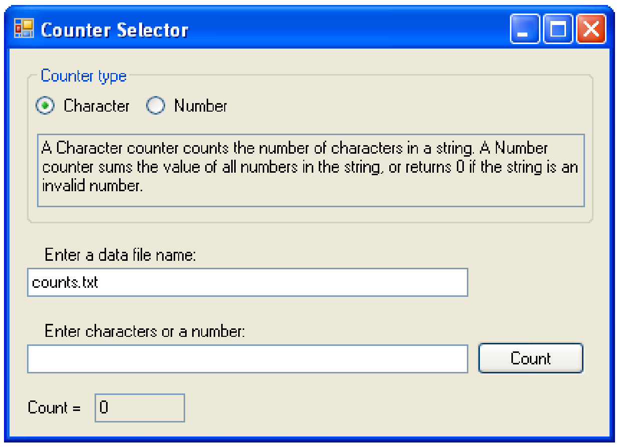
This Individual Project will have you coding to a UML design specification. Attached you will find both a Requirements Specification and a Design Specification. Your responsibility is to implement all of the design, as specified, while meeting all of the requirements, as specified. Note that all of the design must be implemented, all of the requirements must be met, and the code must be functional in order for you to receive full credit. During your implementation of the design, you must also fill out the PSP0 forms for the appropriate sections. Furthermore, your code must conform to the Class Coding Standards.

Change Summary From Original Version

- Updated “Functionality” class diagram
- Added note about interfaces implemented using only “.h” files
- Modified submission requirements to include Moodle and Subversion
- Added method to SelectorForm

You may add to both the requirements specification and to the design specification. You must submit Word documents containing the updated specifications that detail your additions. Note, for example, that the class diagrams do not include constructors. These should be added as you define them for specific classes **for any classes that do not use the default constructor**.

The exception is the Selector Form. You need not model any standard C++ windows form functionality. However, you **must** create a GUI and update the model for program specific methods and attributes (*e.g.*, `getStringEntry()`). Your form should look similar to the following:



A Character counter counts the number of characters in a string. A Number counter sums the value of all numbers in the string, or returns 0 if the string is an invalid number.

counts.txt

0

Note that after the file is first written, the data file text box will be disabled.

The entry point for your program is CS325_09Fa_IP04_[NAME], where [NAME] is your last name and first letter of your first name (*e.g.*, MAYERG). Please do not include the braces (‘[‘ and ‘]’). Note that, as the “entry point”, this means that `main()` is included in the implementation file. Also, there will be both a ‘CS325_09Fa_IP04_[NAME].h’ and a ‘CS325_09Fa_IP04_[NAME].cpp’ file. Note that interface classes will have just a “.h” file (there is no implementation and, therefore, no implementation (“.cpp”) file.

A revised requirements and/or design specification file should be named "IP04_[NAME]_Specs.doc". Your code and specifications should be archived as a zip file named "IP04_[NAME].zip". This zip archive is due to be uploaded to Moodle no later than 11:00 AM on Monday, 19 October 2009. Your code, and associated revisions, must also appear on your specific Subversion server. The latest version, time-stamped before 11 AM, will be used to test your code. Keep configuration control. Your Moodle submission must match your Subversion submission. Late submissions will be penalized per syllabus.

