Assignment 03 - StringHandler

CSCI 242 - Computer Science II

Version 2.0.0

Due date is posted in D2L.

Please read the entire project description before beginning the project.

Objective

The objectives of this assignment are:

1. To reinforce the concepts of Abstract Classes and Interfaces in Java.
2. To reinforce understanding of how to implement a class given its UML class diagram.

To accomplish these objectives, you will implement a class hierarchy dealing with strings of two specific kinds. You will also write a driver class that instantiates the objects and delivers the program output.

Introduction

In programming, we need to parse strings all the time. If we consider the problem in general, there are many, many different kinds of strings. Of these general kinds of strings, two such more specific types of strings might be strings of hexadecimal numbers and password strings. Generally, they are just strings – like every other kind of string. Specifically, though, their compositions are quite different.

To say it slightly differently, all strings have some common behaviors such as parsing letters, digits and other characters. However, the way in which each specific type of string processes the end result can be quite different!

The Assignment

Your assignment is to implement a string parsing system, called `StringHandler`, in Java that uses interfaces to specify common behavior and interface implementations to specify specific behavior. To do this, you will be given a UML class diagram. From there, you will start by writing two interfaces from which the specific implementations will be derived. You will then implement two concrete classes from the interfaces as well as a string parser that calls the specific parsing methods given their implementations. Finally, you will write a driver program to instantiate the classes and use them.

All the project requirements are listed below. This is an *individual* assignment so you are not to show your code to any other student. You are allowed to discuss the assignment with classmates but you are not to show your code in any way to any other student.

In general:

- Your project must compile error-free to be considered for grading. If your program does not compile without errors, you will receive a grade of 0 (zero).
- Your project must comply with the course guidelines found in "Coding Guidelines" in D2L, **General > Coding Guidelines**. This includes documenting using Javadoc. Deductions will be made for programs that to not comply with the document.
- The format of your final output must match exactly that of your instructor's. See the section titled "Sample Run" below. Deductions will be made for programs that to not comply with output requirements. If you think there is a possible typo, please talk to your instructor.

Remember what the UML composition symbol (the diamond) means: - a "has a" relationship exists between a class and a subclass.

Finally, use the `Character` class to your advantage!

The Project

You are required to use IntelliJ IDEA for your assignment.

Your project must be named: `A03-StringHandler`.

You should always use good Software Engineering (SE) techniques. We will discuss what "good software engineering techniques" are as we progress through the semester. Use constants where needed; use lots of methods and comments!

The System

The following class diagram describes the classes as well as the relationships you are to use to implement your `StringHandler`.

