OOP_PROJECT
Checkpoint3

學號：A1093321

姓名：時維駿

指導教授：楊子賢

日期：111 年 6 月 20 日

一、 需求描述

1. 完成所有種類卡牌（包含寶石牌、神器牌與災難牌）的各個方法。

從 deck 抽牌後，判斷抽出的牌哪個種類，寶石牌使用 share 方法；神器牌則是在有唯一玩家選擇離開時加入收集之中；災難牌則是判定 path 中是否已經存在，如果出現兩張同樣的災難，視為災難發生，所有玩家直接逃離，放棄收集。

2. 依照現有實作內容與標準遊戲規則，建構出完整的遊戲運行架構。

遊戲將進行五個回合，所有玩家離開或是災難發生將會結束回合，不再像上次一樣限制 path 的牌數。

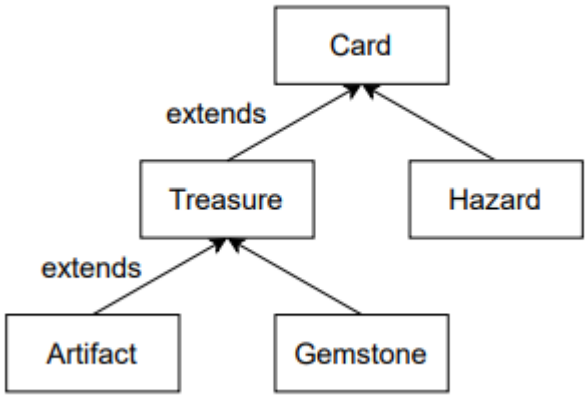
3. 於遊戲結束時，依據玩家所收集的寶藏總數決定最終勝者。
4. 於 Player.java 自訂選擇留下與否的邏輯，達到最高勝率。

二、 OO 的考量

Project2 分為 Main、Game、Environment、Gemstone、Agent、Artifact、Card、Hazard、Treasure、Player、Computer 十一個檔案。

1. Main：作為主程式使用。
2. Game：遊戲主體，規劃遊戲的流程以及使用各種方法達成目標。
   有著 setUpCards(使用 deck.clear 後可重設 deck 的寶石數量)和 suffleDeck(重設 deck 後可將 deck 洗牌)兩種方法。
3. Environment：定義各玩家執行動作(stay or leave)的機率。
4. Gemstone：繼承了 Treasure，分為種類和寶石的數量，遊戲中會有 deck 和 path 兩種屬於此 class 的 object，也包括 share 方法計算分配的寶石數量 (remainValue 和 value)。.
5. Agent：玩家所有資訊，有各玩家的編號，分別擁有的寶石(陵墓中或是帳篷中)，以及狀態(stay or leave)。有著將寶石加上玩家所擁有的方法 addCollectedGems 和將擁有的寶石存入帳篷的方法 storeGemsIntoTent，以及決定玩家狀態(stay or leave)的方法 act。
6. Player：繼承 Agent，讀取 fileName 檔案中的資訊，定義 Player 選擇的邏輯。
7. Computer：繼承 Agent，定義電腦的選擇邏輯。

Card, Hazard, Treasure, Gemstone, Artifact 五個檔案的關係以下圖說明



OOA & OOD：

這次的需求相比於 Project1，我認為不需做太多的修改，更多的是要加上判斷，知道抽到的牌需要作出甚麼相對應的動作，重點將會放在 Card 的各個 Child Class，因為都會放在屬於 Card 的 arraylist(deck&path)，因此在操作時可能會出現一些問題。

在我實作這次的 project 時，我碰到的問題是 Hazard 無法直接從 deck 之中 remove，因此我最後想出的辦法是以 Hazard 和 deck 都有的 name()來判斷，直接使用 deck.get(j).remove，而不是以 Hazard 的 object 直接從 deck 移除。

OOP：

(1) Encapsulation：在本次 project 中，許多 class 都有 private 變數，如果要從 class 外使用、操作這些變數，就要透過 public 的方法。

(2) Inheritance：這次的繼承主要是上面那張圖的部分

I. Card 屬於 abstract class，遊戲中會有 deck 和 path 兩種屬於此 class 的 object。
II. Hazard 繼承了 Cards，每個災難種類是一個 type。
III. Treasure 繼承了 Cards，使用 Cards 的 type，再加上 value，因為繼承 Treasure 的兩個子類別 Gemstore 和 Artifact 都是有著所代表的寶石數量的。
IV. Artifact 繼承了 Treasure，分為種類和代表的寶石的數量。
V. Gemstone 繼承了 Treasure，分為種類和寶石的數量。

