Q1: (60 minutes, 08.30-09.30)

1. Instruction

1) Create a Java Project named "**2110215_Final_Q1**".
2) Copy all folders in **"toStudent/Q1"** to your project directory src folder.
3) You are to implement the following classes (detail for each class is given in section 2 and 3.
    a) **Ordinary** (package role.derived)
    b) **Master** (package role.derived)
    c) **Seer** (package role.derived)
    d) **Gambler** (package role.derived)
    e) **Player** (package game.object)
4) Make UML class diagram for classes in package role.base and role.derived, using ObjectAid or another UML generator program. Save its picture file in your role.derived folder.
5) **JUnit for testing is in a package test.**
6) **To submit:**
    6.1. go to src folder that you actually do the coding for this question.
    6.2. Zip this question's src folder. Name it YOUR-ID_Q1.zip (for example, 6332112121_Q1.zip)
    6.3. Submit the zipped file as an assignment on MyCourseville.

2. Problem Statement: Marble Master

In Marble Master, a group of **4 *players*** is playing a game of fortune. At the start, each player initially has **20 *marbles***, and is randomly assigned to one of the following **4 *roles***: **Master**, **Seer**, **Gambler**, and **Ordinary** (every player has a distinct role and does not know the role of each other).

The game is played in ***rounds***. At the beginning of each round, there are **7 *cards*** to be chosen and picked up by the players. Each card has its own hidden value, called ***multiplier***. The cards are placed in ***slots***, numbered from 0, 1, 2, 3, 4, 5, 6. The **Master** is the one who decides where to place them; thus, only the **Master** knows which cards are placed in which slots. Then the players take ***turns*** to play by following some specific ***order***, which is determined by the **Gambler**.

The order of playing is numbered as follows: 0, 1, 2, 3. That is, the player who takes the first turn of each round is in order number 0. For each player, when it is in his/her turn, the player chooses one of the remaining slot numbers and picks up the card in that slot. The information of

the chosen card will be announced to every player, e.g., the slot number and the multiplier of that card.

However, some players have an extra action before playing his/her turn. Firstly, the Seer can secretly choose two cards to see their multiplier and swap them before returning them to the slots. It is possible that the card he will choose and pick up in his/her normal playing turn is not from the two cards he/she viewed earlier. Moreover, the Gambler can secretly choose ***two target players***. It is possible that the Gambler may choose him/herself as one of the targets.

At the end of the round, i.e., after every player has played their turn, the ***two target players*** (chosen by the Gambler) will have to *swap their role and card*. Then every player will gain or lose some number of marbles, which is determined by the product of the multiplier and the slot number of their chosen card. The first player who successfully collects at least **100** marbles wins and becomes the **Marble Master**! If there is a tie, an extra round will be played until the tie is broken.

3. Implementation Detail

The class package is summarized below.

*** In the following class description, only details of IMPORTANT fields and methods are given.***

3.1 Package role.base

3.1.1 abstract class Role /* ALREADY PROVIDED */

This class is an abstract class and also a **base class** for **all roles** of Player. It contains all common actions which each role must do.

3.1.2. Interface PreRoundActable /* ALREADY PROVIDED */

Method

Name	Description
+ abstract void preRoundActs(GameAction action)	Method to take an action before entering each round.

3.1.3. Interface PreturnActable /* ALREADY PROVIDED */

Method

Name	Description
ostract void preTurnActs(GameAction ction)	Method to take an action before entering each turn.

3.2 Package game.logic

3.2.1. class GameAction /* ALREADY PROVIDED */

This is a wrapper class for the action's information. The information is an array of **GameObject**s. The type and the length of **GameObject**s are dependent on the **role** of the player and when the action takes place (pre-round or pre-turn). For more details, see the description of each role.

Fields

Name	Description
+ final int length	The number of GameObjects related to this action.
- ArrayList<GameObject> info	A list of GameObjects related to this action.

Method

Name	Description
+ GameAction(ArrayList<GameObject> info)	This is the Constructor. Initialize the GameAction by setting length and info
+ ArrayList<GameObject> getInfo()	This method return the ArrayList of GameObject
+ GameObject getInfo(int slot)	This method return GameObject at index slot

3.2.2. class GameLogic /* ALREADY PROVIDED */

3.2.3. class GameRound /* ALREADY PROVIDED */

3.3 Package role.derived /* You must implement this ENTIRE package from scratch */

3.3.1. class Ordinary /* You must implement this class from scratch */

Ordinary is a **Role** with no special actions.

