The Card Game

Programming Fundamentals 2

Goals

- ✯ Using a simple programming workflow: code, compile and run, commit, push.
- ✯ Basics of imperative Java: variables, control structures (conditional and loop), arrays.
- ✯ Simple object creation and manipulation: constructors, methods, attributes.

Deliverables

1. The code on your Github repository generated by clicking here: `https://classroom.github.com/a/SUtCnFRd`
2. **Reviewer**: Léo Gamboa dos Santos (LeoLuxo on Github).
3. **Documentation**: Comment the methods you write using JavaDoc.

Exercise 1 – Using a build automation tool (Maven)

For this laboratory, you compile and execute your code with *Maven*. Install Maven with `sudo apt install maven` on Windows or Ubuntu in the terminal, and with `brew install maven` on OSX. We already provide the file `pom.xml` at the root of the project, which describe how the code is compiled and executed.

- To compile, write `mvn compile` at the root of your project.
- To run the program, write `mvn exec:java -Dexec.mainClass="cardgame.CardGame"`.

Exercise 2 – Pre-requisites

For this exercise, you will need to use both *arrays* and *ArrayLists*. As such, it is recommended you complete the corresponding data structures lab before continuing onto this one.

Here is a small refresher on the usage of both:

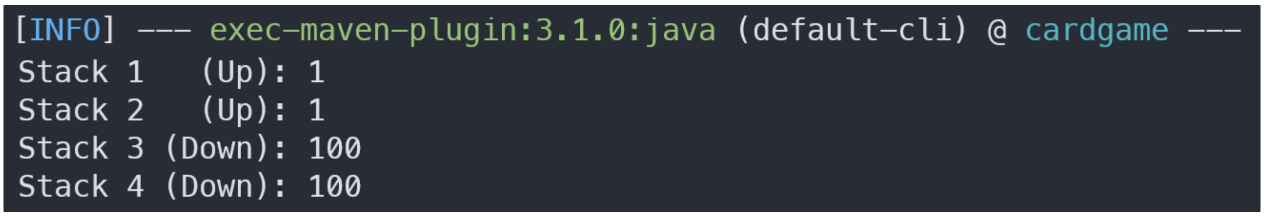
```java
int[] myArray = new int[3];
myArray[0] = 42;
System.out.println(myArray[0]);      // prints 42
System.out.println(myArray.length); // prints 3

ArrayList<Integer> myList = new ArrayList<Integer>();
myList.add(42);
myList.add(10);
System.out.println(myList.get(0));  // prints 42
System.out.println(myList.size());  // prints 2 (because we only added 2 elements).
myList.remove(0); // This removes the element at index 0, so it removes 42
// You can also remove a specific element from the list:
myList.remove(new Integer(10)); // needs to use 'Integer' to distinguish with previous method call.
```

Don't forget to add `import java.util.ArrayList` at the beginning of the file.

Exercise 3 – Game development: The Card Game

The goal of this exercise is to develop a command line version of *The Game*. You can read the rules you are basing yourself on here: `https://middys.nsv.de/wp-content/uploads/2018/01/the-game-english.pdf`. You will program a single player version.

As a beginner, it is often difficult to know how to start coding, because the task seems overwhelming and complicated. The key is to simplify the task by breaking it into numerous smaller and easier-to-code components. For sure, you do not feel confident to implement the whole game, but what about a program that creates an array representing the four stacks and prints them?



```
[INFO] --- exec-maven-plugin:3.1.0:java (default-cli) @ cardgame ---
Stack 1   (Up): 1
Stack 2   (Up): 1
Stack 3 (Down): 100
Stack 4 (Down): 100
```

Figure 1: A simple program that prints the 4 stacks.

Do you find this task even too complicated? Then you can break it down into a smaller set of tasks again, for instance by only printing a single stack, then two, etc.

Only beginners code without intermediate steps. When you start coding, you must think about an easy and reachable task, that you think you can complete in 20 minutes. Your coding workflow will be to code this task, compile and run the program, possibly debug what you coded, and then commit to git the changes. A task you think you can complete in 20 minutes will probably take longer, so don't be too eager.

In this laboratory, the decomposition into small tasks is done for you. This is to show you how to develop a larger application by small successive steps. For the next laboratories, you will need to do this decomposition yourself.

1. The gameplay is simple and is illustrated in Figure 2. The name of the main class must be `CardGame`. You will have other files with other classes.

