CS2030 Practice Assessment #2 (ungraded)

Tags & Categories

Tags:

Categories: CodeCrunch

Related Tutorials

Task Content

CS2030 Practice Assessment #2

To fight against the spread of COVID-19, the local government has implemented TraceTogether tokens for contact tracing. Each token constantly pings other tokens nearby and keeps a record within the device. As data privacy is a major concern, only the required data will be uploaded in the event of a contact tracing exercise.

Task

In this task, we shall implement a simplified model of the contact tracing process. A token tracks another token by pinging it, and then store its identifiers and the time of ping. All tokens are registered to a central registry. This registry contacts all tokens to upload specific data. Data is then uploaded to a data store for later retrieval by an external party (say, the health ministry). Hence, no data is stored in the registry.

Take note of the following

- You are **NOT** allowed to use any java libraries, other than those from **`java.lang`**, **`java.util.List`** and **`java.util.Comparator`**.
- Ensure that there are **NO** cyclic dependencies in your implementation.
- Write each **`class`**/**`interface`**/**`enum`** in its own file.
- Ensure that **ALL** object properties and class constants are declared **`private final`**.
- You are **NOT** allowed to use Java reflection, i.e. **`getClasses`**, **`getName`**, etc.
- You are **NOT** allowed to use **`instanceof`**, other than within the same class (e.g. **`obj instanceof A`** within class **`A`**).
- You may assume that all tests provide valid arguments to methods; hence there is no need to validate method arguments.
- You may assume that each token will only ping each separate token once.

Level 1

Create a **`Token`** class to instantiate tokens with a unique integer identifier. Include a **`toString`** method that outputs the token identifier followed by **`none`** to signify no other tokens were pinged.

```
jshell> Token t1 = new Token(1)
jshell> Token t2 = new Token(2)
jshell> new Token(3)
$.. ==> Token #3: none
jshell> t1
t1 ==> Token #1: none
jshell> t2
t2 ==> Token #2: none
jshell> /exit
```

Level 2

A token can ping another token via the **`ping`** method. When this method is called, the identifier of the target token is noted down, together with a positive (`> 0`) integer time value. The output will now include the pinged tokens (identifier followed by time) in ascending order of the identifier.

Note that a ping from token A to token B will only make a record of B in A, but not the other way round. Moreover, you may assume that a token will not ping itself.

```
jshell> Token t1 = new Token(1)
jshell> Token t2 = new Token(2)
jshell> t1.ping(t2, 10)
$.. ==> Token #1: #2@10
jshell> t1
t1 ==> Token #1: none
jshell> t2
t2 ==> Token #2: none
jshell> t1 = t1.ping(t2, 10).ping(new Token(3), 5)
jshell> t1
t1 ==> Token #1: #2@10 #3@5
jshell> t2
t2 ==> Token #2: none
jshell> new Token(1).ping(new Token(3), 5).ping(new Token(2), 10)
$.. ==> Token #1: #2@10 #3@5
jshell> /exit
```

Level 3

A token can be registered to a registry. Additionally, a registry has write access to a data store via the **`DataStore::write`** method. The **`DataStore`** class is provided for you and is used to simulate writing information into the data store by invoking a **`println`** output statement instead. **You are NOT allowed to modify the `DataStore` class**.

```
class DataStore {
    void write(String info) {
        System.out.println("DataStore: " + info);
    }
}
```

You may assume that invoking **`new DataStore()`** will establish a connection to the same data store. With a **`DataStore`** object created, the registry can write to the data store via the **`void store(String message)`** method.

Once a token is registered, a **`void contact()`** method can be invoked within the token to test the contact with the registry. The test method will write a string **`"Test contact"`** to the data store.

While adhering to good abstraction principles, create a **`RegisteredToken`** class and implement the **`contact`** method. At this juncture, we will also need to create a **`RegistryStub`** object in place of a **`Registry`** object in order to test the functionality of our classes. In particular, **`RegistryStub`** has two constructors: one that takes in a string identifier, and the other with no arguments (i.e. the identifier is an empty string).

```
jshell> DataStore db = new DataStore()
jshell> db.write("Hello")
DataStore: Hello
jshell> Token t1 = new Token(1)
jshell> Token t2 = new Token(2)
jshell> t1 = t1.ping(t2, 10)
jshell> t1
t1 ==> Token #1: #2@10
jshell> new RegistryStub()
$.. ==> []
jshell> Registry stub = new RegistryStub("Test Stub")
jshell> stub
stub ==> [Test Stub]
jshell> stub.store("Hello world")
DataStore: Hello world
jshell> new RegisteredToken(t1, stub)
$.. ==> Token #1: #2@10
jshell> new RegisteredToken(t1, stub).contact()
DataStore: Test contact
jshell> Token t = new RegisteredToken(t1, stub)
jshell> t
t ==> Token #1: #2@10
jshell> /exit
```

Level 4

We are now ready to write the actual registry implementation class **`RegistryImpl`**. This object is created whenever we need to perform contact tracing; that is to say, the tokens would already have pinged one another. A list of (unregistered) tokens is passed to the **`RegistryImpl`** constructor. Then, each token needs to be registered to the registry (see level 3 above).