Method

Name	Description
+ String toString()	/* FILL CODES */ This method return “Ordinary”

3.3.2. class Gambler /* You must implement this class from scratch */

Gambler is a **Role** which has to set the playing order of all the players before the round starts. Thus, the **GameAction** that Gambler will work on will contain **4** GameObjects where each **GameObject** is a **Player**. These Players are stored in the specific playing order. For example, if the GameAction contains 4 players: <**a**, **b**, **c**, **d**>, then **a** is the first player to play in this round. You have to set the order of player **a** as **0**, set the order of player **b** as **1**, and so on.

Also, before playing his/her turn, the **Gambler** has to choose two target players. The target players will have to swap their role and their card at the end of the round. For the purpose of testing, the system has already chosen the two players for you. Hence, the **GameAction parameter** will contain **2** GameObjects where each **GameObject** is a **Player**. Then, you only have to set these two players as to-be-swapped.

Method

Name	Description
+ void preRoundActs(GameAction action)	/* FILL CODES */ Use this method to shuffle the order of all the players by setting each Player's order based on the given action's info. HINT: Use the method **action.getInfo(i)** to get the i^{th} **GameObject** of this action. For each **GameObject**, consider it as a **Player**, and use the method **setOrder(i)** from the class **Player** to set the playing order of this player as **i**.

	Noted: **order** starts with **0** and ends with **action.length-1**
+ void preTurnActs(GameAction action)	/* FILL CODES */ Use this method to mark 2 players in the given GameAction as to-be-swapped. HINT: Use the method **action.getInfo(i)** to get the i^{th} **GameObject** of this action. For each **GameObject**, consider it as a **Player**, and use the method **setToBeSwapped(true)** from the class **Player** to set this player as to-be-swapped.
+ String toString()	/* FILL CODES */ This method return “Gambler”

3.3.3. class Master **/* You must implement this class from scratch */**

Master is a **Role** which has to decide where to place all the cards before the round starts. Thus, the **GameAction** that it works on will contain **7** GameObjects where each **GameObject** is a **Card**. These Cards are stored in the specific slot. For example, if the GameAction contains 7 cards: <**c4**, **c2**, **c1**, **c3, c5**, **c0**, **c6**>, you have to set the slot of card **c4** as **0**, set the slot of card **c2** as **1**, and so on.

Method

Name	Description
+ void preRoundActs(GameAction action)	**/* FILL CODES */** Use this method to set the cards' slot in the given GameAction HINT: Use the method **action.getInfo(i)** to get the i^{th} **GameObject** of this action. For each **GameObject**, consider it as a **Card**, and use the method **setSlot(i)** from the class **Card** to set the slot number of this card as **i**. Noted: **slot** number start with **0** and ends with **action.length-1**
+ String toString()	**/* FILL CODES */** This method return "Master"

3.3.4. class Seer /* You must implement this class from scratch */

Seer is a **Role** which has to secretly choose two cards to see their multiplier and swap them before playing his/her turn. For the purpose of testing, the two cards are already chosen for you. Thus, the **GameAction** will contain **2** GameObjects where each **GameObject** is a **Card**. Then, you only have to swap them.

Method

Name	Description
+ void preTurnActs(GameAction action)	/* FILL CODES */ Use this method to swap 2 Cards in the given GameAction. HINT: Use the method **action.getInfo(i)** to get the i^{th} **GameObject** of this action, and consider it as a **Card.** Then, Swap them by using the method from package game.logic **GameLogic.*getInstance*().getCurrentRound().swapCards(..., ...)**
+ String toString()	/* FILL CODES */ This method return “Seer”

3.4 Package game.object

3.4.1. class Card /* ALREADY PROVIDED */

Card is a GameObject that every role has to pick one and gain (or lose) marbles from it. The cards are placed in slots, numbered from 0, 1, 2, ..., 6. This class contains the method which will be used in role.derived

Fields

Name	Description
- String id	id of the Card
- int slot	slot number of the Card (0,1,2,...,6)
- int multiplier	Each card has its own hidden value, called multiplier
- boolean pickedUp	True if the Card is picked up

Method

Name	Description
+ Card(String id, int multiplier)	This is the Constructor. Initialize the card by setting slot to -1, set id, setting multiplier, and setting pickedUp as false
+ String toString()	Return Card's id and its multiplier Ex. [C001: -2]
+ String getStatus()	This method return only Card slot if it's not picked up, otherwise return Card slot and its multiplier (Calling revealInfo())

	Ex. [S0|####] or [S1| -2]
+ String revealInfo()	Return Card slot and its multiplier Ex. [S1| -2]
getter/setter for all fields	

3.4.2. class abstract GameObject /* ALREADY PROVIDED */

GameObject is the abstract base class for **Player** and **Card**.

