ACS-2947-002

Assignment 1

Due by Friday, February 11, 11:59 pm

Instructions

- Submit your `.java` files (together in a `Assign1.zip` file) via Nexus.
- Include your name and student number as a comment in every file.
    - Document the classes using Javadoc notation.
    - Include comments as needed.
    - Use appropriate exception handling where necessary.

PART A: LinkedLists (55 marks)

The classic card game Uno can be seen as a simple turn-based multiplayer game, where each player takes a turn playing a card from their hand. The first player to discard all their cards from their hand wins the round.

Create a program that simulates the modified game of Uno as described below. *Note that the rules of the game are simplified for this assignment.*

Cards:

Colour	**Number or Symbol**	**Quantity**
Yellow, Green, Blue, red	Numbers 0 – 9	2 of each colour, for each number
	Skip	2 of each colour
	Reverse	2 of each colour
Black	Wild	4
		Total: 100 Cards

Players are dealt 7 cards each. The first card is drawn (played) from the deck. The first player selects a card to play from their hand if it matches the colour, number, or symbol on the card that was last played. The next player then takes their turn, and so on. If the player has no card to play, they draw a card from the deck. The card is played if it is playable, otherwise it is added to the cards in the players hand and the next player gets their turn. Player only play Wild card if they have no other card to play.

Special cards:

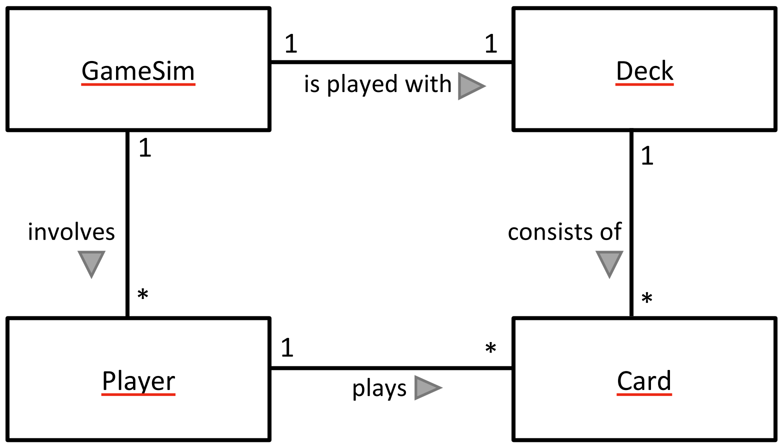
- Skip: the next player misses a turn.
- Reverse: the game reverses direction.
- Wild: the player chooses the colour to play next.

When a player has one card left in their hand, they must yell “Uno”. The first player to get rid of all their cards wins and the game ends. In the unlikely event that a player is unable draw a new card from the deck because there are no more cards, the game ends with no winner.

1. Program the generic `CircularDoublyLinkedList`
   - Use the circularly and doubly linked list classes from our notes as a basis.
   - Start with the DLL and think of how it can work circularly.
     - Sentinels are not required, only reference to the last node.
     - `Node` will have references to the next and previous nodes.
     - `addBetween` and `remove` private utilities will be used, all adding and removing will work like doubly.
   - Your CircularlyDoubleLinkedList class will support the following public methods:

`size()`	Returns the number of elements in the list.
`isEmpty()`	Returns true if the list is empty, and false otherwise.
`first()`	Returns (but does not remove) the first element in the list.
`last()`	Returns (but does not remove) the last element in the list.
`addFirst(e)`	Adds a new element to the front of the list.
`addLast(e)`	Adds a new element to the end of the list.
`removeFirst()`	Removes and returns the first element of the list.
`removeLast()`	Removes and returns the last element of the list.
`rotate()`	Advances to the next element in the list based on direction
`reverse()`	Changes the direction in which rotation which occur

2. Create the following classes:



a) `Card` contains 2 enum types:
   - `Colour`: BLUE, RED, GREEN, YELLOW, BLACK
   - `Face`: numbers ZERO to NINE, symbols SKIP, REVERSE, WILD
   - Implement a `playable()` method that checks if a card is playable i.e., it matches the last card played in colour, number, or symbol.

b) `Deck` stores an `ArrayList` (from Java API) of Cards
   - Upon initialization a deck of cards should be built according to the chart above.

