ITI 1121. Introduction to Computing II
Winter 2019

Assignment 1
(Last modified on February 1, 2019)

Deadline: February 3, 2019, 11:30 pm

[PDF]

Learning objectives

- Edit, compile and run Java programs
- Utilize arrays to store information
- Apply basic object oriented programming concepts
- Understand the university policies for academic integrity

Introduction

For this assignment, we remake the Fall 2018 ITI 1120/1520 assignment 3 using Java instead of Python. The goal is to learn the syntax and semantics of Java. The original assignment, along with a proposed solution given in Python, can be downloaded from here:

- `https://www.eecs.uottawa.ca/~turcotte/teaching/iti-1121/assignments/01/A3F2018.zip`

Every programming language has its own syntax and semantics, which one needs to learn in order to use the language effectively. In part 2, we will adapt the game of Rummy to better use object-oriented programming concepts.

1 Arrays

In Java, an **array** is a data structure to hold a **fixed** number of elements all of the **same type**. The length of the array is specified when the array is created and it cannot be changed after creation. Each element of the array is accessed using its **index**. The index of the first element is 0. Since the elements are stored in contiguous locations in memory, the access to any element is efficient — we say "constant time". Here is a Java tutorial on arrays:

- `https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html`

1.1 countPositive (5 marks)

In Java, implement the class method **countPositive** that takes as input the reference of an array of numbers (all of type **double**) and returns the number of elements of the designated array that are positive (i.e. > 0). You will store this method in a class called **A1Q1**. You can test your code using the program **A1Q1Test**. This method does not read information from the user. It obtains the necessary information through its parameter. The same is true for all the methods in this part of the assignment.

- 01/Utils.java
- 01/A1Q1Test.java

1.2 hasLengthTwoRun (5 marks)

In Java, implement a class method called **hasLengthTwoRun** that takes as input the reference of an array of numbers (all of type **double**) and returns **true** if the given array has at least one run (of length at least two) of repeated and consecutive values, and **false** otherwise. Make sure that the method is efficient (i.e. it stops as soon as the answer is known). You will store this method in a class called **A1Q2**. You can test your code using the program **A1Q2Test**.

- 01/Utils.java
- 01/A1Q2Test.java

1.3 getLongestRun (10 marks)

As mentioned above, a run is a sequence of consecutive repeated values. In Java, implement two class methods, both called **getLongestRun**, taking as input the reference of an array and returning the length of the longest run. The first method takes as input the reference of an array of numbers (all of type **double**), whereas the second method takes as input the reference of an array of strings (objects of the class **String**). Store these two methods in a class called **A1Q3**. You can test your code using the program **A1Q3Test**.

- 01/Utils.java
- 01/A1Q3Test.java

2 Game of Rummy

We revisit the single player game Rummy introduced in ITI 1120/1520. We use the same set of rules, but we take this opportunity to apply the object-oriented programming concepts that we have just learnt.

In the game of Rummy, the main goal is to build **melds**, which consists of sets: two, three or four **of a kind** of the same rank; or **progression**, three or more cards in a sequence of consecutive ranks, of the same suit. So the set ♡10, ♢10, ♣10 forms three of a kind. And the set/sequence ♢7, ♢8, ♢9, ♢10, ♢11 forms a progression. The goal of the player is to get rid of all of the cards in as few rounds of the game as possible.

One of the first changes that we make will be to create a data type (a class) to represent the cards. Just like in ITI 1120/1520, the number of suits is fixed (4), but the number of ranks can be any value in the range from 3 to 99.

Next, we also create a data type to represent a deck of cards. Using arrays to store the cards would require you to write a large amount of code since the arrays have a fixed size and support a limited number of operations. Since you are already familiar with the concept of lists, the class **Deck** will use an object of the class **ArrayList** to store cards. The fact that one can use a class without knowing the details of its implementation is one of the greatest strengths of object-oriented programming. In the second part of the semester, we will learn about the implementation of lists.