3.4.3. class Player /* You must implement some methods of this class*/

Player is GameObject. This class represents a player. It contains player information and status.

Fields

Name	Description
- String name	The name of the player
- int order	The order to play in each round
- Role role	The role of the player in each round
- int marblesEarned	Amount of the marble that player has earned during the game. At the start, each player has **20** marbles.
- boolean toBeSwapped	True when the player is chosen to be swapped role with another player by the Gambler

Method

Name	Description
+ Player(String name, Role role)	This is the Constructor. Initialize all fields of Player
+ void doPreRoundAction(GameAction action)	/* FILL CODES */ This method is called to do an action before entering each round. If the player's role has the action to do before entering each round, you have to consider the player's role as **PreRoundActable**, and then call the method **role.preRoundActs(...)**
+ void doPreTurnAction(GameAction action)	/* FILL CODES */ This method is called to do an action before entering each turn. If the player's role has the action to do before entering his/her turn, you have to consider the player's role as **PreTurnActable**, and then call the method **role.preTurnActs(...)**
+ int getRoundRewards()	Return the marbles that player's going to receive calculated by slot number multiply by multiplier of the chosen Card
+ void takesRoundRewards()	This method set Player's marblersEarned at the end of each round
+ String toString()	Return player's name and his/her marblesEarned Ex. [Anny: 20]
getter/setter for all fields	

3.5 Package main

3.5.1 class Main /* ALREADY PROVIDED */

This class is the main program. You don't have to implement anything in this class. You can test the program by running this class.

3.6 Package test /* ALREADY PROVIDED */

This package provides JUnit test for each of your class.

4. Finished Code Run Example (this is one of possible runs)

Just run and the game logic will do everything at random. You don't have to interact with anything after running the code.

4.1. Start the game

```
[ANNOUNCEMENT] ##########################################################################################
[ANNOUNCEMENT] GAME STARTED
[ANNOUNCEMENT] Players: [Anny: 20] [Beth: 20] [Carl: 20] [Doug: 20]
[ANNOUNCEMENT] Cards:   [C001: -2] [C002: -1] [C003:  0] [C004:  1] [C005:  3] [C006:  5] [C007:  7]
[ANNOUNCEMENT] ##########################################################################################
```

4.2. Each round

```
[ANNOUNCEMENT] ==========================================================================
[ANNOUNCEMENT] ROUND  1
[ANNOUNCEMENT] Players: [Anny: 20] [Beth: 20] [Carl: 20] [Doug: 20]
[ANNOUNCEMENT] ==========================================================================
[ANNOUNCEMENT] > Pre-round Actions
 [TO: Carl<M>]    You have set the cards' slot as follows:
 [TO: Carl<M>]      [S0| -1 ] [S1| -2 ] [S2|  1 ] [S3|  0 ] [S4|  7 ] [S5|  3 ] [S6|  5 ]
 [TO: Doug<G>]    You have set the players' order as follows:
 [TO: Doug<G>]      [Doug: 20] [Beth: 20] [Anny: 20] [Carl: 20]
[ANNOUNCEMENT]
[ANNOUNCEMENT] > Current Slots: [S0|####] [S1|####] [S2|####] [S3|####] [S4|####] [S5|####] [S6|####]
[ANNOUNCEMENT]
[ANNOUNCEMENT] > [Turn: 1] Now it is Doug's turn.
 [TO: Doug<G>]    You have chosen to swap (at the end of this round) the role and card of these players:
 [TO: Doug<G>]      [Carl: 20] [Anny: 20]
 [TO: Doug<G>]    You have chosen a card at slot (S1).
[ANNOUNCEMENT]   Doug has chosen a card at slot (S1) which has multiplier (-2).
[ANNOUNCEMENT]
[ANNOUNCEMENT] > Current Slots: [S0|####] [S1| -2 ] [S2|####] [S3|####] [S4|####] [S5|####] [S6|####]
[ANNOUNCEMENT]
[ANNOUNCEMENT] > [Turn: 2] Now it is Beth's turn.
 [TO: Beth<S>]    You have chosen to view and swap the slot of these cards:
 [TO: Beth<S>]      [S6|  3 ] [S5|  5 ]
 [TO: Beth<S>]    You have chosen a card at slot (S6).
[ANNOUNCEMENT]   Beth has chosen a card at slot (S6) which has multiplier ( 3).
[ANNOUNCEMENT]
[ANNOUNCEMENT] > Current Slots: [S0|####] [S1| -2 ] [S2|####] [S3|####] [S4|####] [S5|####] [S6|  3 ]
[ANNOUNCEMENT]
[ANNOUNCEMENT] > [Turn: 3] Now it is Anny's turn.
 [TO: Anny<O>]    You have chosen a card at slot (S2).
[ANNOUNCEMENT]   Anny has chosen a card at slot (S2) which has multiplier ( 1).
[ANNOUNCEMENT]
[ANNOUNCEMENT] > Current Slots: [S0|####] [S1| -2 ] [S2|  1 ] [S3|####] [S4|####] [S5|####] [S6|  3 ]
[ANNOUNCEMENT]
[ANNOUNCEMENT] > [Turn: 4] Now it is Carl's turn.
 [TO: Carl<M>]    You have chosen a card at slot (S0).
[ANNOUNCEMENT]   Carl has chosen a card at slot (S0) which has multiplier (-1).
[ANNOUNCEMENT]
[ANNOUNCEMENT] > Current Slots: [S0| -1 ] [S1| -2 ] [S2|  1 ] [S3|####] [S4|####] [S5|####] [S6|  3 ]
[ANNOUNCEMENT]
[ANNOUNCEMENT] > [Round Summary]
[ANNOUNCEMENT]   The role & card of (Anny) and (Carl) are swapped.
[ANNOUNCEMENT]   Rewards: [Anny: + 0 ] [Beth: +18 ] [Carl: + 2 ] [Doug: - 2 ]
```