(3) Polymorphism：上圖中，每個 class 都有的就是 name()，由於在 Card 中是宣告為 abstract 的型態，因此每個繼承的 class 都要重新定義，也就造成了一字多義的結果。

三、 功能/邏輯說明

```
Scanner input = new Scanner(Paths.get(fileName));
while (input.hasNext()){
    String str = input.nextLine();
    if (str.contains(s: "<A: ")){
        return false;
    }
}
return true;
```

首先介紹這次 checkpoint 主要實作的 Player 部分，除了一開始就有的 java.io.IOException，我還額外 import 了 java.nio.file.Paths 和 java.util.Scanner ，用以讀取 fileName 的內容，並在第一次讀到神器出現就 return false 離開，以這樣的方式奪取神器。

```
int obtain = 0;
int IN_count = 0;
int OUT_count = 0;
int remain_distributed = 0;

//IN & OUT counts
for (int i = 0; i < receivers.size(); i++){
    if ( receivers.get(i).isInExploring() == true)
        IN_count++;
    else
        OUT_count++;
}

if (IN_count != 0 ){ //傳進的receivers是所有的玩家
    obtain = value / IN_count;
    remainValue = value % IN_count;
}

if (OUT_count != 0 && IN_count == 0 ){ //傳進的receivers是決定離開的玩家
    remain_distributed = remainValue / OUT_count;
    remainValue = remainValue % OUT_count;
}

for (int i = 0; i < receivers.size(); i++){ //將玩家此輪獲得的寶石加入擁有的寶石中
    if ( receivers.get(i).isInExploring() == true)
        receivers.get(i).addCollectedGems(obtain);
    else
        receivers.get(i).addCollectedGems(remain_distributed);
}
```

share 方法(標示於流程圖第 5 點)，其中宣告了四個 integer :

1. obtain 是陵墓中玩家能拿到的數量。

2. remain_distributed 則是離開的玩家能拿到的數量。
3. IN_count 是在陵墓中的玩家數量。
4. OUT_count 則是離開陵墓的玩家數量。

先計算陵墓中和離開的玩家數量，因為丟進來的 receivers 會有兩種，一種是所有的玩家，另一種則是每次決定要離開的玩家。所以在這邊先統計數量，以便決定運算方式和運算的進行。

接下來分為 6 個部分，標示於下方流程圖中。

1. (第一點)回合開始(初始化)：

   這次的初始化包括五個動作

   (1)清空 path 使用 path.clear()來完成。

   (2)清空 deck 則使用 deck.clear()來完成。

   (3)加入寶石和災難牌，使用 setUpCards()來完成。

   (4)加入神器牌，使用 deck.add(artifacts.get(round))來完成

   (5)將上回合發生的災難牌，從 deck 中移除。使用以下的方法：

```
for (int i = 0; i < occurredHazards.size();i++){
    for (int j = 0; j < deck.size(); j++){
        if (deck.get(j).name() == occurredHazards.get(i).name()){
            deck.remove(j);
            break;
        }
    }
}
```

   (6)最後是將完成的牌庫洗牌，使用 shuffleDeck()完成。

2. (第二點)判斷抽出的牌是哪個種類：

   這邊我使用了兩種方法，第一種是使用 deck.get(0).name()來判斷，因為寶石牌的名稱只有 Gemstone 一種，因此能使用這個方式。

   但是神器和災難牌的名稱有許多種，因此我使用了 instanceof，由此得知抽出的牌的類型，並決定要做出哪些動作。

3. (第三點)使用 act(environment)讓每個人決定 stay or leave。如果已經選擇離開，便不會再有新的動作。

```
for (int i = 0; i < 6 ; i++)
    explorers.get(i).act(environment);
```

4. (第四點)印出"X hazard occurs, all explorers attempt to flee!"，使用 break 中斷這次的 while (this.isAnyoneStay())，進入下一回合。

```
if (HazardThisTurn.isEmpty() == false){
    System.out.printf(format: "%s %s%n%n", HazardThisTurn.get(index: 0).name(), H);
    break;
}
```