For the purpose of testing, the output of a registry will include the identifier of the registry, the number of tokens registered, and the corresponding output of each token. The tokens are output in the same order as the list presented to the constructor. If you are concerned about possible data leaks, we can always make the **`toString`** methods less verbose later on.

```
jshell> Token t1 = new Token(1)
jshell> Token t2 = new Token(2)
jshell> t1 = t1.ping(t2, 10)
jshell> Registry r1 = new RegistryImpl("r1")
jshell> r1
r1 ==> [r1]: 0 tokens registered
jshell> Registry r2 = new RegistryImpl("r2", List.<Token>of(t1, t2))
jshell> r2
r2 ==> [r2]: 2 tokens registered
Token #1: #2@10
Token #2: none
jshell> Registry r3 = new RegistryImpl("r3", List.<Token>of())
jshell> r3
r3 ==> [r3]: 0 tokens registered
jshell> /exit
```

Level 5

Once the tokens are registered into the registry, a **`void alert(int time)`** method that takes a positive (`> 0`) integer **`time`** as the argument can be called to alert all tokens to write their data to the data store **only if the ping-time recorded and the specified time coincide exactly**.

Ensure that only the **`Registry`** class has a dependency on the **`DataStore`** class.

Hint: Keep in mind the Tell-Don't-Ask principle. Specifically, the role of **`RegistryImpl`** is just to initiate the alert, and tell each token to proceed with writing data to the data store. Once the token is alerted, it initiates the process to write data to the data store.

```
jshell> Token t1 = new Token(1)
jshell> Token t2 = new Token(2)
jshell> t1 = t1.ping(t2, 10).ping(new Token(3), 5)
jshell> t1
t1 ==> Token #1: #2@10 #3@5
jshell> t2
t2 ==> Token #2: none
jshell> new RegistryImpl("r1", List.of(t1, t2)).alert(5)
DataStore: Token #1: #3@5
DataStore: Token #2: none
jshell> new RegistryImpl("r2", List.of(t1, t2)).alert(8)
DataStore: Token #1: none
DataStore: Token #2: none
jshell> t2 = t2.ping(t1, 5)
jshell> new RegistryImpl("r3", List.of(t1, t2)).alert(5)
DataStore: Token #1: #3@5
DataStore: Token #2: #1@5
jshell> /exit
```

Level 6

We are now ready to include more devices into the registry. A safe-entry device with a positive (`> 0`) identifier can be registered into the registry. Unlike a token, a safe-entry device has a direct connection with the data store.

Include **`add`** methods into the registry so it can have either another safe entry device or an unregistered token added.

Ensure that only **`Registry`** and **`SafeEntry`** classes have dependencies on the **`DataStore`** class.

For simplicity, you may assume there is only one single positive time recorded on a safe-entry device.

```
jshell> Token t1 = new Token(1)
jshell> Token t2 = new Token(2)
jshell> t1 = t1.ping(t2, 10).ping(new Token(3), 5)
jshell> t1
t1 ==> Token #1: #2@10 #3@5
```

```
jshell> t2
t2 ==> Token #2: none
jshell> new RegistryImpl("r1", List.of(t1, t2)).alert(5)
DataStore: Token #1: #3@5
DataStore: Token #2: none
jshell> new RegistryImpl("r2", List.of(t1, t2)).alert(8)
DataStore: Token #1: none
DataStore: Token #2: none
jshell> t2 = t2.ping(t1, 5)
jshell> Registry r = new RegistryImpl("r3", List.of(t1, t2))
jshell> r.alert(5)
DataStore: Token #1: #3@5
DataStore: Token #2: #1@5
jshell> new SafeEntry(10, 5)
$.. ==> SafeEntry #10@5
jshell> Registry s = r.add(new SafeEntry(10, 5)).add(new Token(4).ping(t2, 5))
jshell> s
s ==> [r3]: 4 tokens registered
Token #1: #2@10 #3@5
Token #2: #1@5
SafeEntry #10@5
Token #4: #2@5
jshell> s.alert(5)
DataStore: Token #1: #3@5
DataStore: Token #2: #1@5
DataStore: SafeEntry #10@5
DataStore: Token #4: #2@5
jshell> s.alert(8)
DataStore: Token #1: none
DataStore: Token #2: none
DataStore: SafeEntry #10: none
DataStore: Token #4: none
jshell> r
r ==> [r3]: 2 tokens registered
Token #1: #2@10 #3@5
Token #2: #1@5
jshell> /exit
```