Lab 3: Abstract Class

Instruction

1. Click the provided link on MyCourseVille to create your own repository.
2. Create an IntelliJ Project called **LabProgmethIntelliJ**.
3. Inside **LabProgmethIntelliJ** , make an **IntelliJ module Lab3_2110215_2023_1_{ID}_{FIRSTNAME}**
   - Example: **Lab3_2110215_2023_1_6531234521_Jolyne**
4. Initialize git in your **module** directory
   - **Add .gitignore. (for IntelliJ)**
   - Commit and push initial codes to your GitHub repository.
5. Implement all the classes and methods (copying initial code from the given lab file) following the details given in the problem statement file which you can download from CourseVille.
   - You should create commits with meaningful messages when you finish each part of your program.
   - Don't wait until you finish all features to create a commit.
6. Test your codes with the provided JUnit test cases, they are inside package **test.student**
   - **You have to write some tests by yourself**. Put them inside package **test.student**
   - Aside from passing all test cases, your program must be able to run properly without any runtime errors.
   - There will be additional test cases (NOT given to you) to test your code after you submit the final version, **make sure you follow the specifications in this document**.
7. After finishing the program, create a UML diagram and put the result image (**UML.png**) at the **src** folder of your module.
8. Export your project into a jar file called **Lab3_2023_1_{ID}** and place it at the **src** folder of your module. **Include your source code in the jar file**.
   - Example: **Lab3_2023_1_6531234521.jar**
9. Push all other commits to your GitHub repository.

1. Problem Statement: no u

In this lab, we will implement the popular friendship-destroying card game ***no u (definitely not known as UNO or anything)***. The game contains a deck of 6 different types of cards in 4 colors - Red, Yellow, Green, and Blue. Each player is dealt 7 cards and then a card is placed at the center pile. Players take turns playing a card that has either a matching color or matching symbol to the top card of the pile, and the loser is the last person to get rid of all their cards. Some cards will have special effects that can sabotage other players' efforts, thus it is a game of backstabbing your friends!

Here you can only play against dummy opponents with pre-programmed behavior, so, fortunately, no friendships will be destroyed in this lab.

1. Game Flow

1. At the start of the game, the deck is shuffled and each player is dealt 7 cards. 1 card is drawn from the top of the deck to be the first *top card*, which players will have to match their cards with. The rest of the deck is placed face down next to the top card, then the game begins. You play as player 0, and start the game.
2. On each player's turn, the player chooses a card to play.
    - If there exists at least one playable card in the player's hand, the player chooses one of the playable cards.
    - If no cards in the player's hand are playable, draw a card from the deck.

        i. If the drawn card is playable, the player will be forced to choose it.
        ii. If not, no card will be chosen.
3. Play the chosen card (or play nothing if no card can be chosen). If the card has an effect i.e. skip, reverse, the effect is performed. This effect may cause changes in the order of the next player (see the card details below). The turn ends.
4. The next player is determined by the current play direction, which is forward by default.
5. Steps 2-3 are repeated for each player until the game ends.
6. The game ends when you have no cards left in your hand (you win!) or you are the last player with cards in your hand (you lose).

2. Cards

1. Number Card
    - Each Number Card has a number (0-9) and color (Red Yellow, Green, Blue).
    - Can be matched with a card that has either the same number or same color.
    - There is no effect when played.
2. Reverse Card
    - Each Reverse Card has the reverse symbol and a color (Red Yellow, Green, Blue).
    - Can be matched with a card that has either the same symbol (reverse) or same color.
    - When played, the play direction is changed to the opposite direction. There are 2 directions:
        i. Forward: the player turns go from 0, 1, 2, 3, 0 …
        ii. Backward: the player turns go from 0, 3, 2, 1, 0 …
3. Skip Card
    - Each Skip Card has the skip symbol and a color (Red Yellow, Green, Blue).
    - Can be matched with a card that has either the same symbol (skip) or same color.
    - When played, the next player with a non-empty hand is skipped. The next player is determined by the current play direction.
        i. For example, if the current direction is forward and all 4 players still have cards in their hand, when player 0 plays skip, player 1 will be skipped and player 2 will take the next turn.

        ii. From above, if player 1 already has an empty hand, player 2 will be skipped and player 3 will take the next turn.