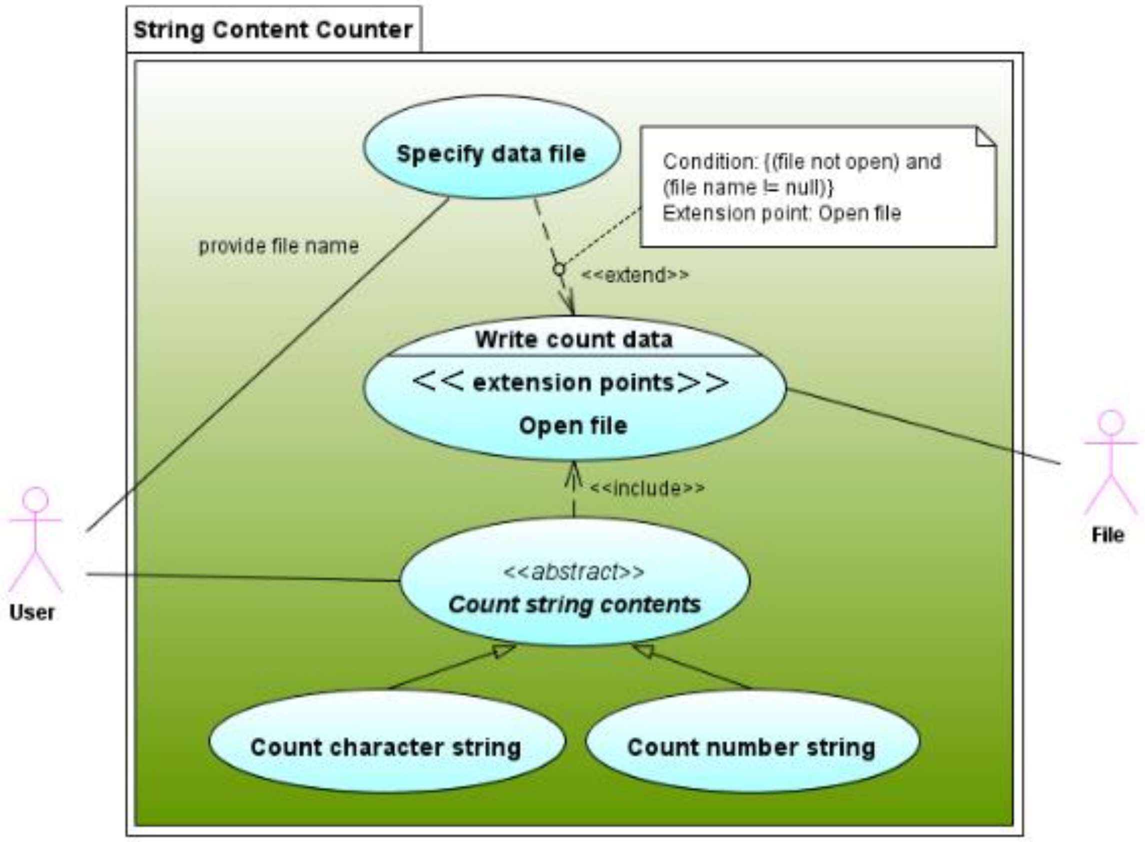
This assignment is worth 25 points. A maximum of 15 points will be assigned for your correct implementation of the UML design and your code's functionality. A maximum of 5 points will be assigned for your PSP0 documentation. A maximum of 5 points will be assigned for your revised specifications.

IP04 Requirements Specification

Requirements:

1. Count character string contents as the number of characters (including whitespace characters) in the string. Character strings do not contain numbers (*i.e.*, [0..9]).
2. Count numeric string contents (strings with all numbers [0..9] as characters) as the sum of each number character in the string. For example, "9245" would have a count of 20 (9 + 2 + 4 + 5 = 20).
3. Show the count to the user and write count data to a file.

Use Case and Descriptions:



Use Case Name:	**<< abstract >> Count string contents**	
Related Requirements	1, 2, 3	
Goal in Content	A string is provided to the system to be counted and the count is written to a file	
Preconditions	[none]	
Successful End Condition	The characters in the string are properly counted and written to the file	
Failed End Condition	The string is not counted	
Primary Actors	User	
Secondary Actors	File	
Trigger	The user provides a string to the system to be counted	
Main Flow	**Step**	**Action**
	1	The user specifies the type of string
	2	The user provides a string to be counted
	3	The string is counted per specified count type
	4	The count is provided to the user
	5	The count is written to the file
	include::Write count data	
Extensions	**Step**	**Branching Action**
	2.1	The provided string is not of the specified type or is null
	2.2	The string is not counted

