the figure with center (5, 5), r=2, and 7 darts to get a better idea.

### Input

The input comes from the console. The first and the second numbers hold the coordinates **CX and CY** of the center of the figure. The next two numbers are **r of figure** and the **count n**, followed by **n** coordinates **X** and **Y** of the thrown darts. All input numbers will be separated one from another by whitespace (one or more spaces / new lines). The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console the result “**yes**” or “**no**” for each dart in the same order, each at a separate line.

### Constraints

* The coordinates **CX** and **CY** of the **center** and **darts** coordinates (**X**, **Y**) will be integers in range [-1000…1000].
* The **r** will be positive integer in the range [0…500].
* The **count n** will be **positive** integer in the range [0…20].
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 5  2  7  2 8 2 3 4 5 5 2 5 6 7 4 9 4 | no  no  yes  no  yes  yes  no | center = (5, 5)  r = 2  n = 7  (2, 8) 🡪 no; (2, 3) 🡪 no;  (4, 5) 🡪 yes; (5, 2) 🡪 no;  (5, 6) 🡪 yes; (7, 4) 🡪 yes;  (9, 4) 🡪 no |
| -3 6  5  8  -5 2 -5 1 10 1 9 1 1 4 6 6  -100 100 3 -3 | yes  yes  no  no  yes  no  no  no | **center = (-3, 6)**  **r = 5**  **8 = 7**  **(-5, 2)** 🡪 yes; (-5,1) 🡪 yes;  (10, 1) 🡪 no; (9,1) 🡪 no;  (1, 4) 🡪 yes; (6, 6) 🡪 no;  (-100, 100) 🡪 no; (3, -3) 🡪 no |
| 5 16  2  4  3 3 5 14 6 7 8 6 | no  yes  no  no | center = (5, 16)  r = 2  n = 4  (3, 3) 🡪 no; (5, 14) 🡪 yes;  (6, 7) 🡪 no; (8, 6) 🡪 no |

# Problem 14\*\* – Exam Score

We are given a table of **students** with **exam score** and **grades** in the following form:

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

| Name | Exam Score | Grade |

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

| Peter Ivanov | 306 | 5.26 |

| George Stefanov | 120 | 3.12 |

| Maria Petrova | 400 | 6.00 |

| Petya Georgieva | 400 | 6.00 |

| Diana Kirova | 120 | 3.23 |

| Darin Mihaylov | 400 | 5.00 |

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

Write a program to aggregate the exam score data and print for each **exam score** all **students**, which have that score and the **average** **grade** for these students. Use the following format **<score> -> [<student1>, <student2>, …]; avg=<avg grade>**". Order the **score in ascending order**. Order the **students alphabetically**. Print the average grade rounded with exactly 2 digits after the decimal point. In our example, the output should be the following:

|  |
| --- |
| 120 -> [Diana Kirova, George Stefanov]; avg=3.18  306 -> [Peter Ivanov]; avg=5.26  400 -> [Darin Mihaylov, Maria Petrova, Petya Georgieva]; avg=5.67 |

### Input

The input comes from the console. At the first 3 lines stays the header of the form that don’t have important information for you, followed by a few lines holding exam information in format **<name> | <score> | <grade>**, separated by whitespaces and pipes. Student names are **unique**. The last line is the footer and consists of '**-**' only. The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print for each **exam score** (increasingly) all **students** (alphabetically), which have that score and the **average** **grade** for these students in the above described format (see also the examples).

### Constraints

* The **count** of the input lines is in the range [5…1000] including the table header and borders.
* The **<score>** is an integer in the range [0…400].
* The **<name>** consists of only of **Latin characters and spaces**, with length of [1…50].
* The **<grade>** is a number number in the range [2.00…6.00].
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| --------------------------------------------  | Name | Exam Score | Grade |  --------------------------------------------  | George Ivanov | 306 | 5.26 |  | George Stefanov | 120 | 3.12 |  | Petya Koleva | 400 | 6.00 |  | Aleksandar Stoyanov | 300 | 5.00 |  | Diana Kirova | 120 | 3.23 |  | Ivan Ivanov | 0 | 2.00 |  | Kalin Petrov | 300 | 5.40 |  | Stoyan Kotsev | 400 | 5.00 |  | Krasimir Mihaylov | 400 | 5.98 |  -------------------------------------------- | 0 -> [Ivan Ivanov]; avg=2.0  120 -> [Diana Kirova, George Stefanov]; avg=3.18  300 -> [Aleksandar Stoyanov, Kalin Petrov]; avg=5.20  306 -> [George Ivanov]; avg=5.26  400 -> [Krasimir Mihaylov, Petya Koleva, Stoyan Kotsev]; avg=5.66 |

# Problem 15\*\*\* – Straight Flush

Nakov loves to play [poker](http://en.wikipedia.org/wiki/List_of_poker_hands), but unfortunately, he does not know when he has a strong hand. One of the strongest hands in poker is **Straight Flush**: a sequence of 5 consecutive cards of the same suit. The poker game uses a **standard deck of 52 cards**. The card faces in increasing order are: **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **J**, **Q**, **K** and **A**. The cards suits are denoted by the letters **S** (spades), **H** (hearts), **D** (diamonds) and **C** (clubs).

Help Nakov to make some money for food! Write the program that check if his hand holds one or more Straight Flushes.

For example, the hand "**9D, 2S, 10D, AD, 10H, JD, QD, KD**" contains two Straight Flushes: **[9D, 10D, JD, QD, KD], [10D, JD, QD, KD, AD]**.

*Note:* The A can be only after K, not before 2.

### Input

The input comes from the console as a **single line, holding the hand of cards**, separated by comma and space between each two cards. The input cards are **unique** (there are **no duplicates**). The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* If there are Straight Flushes – print them, each at a new line in format **[card1, card2, card3, card4, card5]**. The **ordering** of the output lines is not significant (the Straight Flushes can be ordered in arbitrary way).
* If no Straight Flushes exist among the given cards, print a message “**No Straight Flushes**”.

### Constraints

* The **count** the cards will be in the range [1…52].
* The **card faces** will beone of the following values: [**2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **J**, **Q**, **K**, **A**].
* The **card suits** will beone of the following values: [**S**, **H**, **D**, **C**].
* Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9D, 2S, 10D, AD, 10H, JD, QD, KD | [9D, 10D, JD, QD, KD]  [10D, JD, QD, KD, AD] |
| AS, KH, 10C | No Straight Flushes |
| 2S, 2C, 2D, 2H, AS, KH, 10C | No Straight Flushes |
| 5H, AS, 10C, 8H, KS, KH, KD, 9H, JH, QS, 3H, QD, 4H, QH, 8S, 10D, 6H, 10S, 10H, 7C, JD, JS, 2H, 7S, 7D | [8H, 9H, 10H, JH, QH]  [9H, 10H, JH, QH, KH]  [10S, JS, QS, KS, AS]  [2H, 3H, 4H, 5H, 6H] |