Course Project

CSE 460
Software Analysis and Design
(Fall 2022)

Assigned Date: Nov 04, 2022
Due Date: Nov. 26, 2022

Posting ID |__|__|__|__|-|__|__|__|

REVISION HISTORY

Version	Release Date	Updates
1.0	11/04/2022	(Initial release of description and scope of the project)
1.01	11/11/2022	See page 5

Please post and ask questions **early**. This will help everyone!

Vehicle Manufacturers and Rental Companies

Supply-chain systems use software systems to connect buyers and suppliers. **Supplier companies are producers of products that buyer companies may want.** Suppliers announce their products. Buyers may subscribe to products. *For example*, Chrysler and Honda brands are suppliers (manufacturers) and sell vehicles in three *categories*: car, sport utility, and truck. In each category, a few different models can be manufactured. **For example**, Accord and Civic are two of many *types* of cars manufactured by Honda. Each vehicle uses one of three *types* of fuel: electric, gasoline, or hybrid. Therefore, each vehicle in a *category* has a model *type* and a fuel *type*. Rental companies (referred to as buyers) **such as** Avis, Budget, and Hertz can buy vehicles from vehicle manufacturers (referred to as producers).

Supply-Demand software systems can be used by vehicle manufacturers and rental companies. Manufacturers can *publish* their products and buyers (rental companies) can *subscribe* to them. Notifications of available products from manufacturers are relayed to the buyers. Rental companies can subscribe to different vehicle categories and receive notification events (i.e., rental companies are notified when vehicles become available for purchase). Rental companies can stop receiving notifications by unsubscribing.

Scope

Develop a producer-supplier software system. This system should support three types of events named "`publish`", "`subscribe`", and "`unsubscribe`". Each subscribe event includes a buyer and a product category in which a subscriber is interested. Similarly, an unsubscribe event includes a buyer and a product category. Publish events include the manufacturer names, category model types, and fuel types

When a manufacturer publishes a product belonging to a *product category* and *model types, fuel type*, all buyers that are currently subscribed to that category will receive a notification for that product. A notification would be one or more lines of output (see the **Coding and Testing** section).

The subscribe, unsubscribe, and publish events are to be processed sequentially. A buyer will not receive notifications for a given product unless they subscribe to it. Once a buyer unsubscribes from a category, it will no longer receive any notifications for it unless the buyer re-subscribes to the same category. Multiple subscribers to a product category are notified according to the order of subscriptions; this is required to have a fixed order of output events. The syntax for the publishers and subscribers are:

- `publish, [producer], [product category], [model type], [fuel type]`
- `subscribe, [buyer], [product category]`
- `unsubscribe, [buyer], [product category]`

The name for producers, buyers, product categories, model types, and fuel types may contain any character except comma and leading/trailing spaces. The software should treat upper-case and lower-case names to be the same (e.g., “Sport Utility ” and “Sport utility ” are handled the same).

Your program should not produce any output other than those expected. Therefore, please do not include welcome messages or other extraneous information. Additionally, the program should not produce any error messages during the entire execution. An example is when a buyer is trying to unsubscribe from a category s/he has not subscribed to. Duplicate subscriptions should be ignored. There are no error messages if a buyer subscribes multiple times to a category.

Analysis and Design Artifacts

The design for the software system must follow the *publisher-subscriber design pattern*. This pattern has a **broker** to manage the operations of the **publishers** and **subscribers** while ensuring they do not have any direct relationship (access) to one another. Every publisher is independent of all other publishers just as every subscriber is independent of all other subscribers. This means that both the structure of your system and the behavior of the components of the software system match the publisher-subscriber design pattern.

You need to develop specifications according to UML using the Astah tool prior to implementing them. You should include appropriate numbers of class, use-case, and sequence diagrams and one state machine diagram for the broker.

The producers should implement the provided interface, `IPublisher`, shown in Figure 1. Similarly, the buyers should implement `ISubscriber`, as in Figure 2. You may implement additional methods, but they will not be used in the testing process.

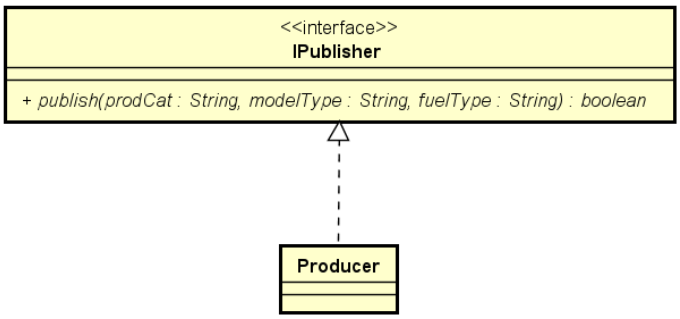

Figure 1: Partial class diagram for the publisher in the Publish-Subscribe design pattern

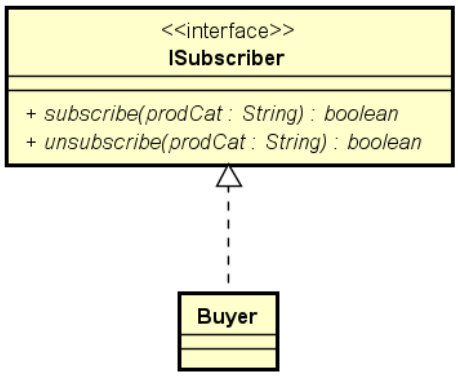

Figure 2: Partial class diagram for the subscriber in the Publish-Subscribe design pattern

Coding and Testing

You must use forward engineering to export your UML design specification to Java code. Then, you must complete the partially generated Java code by adding your own code to the stubs generated by Astah. Do not delete or change any of the Java code produced by Astah. If you need to change any line of automatically generated code, such as for a return statement, keep a commented out copy of the original line in the program. Each section of the code that you add should be delineated using the **//begin** and **//end** comments.

