20169321

Please do not share this file. It has been fingerprinted with your ID (20169321).

20169321 20169321 20169321

Assignment 2

Object Oriented Software Engineering (COMP2003/6005)
Semester 1, 2021

Due: Thursday 3 June, 23:59
Weight: 45% of the unit mark

1 Your Task

Design and implement the system described below under Problem Description. Specifically:

- Use either Java, C#, C++ or Python to implement the system described.
  - Your code should be of high quality, with consistent formatting and meaningful commenting.
  - If using Java, it is highly recommended that you use Gradle or Ant.
  - If using Python, use type hinting for all method/function parameters and return types.
  - If using C++, be sure you've understood the Base64 caveat.
- Make good use of polymorphism. Specifically, your design should include *two distinct inheritance hierarchies* such that:
  - They use polymorphism to help create loose coupling (and/or extensibility).
  - They address the actual requirements of the system (i.e., they're part of the solution to the problem description, not just added on).
  - They are substantially different from one another (not just repetitions of a single idea applied in slightly different places).

  You are encouraged to consider the use of the Observer Pattern and State Pattern for these purposes, but you are not limited to these choices (and there are many possible solutions even using only these patterns).
- Further, with respect to your design:
  - Ensure sensible division of responsibilities, including a sensible use of packages or namespaces. (MVC is not required, and may not be a good choice for this problem.)
  - Implement dependency injection, and *avoid* using Singletons. (You may or may not decide that a factory is necessary; this is up to you.)
  - Perform appropriate error handling.

The following requirements are the same as for Assignment 1:

- Provide a UML class diagram giving a structural overview of your design.
  - Your UML must accurately represent your design.

20169321

    - Use a proper tool; e.g., Umlet, draw.io, PlantUML or others. Avoid tools that have no explicit support for UML.
    - Make the diagram layout as neat and logical as practical. Minimise crossing lines.
    - If in doubt as to whether or not to represent certain things on the diagram, err on the side of more detail. But it is generally unnecessary to show constructors and (in most cases) usage dependencies.
- Provide a `README.txt` (or `README.md`) that briefly explains how to compile and run your code, if there are any facets of this that the marker needs to be aware of.
- Provide a response to each of the marking criteria. Record this in a file called `criteria.txt` (or `criteria.md`).

20169321

2 Submission

Submit all of the above, plus a signed declaration of originality, to the appropriate area on Blackboard. Include everything in a single .zip/.tar.gz file (*please avoid* .rar, .zipx or .7z). You do not need to include any compiled code.

You must verify that your submission is correct and not corrupted. Once you have submitted, please download your own submission and thoroughly check that it is intact. You may make multiple submissions. Only your last one will be marked.

20169321

3 Marking Criteria

The assignment will be marked out of 30. There are six core marking criteria and an additional bonus, each worth 5 marks:

1. Clear and distinct package/class/interface/method responsibilities.
2. Appropriate error handling.
3. Implementation of dependency injection.
4. Use of polymorphism #1
5. Use of polymorphism #2
6. Clear and correct UML
7. Bonus: meaningful use of generics (in the sense of creating your own class/interface with generic type parameter(s), such that it contributes to code reuse).

20169321

3.1 Does the program run correctly?

This isn't strictly one of the criteria, but rather a cross-cutting concern that affects all of them.

The marker will attempt to compile and run your code. If it does not work correctly, this will have implications for how we assess the other criteria.

3.2 Your Responses to the Criteria

You must provide a response to each of the six critera, as part of your submission. That is, for each individual criterion, write a paragraph or two justifying the choices you have made. Include all your

20169321

20169321

discussion in `criteria.txt`.

This fulfils two purposes:

1. We want to know that you can articulate, in plain English, what you are doing. This is part of the task.
2. If you do something unusual/unorthodox, this is your opportunity to justify it.

One way to think about what you're doing here is this: imagine that the marker is slightly sceptical about whether to award you the marks for a given criterion. What would you say to convince them?

Make your responses concise and to-the-point. Address each criterion separately.

20169201693321

4 Problem Description

Your task is to design and implement a software system for controlling a Mars rover; that is, a robotic vehicle for exploring Mars.

4.1 Commands from Earth

The rover must take action in accordance with commands sent over radio from the Space Agency's top scientists on Earth. Each command will be sent as a single string, as follows:

- "`D 23.1`" – a command to drive forward 23.1 metres (for instance). Any positive distance is allowed.
- "`T 75.5`" – a command to turn 75.5 degrees anticlockwise. Any angle from -180 to 180 is allowed.
- "`P`" – a command to take a photo.
- "`E`" – a command to report environmental status.
- "`S`" – a command to perform soil chemistry analysis.