Our implementation of the game consists of 6 classes: **Die**, **Card**, **Deck**, **Game**, **Run**, and **Utils**. In the following sections, we describe the requirements for each class.

2.1 Die (10 marks)

In our game, each turn begins with the roll of die. In order to support this element of the game, you must create the class **Die**.

- A die stores an integer value in the range 1 to 6, inclusively.
- The class **Die** declares a single constructor, which has no parameters. The constructor initializes the value of this die with a random number in the appropriate range.
- The accessor, **getValue**, returns the current value of the die.
- The method **roll** randomly assigns a new value to this die. Hint: you can use an object of the class **java.util.Random** to generate pseudorandom numbers.
- The method **toString** returns a **String** representation of this object. The expected format is shown below.
- Finally, the class **Die** declares a constant called **MAXVALUE** with value 6. Make sure that the visibility is **public** and the variable is **immutable**.

Here is a terse example showing the intended use:

```
1 Die d;
2 d = new Die();
3 System.out.println(d.getValue());
4 d.roll();
5 System.out.println(d.getValue());
6 System.out.println(d);
```

Line 1 of the above program declares a reference variable of type **Die**. Line 2 creates an object of the class **Die** and stores its reference in the reference variable **d**. Line 3 prints the current value of the die. Line 4 calls the method **roll** of the object designated by **d**. Line 5 prints the current value of the die. Finally, Line 6, the method **println** implicitly calls the method **toString** of the object designated by **d** and prints the result. Since the values are randomly generated, each run is likely to produce a different result. Here is the result of a possible run.

```
1
3
Die {value:3}
```

2.2 Card (15 marks)

You must implement the class **Card**.

- An object of the class **Card** stores a suit and a rank, both of type **int**.
- The class declares one constructor. It receives the initial value for the suit and the rank as parameters.
- Each object has two "getters", **getSuit** and **getRank**, which returns the value of the suit and the rank, respectively.
- An object of the class **Card** has a method **equals** with a parameter of type **Object**. If the object designated by the parameter is not of type **Card**, the method returns **false**, as these objects cannot be considered "equals". Otherwise, the method returns **true** if and only if the object designated by parameter is a **Card** that has the same suit and the same rank. Here is a starting point for your implementation:

```
public boolean equals(Object object) {

    if (! (object instanceof Card)) {
        return false;
    }

    Card other;
    other = (Card) object;

    // Complete the implementation...
}
```

- The method **toString** returns a string with the suit and the rank of this card. In order to keep the representation compact when printing an entire deck of cards, no other information than the suit and the rank is used. See below.
- Finally, the class **Card** declares four constants: **DIAMOND**, **CLUB**, **HEART**, and **SPADE**, with values **0**, **1**, **2**, and **3**, respectively.

Here is a small example showing the intended use.

```
Card a, b, c;
a = new Card(2, 7);
b = new Card(2, 7);
c = new Card(0, 9);
System.out.println(a);
System.out.println(a.equals(b));
System.out.println(a.equals(c));
```

```
{2,7}
true
false
```

2.3 Deck (30 marks)

In this application, the programmer must often manipulate a collection of cards: the main deck, the player's hand, the cards that are dealt, and the cards that user wants to discard. Having a specific data structure for a collection of cards will be useful when implementing the logic of the game. Accordingly, you must implement the class **Deck** following these instructions:

- A **Deck** uses an object of the class **java.util.ArrayList** to store the cards of this deck.
- The class declares two constructors. One of them has no parameters. It is used when creating an empty deck. The other constructor has one parameter, of type **int**. The parameter specifies the number of ranks for the cards. It also initializes this deck to contain $4 \times n$ cards, where n is the value of the parameter.
- **int size():** returns the number of cards in this deck.
- **boolean hasCards():** returns **true** if and only if this deck has one or more cards.
- **Card get(int pos):** returns the card at the specified position in the deck.
- **void add(Card card):** adds the specified card at the end of this deck.
- **void addAll(Deck other):** appends all the cards from **other** at the end of this deck. The cards are also removed from **other**. Consequently, the deck designated by **other** is empty after the call.
- **Card removeLast():** removes and returns the last card of this deck.
- **Card removeFirst():** removes and returns the first card of this deck.
- **boolean remove(Card card):** removes the first occurrence of the specified card from this deck, if it is present. Returns **true** if and only if this deck contains the specified card.
- **void removeAll(Deck other):** removes from this deck all of its cards that are contained in the deck designated by the parameter **other**. The cards are not removed from the deck designated by **other**.
- **void shuffle():** randomly permutes the cards.
- **Deck deal(int n):** removes a maximum of **n** cards from the end of this deck. The cards are returned in a new deck.
- **boolean contains(Card card):** returns **true** if and only if this deck contains the specified card.
- **boolean containsAll(Deck other):** returns **true**if and only if this deck contains all the cards in the specified deck.
- **boolean isKind():** returns **true** if and only if this deck is a discardable kind. Specifically, the method returns **true** if this deck has at least two cards and all the cards have the same rank. Otherwise, the method returns **false**.
- **boolean isSeq():** returns **true** if and only if this deck is a discardable sequence. Specifically, the method returns **true** if this deck has at least three cards, the cards all have the same suit, the cards form a sequence of consecutive ranks. Otherwise, the method returns **false**.
- **void sortBySuit():** sorts the cards of this deck by suit.
- **void sortByRank():** sorts the cards of this deck by rank.
- **void print():** prints the content of this deck in two ways. First, the content is printed by suit. Next, the content is printed by rank. Please note that this method has a side effect, the order of the cards is not preserved. Consequently, the method should not be called on the main deck during a game!
- **String toString():** returns a string representation that contains all the cards in this deck.

Here is a link to the documentation of the class **ArrayList**:

- https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

2.4 Game (20 marks)

The class **Game** implements the logic of the single player game of Rummy.

- An object of the class **Game** has two decks of cards, one is the main deck, whereas the second one represents the player's hand. An object of the class **Game** also has a die (reference to an object of the class **Die**).
- The class declares one constructor. This constructor has one parameter, which specifies the number of ranks for this game. When a **Game** object is created, it also creates the main deck and the die.
- A **Game** object has only one public instance method, which is called **play**. The method implements the logic of the game.

Here is a description of the game:

- Step 0. The strange deck is created and shuffled. (In order to test your game more quickly you can reduce the number of ranks to less than 13 and more than 3). In your implementation, the created deck is an object of the class **Deck**. Top of the deck is considered the last card.
- Step 1. The player is dealt 7 cards from the top of the strange deck.
- Step 2. The following steps are repeated until the player runs out of the cards i.e. until the player's hand is empty:
  - The player rolls a die:
    - If the player gets a 1:
      - The player can discard any one card she likes. After that, the current round is over and the game goes back to Step 2.
    - If instead the player gets 2, 3, 4, 5 or 6:
      - The player is first delt, from the top of the deck, a number of cards corresponding to the value of the die, or size of the deck of cards, whichever is smaller. The player then keeps on discarding melds from her hand until she has no more melds. You program has to check that the set of cards that the player chooses indeed forms a valid meld before discarding them from the player's hand. Once she decides she is out of melds, the round is over and the game goes back to Step 2.
- Finally, once the player is out of cards (i.e. once the player's hand is empty), the total number of rounds is reported and the game is over.

Note that if the deck is empty and the player has no more melds, no melds can ever be created again. Thus the player has to wait for 1 on the die. In order to avoid that frustration, your game should roll 1 i.e. set num to 1, in each round that starts with an empty deck.

