# Staying healthy

Some people use fitness gyms to maintain and improve their health. A gym can have multiple locations in cities. Individuals may choose to go to any gym located in different cities. Assume a person can make one or more reservations to use any of three exercise facilities: cardio, swimming, and yoga. A gym can disseminate information about the availability of its exercise facilities at three locations: Mesa, Phoenix, and Tempe. Exercise facilities are available during specific time slots, such as 7:00 to 9:00 or 14:00 to 16:00 (i.e., time slots are in 24-hour clock format). Assume time slots can be within a 24-hour day.

Software systems can be used to “connect” gyms (providers) and people (consumers). Gyms (e.g.,

Planet Fitness, LA Fitness, 24 Hour Fitness) can informany consumer (such as James, Mary, or Robert) who might be interested in using their facilities. Considering the publisher-subscriber design pattern, gyms can be considered as *publishers* and people as *subscribers*. Individuals may subscribe to any kind of exercise facility and gyms notify the availability of each exercise facility. Individuals can stop receiving notifications by unsubscribing.

## Scope

Develop a publisher-subscriber software system. This system should support three types of events named “publish”, “subscribe”, and “unsubscribe”. Each published event includes a gym (i.e., publisher) such as Planet Fitness, exercise facility such as swimming, facility location such a Tempe, and available time slots such as 11:30 to 12:30. Each subscribed event includes an individual and an exercise facility in which the individual is interested. Similarly, an unsubscribed event includes an individual and an exercise facility. The time slots should be in a 24-hour format .

When a fitness gym publishes its *locations*, *exercise* *facilities* *and timings*, all subscribers who are currently subscribed to exercise facilities will receive notifications. 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 subscriber will not receive notifications for a given exercise facility unless s/he has subscribed to it. Once a subscriber unsubscribes from an exercise facility, s/he will no longer receive notifications unless it is resubscribed. Multiple subscribers to an exercise facility are notified according to the order of their subscriptions; this is required to have a fixed order of output events. The syntax for the publishers and subscribers is:

* publish, [publisher], [exercise facility], [gym location], [gym timings]
* subscribe, [subscriber], [exercise facility ]
* unsubscribe, [subscriber], [exercise facility ]

The names for publisher, subscriber, facility categories, and available locations may contain any character except comma and leading/trailing spaces. The software should treat upper-case and lowercase names to be the same (e.g., “Swimming Pool ” and “swimming Pool” 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 subscriber 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 consumer 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 UML specifications using the Astah tool prior to implementing them. You should include an appropriate number of class, use-case, and sequence diagrams and one state machine diagram for the broker. The design should have documentation.

The providers should implement the IPublisher interface 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.*



*Figure 3: Provided ProviderConsumer class skeleton*

Your program must implement the provided class, **ProviderConsumer**, 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 ProviderConsumer 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:

Subscriber: James; Facility category: Swimming Pool.

Publisher: Planet Fitness; Category name: Swimming Pool; Location: Tempe, Timing: 17:00 to 19:00.

|  |  |
| --- | --- |
| Sample input  | processInput(“subscribe, James, swimming pool”);  |
| processInput(“publish, planet fitness, Swimming Pool, Tempe, 17:00 to 19:00”);  |
|  |    |

*Table 1: A Sample input with subscribe and publish commands.*

The getAggregatedOutput() operation returns a list. The sample notification could be seen below:

|  |  |
| --- | --- |
| Sample output  | James notified swimming pool: planet fitness, tempe, 17:00 to 19:00  |

*Table 2: A sample notification involving James as consumer and planet fitness as provider*

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 ProviderConsumer (i.e., all \*.java files are in the folder ProviderConsumer). 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.**

|  |  |  |  |
| --- | --- | --- | --- |
| ProviderConsumer  |   |   |   |
|   | src  |   |   |
|   |   | ProviderConsumer  |   |
|   |   |   | utils  |
|   |   |   | IPublisher  |
|   |   |   | ISubscriber  |
|   |   |   | …  |
|   |   |   | …  |
|   | test  |   |   |
|   |   | ProviderConsumer  |   |
|   |   |   | ProviderConsumerTests  |

Common Q&A

Q: 1. By publisher-subscriber, it actually means the observer pattern as per the gang of four correct?
2. As long as the given pattern is implemented as well as the interfaces (Meaning they are required) mentioned in the assignment, we have the freedom t o make any other objects or interfaces.
3. In a practical scenario, the publisher/observer would act on their own, however we are using the ProviderConsumer class to "simulate" or automate their interactions via the methods given correct?

Question

4. What is the purpose of the ProviderConsumer method. If its a testing tool, wouldn't it make more sense to make that a static class? I don't think I understand the purpose of insatanciating a ProviderConsumerObject? From what I understand, this would allow a providerConsumer to instantiate a ProviderConsumer object, but isn't that the purpose of the "new" key word?

1. A: By publisher-subscriber, it actually means the observer pattern as per the gang of four correct?

--> The Observer design pattern (see Ch. 2: Concepts and Techniques + Observer Design Pattern) is different than the publisher/subscriber design pattern (see Ch. 4: Publisher-Subscriber Design Pattern).

1. As long as the given pattern is implemented as well as the interfaces (Meaning they are required) mentioned in the assignment, we have the freedom t o make any other objects or interfaces.

--> Right. The interfaces are for the publishers and subscribers. They are required. They do not have the functionality needed for the broker.

1. In a practical scenario, the publisher/observer would act on their own, however we are using the ProviderConsumer class to "simulate" or automate their interactions via the methods given correct?

--> see the response to your question below.

Question

1. What is the purpose of the ProviderConsumer method. If its a testing tool, wouldn't it make more sense to make that a static class? I don't think I understand the purpose of insatanciating a ProviderConsumerObject? From what I understand, this would allow a providerConsumer to instantiate a ProviderConsumer object, but isn't that the purpose of the "new" key word?

--> The ProviderConsumer is not a static class (see the Astah file for this class). This class acts as a test frame (see the above response).

…………………………………………………….

Q: The subscribe, unsubscribe, and publish events are to be processed sequentially.
This entails that the consumer will be able to subscribe to events before they are even created, does that mean that the consumer is the one that essentially makes the broker build the provider and facility. And the provider only then supplies with the city and time?

A: Yes, that is indeed correct. When a consumer subscribes to a facility, they only receive notifications if the provider or publisher publishes that facility. One can see the sample input processing commands with their respective outputs that I have pasted above.