4. Change Color Card
    - Each Change Color has the change color symbol and no color.
    - Can be matched with any card.
    - When played, a color will be set to the card. Here we set the color to the color of the first card in the player's hand after this card has been discarded.
        i. If that first card has no color i.e. it's a Change Color Card or Draw Four Card, or there are no cards left in the player's hand, set the color to **Red**.
5. Draw Two Card
    - Each Draw Two Card has the draw two symbol and a color (Red Yellow, Green, Blue).
    - Can be matched with a card that has either the same symbol (draw two) or same color.
    - When played, increase the *draw amount* by 2, then the turn moves to the next player with a non-empty hand.
        i. If the player has a **Draw Two or Draw Four Card**, play the first one in hand. The player won't have to draw.
        ii. If the player doesn't have either card, draw *draw amount* number of cards, then reset the *draw amount* to 0.
6. Draw Four Card
    - Each Draw Four Card has the draw four symbol and no color.
    - Can be matched with any card.
    - When played:
        i. First a color will be set to the card. Here we set the color to the color of the first card in the player's hand after this card has been discarded.
            1. If that first card has no color i.e. it's a Change Color Card or Draw Four Card, or there are no cards left in the player's hand, set the color to **Red**.
        ii. Then increase the *draw amount* by 4. The turn moves to the next player with a non-empty hand.
            1. If the player has a **Draw Four Card**, play the first one in hand. The player won't have to draw.
            2. If the player doesn't have either card, draw *draw amount* number of cards, then reset the *draw amount* to 0.

2. Implementation Details:

To complete this assignment, you need to understand **Abstract Classes** and **Junit Test Cases**.

To test your understanding about abstraction, **we will not provide a class diagram for this assignment, and we will not indicate which methods and classes are abstract.** Try your best to figure them out. There are **two** abstract classes and **four** abstract methods.

There are **five** packages in the provided files: *application*, *logic.card*, *logic.game*, *test* and *utils*.

You will be implementing all of the classes in the *logic.card* package and some methods in *logic.game.* (Every class is partly given.) Note that only important methods that you need to write and methods that you need to complete this question are listed below.

There are some test cases given in package *test.student*. These will help test your code whether it will be able to run or not. However, **some conditions are not tested** in these test cases. **Look for those conditions in the class details**. **You must create your own test cases for such cases.**

You can define any additional number of <u>private</u> (but not public, protected or package) fields and methods in addition to the fields and methods specified below. You are encouraged to try to group your logic into private methods to reduce duplicate code as much as possible.

** Noted that Access Modifier Notations can be listed below*

+ ***(public), # (protected), - (private)***

2.1 package logic.card

2.1.1 Class BaseCard /*This class is partially given*/

This class is a base class for all the Cards. It contains all common elements of a card in this game. Note that this card should never be instantiated. It acts like a base code for other types of cards.

Fields

- CardColor color	The color of this card.
- CardSymbol symbol	The symbol/number of this card.

Methods

+ BaseCard(CardColor color, CardSymbol symbol)	Constructor. Set fields to the given parameter values.
+ String toString()	Returns this card name as a string. The format depends on the type of card. *Note that this method doesn t have body and depends on its subclass.*
+ boolean canPlay()	Returns whether or not this card can be played, by checking if can be matched with the current top card. *Note that this method doesn t have body and depends on its subclass.*
+ String play()	Plays this card and returns a message to be printed in the game loop. *Note that this method doesn t have body and depends on its subclass.*
+ CardColor getColor() + void setColor() + CardSymbol getSymbol() + void setSymbol()	Getter and setter methods.

2.1.2 Class NumberCard /*This class is partially given*/

This class represents a Number Card, which can be matched with another card that has either the same number or same color. It has no effect when played.

Methods

+ NumberCard(CardColor color, CardSymbol symbol)	Constructor. Set fields of the super class.
+ String toString()	Returns this card name as a string. In the format **"COLOR SYMBOL"**. For example, **RED ONE**. *Note: You can use the toString() method of CardColor and CardSymbol to get the color and symbol string.*
+ boolean canPlay()	Returns whether or not this card can be played, by checking if this card has the same color or symbol as the current top card. (see class GameLogic in logic.game)
+ String play()	Plays this card, by setting the top card to this card and then removing this card from the current player's hand (see class GameLogic in logic.game). Returns null.

2.1.3 Class EffectCard /*This class is partially given*/

This class is a base class for all Cards with an effect. It contains common behavior for cards with effects. Note that this card should never be instantiated. It acts like a base for other types of cards that have effects.