Some further constraints and clarifications:

- If any other kind of string is provided, it is an invalid command.
- The rover cannot start soil analysis while driving or already performing soil analysis. An "`S`" command is invalid under these conditions.
- The rover cannot turn or start driving while performing soil analysis. The "`D`" and "`T`" commands are invalid under this condition.
- A subsequent "`D`" command can override a previous one. If the rover receives a "`D`" command while already driving, it must not stop, but rather drive the newly indicated distance from its current location (disregarding whatever distance it had been given previously).
- A "`T`" command is valid while either stopped or driving. "`P`" and "`E`" are valid regardless of the situation.
- For "`T`", "`P`" and "`E`", those actions can be completed immediately, so there is no question of what happens "during" any of them.

4.2 Return Messages

20169321

69321

The rover must also send various information back to Earth, via a similar set of messages:

20169321

- "`! ...`" – a return message indicating that the rover received an invalid command, where "`...`" is text of the invalid command. If a command is invalid *for any reason*, it must trigger such a return message.
- "`D`" – a return message indicating that the rover has reached its destination and stopped driving.
- "`P ...`" – a return message containing a photograph, where "`...`" is the image data encoded in Base64. (The full message is likely to be quite long.)
- "`E -15.8 7.1 13774`" – a return message reporting a temperature of -15.8 °C, a visibility of 7.1 km, and an ambient light level of 13,774 lux.
- "`S ...`" – a return message containing completed soil chemistry analysis, where "`...`" is the analysis data encoded in Base64.

These return messages have different timing/triggering requirements:

- The "`!`", "`P`" and "`E`" messages can be sent almost right away after receiving the corresponding command (allowing for a trivial amount of time needed to take a photo or measure environmental conditions). The software doesn't need to wait for a second event to occur.
- An "`E`" return message must *also* be sent, even if no "`E`" command was received, *if and when* the visibility falls below 4 km, or rises above 5 km (i.e., whenever it crosses those thresholds). Such automatic messages must still specify all three values as above.
- A "`D`" return message must only be sent once the rover *stops* driving, which will be some time after the "`D`" command was received. Other events/commands (except for soil analysis) may happen in the mean time.
- Similarly, an "`S`" return message must only be sent once soil analysis is complete, which will be some time after the "`S`" command was received. "`P`" and "`E`" commands/events may happen in the mean time.

20169321

4.3 Rover API

Several predefined classes will allow your code to interact with the rover's hardware systems.

Important: You must create *one*, and *only one*, object of each class (`EarthComm`, `Sensors`, `EngineSystem` and `SoilAnalyser`). The methods are all non-static, but the rover's hardware cannot be controlled from multiple instances of these classes at the same time. (This doesn't mean you have to use the Singleton pattern, and in fact it's better if you don't. You just need to make one object of each class.)

The declarations are shown in Java, but translate easily to the other languages. You can modify the method names to conform to local naming conventions. The Python 3 equivalent to `byte[]` is a `bytes` object.

(See further notes underneath the declarations.)

API Declarations (Java)

```
public class EarthComm
{
    /**
```

20169321

```
     * Return the next command received from Earth, or null if no further command
     * has been received. If multiple commands arrive in between calls to
     * pollCommand(), they will be buffered (stored temporarily), and subsequent
     * calls to pollCommand() will return one command at a time, in the order of
     * arrival.
     *
     * There is no risk of commands being lost. But pollCommand() *will not wait*
     * for a command to be received if none has been yet.
     */
    public String pollCommand() {...}

    /** Sends a return message to Earth. */
    public void sendMessage(String msg) {...}
}

public class Sensors
{
    /** Performs a temperature reading and returns the result in °C. */
    public double readTemperature() {...}

    /** Performs a visibility reading and returns the result in km. */
    public double readVisibility() {...}

    /** Performs a light-level reading, and returns the result in lux (units). */
    public double readLightLevel() {...}

    /** Takes a photo and returns the binary data making up the image. */
    public byte[] takePhoto() {...}
}

public class EngineSystem
{
    /**
     * Begins driving forward. The effect is *not* to drive a fixed distance, but
     * to simply start driving. The rover will not stop until the stopDriving()
     * method is subsequently called.
     *
     * If startDriving() is called while the rover is already driving, it will
     * throw an exception.
     */
    public void startDriving() {...}

    /**
     * Stops driving.
     *
     * If stopDriving() is called while the rover is already stopped, it will
     * throw an exception.
     */
    public void stopDriving() {...}

    /**
     * Immediately turns the rover by the specified angle anticlockwise (negative
     * for clockwise).
     */
    public void turn(double angle) {...}

    /**
     * Returns the total distance that the rover has ever driven, since it first
     * landed on Mars. This figure is never reset. It remains constant while the
     * rover is stopped, and increases while the rover is driving.
```

