# HW-6 advanced unit test mock

## Write the mock for a class to assert your class correctness

**說明:**

有時候，你的class under test 的正確行為並不是展現在 method 的 return values。這種例子並不少。課堂上老師有解說，如果你不懂 interaction testing 跟 state testing 的差別，請再回去看懂投影片，再來進行這個作業 。

開啟專案會發現無法執行，少了一個Interface ，請先把它修好。然後撰寫 Junit tests。但是當你真的在進行 Junit tests 的時候，你應該會發現你卡住了。你無法用傳統的方法驗證 method 的回傳值來判斷正確性。在這個lab 請你手動寫出 mock 來幫助你這件事情。

**你的任務:**

1. 先嘗試撰寫你的單元測試，但是你應該會發現靠method 的回傳值 assert 的老招不再成立
2. Refactor 你的class under test (註: NO idea? 上課恍神? 請參考老師的投影片，或儘快請教同學了解背後的原理) 準備讓mock 物件能夠開始運作
3. 撰寫 mock 來驗證你的測試
4. 你的成績將由助教根據
   1. 是否撰寫了夠完整，夠多的 unit tests for class under test
   2. unit tests 中是否正確地使用了 mock。

**提示:**

1. 為了正確控制秒數，理論上你應該用執行續來完成某些指令。但是為了簡化這個作業，你如果不用執行續來進行，並不會扣分。你可以先蒐集分析所有指令的觸發時間點，然後排序一下，然後再巧妙地算出等待時間間隔，最後用一個迴圈執行所有指令，迴圈每一次 sleep(1) 睡覺一秒就可以了，時間點到了，就執行該被執行的指令。執行續的使用與否並不是這lab的主要目的。(暗示已經夠多了，你們再看不懂我也沒辦法)

|  |
| --- |
| **The specs of your class to be implement** |
| 你負責為美國太空總署的火箭實驗室撰寫一個class。這個 class 的功能是當接到命令的時候，它會去控制火箭燃料噴嘴，在飛行中就可以控制方向。你要控制的火箭總共有4個噴嘴(nozzle)。噴嘴的系統 (rocket system) 不是你寫的，你只是負責呼叫它。  RocketSystem rocket = new RocketSystem() ;  rocket.ignite(0, 1000);  // 點燃0號噴嘴，以每秒1000磅的壓力進行噴發  rocket.ignite(1,500);  // 點燃1號噴嘴，以每秒 500 磅的壓力進行噴發  ……… // burn for some time  rocket.shutoff(0); // 關掉0 號噴嘴  rocket.shutoff(1); // 關掉1號噴嘴  你的任務是撰寫一個底層的 class，更高層的模組會經過精密的科學計算傳來一條控制點火字串。一個點火字串的範例如下:  IGNITE 0 1000 AT 0  IGNITE 1 500 AT 2  SHUTOFF 0 AT 5  SHUTOFF 1 AT 6  字串的格式非常簡單，字串內的每一行以換行符號 (\n)作為間隔。  第一行指令請在第0秒的時候點火0號噴嘴以每秒1000磅的壓力進行噴發。第二指令則請你的在第2秒的時候點火1號噴嘴，以每秒500磅的壓力進行噴發。第三行指令則請在第5秒關掉0號噴嘴。第四行指令則請在第6秒的時候關掉1號噴嘴。每一次收到一個點火字串，就是一個新的第0秒。  The interfaces of your class under test is  class YourRocketController {  public void fireUp(string control\_string)  throw ERROR\_STRINGFORMAT ;  }  請注意，當你處理這一條指令字串時都是一個重新的第0秒，你無須管理絕對的時間。你的 fireUp method 在執行完一個指令字串之後，有可能火箭的噴嘴繼續處於噴發的狀態。舉例來說，你的指令中有一個最晚開始的指令，例如在第25秒點火一個噴嘴。你的method 應該等到第25秒，執行噴嘴指令之後，才結束你的 method，然後 return。 如果你的 method 又被呼叫了，請你執行新的指令，不用管噴嘴的先前狀態。 |