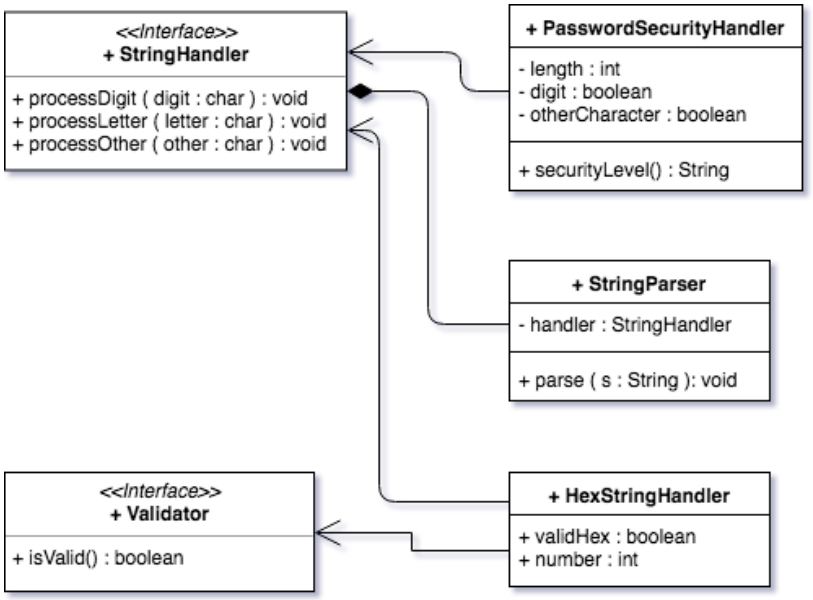

Figure 1 - StringHandler UML Diagram

This figure shows two interfaces, `StringHandler` and `Validator`. The two interfaces describe the methods the concrete classes are required to implement. Two concrete classes, `PasswordSecurityHandler` and `HexStringHandler` implement the `StringHandler` interface. `HexStringHandler` also implements the `Validator` interface. Finally, the `StringParser` contains a `StringHandler` as a member variable.

Each implementer of the `StringHandler` interface will need to process the characters of the string differently. For example, the `PasswordSecurityHandler` will not process the characters of the string the same way that the `HexStringHandler` does. Thus, it makes sense to separate the processing of the characters (`StringHandler`) from the processing of the string (`StringParser`). To this end, the `StringParser` determines what each character is and hands the processing of the character off to the implementer of the `StringHandler`. Since the `StringParser` contains a `StringHandler` as its member variable, it is easy for the `StringParser` to call the appropriate `StringHandler`. The call is accomplished by the `parse()` method of the `StringParser`.

As an example of how this might work, examine the following lines of Java:

```
01  HexStringHandler hsh = new HexStringHandler();
02  sp = new StringParser ( hsh );
03  sp.parse ("2D3F" );
```

Line 01 shows the instantiation of a `HexStringHandler`. Line 02 shows an instantiation of a `StringParser` with the afore mentioned `HexStringHandler` as its constructor parameter. Finally, the `parse()` method is called to run the proper `StringHandler` methods implemented in `HexStringHandler` in line 03.

A final couple of notes about the figure:

- Typically, constants are not shown in ULM diagrams. Thus, they have not been shown above.
- The object constructors have also been omitted from the diagram. You must implement proper constructors.
- The "getters" and "setters" have also been omitted from the diagram. You must implement proper "getters" and "setters".

The Interfaces & Classes

The following describe the classes you are to use to implement your `StringHandler`.

The `StringHandler` Interface

Refer to the class diagram above to see the design of the `StringHandler` interface. Remember, each implementer of the `StringHandler` interface must provide the specialized implementations of `processDigit()`, `processLetter()` and `processOther()`.

The `Validator` Interface

Refer to the class diagram above to see the design of the `Validator` interface. Remember, each implementer of the `Validator` interface must provide the specialized implementation of `isValid()`.

The `PasswordSecurityHandler` Class

The class diagram describes the design of the `PasswordSecurityHandler` class.

Member Constants

The member constants are as follows:

- `SECURITY_LEVEL_WEAK`. Constant value of '`weak`'.

- `SECURITY_LEVEL_MEDIUM`. Constant value of '`medium`'.
- `SECURITY_LEVEL_STRONG`. Constant value of '`strong`'.

Member Variables
The member variables are as follows:

- `length` holds the number of characters in the password.
- `digit` is a flag that indicates a digit was found in the password.
- `otherCharacter` is a flag that indicates an "other" character was found in the password. An "other" character is defined as being neither a letter nor a digit.

Member Methods
The class has `protected` getters. This means only classes in the package have access to the getters. The application has no need for getters and setters as the other methods provide the needed information to the application.

The `securityLevel()` method should return a `String` that represents the security level of the password. Valid security levels are, "`weak`", "`medium`" and "`strong`". A ***weak*** password is defined as a password with less than eight characters. A ***medium*** password is defined as a password having a length of eight or more characters and having either a digit or an "other" character (see `processOther()` above for what an "other" character is). A ***strong*** password is defined as a password having a length of eight or more characters and having both a digit and an "other" character.

When processing a digit, make sure the argument is a digit, then update `length` and `digit` appropriately. When processing a letter, you should make sure the argument is a letter and update `length` appropriately. Finally, when processing an "other", you should make sure the argument is not a digit, letter or control character and update `length` and `other` appropriately.