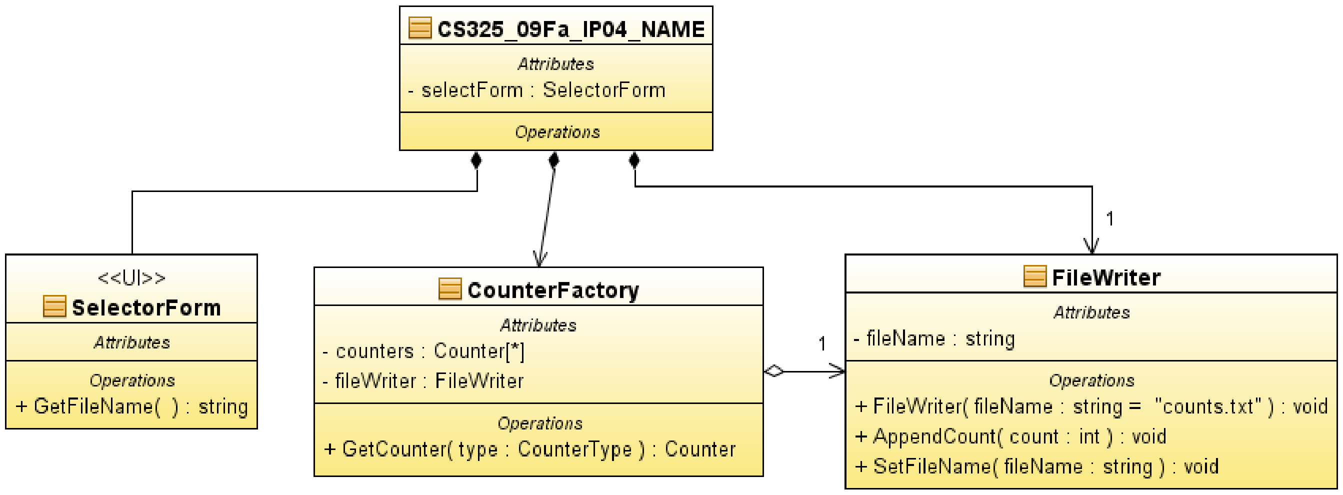
Use Case Name:	**Write count data**	
Related Requirements	3	
Goal in Content	A string count is written to a file	
Preconditions	The count is a valid integer	
Successful End Condition	The count is written to the file	
Failed End Condition	The count is not written to the file	
Primary Actors	File	
Secondary Actors	[none]	
Trigger	A "Count string contents" use case provides a string count result	
Main Flow	**Step**	**Action**
	1	A count is provided to the system
	2	Ensure the file to be written is open
	3	The count is written to the file
Extensions	**Step**	**Branching Action**
	2.1	The file to be written is not open
	2.2	Open file
	extend::Specify data file, Condition:{file name != null}	

Use Case Name:	**Write count data**	
Related Requirements	3	
Goal in Content	The name of the file to which count data is written is provided to the system	
Preconditions	The file to which count data is written is not yet open	
Successful End Condition	A file name is provided to the system	
Failed End Condition	The default file name is used	
Primary Actors	User	
Secondary Actors	[none]	
Trigger	The user provides a file name string to the system	
Main Flow	**Step**	**Action**
	1	A file name is provided to the system
	2	The system stores the user-specified file name
	3	A file with the specified name is opened and used to write count data
Extensions	**Step**	**Branching Action**
	3.1	The count data file is already open
	2.2	The existing file is used to store count data

