軟體測試期中考_2

連結

https://hackmd.io/@ruserxd/B1P_Fmj-1x

介紹 Introduce

主要修改 github 上的程式碼並自訂義資料的型態 + 測試程式碼的可信度

程式設計 Program

1. 新增的資料型別

這邊用 lombok 省去 set, get的撰寫
將每個州的投票資訊設計為一個 class

```
@lombok.Data
public class ticketData {
    String state;
    String ElectoralVotes;
    String candidate1Vote;
    String candidate2Vote;
}
```

2. 針對不滿足總選舉人票數進行拋出例外

因為規格上有寫說總選舉人票數必須滿足 538 的條件
因此設計一個常量，日後修改也較為方便

```
public final static int TOTALVOTES = 538;

if (c1_electoral_votes + c2_electoral_votes != TOTALVOTES)
    throw new IllegalAccessException("c1_electoral_votes + c2_electoral_votes
```

3. 針對 Nebraska 與 Maine 的比例分配

實際上是按照各候選人於各個國會議員選區的得票來分配
但根據規格書上寫的比例原則，因此特別針對這兩州進行票數的比例分配

- 程式邏輯
  我們透過 獲得票數 g /總票數 t 得出一個 比例 p

接著透過 p * t 得到我們這次獲得的票數 (currentGet)
但這時會出現小數的問題
先省略不計
接著將兩個 currentGet 加起來，若不等於總票數 t
我們就依據比例的大小，較大的 + 1

```
if (Objects.equals(state, "Nebraska") || Objects.equals(state, "Maine")) {
    double totalPopularVotes = vote_arr[1] + vote_arr[2];
    double probability1 = (double) vote_arr[1] /totalPopularVotes,
            probability2 = (double) vote_arr[2] /totalPopularVotes;

    int currentTakeVote1 = (int)Math.floor(vote_arr[0]*probability1),
            currentTakeVote2 = (int)Math.floor(vote_arr[0]*probability2);
    if (currentTakeVote1 + currentTakeVote2 != vote_arr[0]) {
        if (probability1 > probability2) {
            currentTakeVote1 += 1;
        } else {
            currentTakeVote2 += 1;
        }
    }
    log.info("比例原則 {} 總票數 {}", state, vote_arr[0]);
    log.info("c1 機率 {} 獲得票數 {} ,c2 機率 {} 獲得票數 {}"
            , probability1, currentTakeVote1, probability2, currentTakeVote2);
    c1_electoral_votes += currentTakeVote1;
    c2_electoral_votes += currentTakeVote2;
}
```

4. 針對同票的情況進行拋出例外

因為除了兩州透過比例去做計算選舉人票數，其餘是勝者全拿
因此必須針對同票的情況拋出例外

```
if (vote_arr[1] > vote_arr[2]) // 選舉人票
    c1_electoral_votes += vote_arr[0];
else if (vote_arr[2] > vote_arr[1])
    c2_electoral_votes += vote_arr[0];
else
    throw new IllegalAccessException("不該出現同票的情況，請重新驗票");
```

5. log

這邊特別提到 log 的原因，因為對於後來檢查幫助很大
人口票, 選舉人票 我時常會搞混

```
log.info("c1 get {} {}", c1_electoral_votes, c1_popular_votes);
log.info("c2 get {} {}", c2_electoral_votes, c2_popular_votes);
```

透過這樣的 log 日誌，就比較好去檢查，這些變數的狀態是不是我所期望的

考試後透過 log 的重大發現

log 發現票數落差很大，原來是 index 的地方改錯了，難怪票數差很多

```
for (Map.Entry<String, int[]> voteEntry : state_votes.entrySet()) {
    ...省略大量程式碼
    if (vote_arr[1] > vote_arr[2]) // 選舉人票
        c1_electoral_votes += vote_arr[0];
    else if (vote_arr[2] > vote_arr[1])
        c2_electoral_votes += vote_arr[0];
}
```

測試 Test

模組化 getData

避免過多的重複程式碼

```
private void getData(String path, List<ticketData> data) {
    try (InputStream is = USAElectionTest.class.getResourceAsStream(path)) {
        assert is != null;
        try (BufferedReader br = new BufferedReader(new InputStreamReader(is)
            String line;

            while ((line = br.readLine()) != null) {
                // 移除引號並分割每行的欄位
                String[] values = line.split(",");

                // 將每行的資料轉為 Object[]
                ticketData row = new ticketData();
                row.setState(values[0]); // 州名 (String)
                row.setElectoralVotes(values[1]); // 選舉人票數
                row.setCandidate1Vote(values[2]); // 一號候選人 得票數
                row.setCandidate2Vote(values[3]); // 二號候選人 得票數
                log.info("測試獲得: {}", String.valueOf(row));
                data.add(row);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

以下簡略描述測試的範圍

1. 原始基本的測試資料 (Nebraska && Maine 進行比例分配)
   原先就有的測試資料，對其進行測試
   修改 306 -> 307，因為按照比例原則非各個國會議員選區的得票來分配
2. 非 Nebraska 與 Maine 同票的測試資料
   因為除了上述兩州，其餘為勝者全拿，因此必須避免掉同票數的出現

測試是否有正確的拋出例外以及訊息是否符合

```
String expected = "不該出現同票的情況，請重新驗票";
IllegalAccessException illegalAccessError = assertThrows(IllegalAccessExc
assertEquals(expected, illegalAccessError.getMessage());
```

3. 平手的測試資料

   全部設置為 1,1,0

   最後的

   ```
   Wyoming,220,1,0
   Washington D.C.,269,0,1
   ```

   測試是否兩個的得票數有一樣

4. Biden Split 的測試資料

   修改平手的測資，改成 Biden 獲勝，然後刻意調高 Trump 的得票數測試 Split

   ```
   Wyoming,220,0,10000
   Washington D.C.,269,1,0
   ```

5. Trump Align 的測試資料

   修改平手的測資，改成 Trump 獲勝，然後刻意調高 Trump 的得票數測試 Align

   ```
   Wyoming,220,0,100
   Washington D.C.,269,0,1
   ```

6. Trump Split 的測試資料

   修改平手的測資，改成 Trump 獲勝，然後刻意調高 Biden 的得票數測試 Split

   ```
   Wyoming,220,0,1
   Washington D.C.,269,0,1
   ```

7. 測試 (Nebraska && Maine 比例分配) 是否有符合比例較大者 + 1 的條件

   將最一開始的測資進行修改

   調成比例一致

   `Maine,4,50,50`

   接著調成 Biden 多一張

   `Maine,4,51,49`

自評

(O) 能修正程式，使之可以進行測試
(O) 測試策略說明完整，能夠因此產生豐富而有效的測試案例
(O) 能夠修改 Nebraska/Maine 的特殊處理，並進行測試
(O) 透過 JUnit 進行自動化的測試，把各種情境都確認無誤
(O) 改善 JUnit 的程式使之模組化，方便進行多個資料測試