CMPT220 – Program 10 – Pokemon Battle!

Program Due: Monday, May 4th, before noon. (saved, submitted and printed)

Absolutely no late programs will be accepted!

Name the project **Prog10YourLastName**

Name the main class **BattleDemoYourLastName.java**

Name the Card class **PokemonCardYourLastName.java**

Name the Node class **NodeYourLastName** (if you have one)

Name the Stack class **StackYourLastName.java**

Be sure to read these specifications ***CAREFULLY*** so that your program works **EXACTLY** as expected on a given set of data, because that's how it will be tested. (In the description, I'll use "**Stack**" instead of "StackYourLastName" and "**Card**" instead of "PokemonCardYourLastName".

This programming assignment requires the use of the **`Stack`** abstract data type. You may choose **either** the array **or** the linked-list implementation.

The Stack will contain objects of class **`Card`**. A **`Card`** object contains two attributes: an integer representing the powerLevel of the Pokemon and a String representing the Pokemon's name.

The Pokemon Battle Game

These are the rules for our updated Pokemon Battle Game:

Take a deck of **up to** 52 cards and deal them to 2 players. To deal means to give player1 the first card, player2 the next, player1 the next, and so on until there are no more cards in the deck.

Each player will maintain two stacks of cards: a *play stack* and a *discard stack*. Initially all cards dealt to each player are in that person's play stack while his or her discard stack is initially empty.

Proceed as follows until either player runs out of cards, or the game takes too long:

(1) Each player takes ("pops") the top card from her play stack and puts it down. If a player's play stack is empty, but her discard stack is not, copy the contents of the discard stack into the player's play stack but do so in a way that preserves the same bottom-to-top order of the cards. The result of the copy from discard into play is that the play stack now contains exactly what the discard stack once did and the discard stack is empty.

(2) If the two cards have DIFFERENT powerLevels, then the player with the card of higher powerLevel wins and she places ("pushes") both cards on her discard stack **in the following order**: higher card **first**, then lower card.

(3) Else if the two cards have the SAME powerLevel, then a "battle" occurs: Since the cold war has ended, and the world doesn't need any more battles, and I want to keep this program "do-able", we have "nice battles", which means each player puts ("pushes") her own card onto her own discard stack.

We will assume that a game "takes too long" if there are more than 1000 plays.

Ending the Game

If only one player has cards remaining, then that player is the winner. Otherwise, if the play limit is reached, then the player with the most cards is declared the winner. If both players ended up with the same number of cards, then the game ends in a tie.

Output
The program should play the game to its conclusion and then print out the following:

```
The game started with __ cards.
There were __ plays in the game.
The game ended with a clear winner. or   The game took too long.
Player 1 ended up with ___ cards.
Player 2 ended up with ___ cards.
The winner was __.  (Player1 or Player2 or No one)
```

Your program will begin by prompting the user to enter the name of an input file that contains a **pre-shuffled** deck of cards. Write a method that reads in a pre-shuffled deck of cards from an input file. A sample input file is formatted as integer/name pairs (one value per line) with end-of-file signaling the end of the input. Assume that each integer and String appears on a line by itself. You may assume that the input file will contain at most 52 cards and will be properly formatted.

```
2
Pikachu
10
Eevee
4
Snorlax
5
Charizard
9
Celebi
…
12
Ditto
```

As you read-in each card you will deal them out alternatively to each player. So the 2Pikachu goes into the "Play" stack for player1, the 10Eevee goes into the "Play" stack for player2, 4Snorlax to player1, and so on. As a card is dealt to a player you will push that card onto that player's play stack. Both players begin with empty "Discard" stacks.

Hint
Try testing your program with small input files that represent a shuffled subset of a deck of 52 cards. This will allow you to trace out the contents of the stacks to verify that the program is behaving correctly.

Sample Program Run

Suppose the input file contains the following cards: 3H 2H 4D 5S 6C 7S. (I'm using single-letter names for brevity for this example. You should, of course, expect longer names in the actual data.) Then at the beginning the cards are dealt and the players have:

	Play	Discard		Play	Discard
player 1 :	6C	---	player 2 :	7S	---
	4D			5S	
	3H			2H	

Play begins: player1 plays a 6, player2 plays a 7. Player 2 wins taking both cards (6, 7) and adding them into the discard stack. Put in the higher valued card first.

	Play	Discard		Play	Discard
player 1 :	4D	---	player 2 :	5S	6C
	3H			2H	7S

Then player1 plays a 4, and player2 plays a 5. Player 2 wins.

	Play	Discard		Play	Discard
player 1 :	3H	---	player 2 :	2H	4D
					5S
					6C
					7S

Then player1 plays a 3, and player2 plays a 2. Player 1 wins.

	Play	Discard		Play	Discard
player 1 :	---	2H	player 2 :	---	4D
		3H			5S
					6C
					7S

It just so happens that each player is out of cards. So each player copies the discard stack to her play stack. Many times, only one player needs to do this. Notice that the stack REMAINS IN THE SAME ORDER after it is copied!

	Play	Discard		Play	Discard
player 1 :	2H	---	player 2 :	4D	---
	3H			5S	
				6C	
				7S	

Then player1 plays a 2, and player2 plays a 4, player 2 wins & takes the cards.

Then player1 plays a 3, and player2 plays a 5, player 2 wins & takes the cards. Player 1 is out of cards (both her play stack and her discard stack are empty). So player2 is the winner.

When the program is run, we would get the following game summary output. (As you are developing your program, you might want it to print out details of each play, so that you can determine whether the game is progressing correctly.) **Note that it is NOT necessary to have the program print a turn-by-turn trace as shown above.** Simply print out the game summary.

```
Battle Card Game Summary
========================
The game started with 6 cards.
There were 5 plays in the game.
The game ended with a clear winner.
Player 1 ended up with 0 cards.
Player 2 ended up with 6 cards.
The winner was Player 2.
```

You need to develop good algorithms for this program and do good top-down design. It should be obvious that you will need to write many methods for your **`BattleGame`** class. A set of methods (which may or may not need parameters) includes, but is not limited to, the following.

deal(): read the cards from the input file and deal the cards to the players "Play" stacks
play(): get a card from each player
compare(): compare the two cards and determine the winner
winPlay(): give the cards to the winner
copy(): copy a player's discard stack into her play stack
countCards(): count the number of cards in a single stack
printResults()

Submit .PDFs instead of printouts, of course. Also, be sure to submit your input file(s) – the easiest way to do this is to copy your input file(s) into the project directory before you .zip it.