5. (第六點)使用神器的 share 方法，判斷此輪離開的人數是否為一人，如果是一人，該人獲得神器，神器也將不存在於 path 之中。

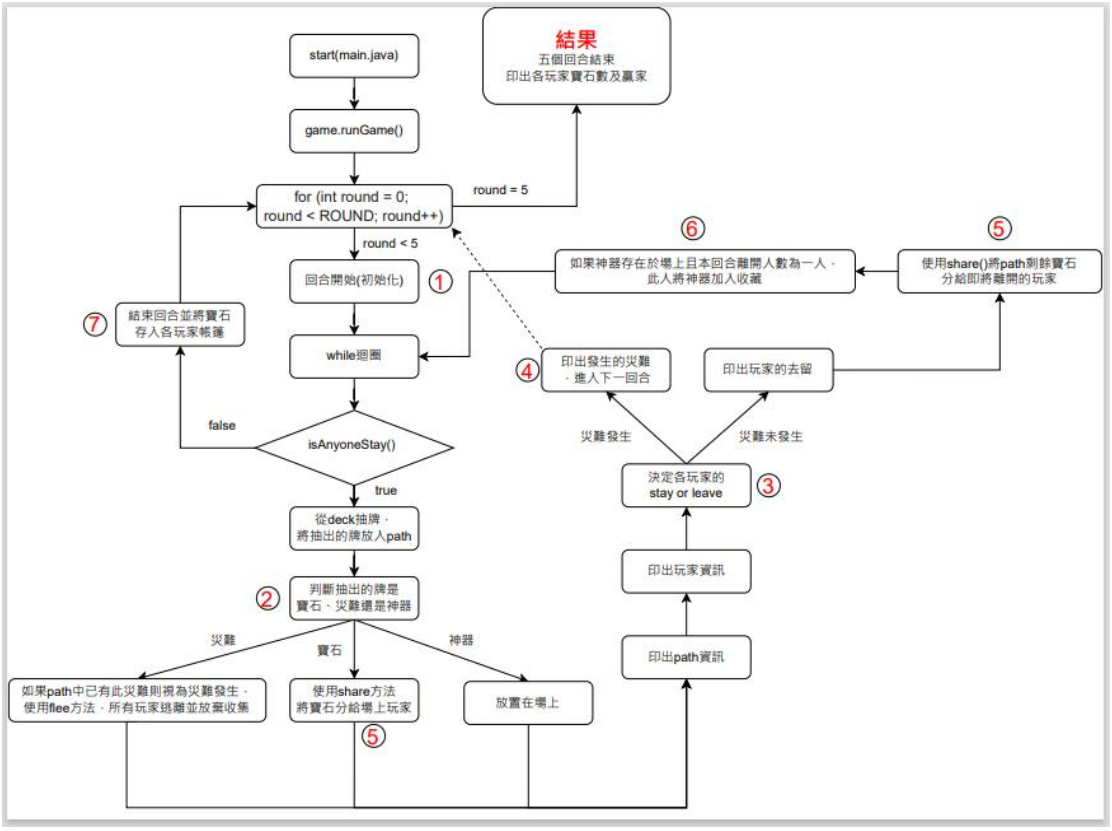
```
int OUT_count = 0;
int OUT_NUM = 0;

for (int i = 0; i < receivers.size(); i++){
    if ( receivers.get(i).isInExploring() == false)
        OUT_count++ ;
        OUT_NUM = i;
}
if ( OUT_count == 1 ){
    receivers.get(OUT_NUM).getOwnedArtifacts().add(new A1093321_Project2_Artifact(this.getType(), this.getValue()));
    this.inTomb = false;
}
```

6. (第七點)回合結束時，將每個人擁有的寶石存入帳篷

```
for (int i = 0; i < 6 ; i++)
    explorers.get(i).storeGemsIntoTent();
```

四、 程式流程圖



五、 使用說明

因為 main 有包含主程式 public static void main(String[] args)，因此只要對它進行動作，其他附屬的檔案也會一起編譯、執行。

但這次的情形比較特殊，Player 和 Computer 無法和主程式一起被編譯，因此必須個別編譯，才能執行程式。

在 linux 環境中編譯之後，執行時，要在指令後方加入參加者，例如 java Project3_Main Project3_Player Project3_Computer Project3_Computer，表示參加者包括一位 Player 和兩位 Computer(必須符合 3~8 人)。

輸出結果方面則和 checkpoint2 完全相同。

六、 其他

無。