Methods

+ EffectCard(CardColor color, CardSymbol symbol)	Constructor. Set fields of the super class.
+ String performEffect()	Performs this card's effect and returns a message to be printed in the game loop. *Note that this method doesn t have body and depends on its subclass.*
+ String play()	Plays this card, by setting the top card to this card, removing this card from the current player's hand (see class GameLogic in logic.game), then performing this card's effect. Returns the message from performing the effect (see method above).

2.1.4 Class ReverseCard /*This class is partially given*/

This class represents a Reverse Card, which can be matched with another card that has either the same symbol or same color, and will change the play direction to an opposite direction when played.

Methods

+ ReverseCard(CardColor color)	Constructor. Set color of the super class. Set symbol to REVERSE.
+ String toString()	Returns this card name as a string. In the format **"COLOR SYMBOL"**. For example, **GREEN REVERSE**. *Note: You can use the toString() method of CardColor and CardSymbol to get the color and symbol string.*
+ boolean canPlay()	Returns whether or not this card can be played, by checking if this card has the same color or symbol as the current top card.
+ String performEffect()	Performs this card's effect by setting the play direction to the opposite of the current direction (see class GameLogic and PlayDirection in logic.game). Then, return **"Set direction to D"** when D is the new direction. *Note: You can use PlayDirection.getOpposite() method.*

2.1.5 Class SkipCard /*This class is partially given*/

This class represents a Skip Card, which can be matched with another card that has either the same symbol or same color, and will skip the next player when played.

Methods

+ SkipCard(CardColor color)	Constructor. Set color of the super class. Set symbol to SKIP.
+ String toString()	Returns this card name as a string. In the format **"COLOR SYMBOL"**. For example, **GREEN SKIP**. *Note: You can use the toString() method of CardColor and CardSymbol to get the color and symbol string.*

+ boolean canPlay()	Returns whether or not this card can be played, by checking if this card has the same color or symbol as the current top card.
+ String performEffect()	Performs this card's effect by going to the next player whose hand is not empty. Then, return **"Skipped player N"** when N is the skipped player. *Note: You can use the GameLogic.getInstance().goToNextPlayer() method somehow.*

2.1.6 Class ChangeColorCard /*This class is partially given*/

This class represents a Change Color Card, which has no color before playing, can be matched with any card, and a color will be set when played.

Methods

+ ChangeColorCard()	Constructor. Set color to null. Set symbol to CHANGE_COLOR.
+ String toString()	Returns this card name as a string. If the color is not set, return **"CHANGE COLOR".** If the color is set, return **"CHANGE COLOR (COLOR color selected)"**. For example, **CHANGE COLOR (RED color selected)**.
+ boolean canPlay()	Returns whether or not this card can be played, which is always true.
+ String performEffect()	Performs this card's effect by setting this card's color. • Set the color to the color of the first card in the player's hand (the first card is counted after removing this card). • If that first card has no color i.e. it's a Change Color Card or Draw Four Card, or there are no cards left in the player's hand, set the color to **Red**. Then, return **"Set color to C"** when C is the set color.

2.1.7 Class DrawTwoCard /*This class is partially given*/

This class represents a Draw Two Card, which can be matched with another card that has either the same symbol or same color, and will make the next player have to draw cards unless they have a Draw Two or Draw Four Card.

Methods

+ DrawTwoCard(CardColor color)	Constructor. Set color of the super class. Set symbol to DRAW_TWO.
+ String toString()	Returns this card name as a string. In the format **"COLOR SYMBOL"**. For example, **BLUE DRAW TWO**. *Note: You can use the toString() method of CardColor and CardSymbol to get the color and symbol string.*
+ boolean canPlay()	Returns whether or not this card can be played, by checking if this card has the same color or symbol as the current top card.
+ String performEffect()	Performs this card's effect. 1. Increment *draw amount* by 2. 2. Go to the next player whose hand is not empty. 3. The player automatically takes actions depending on the state of their hand. • If the player has a Draw Two or Draw Four Card, select the first one in hand. The message to be returned **"Player N played CARD NAME. X cards remaining."**, when N is the current player, CARD NAME is the card name, and X is the number of cards that will be left in the current player's hand after playing. Then play the selected card and concatenate that card's message to this card's message (display on separate lines). • If the player doesn't have either card, draw *draw amount* number of cards. The message to be returned is **"Player N drew Y cards. X cards remaining."**, when N is the current player, Y is the draw amount, and X is the number of cards that will be left in the current player's hand after drawing. Then reset the draw amount to 0. 4. Return the message. *Note: You can use the methods GameLogic.getInstance().goToNextPlayer(), GameLogic.getInstance().incrementDrawAmount() GameLogic.getInstance().draw() somehow.*

2.1.8 Class DrawFourCard /*This class is partially given*/

This class represents a Draw Four Card, which has no color before playing, can be matched with any card, and a color will be set when played and will make the next player have to draw cards unless they have a Draw Four Card.

