# Unit 1 Assignment

## GENERAL

1. Save the work of each exercise in a separate Java source code file and name it *StuPrefixClassName.java*.

The *StuPrefix* portion should be the first letter of your last name and the first letter of your first name followed by the last two digits of your Park ID. For example, student John Doe with Park ID 12345670 will add prefix DJ70 to the names of all source code files. If one program should be called TestInput, he needs to put his work in DJ70TestInput.java.

See DJ70TestInput.java under Files > Student Files > Tutorials for required comments and recommended coding style.

2. Overall comment your program appropriately (file prolog comment, method prolog comment, and comments on the code). Pay attention to the standard stuff like coding style, meaningful identifier names, indention, and locations of braces.

## EXERCISES

1. (Name this program StuPreInputStats) Write a program that takes any number of non-negative integers as input, and output statistics on the data. A negative integer ends the input and is not included in the statistics. The numbers are ratings on a recently released video game and should be in the range of [0, 5]. Assume there will be at least one rating input. The statistics generated by the program should include the total number of ratings entered, and the number and percentage of each of the following types of ratings:

* Liked: rating is 4 or 5
* Neutral: rating is 3
* Disliked: rating is in 0 ~ 2
* Invalid: > 5. Invalid data may exist due to input errors.

Here is a sample run of this program (green italic text indicates user input):

Please enter the ratings (end with a negative value):

*4 5 1 0 3 4 6 5 4 7 3 4 5 4 2 -2*

Total # of ratings: 15

Positive ratings (4~5): 8 (53%)

Neutral ratings (3): 2 (13%)

Negative ratings (0~2): 3 (20%)

Invalid ratings (>5): 2 (13%)

Another sample run with data entered one at a line:

Please enter the ratings (end with a negative value):

*4*

*5*

*1*

*0*

*3*

*4*

*6*

*5*

*4*

*7*

*3*

*4*

*5*

*4*

*2*

*-1*

Total # of ratings: 15

Positive ratings (4~5): 8 (53%)

Neutral ratings (3): 2 (13%)

Negative ratings (0~2): 3 (20%)

Invalid ratings (>5): 2 (13%)

A sample run with a different set of numbers:

Please enter the ratings (end with a negative value):

*2*

*1*

*3*

*4*

*-5*

Total # of ratings: 4

Positive ratings (4~5): 1 (25%)

Neutral ratings (3): 1 (25%)

Negative ratings (0~2): 2 (50%)

Invalid ratings (>5): 0 (0%)

**How to Print a Number in Percentage**

System.out.printf() introduced in Ch2.7 of our textbook can be used to format floating-point values for output. Here is an example segment of code to print out one integer (formatted with format specifier %d) and one percentage (%.0f is to print a floating-point number with no digits after the decimal point. %% is to print out % itself).

int count = 4;

int total = 15;

System.out.printf("We want to see count %d and percentage %.0f%%.\n", count, 100.0\* count / total);

It will print out:

We want to see count 4 and percentage 27%.

Here is how it works. 4 divided by 15 is 0.2666…. Since we want to see the value in percentage (26.66…) instead of 0.2666…, we use 100.0 \* count / total. Format specifier %.0f turns 26.66… into 27 and extra digits are dropped after rounding. We have to print % manually. Because % is used by printf() to start a format specifier (%d, %f, …), we need to use %% to print out the % character itself. See ch9.2 Output Formatting if you want to know more about printf().

**How to Build This Program**

Here is a possible idea to build this program incrementally. Makes sure your program works properly (compiles and generates proper results) at each step before adding more code.

* Set up a program with an empty main() method.
* Add data input portion. To test your work, add temporary prints to print out the data so you can verify that the input is done properly. Remove those temporary prints (or at least comment them out) after you’re done with this step.
* Add code to count and print total.
* Add code to count and print different types of ratings.
* Add code to print percentages.

Once completed, run your program as shown in the three sample runs and include a screenshot of each run (showing execution result) in your assignment report document. This is to remind you that the final version of your program should work as shown in the sample runs, including prompt messages and data formatting.

1. (Name this program StuPreIPv6) Each network device must be assigned an IP address. An IPv6 (IP address version 6) address is represented in the preferred format as eight groups of four hexadecimal digits. A hexadecimal digit may be 0 ~ 9 or a ~ f (or A ~ F). The groups are separated by colons (:). An example of an IPv6 address is:

2001:0db8:85a3:0000:0000:8a2e:0370:7001

Write a method to decide whether a string contains a valid IPv6 address in the preferred format (each h represents a hexadecimal digit):

hhhh:hhhh:hhhh:hhhh:hhhh:hhhh:hhhh:hhhh

For this assignment, your method must loop through individual characters in the string. Approaches using regular expression and/or API classes like Pattern, Matcher will receive zero credit.