As usual, whenever you ask the player for some input you should make sure they give you the required kind of input. You may assume that the player will follow instructions and give you a correct type of data, but not the correct values. For example, if you are asking for an integer between 3 and 99, you may assume that the player will give you an integer, but not that she will give you an integer in the correct range. Thus you should keep on repeating the question until you get a valid answer. Similarly if you ask the player for a meld, you may assume that the player will give you a set of 2 or more cards, but you will need to test if these cards are indeed in the player's hand and that they form a meld. We provide you with several methods to help you with this part of the assignment. See 02/Utils.java.

2.5 Run

We provide you with a class called **Run**. It contains the main method for this application.

2.6 Test

We provide you with a class called **Test**. It contains a small number of tests to help you better understand the requirements. Those tests are by no means exhaustive.

2.7 Utils

The class **Utils** contains a number of methods to assist you to solve this assignment. Namely, you can use **Utils.readCard()** and **Utils.readCards()** to obtain one or several cards from the user, respectively.

Warnings

The implementation herein has a number of changes compared to that of ITI 1120/1520. You have to follow our directives in order to get the full marks. In particular, you cannot claim that your implementation meets the requirements of ITI 1120/1520 and thus deserves the full marks.

Academic Integrity

This part of the assignment is meant to raise awareness concerning plagiarism and academic integrity. Please read the following documents.

- https://www.uottawa.ca/administration-and-governance/academic-regulation-14-other-importa
- https://www.uottawa.ca/vice-president-academic/academic-integrity

Cases of plagiarism will be dealt with according to the university regulations. By submitting this assignment, you acknowledge:

1. I have read the academic regulations regarding academic fraud.
2. I understand the consequences of plagiarism.
3. With the exception of the source code provided by the instructors for this course, all the source code is mine.
4. I did not collaborate with any other person, with the exception of my partner in the case of team work.
   - If you did collaborate with others or obtained source code from the Web, then please list the names of your collaborators or the source of the information, as well as the nature of the collaboration. Put this information in the submitted README.txt file. Marks will be deducted proportional to the level of help provided (from 0 to 100%).

Rules and regulation (5 marks)

- Follow all the directives available on the assignment directives web page.
- Submit your assignment through the on-line submission system virtual campus.
- You must preferably do the assignment in teams of two, but you can also do the assignment individually.
- You must use the provided template classes below.
- If you do not follow the instructions, your program will make the automated tests fail and consequently your assignment will not be graded.
- We will be using an automated tool to compare all the assignments against each other (this includes both, the French and English sections). Submissions that are flagged by this tool will receive the grade of 0.
- It is your responsibility to make sure that Blackboard has received your assignment. Late submissions will not be graded.

Files

You must hand in a **zip** file (no other file format will be accepted). The name of the top directory has to have the following form: **a1_3000000_3000001**, where 3000000 and 3000001 are the student numbers of the team members submitting the assignment (simply repeat the same number if your team has one member). The name of the folder starts with the letter "a" (lowercase), followed by the number of the assignment, here 1. The parts are separated by the underscore (not the hyphen). There are no spaces in the name of the directory. The archive a1_3000000_3000001.zip contains the files that you can use as a starting point. Your submission must contain the following files.

- README.txt
  - A text file that contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- 01/A1Q1.java
- 01/A1Q1Test.java

- 01/A1Q2.java
- 01/A1Q2Test.java
- 01/A1Q3.java
- 01/A1Q3Test.java
- 01/StudentInfo.java
- 01/Utils.java
- 02/Card.java
- 02/Deck.java
- 02/Die.java
- 02/Game.java
- 02/Run.java
- 02/StudentInfo.java
- 02/Utils.java
- 02/Test.java

Questions

For all your questions, please visit the Piazza Web site for this course:

- `https://piazza.com/uottawa.ca/winter2019/iti1121/home`

Acknowledgement

We thank our colleagues, Vida Dujmovic and Diana Inkpen, for providing us with text description and source of their assignment. Several sections of their work have been reproduced with their permission!

Last modified: February 1, 2019