Methods

+ DrawFourCard()	Constructor. Set color to null. Set symbol to DRAW_FOUR.
+ String toString()	Returns this card name as a string. If the color is not set, return **"DRAW FOUR".** If the color is set, return **"DRAW FOUR (COLOR color selected)"**. For example, **DRAW FOUR (RED color selected)**. *Note: You can use the toString() method of CardColor and CardSymbol to get the color and symbol string.*
+ boolean canPlay()	Returns whether or not this card can be played, which is always true.
+ String performEffect()	Performs this card's effect. 1. Set this card's color. • Set the color to the color of the first card in the player's hand (the first card is counted after removing this card). • If that first card has no color i.e. it's a Change Color Card or Draw Four Card, or there are no cards left in the player's hand, set the color to **Red**. • Then, print **"Set color to C"** when C is the set color. 2. Increment *draw amount* by 4. 3. Go to the next player <u>whose hand is not empty</u>. 4. The player automatically takes actions depending on the state of their hand. • If the player has <u>a Draw Four Card</u>, select the first one in hand. Print **"Player N played CARD NAME. X cards remaining."**, when N is the current player, CARD NAME is the card name, and X is the number of cards that will be left in the current player's hand after playing. Then play the selected card and concatenate that card's message to this card's message (display on separate lines). • If the player doesn't have either card, draw *draw amount* number of cards. Print **"Player N drew Y cards. X cards remaining."**, when N is the current player, Y is the draw amount, and X is the number of cards that will be left in the current player's hand after drawing. Then reset the draw amount to 0. 4. Return the message. *Note: You can use the methods* *GameLogic.getInstance().goToNextPlayer(),* *GameLogic.getInstance().incrementDrawAmount()* *GameLogic.getInstance().draw()* *somehow.*

2.2 package logic.game

2.2.1 Class GameLogic /*This class is mostly given*/

Fields

- final int initialPlayerCount	The number of players in the game.
- final ArrayList<BaseCard> deck	The deck of cards to draw from.
- final ArrayList<ArrayList<BaseCard>> playerHands	The hands of all players. Player 0's hand is at index 0, player 1's hand is at index 1, etc.
- int currentPlayer	Denotes the current player as of that turn.
- PlayDirection playDirection	The current play direction. Set to FORWARD by default.
- int drawAmount	The current cumulative draw amount, used for stacking Draw Two and Draw Four cards. Set to 0 by default.
- BaseCard topCard	The current top card of pile. Used to match with the player's cards. Set to null by default.
- GameLogic instance	The current game instance.

This class contains the logic of the game system, including player's hands, game initialization, game over logic and global parameters. **You may use these methods in your work, so the explanations of useful fields and methods in this class are provided here. You must also implement 3 methods in this class, specified below.**

Methods to Implement

+ boolean isHandPlayable(int playerIndex)	Returns whether the player's hand specified by *playerIndex* is playable. A hand is playable if at least one card can be played, otherwise the hand is not playable.
+ ArrayList<BaseCard> draw(int amount)	Returns an ArrayList of size *amount* containing the top *amount* cards from the deck. All of those cards are removed from the deck and added to the current player's hand. If amount is more than the number of cards in the deck, return just the remaining cards.

Useful Methods Provided

+ <u>GameLogic getInstance()</u>	Used to get the game's instance. There is only one instance per game.
+ int getCurrentPlayer()	Returns the current player.
+ ArrayList<BaseCard> getCurrentPlayerHand()	Returns the hand of the current player.
+ public void goToNextPlayer()	Set the current player to the next player, depending on the current play direction.
+ PlayDirection getPlayDirection() + void setPlayDirection(PlayDirection playDirection) + int getDrawAmount() + void setDrawAmount(int drawAmount) + BaseCard getTopCard() + void setTopCard(BaseCard topCard)	Getter and setter methods for *playDirection*, *drawAmount*, and *topCard*.
+ void incrementDrawAmount(int amount)	Increases the value of *drawAmount* by *amount*.