IP04 Design Specification

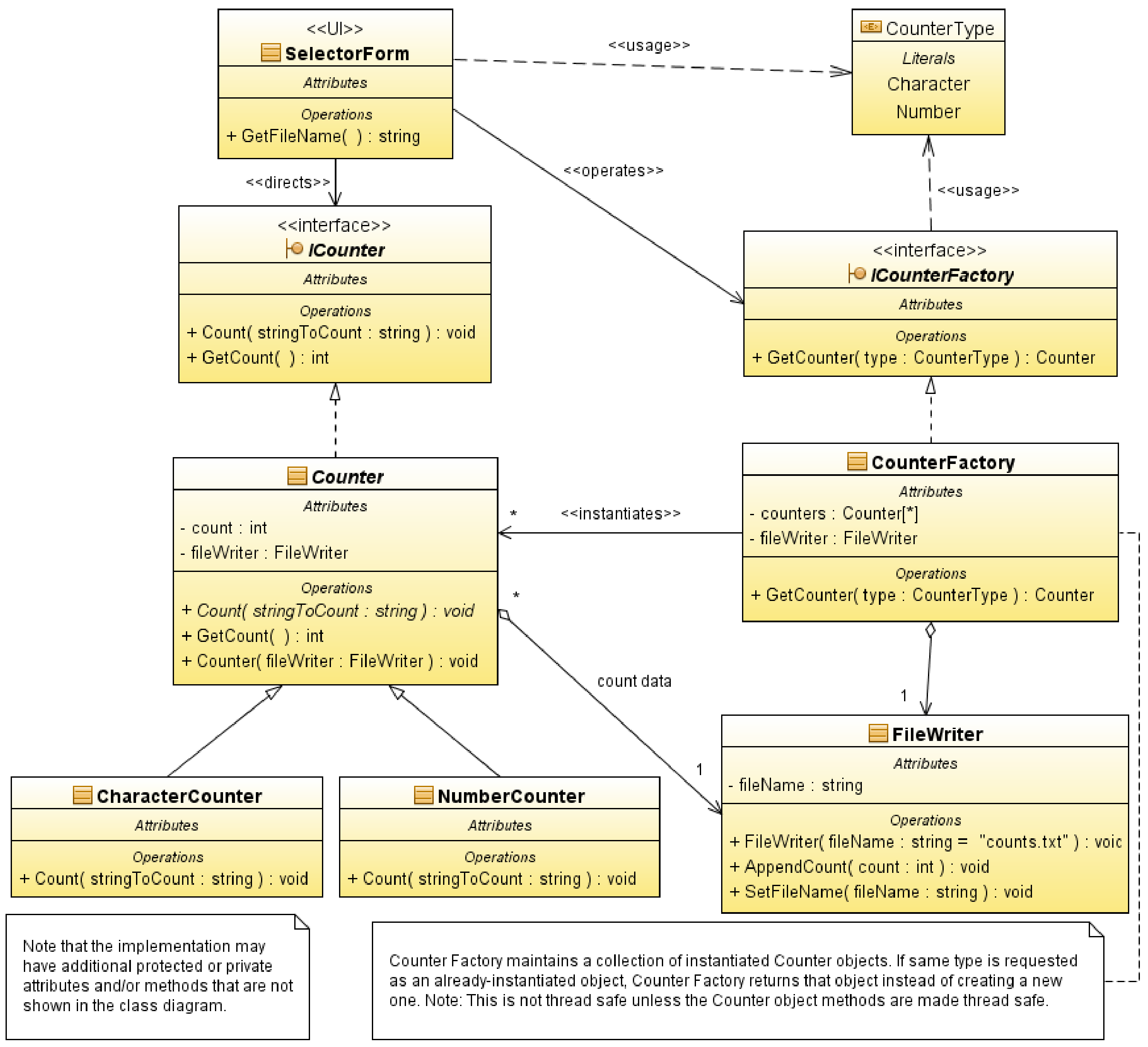
Class Diagrams:

Program Entry View:


CS325_09Fa_IP04_NAME
Attributes
- selectForm : SelectorForm
Operations
<<UI>>
SelectorForm
Attributes
Operations
+ GetFileName( ) : string
CounterFactory
Attributes
- counters : Counter[*]
- fileWriter : FileWriter
Operations
+ GetCounter(type : CounterType) : Counter
FileWriter
Attributes
- fileName : string
Operations
+ FileWriter(fileName : string = "counts.txt") : void
+ AppendCount(count : int) : void
+ SetFileName(fileName : string) : void
1
1

Note: While a "string" is specified, which might lead one to use the C string type, the developer may use the "String" class, as long as a single string-type is used consistantly throughout the program.

Functionality View:



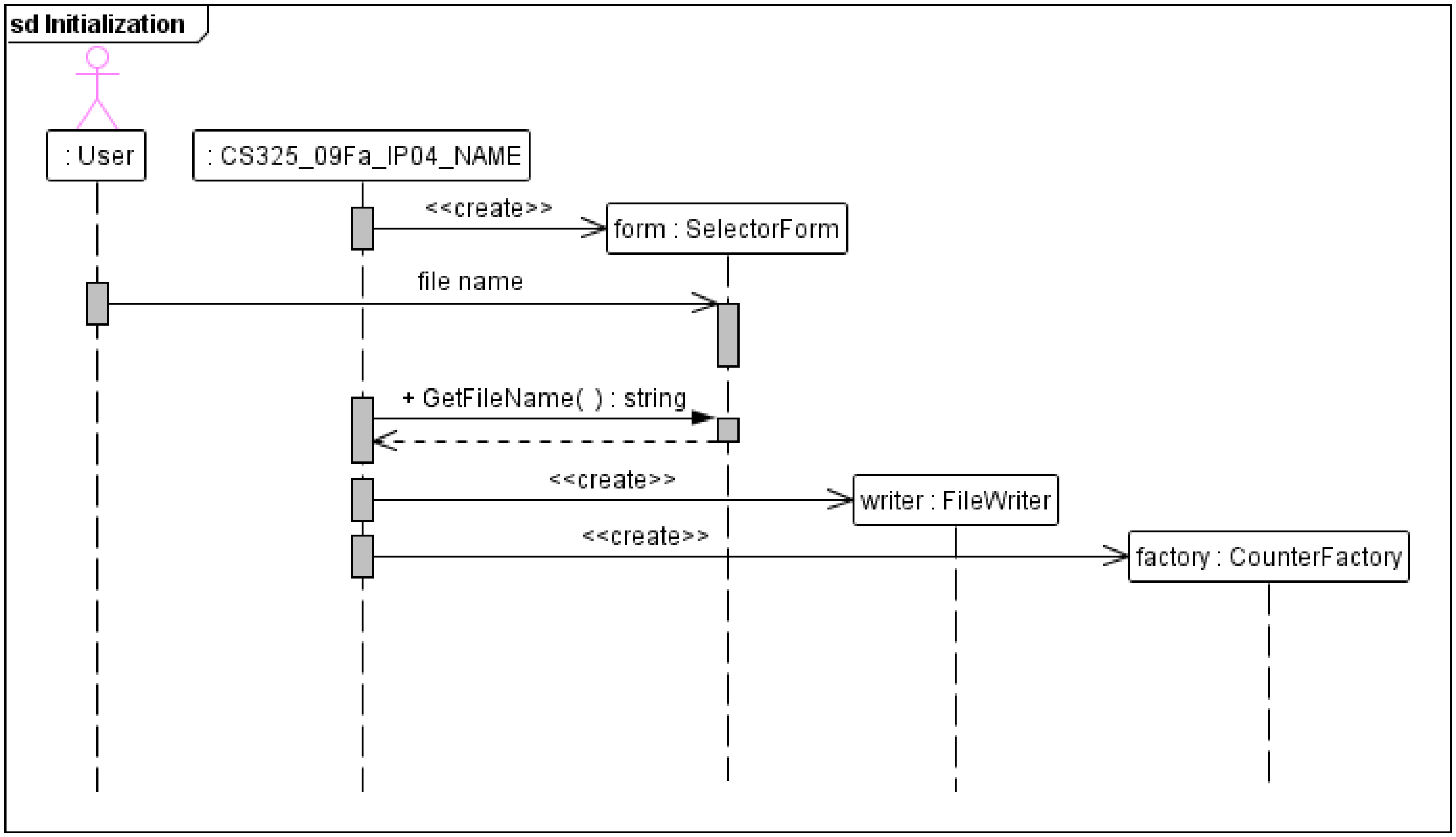
<<UI>>
SelectorForm
Attributes
Operations
+ GetFileName( ) : string
<<usage>>
CounterType
Literals
Character
Number
<<operates>>
<<directs>>
<<usage>>
<<interface>>
ICounter
Attributes
Operations
+ Count(stringToCount : string) : void
+ GetCount( ) : int
<<interface>>
ICounterFactory
Attributes
Operations
+ GetCounter(type : CounterType) : Counter
Counter
Attributes
- count : int
- fileWriter : FileWriter
Operations
+ Count(stringToCount : string) : void
+ GetCount( ) : int
+ Counter(fileWriter : FileWriter) : void
<<instantiates>>
*
*
count data
1
1
CounterFactory
Attributes
- counters : Counter[*]
- fileWriter : FileWriter
Operations
+ GetCounter(type : CounterType) : Counter
FileWriter
Attributes
- fileName : string
Operations
+ FileWriter(fileName : string = "counts.txt") : voic
+ AppendCount(count : int) : void
+ SetFileName(fileName : string) : void
CharacterCounter
Attributes
Operations
+ Count(stringToCount : string) : void
NumberCounter
Attributes
Operations
+ Count(stringToCount : string) : void