- Include a `shuffle()` method that shuffles the cards
- Include a deal method that removes a card from the top (front) of the deck and return the Card instance.

c) `Player` stores the player's name and an `ArrayList` (from Java API) of Cards as their hand. Include a method to play a card (removes and returns a given card from the hand).

d) `GameSim` class that simulates a game among a number of players.
Note: *most of the work will be done here.* `Assign1PartA_Driver` *should have only minimal amount of code: e.g., declare/create a Game instance, add players, initiate game play.*
- Fields
    - a `CircularDoublyLinkedList` of players
    - a `Deck` of cards
    - a `Card` that stores the card last played (i.e., represents the top card of the discard pile)
    - any other field required to manage the game
- Upon instantiation, set up the list of players (use 4 players for your demo/sample data), create the deck, shuffle, and deal 7 cards to each player. Set the first card and begin the game with the first player in the list. Each player will have their turn when positioned at the head of the list.

3. Write a driver class called `Assign1PartA_Driver` that creates a list of four players of your choice and invokes a simulation of the game.

Suggestions:

- Have a method for the player's turn that determines which card the player will play
    - Use a simple algorithm that finds a playable card i.e., check each card and play the first one that matches either colour or face, or check if there's a matching colour first, then check for matching face
- Use a different method that manages the game
    - E.g., advancing to next player

Notes:

- No need to keep a discard pile (list)
- If the first card drawn when game begins is wild, simply draw another card
- Include any other methods as you see necessary (for code reuse, modularity, etc.)
- To format output, include a String field `display` and override the enum's toString().
- For special symbols, you may also use \u21c4 for ⇄ (reverse) and \u2205 for ∅ (skip)

Sample output:

```
Total Cards in Deck: 100

Let's play UNO!!!

First card: Red Skip

Nala misses a turn

(Simba's hand: Red 2, Yellow Reverse, Yellow 7, Red 6, Yellow 9, Red 9, Blue
2)
Simba plays Red 2

(Kion's hand: Green 0, Red 8, Red 3, Wild *, Blue 9, Green 6, Red Skip)
Kion plays Red 8

(Kiara's hand: Red 8, Wild *, Green 6, Green 9, Red 5, Green 5, Blue Skip)
Kiara plays Red 8

(Nala's hand: Blue 8, Blue 7, Blue 9, Red 4, Red 2, Red 1, Green Reverse)
Nala plays Blue 8

(Simba's hand: Yellow Reverse, Yellow 7, Red 6, Yellow 9, Red 9, Blue 2)
Simba plays Blue 2

(Kion's hand: Green 0, Red 3, Wild *, Blue 9, Green 6, Red Skip)
Kion plays Blue 9

(Kiara's hand: Wild *, Green 6, Green 9, Red 5, Green 5, Blue Skip)
Kiara plays Green 9

(Nala's hand: Blue 7, Blue 9, Red 4, Red 2, Red 1, Green Reverse)
Nala plays Blue 9

(Simba's hand: Yellow Reverse, Yellow 7, Red 6, Yellow 9, Red 9)
Simba plays Yellow 9

(Kion's hand: Green 0, Red 3, Wild *, Green 6, Red Skip)
Kion plays Wild *
Color is now Blue

(Kiara's hand: Wild *, Green 6, Red 5, Green 5, Blue Skip)
Kiara plays Blue Skip

Nala misses a turn

(Simba's hand: Yellow Reverse, Yellow 7, Red 6, Red 9)
Simba has no play, draws Yellow 0
Simba can't play it
```

```
(Kion's hand: Green 0, Red 3, Green 6, Red Skip)
Kion plays Red Skip

(Kiara's hand: Wild *, Green 6, Red 5, Green 5)
Kiara plays Red 5

(Nala's hand: Blue 7, Red 4, Red 2, Red 1, Green Reverse)
Nala plays Red 4

(Simba's hand: Yellow Reverse, Yellow 7, Red 6, Red 9, Yellow 0)
Simba plays Red 6

(Kion's hand: Green 0, Red 3, Green 6)
Kion plays Red 3

(Kiara's hand: Wild *, Green 6, Green 5)
Kiara plays Wild *
Color is now Green

(Nala's hand: Blue 7, Red 2, Red 1, Green Reverse)
Nala plays Green Reverse
Game reverses direction

(Kiara's hand: Green 6, Green 5)
Kiara plays Green 6
Kiara yells "UNO"!

(Kion's hand: Green 0, Green 6)
Kion plays Green 0
Kion yells "UNO"!

(Simba's hand: Yellow Reverse, Yellow 7, Red 9, Yellow 0)
Simba plays Yellow 0

(Nala's hand: Blue 7, Red 2, Red 1)
Nala has no play, draws Blue 1
Nala can't play it

(Kiara's hand: Green 5)
Kiara has no play, draws Yellow 2
Kiara plays Yellow 2
Kiara yells "UNO"!

(Kion's hand: Green 6)
Kion has no play, draws Yellow 3
Kion plays Yellow 3
Kion yells "UNO"!

(Simba's hand: Yellow Reverse, Yellow 7, Red 9)
```

```
Simba plays Yellow Reverse
Game reverses direction

(Kion's hand: Green 6)
Kion has no play, draws Blue 2
Kion can't play it

(Kiara's hand: Green 5)
Kiara has no play, draws Yellow 1
Kiara plays Yellow 1
Kiara yells "UNO"!

(Nala's hand: Blue 7, Red 2, Red 1, Blue 1)
Nala plays Red 1

(Simba's hand: Yellow 7, Red 9)
Simba plays Red 9
Simba yells "UNO"!

(Kion's hand: Green 6, Blue 2)
Kion has no play, draws Yellow 9
Kion plays Yellow 9

(Kiara's hand: Green 5)
Kiara has no play, draws Yellow 6
Kiara plays Yellow 6
Kiara yells "UNO"!

(Nala's hand: Blue 7, Red 2, Blue 1)
Nala has no play, draws Blue 0
Nala can't play it

(Simba's hand: Yellow 7)
Simba plays Yellow 7
Simba wins!

Total Cards in Deck: 62
```