2.2.2 Enum CardColor

This enum contains values of the possible card colors which are red, yellow, green, and blue. This enum also contains a method to get a random color. **You do not need to modify this class.**

2.2.3 Enum CardSymbol

This enum contains values of the possible card symbols which are 0-9, Skip, Reverse, Change Color, Draw Two, and Draw Four. This enum also contains a method to get a random symbol and a method to get only the numbers. **You do not need to modify this class.**

2.2.4 Enum PlayDirection

This enum contains values of the possible play directions which are forward and backward. This enum also contains a method to get the opposite direction (forward and backward are defined as opposite). **You do not need to modify this class.**

2.3 package test.student

2.3.1 Class NumberCardTest

This class tests NumberCard class from *logic.card* package. All constructors and test cases have already been implemented. You don't have to implement any test case in this class, but you can use it as a reference.

2.3.2 Class ReverseCardTest

This class tests ReverseCard class from *logic.card* package. All constructors and some test cases has already been implemented, **except for 2 test cases which student needs to implement.**

Methods (Test cases to implement)

void testConstructor()	Check all fields of each Reverse Card.
void testCanPlay()	The card to be played is a reverse card. Check if ReverseCard's canPlay() returns the correct value for the following cases: • The color matches the top card. • The symbol matches the top card. • Both the color and symbol matche the top card. • Both the color and symbol do not match the top card.

2.3.3 Class SkipCardTest

This class tests SkipCard class from *logic.card* package. All constructors and some test cases has already been implemented, **except for 2 test cases which student needs to implement.**

Methods (Test cases to implement)

void testConstructor()	Check all fields of each Skip Card.
void testCanPlay()	The card to be played is a skip card. Check if SkipCard's canPlay() returns the correct value for the following cases: • The color matches the top card • The symbol matches the top card • Both the color and symbol matches the top card • Both the color and symbol do not match the top card

2.3.4 Class ChangeColorCardTest

This class tests ChangeColorCard class from *logic.card* package. All constructors and some test cases has already been implemented, **except for 2 test cases which student needs to implement.**

Methods (Test cases to implement)

void testPerformEffectHasColor()	Test the case where the new color can be determined by the first card in the player's hand.
void testPerformEffectNoColor()	Test the case where the first card in the player's hand has no color.
void testPerformEffectNoCard()	Test the case where the player's hand has no cards left.

2.3.5 Class DrawTwoCardTest

This class tests DrawTwoCard class from *logic.card* package. All constructors and some test cases has already been implemented, **except for 1 test case which student needs to implement.**

Methods (Test cases to implement)

void testPlay()	Test the case where there are 4 players. Player 0 plays DrawTwo, and the following players are forced to play DrawTwo, DrawTwo, and DrawFour respectively, resulting in player 0 having to draw 10 cards.

2.3.6 Class DrawFourCardTest

This class tests DrawFourCard class from *logic.card* package. All constructors and some test cases has already been implemented, **except for 1 test case which student needs to implement.**

Methods (Test cases to implement)

void testPlay()	Test the case where there are 4 players. Player 0 plays DrawFour, and the following players are forced to all play DrawFour, resulting in player 0 having to draw 16 cards.

2.3.7 Class GameLogicTest

This class tests GameLogic class from *logic.card* package. **You need to implement all the test cases for the methods you implemented.**

Methods (Test cases to implement)

void testIsHandPlayableTrue()	Test 2 different cases of when a player's hand is playable.
void testIsHandPlayableFalse()	Test when a player's hand is not playable. One case is enough.
void testDrawLessThanDeck()	Test when the draw amount is less than the size of the deck.
void testDrawMoreThanDeck()	Test when the draw amount is more than the size of the deck.

3. Score Criteria (will be scaled down to the total of 2.5)

Implement BaseCard correctly	3 points
NumberCardTest	**3 points**
SkipCardTest	**5 points**
ChangeColorCardTest	**6 points**
ReverseCardTest	**4 points**
DrawTwoCardTest	**7 points**
DrawFourCardTest	**6 points**
GameLogicTest	**4 points**
UML	**3 points**
Write Junit	**9 points**