COMPSCI 280: Introduction to Software Development

Assignment 1 (Iteration I): Unit Testing *(Version 2)*

Deadline: 11:59pm, 9th August 2019

Introduction

The previous developer for DALSys had started working on the system and has written some basic code. However, this developer has since left the project and you are responsible for taking over their code and continuing development.

To get you started on the project, your team lead has asked you to look at the existing code and ensure everything is sufficiently unit tested. This will help improve the quality of the code and give you an opportunity to familiarise yourself with the systems organization.

Resources

The initial code for this assignment is available on Canvas. The ZIP file contains the following Python code files:

drones.py: initial implementation of the classes relating to drones, including `Drone` and `DroneStore`.
operators.py: initial implementation of the classes relating to operators, including `Operator` and `OperatorStore`.
test_allocation.py: unit tests for the implementation of the `allocate()` method.

Product Backlog

There are three items in the product backlog for this iteration. Items are having different grade marks (4, 7 and 3 respectively).

Note:

- In addition to failing tests cases, there could be mark deductions on making unnecessary changes to the provided code.
- Maintain all your code that you wrote in this iteration as you will be extending the same for the next two iterations (Assignment 2 and Assignment 3).

Submission

Your work must be submitted in a single ZIP file to the ADB (https://adb.auckland.ac.nz) submission system. Submit all the Python files for your work, including the files containing the original code.

Item #1

Title:
As the system owner, I want unit tests for adding and removing drones, so I can have confidence in its completeness.

Description:
The DroneStore class is responsible for storing details on each available drone. Internally, the store contains a list of all the drones.

When a drone is added, the store will validate that the drone does not already belong to the store and add it. If it is already in the store an error will be raised. When a drone is removed, the store will validate the drone exists in the store and removes it. If it doesnt exist in the store, an error will be raised.

The previous developer has written the add and remove methods. You will need to add unit tests to validate they are working correctly.

Done Criteria:
Write four unit tests for DroneStore:

1. Unit test for add() that ensures a drone is successfully added
2. Unit test for add() that checks the method raises an exception if the drone already exists in the store
3. Unit test for remove() that removes an existing drone from the store
4. Unit test for remove() that checks the method raises an exception if the drone does not exist in the store

Item #2

Title:
As a manager, I want the operator data to be valid, so I can know the operators are correct.

Description:
The OperatorStore class is responsible for storing details about drone operators. Like DroneStore, it contains an internal list, one of potential operators. However, unlike DroneStore, OperatorStore needs to validate the data when it is added.

Adding an operator is a two-stage process. For the first stage, OperatorStore will return a pending operation object. This object contains the operator to add and a list of any validation errors. The second stage is to commit the pending operation to OperatorStore. This allows the user to check and correct the errors in operator entry, and subsequently override the validation, as needed.

The following is an example of how to use the two stage process:

```
op = Operator()
# Set the operator properties
act = opStore.add(op)
if act.is_valid():
   act.commit()
else:
   # Handle the validation errors
   # All validation errors are stored in act.messages
```

For an operator to be valid, the operator must pass the following checks:

1. The operator must have a first name.
2. The operator must have a date of birth.
3. The operator must have a drone license.
4. To hold a drone class 2 license, the operator must be at least 20 years old.
5. To hold a rescue drone endorsement, the operator must have been involved in five prior rescue operations.

The previous developer has written most of the code for this process, including the two stages. You will need to add the unit tests to validate the rules, and fix any validation errors in the code.

Done Criteria:
There are at least seven unit tests:
1. The data for the operator passes all the validation rules.
2. Test checking when each validation rule fails (five tests.)
3. Check that the operator is added to the store.

Ensure the code checks all the validation rules.

Item #3

Title:
As a manager, I want to allocate drones to operators, so each operator knows which drone they should control.

Description:
The DroneStore class is also responsible for allocating drones to operators. This operation should check whether the operator is allowed to operate the drone, and then assign the drone to the operator (if allowed.) The operation should also ensure a one to one relationship between drones and operators.

This operation is also a two-stage process. For the first stage, DroneStore will return a pending operation object. This object contains the operator and drone, plus a list of any validation errors. The second stage is to commit the pending operation to DroneStore.

For an allocation to be valid, it must meet the following rules:

1. An operator can only operate one drone.
2. The operator must hold the correct license.
3. For a rescue drone, the operator must hold a rescue drone endorsement.

The previous developer has written the unit tests and started on the two stages. You will need to add the unit tests to validate the rules, and fix any validation errors in the code.

Done Criteria:
Add code so all the unit tests pass.