When processing a digit, letter or "other", you should throw an `IllegalArgumentException` if the argument is not what is expected.

The `HexStringHandler` Class
The class diagram describes the design of the `HexStringHandler` class.

Member Constants
The member constants are as follows:

- `INVALID_NUMBER`. Constant value of `-1`.
- `NUMBER_SYSTEM`. Constant value of `16`.
- `NUMBER_LETTER_MIN`. Constant value of `10`.
- `NUMBER_LETTER_MAX`. Constant value of `16`.

Member Variables
The member variables are as follows:

- `validHex` is a flag that indicates whether the string being parsed contains only valid hex characters.
- `number` holds the decimal equivalent of the hex value.

Member Methods

The class has two "getter" methods: `isValid()` and `getNumber()`. `getNumber()` should return either `number` of `INVALID_NUMBER`.

When processing a digit, make sure the argument is a valid hex digit, then calculate the decimal equivalent of the provided digit. When processing a letter, you should make sure the argument is a valid hex letter and calculate the decimal equivalent of the provided letter. Finally, when processing an "other", you should make sure the argument is not a digit, letter or control character and update `validHex` appropriately.

When processing a digit, letter or "other", you should throw an `IllegalArgumentException` if the argument is not what is expected.

The `StringParser` Class

The class diagram describes the design of the `StringParser` class.

Member Constants

The `StringParser` class does not use any member constants. This relationship is shown above in the class diagram as a solid diamond at the end of the connector.

Member Variables

The `StringParser` class has only a single member variable: `handler`. `handler` is of type `StringHandler`.

Member Methods

The `parse()` method takes the `String` to parse as a parameter and walks through the `String` a character at a time, determining the type of character and calling the appropriate method of the `StringHander` interface for each character. Note that having an object of the `StringHandler` interface in the `StringParser` class makes the `StringParser` class reusable with any class that implements the `StringHandler` interface!

Sample Run

The following is a "correct" sample run. My inputs are shown in bold.

```
run:
Enter a hexadecimal number >
4AD
4AD = 1197
A strong password has at least eight
characters and contains at least one digit
and one special characters.
Enter a password > qwerty#56
qwerty#56's security is: strong
BUILD SUCCESSFUL (total time: 53 seconds)
```

Extra Credit

Once your assignment is up and working, you may attempt the extra credit.

New String Handlers

Two extra credit opportunities exist for this assignment:

1. Five extra credit points will be awarded for a completed and working `UsTelephoneStringHandler` class. This class must correctly parse a telephone number from the United States of America.

2. Ten extra credit points will be awarded for a completed and working `EmailStringHandler` class. Parsing email addresses is quite complex – at least if you try to handle every possibility. The Internet Engineering Task Force (IETF) manages the format of email addresses (as well as other information about email) and publishes a document that describes the format of email addresses in detail. This document is called "Standard for the Format of ARPA Internet Text Messages" (RFC 822) and can be found at `https://tools.ietf.org/html/rfc822`.

Note that these types of strings are trickier as they not only have certain characters that are valid (like `HexStringHandler` and `PasswordStringHandler`) but they also have formatting specifications. In other words, the characters of the string must be in a certain order for them to be correct.

Rules for Extra Credit

The following rules apply for extra credit:

- Extra credit will not be given if the "for credit" functionality does not function. In other words, extra credit is… EXTRA. Meaning, you must complete the "for credit" requirements and they must work properly before being considered for extra credit.
- The extra credit possibilities must be completed in order. You must complete `UsTelephoneStringHandler` before you complete `EmailStringHandler`.

Submission

Zip up your entire IntelliJ IDEA project folder into a single zip file named `A03.zip` and submit that one file. Submit the Zip file to the D2L dropbox labeled `Assignment 03 – StringHandler`. I should be able to unzip the entire project and load the entire project in IntelliJ IDEA without incident.

Grading

See the document titled, "Programming Requirements and Rubric" found in the "Content" section in D2L for details on how the program will be graded.

Ω