Your program must implement the provided class, **`ProducerBuyer`**, shown below for testing.

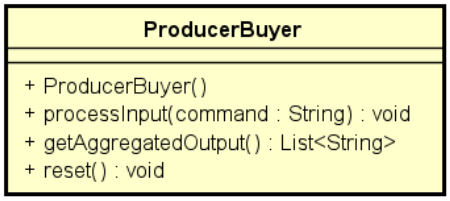

Figure 3: Provided `ProducerBuyer` class skeleton

Your program must implement the provided class, **`ProducerBuyer`**, shown below for testing. Details for this class can be found in the provided Astah file. The following is a general testing procedure using this class for running a set of test scenarios.

1. A `ProducerBuyer` object is instantiated.
2. The `processInput()` operation can be called sequentially multiple times, each time with a single input (see below). The three `publish`, `subscribe`, and `unsubscribe` inputs can be interleaved. All inputs are for the Supply-Demand software system. Note that not every `processInput()` operation may necessarily result in a notification sent to one or more subscribers.
3. The `getAggregatedOutput()` operation returns a list. This output list can have zero or more entries depending on the specific `processInput()` operations that are included in test scenarios. This operation also acts as input for testing the Supply-Demand software system.

The output lines in the returned value of this input must be sequentially ordered according to the order of the `processInput()` operations (see included test scenario). Do not add trailing newline characters (`\n`) to the output.

4. The result from the previous step will be compared against a reference answer.
5. The `reset()` operation will be called to clean up all information in the system. After resetting, the system starts anew. The program should **NOT** carry over information from previous rounds to the next one.
6. Repeat steps 2 to 5. Each round of execution corresponds to one test case.

Sample inputs and outputs

A possible sample test scenario is provided below:

Buyer: Avis; Product category: Car;

Producer: Honda; Category name: Car; Model: Civic, Fuel: Hybrid

Sample input	`processInput("subscribe, Avis, car");`
	`processInput("publish, Honda, car, civic, hybrid");` `getAggregatedOutput();`
Sample output	`Avis notified car: Honda civic, hybrid fuel`

Table 1: A sample subscription from a Avis rental company with an output from Honda manufacturer

Sample inputs and outputs will be released and announced later in Canvas. This is important for developing use-case models and scenarios.

Software and testing environment:

Important! All classes must be in a package named ProducerBuyer (i.e., all *.java files are in the folder ProducerBuyer). The table shows the directory structure (including the JUnit test) to be used. The `Ipublisher` and `Isubscriber` interfaces shown above are required to be used. **Not complying with this requirement will result in failure of compilation in the automated grading system.**

ProducerBuyer			
	src		
		ProducerBuyer	
			utils
			Ipublisher
			Isubscriber
			…
			…
	test		
		ProducerBuyer	
			ProducerBuyerTests

- J2SE must be used — J2EE is not allowed.
- No compiled code or third-party APIs are allowed.
- Source code can be implemented only using the Java programming language.
- Your submission will be tested automatically on GradeScope in an Ubuntu 18.04 virtual machine. Your source code will be compiled and executed using JRE 11.
- After each submission, you will see detailed reports. You may notice some random letters and numbers in place of publishers, subscribers, etc. – this is normal. Test inputs are randomly generated each run.
- You should not need to worry about the actual input, but rather the design with its implementation works with scenarios that have different ~~writers~~producers and ~~readers~~buyers.
- You source code submission must exclude the **`test`** package in the **`ProducerBuyer`** package. This package is provided for your development setup.

Make sure your code compiles and matches the given interfaces.

Submission

- Submit your **source code** in a zip file. It should contain a folder named ProducerBuyer. This folder has the portion of the structure shown in Figure 4 that is shaded with a light green color. Upload this zip file to the project **GradeScope** page (you can also find the link to GradeScope in the project page on Canvas).
- Submit your **project report** to **GradeScope**. A new Canvas page named Project Report Submission was created and it will take you to the right GradeScope page. Note that you need to mark up the sections accordingly.
- Submit your **Astah file** to Canvas. You should name this file *postingid*.asta (e.g. 1234-987.asta). See the rubric below for information on the project report. Submit this Astah file to **Canvas** - Project Astah Submission.

Project Rubric

Parts	Points
Uses publisher-subscriber model	20
Use case diagrams	10
Class diagrams	10
Sequence diagrams	10
State machine diagram(s) for the Broker	10
Detailed description for the Broker state machine diagram(s)	10
Forward engineering (partial code is generated automatically)	10
Code documentation	5
Produces correct outputs	20
Project report quality: this is for the full project report. It focuses on the UML specifications and descriptions, organization, and writing. The report is a single PDF file. A template for this document is provided.	5
Total	110

Note: 10 out of 110 points is extra (bonus).

Important note: be sure to check the Canvas Discussions for updates, questions, answers, and hints. You are responsible for knowing any posted information. If you are not certain about some aspects of this programming assignment, it is very important to ask early, not a few days before the date project is due.