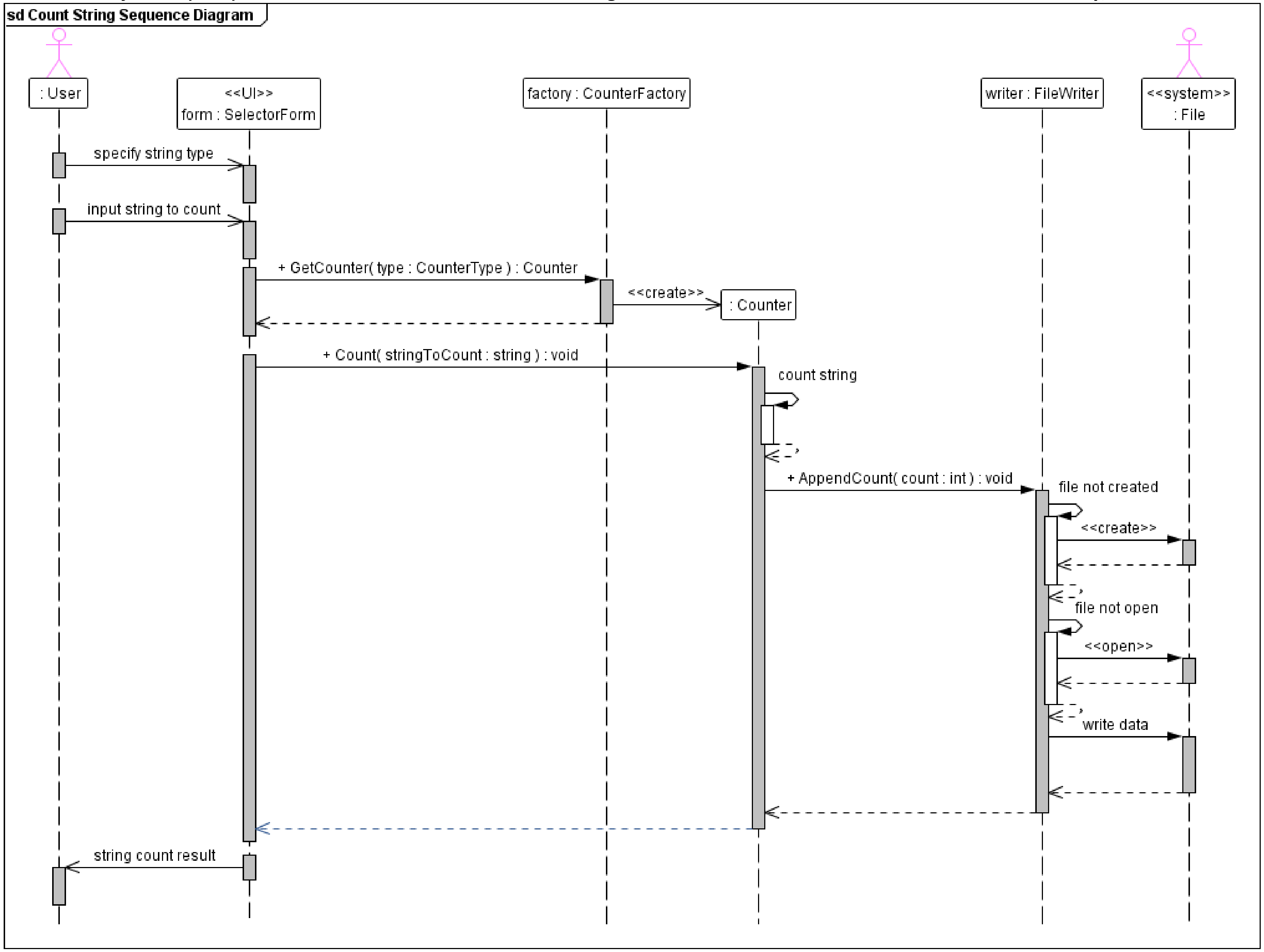
Note that the implementation may have additional protected or private attributes and/or methods that are not shown in the class diagram.

Counter Factory maintains a collection of instantiated Counter objects. If same type is requested as an already-instantiated object, Counter Factory returns that object instead of creating a new one. Note: This is not thread safe unless the Counter object methods are made thread safe.

Sequence Diagrams:

Note: This sequence diagram need not be implemented exactly as shown. It is here to provide you a suggested approach. Depending on your user form, the user could try to send the file name at any time. Your program should handle such cases.




sd Count String Sequence Diagram
: User
<<UI>>
form : SelectorForm
factory : CounterFactory
writer : FileWriter
<<system>>
: File
specify string type
input string to count
+ GetCounter(type : CounterType) : Counter
<<create>>
: Counter
+ Count(stringToCount : string) : void
count string
+ AppendCount(count : int) : void
file not created
<<create>>
file not open
<<open>>
write data
string count result