20169321
20169321

```
     */
    public double getDistanceDriven() {...}
}

public class SoilAnalyser
{
    /**
     * Begins a soil analysis. The soil analysis will complete some time later,
     * and its result can be retrieved by calling pollAnalysis().
     *
     * If startAnalysis() is called while analysis is already underway, it will
     * throw an exception.
     */
    public void startAnalysis() {...}

    /**
     * Retrieves the results of a soil analysis, if they're ready yet. If no new
     * results have been produced, this method returns null.
     */
    public byte[] pollAnalysis() {...}
}
```

These classes are *not* considered 'your work'. Assume that another software development team is working on them. Therefore:

- Don't make any assumptions about what they do, other than what can be seen in the declarations and comments.
- *Do* make stubs of these classes for the purposes of testing your code. (A "stub" is a highly simplified, placeholder version of a class/method used to test whether *another* class or method works.)
- Understand that, if you implement any of these classes, those implementations will (in principle) be deleted and replaced by the other team's work. So don't put any of your actual design into them.

20169321

4.4 Hint: Architecture

You are likely to find that the MVC architecture simply does not apply to this situation. Nonetheless, you should find a sensible way to break up the system into packages/namespaces.

Why no MVC? First, because there's no user interface. And second, because there's virtually no data to model. The system is driven by events and state.

To emphasise one key point: your software is *only* running on the rover itself. You are not writing any of the code that will run on Earth. It's for this reason that your software does not need a UI, and cannot usefully have one. If there was someone on Mars to use the UI, then why did we send the rover at all?

> **Warning**
>
> If you do create an interactive user interface for the rover, you will be forcibly sent on a one-way trip to Mars to use it.

20169321

The requirements will lead inevitably to one key architectural choice: the use of an infinite loop like this:

```
while(true)
{
    poll everything
    perform actions if needed
    sleep for a few milliseconds
}
```

20169321

We call this an "event loop" (or a very basic version of an event loop, anyway); something similar typically appears in games and simulations. In our case, there is no need to exit from this loop, because there are no conditions under which the Rover's software should terminate (though for testing purposes you can type Ctrl-C at the terminal).

"Polling" just means actively checking whether events have occurred. For instance, how do you know when the rover needs to stop driving? You keep checking (polling) the total distance it has travelled, and subtract the distance it had already travelled when it started driving.

20169321

4.5 Hint: Sleeping

Sleeping (i.e., doing nothing for a small period) is necessary so that we don't waste energy or overheat our device by needlessly polling hundreds of millions of times per second. (Try *not* sleeping inside this loop, and you'll hear your machine's CPU fan go into overdrive to dissipate the heat from all the useless effort the CPU has to put in!)

So, here's some relevant API references:

- Java: `Thread.sleep(millisecs)`. (You will have to catch `InterruptedException` and treat it as an internal error. In multithreaded code, this exception has a specific purpose, but it shouldn't happen in the ordinary single-threaded code that you'll be writing.)
- Python: `time.sleep(secs)` (where `secs` is a `float`).
- C#: `Thread.Sleep(millisecs)`.
- C++: `std::this_thread::sleep_for(duration)`.

20169321

4.6 Hint: Base64

Base64 is a scheme for representing binary data using the characters `a-z`, `A-Z`, `0-9`, `+`, `/` and `=`. Here's some relevant API references:

- Java: `Base64.Encoder.encodeToString(byte[])`.
- Python: `base64.b64encode(bytes)`.
- C#: `System.Convert.ToBase64String(byte[])`.
- C++: sorry, you're on your own! You are welcome to find and use a third-party Base64 encoder, provided you reference your source.

20169321

5 Academic Integrity

Please see the *Coding and Academic Integrity Guidelines* on Blackboard.

In summary, this is an assessable task. If you use someone else's work or assistance to help complete part of the assignment, where it's intended that you complete it yourself, you will have compromised the assessment. You will not receive marks for any parts of your submission that are not your own original work. Further, if you do not *reference* any external sources that you use, you are committing plagiarism and/or collusion, and penalties for academic misconduct may apply.

20169321

Curtin also provides general advice on academic integrity at academicintegrity.curtin.edu.au.

20169321

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an academic misconduct inquiry.

20169321