PART B: Stacks (25 marks)

Create a program that simulates the undo/redo features of an application. Implement a simple calculator that asks the user for the basic arithmetic operation that they would like to perform on the *last result*.

Your program will:

- Prompt the user to enter an initial number (first operand)
- Then
    - prompt the user for the next operator and second operand
    - evaluate the expression
    - present the user with the result, which will be the new first operand
- This will continue until the user chooses to quit, or undo (redo) an operation

The undo operation restores the last state. The redo operation restores the next state if an undo was previously performed.

1. Create the generic `ArrayStack` that implements the provided `Stack` interface using an array.
2. Create a driver class called `Assign1PartB_Driver` and any other classes/methods that you may require.

You must use two stack objects to hold items that will restore the states based on the function (undo "`z`" or redo "`y`") selected. Allow the user to quit ("`q`") anytime.

Sample output:

```
Simple Calculator: type z to undo, y to redo, q to quit

Enter the first number:
10
Enter the next operation on 10:
+ 2
= 12
Enter the next operation on 12:
- 3
= 9
Enter the next operation on 9:
* 4
= 36
Enter the next operation on 36:
z
UNDO: 9
Enter the next operation on 9:
z
UNDO: 12
Enter the next operation on 12:
y
```

```
REDO: 9
Enter the next operation on 9:
y
REDO: 36
Enter the next operation on 36:
y
Nothing to redo.
Enter the next operation on 36:
* 2
= 72
Enter the next operation on 72:
z
UNDO: 36
Enter the next operation on 36:
z
UNDO: 9
Enter the next operation on 9:
z
UNDO: 12
Enter the next operation on 12:
z
UNDO: 10
Enter the next operation on 10:
z
Nothing to undo.
Enter a number:
3
Enter the next operation on 3:
q
Goodbye!
```

Suggestions:

- If you need to understand the use of undo and redo a bit more, type a few characters in a text editor and execute a series of undo/redo operation to see its effect.
- Check that your implementation work for the above sample output.

Submission
Submit your **`Assign1.zip`** file that include all the assignment files (`CircularDoublyLinkedList.java, Deck.java, Card.java, GameSim.java, Player.java, Colour.java, Face.java, Assign1PartA_Driver.java, Stack.java, ArrayStack.java, Assign1PartB_Driver.java`) via **Nexus**.