2. Before reading the questions that follows, it would be extremely valuable to figure out yourself how you could code this game with very small steps. That is, put yourself in the shoes of a manager who needs to decompose a project into small tasks, so that he can verify the project is progressing well at each step. Once you have done it, read the next questions and compare with what you had in mind. Did you under- or overestimate the duration of each task? We tend to think that some tasks take less time than what it will actually take.

3. We decompose the game into a series of intermediate results as follows (that we overly describe to help you!):

   (a) Add a welcome message in the main function of the file `CardGame.java`.

   (b) Create the file `Table.java` with an array attribute `private int[] sizeOfStack;`, a constructor, and a `public toString()` method. Create an object `Table` from the main, and print it. Here is how your main function in `CardGame` should look:

   ```
   public static void main(String[] args) {
     Table t = new Table();
     System.out.println(t.toString());
     // Note that 'toString' is called automatically in a 'println' so you can simply write:
     System.out.println(t);
   }
   ```

   (c) In `CardGame.java`, ask the player for the index/ID of the stack on which they want to place a card on. You can already check if the user input a valid stack ID, and if not, ask them again.

   (d) After that, also ask the player for the card they would like to place on the pile they put in previously.

   (e) Add a method

```
public boolean playCard(int stackID, int card)
```

to the `Table` class. It should place the card on the given stack, only if it is a valid move (i.e. the card value is smaller/bigger than the value of the corresponding stack, or it is exactly 10 over/under the value). The method should return whether the move was valid or not.

(f) Combine the `playCard` method and the user inputs from earlier. If the move was not valid, make sure to ask the user again for valid inputs.

(g) Add an attribute `private ArrayList<Integer> hand` in `Table.java`. It should be initialized in the constructor, and you can temporarily add certain cards to it (directly in the constructor) for testing purposes. The contents of the player's hand should be output in the `toString` method.

Also update the `playCard` method so it makes sure that the given `card` is in the player's hand, and removes it from there when adding it to a stack.

(h) Put the input prompts into a loop to ask the user for multiple moves sequentially. A '*move*' constitutes placing a card from the hand to a stack.

(i) Add another attribute `private ArrayList<Integer> drawPile` to the `Table` class. It should be initialized in the constructor and contain the numbers from 2 to 99 included. The size of the draw pile should be put out in the `toString` method.

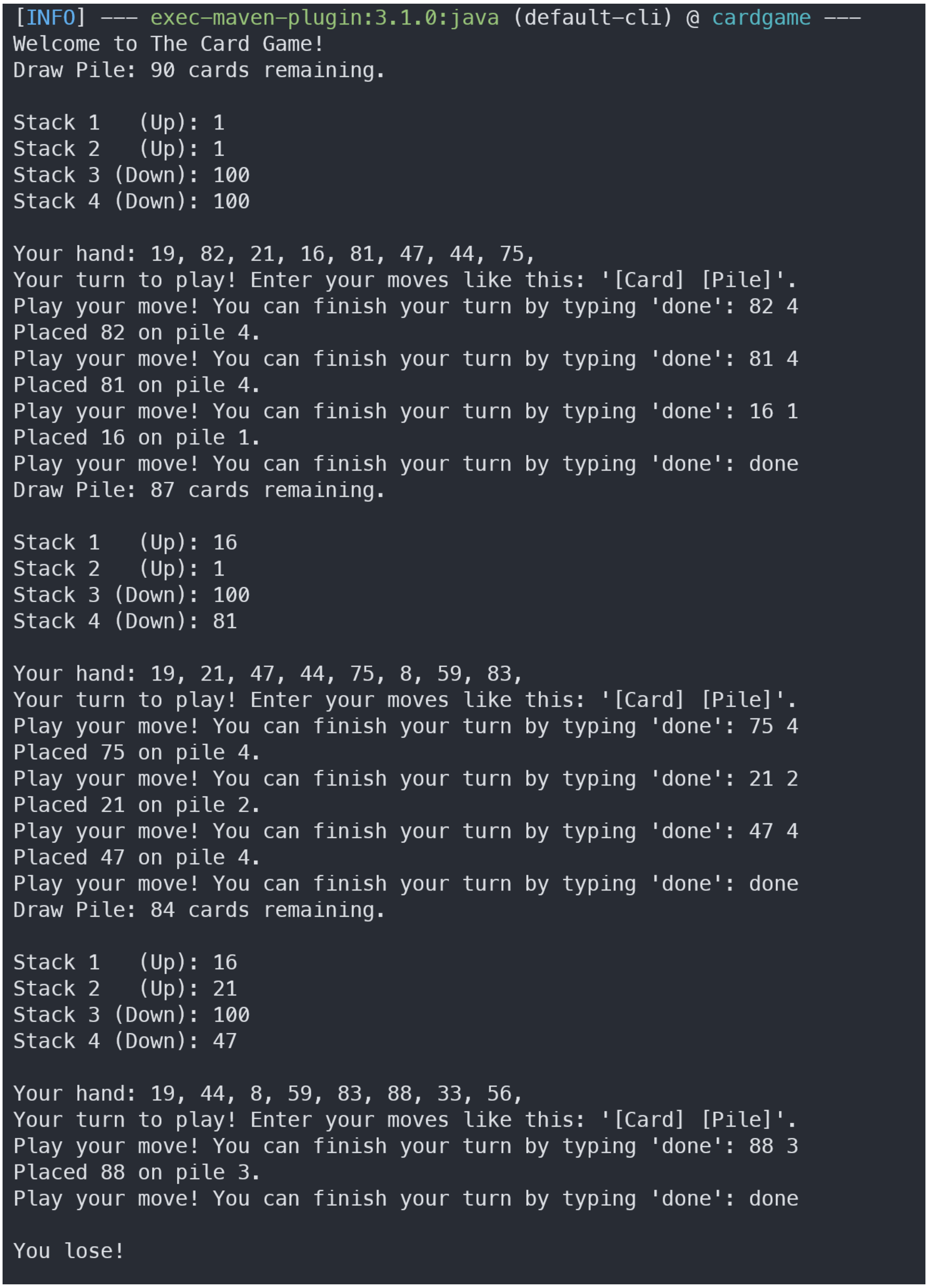
(j) Add a `void dealCards()` method as well, that takes random cards from the draw pile and adds them to the player's hand until it is filled with maximum 8 cards. To clarify, the player should still keep their old cards. To get a random number in the range $[0; e[$, use the following method:

```
ThreadLocalRandom.current().nextInt(e)
```