4.3 Pre round interface

be able to set cards' slot and players' order

```
[ANNOUNCEMENT] > Pre-round Actions
 [TO: Carl<M>]    You have set the cards' slot as follows:
 [TO: Carl<M>]      [S0| -1 ] [S1| -2 ] [S2|  1 ] [S3|  0 ] [S4|  7 ] [S5|  3 ] [S6|  5 ]
 [TO: Doug<G>]    You have set the players' order as follows:
 [TO: Doug<G>]      [Doug: 20] [Beth: 20] [Anny: 20] [Carl: 20]
```

4.4 Pre turn interface

Be able to view and swap the slot of the card.

```
[ANNOUNCEMENT] > [Turn: 2] Now it is Beth's turn.
 [TO: Beth<S>]    You have chosen to view and swap the slot of these cards:
 [TO: Beth<S>]      [S4|  0 ] [S3|  5 ]
 [TO: Beth<S>]    You have chosen a card at slot (S6).
```

Be able to swap the role and card of the players

```
[ANNOUNCEMENT] > [Turn: 4] Now it is Doug's turn.
 [TO: Doug<G>]    You have chosen to swap (at the end of this round) the role and card of these players:
 [TO: Doug<G>]      [Doug: 18] [Carl: 22]
```

4.5. Round summary

```
[ANNOUNCEMENT] > [Round Summary]
[ANNOUNCEMENT]   The role & card of (Carl) and (Doug) are swapped.
[ANNOUNCEMENT]   Rewards: [Anny: -10 ] [Beth: +42 ] [Carl: + 3 ] [Doug: + 2 ]
[ANNOUNCEMENT]
```

4.6. Game Ending

```
[ANNOUNCEMENT]
[ANNOUNCEMENT] ################################################################################
[ANNOUNCEMENT] GAME ENDED
[ANNOUNCEMENT] Players: [Anny: 18] [Beth: 100] [Carl: 31] [Doug: 40]
[ANNOUNCEMENT] The winner is Beth!!
[ANNOUNCEMENT] ################################################################################
```

5. Score Criteria

The maximum score for the problem is 35 and will be adjusted to 5.

5.1 Class Ordinary: **= 4 points**

- testRole = 3 points
- testToString = 1 point

5.2 Class Master: **= 7 points**

- testRole = 3 points
- testPreRoundActs = 3 points
- testToString = 1 point

5.3 Class Seer: **= 7 points**

- testRole = 3 points
- testPreTurnActs = 3 points
- testToString = 1 point

5.3 Class Gambler: **= 10 points**

- testRole = 3 points
- testPreRoundActs = 3 points
- testPreTurnActs = 3 points
- testToString = 1 point

5.4 Class Player: **= 6 points**

- testDoPreRoundAction = 3 points
- testDoPreTurnAction = 3 points

5.5 UML: **= 1 point**