Start with a method stub like this one. The main() will be used for unit testing (see ch6.6).

// Returns true if the string parameter contains a valid IPv6

// address in the preferred format. Otherwise returns false.

public static boolean isValidPreferFormatIpv6(String address) {

return false;

}

// for unit testing

public static void main(String[] args) {

System.out.println("Testing started ...");

String str; // store test strings

str = "";

if (isValidPreferFormatIpv6(str) != false) // if not the expected result

System.out.println( "isValidPreferFormatIpv6(" + str + ") failed";

str = "0123:4567:89ab:cdef:0123:4567:89ab:cdef";

if (isValidPreferFormatIpv6(str) != true) // if not the expected result

System.out.println( "(isValidPreferFormatIpv6(" + str + ") failed";

// add more testing cases

System.out.println("Testing completed.");

}

Here are examples of valid and invalid IPv6 addresses:

Valid:

0123:4567:89ab:cdef:0123:4567:89Ab:cDEf

0000:0000:0000:dddd:eeee:dddd:eeee:ffff

0000:0000:0000:0000:0000:0000:0000:0000

Invalid:

123:4567:89ab:cdef:0123:4567:89ab:cdef0 -> not 4 hex digits in 1st and last groups

0000:0000:0000 -> not enough digits/groups

01hg:4567:89ab:cdef:0123:4567:89ab:cdef -> non-hex digits (in 1st group)

0123:4567::89b:cdef:0123:4567:89ab:cdef -> two colons together

Use a proper incremental development strategy to build your program. It is a good idea to first check whether the parameter string is of the right length.

Following the unit testing examples and add at least 3 more testing cases of your choice to your main(): 1 valid address and 2 invalid ones.

When you are done, include a screenshot of the execution of your program in your assignment report document. If it works properly, your program should only print those two lines:

Testing started ...

Testing completed.

1. Reflection: answer those questions AFTER you’ve completed this assignment:
2. What’s the hardest part of this assignment for you? Please explain.
3. What would you do differently if you could ride a time machine back to day 1 of this week?

## SUBMISSION

Submit two .java files + one word/PDF document. Please put all screenshots and answers to questions (exercise 3) into your word/PDF document.

* Exercise 1: provide the java file and screenshots of the execution results of three required execution runs
* Exercise 2: provide the java file and a screenshot of the execution result
* Exercise 3: Assignment reflection
* Check the completeness of your work against the rubric before turning it in.

## Rubric: Unit 1 Assignment

| **Criteria** | **Ratings** | | | **Pts** |
| --- | --- | --- | --- | --- |
| **Exercise 1 (InputStats)** | 8 pts. Correct method with the required steps. | 7 pts ~ 1 pts. One or more incorrect steps:  (2 pts) Input unknown number of integers ended with a negative value in a loop.  (1 pts) Count total number of ratings,  (2 pts) Classify and count positive, neutral, negative, and invalid ratings.  (2 pts) Calculate percentages of positive, neutral, negative, and invalid ratings.  (1 pts) Print all required counts and percentages in the correct format. | 0 pts. No steps are correct; no submission. | 8 |
| **Exercise 1: two screenshots of execution results** |  | 0.5 pts. Provided three required screenshots. | 0 pts. No submission or missed at least one. | 0.5 |
| **Exercise 2 (IPv6)** | 8 pts. Correct method meeting requirements. | 7 ~ 1 pts. At least one requirement is incorrect or missing:  (1 pts) Handle string parameter of incorrect length.  (1 pts) Loop through the characters in the string parameter.  (2 pts) Handle an invalid IPv6 address string due to invalid characters such as none-hex digits or not colon in a loop setting.  (2 pts) Handle an invalid IPv6 address string due to incorrectly grouped digits in a loop setting.  (1 pts) Return true if and only if a valid IPv6 address string in the preferred format.  (1 pts) 3 additional testing cases (1 valid, 2 invalid) in main().  (Notes) Handling includes a proper return value; approaches using regular expression may receive only the first point (test length) and the last point (testing cases). | 0 pts. Incorrect or missed all requirements; no submission | 8 |
| **Exercise 2: screenshot of execution result** |  | 0.5 pts. Provided. | 0 pts. No submission. | 0.5 |
| **Exercise 3. Assignment reflection** | 1 pts. Answered both questions. | 0.5 pts. Answered only one question. | 0 pts. Didn’t answer the questions; no submission. | 1 |
| **Required program name (StuPre part) + Style Points (Proper comments; meaning identifier names; consistent indentation)** | 2 pts. Correctly named program and proper style in all three areas of style points. | 1 pts. Problems in one of the following areas: program name, three areas of style points. | 0 pts. Problems in two or more of the following areas: program name, three areas of style points. | 2 |
|  |  |  | *Total Points* | 20 |