(k) Now wrap the move logic from earlier into another loop, to write the logic for turns. A '*turn*' is made of multiple *moves*. At the beginning of each turn, the player is dealt new cards, and then can play as many moves as they choose. If they enter `"done"` instead of a valid move, they end their turn.

(l) We must finally take care of the winning conditions. If at the end of the turn, the player has fully emptied their hand and the draw pile, they win. If they have played less than 2 moves in a turn, they lose. Otherwise, the game continues.

4. Is it easy to change the maximum number of cards in the player's hand in your code? And the range of values for the cards of the draw pile and the stacks? If not, then it should.



```
[INFO] --- exec-maven-plugin:3.1.0:java (default-cli) @ cardgame ---
Welcome to The Card Game!
Draw Pile: 90 cards remaining.

Stack 1   (Up): 1
Stack 2   (Up): 1
Stack 3 (Down): 100
Stack 4 (Down): 100

Your hand: 19, 82, 21, 16, 81, 47, 44, 75,
Your turn to play! Enter your moves like this: '[Card] [Pile]'.
Play your move! You can finish your turn by typing 'done': 82 4
Placed 82 on pile 4.
Play your move! You can finish your turn by typing 'done': 81 4
Placed 81 on pile 4.
Play your move! You can finish your turn by typing 'done': 16 1
Placed 16 on pile 1.
Play your move! You can finish your turn by typing 'done': done
Draw Pile: 87 cards remaining.

Stack 1   (Up): 16
Stack 2   (Up): 1
Stack 3 (Down): 100
Stack 4 (Down): 81

Your hand: 19, 21, 47, 44, 75, 8, 59, 83,
Your turn to play! Enter your moves like this: '[Card] [Pile]'.
Play your move! You can finish your turn by typing 'done': 75 4
Placed 75 on pile 4.
Play your move! You can finish your turn by typing 'done': 21 2
Placed 21 on pile 2.
Play your move! You can finish your turn by typing 'done': 47 4
Placed 47 on pile 4.
Play your move! You can finish your turn by typing 'done': done
Draw Pile: 84 cards remaining.

Stack 1   (Up): 16
Stack 2   (Up): 21
Stack 3 (Down): 100
Stack 4 (Down): 47

Your hand: 19, 44, 8, 59, 83, 88, 33, 56,
Your turn to play! Enter your moves like this: '[Card] [Pile]'.
Play your move! You can finish your turn by typing 'done': 88 3
Placed 88 on pile 3.
Play your move! You can finish your turn by typing 'done': done

You lose!
```

Figure 2: Gameplay of *The Card Game*