Computer Science 2 CSCI-142
War Lab 2: Problem Solving

1/18/2021

1 Introduction

War is a two player card game using a deck of cards. Initially the deck is shuffled and the cards are evenly distributed face down to each player's pile. The goal of the game is for one player to win all the cards.

Each player draws a card face up from the top of their pile at the same time. The player with the higher card takes both cards and puts them face up on the bottom of their pile. If the cards are the same rank, it is War. Each player draws three cards face down from their pile and draws the fourth card face up. The player with the higher face up card takes all the drawn cards. If again it is tie, the War continues.

If a face up card comes to the top of a player's pile, it is reshuffled and all cards become face down again before another card is drawn from it.

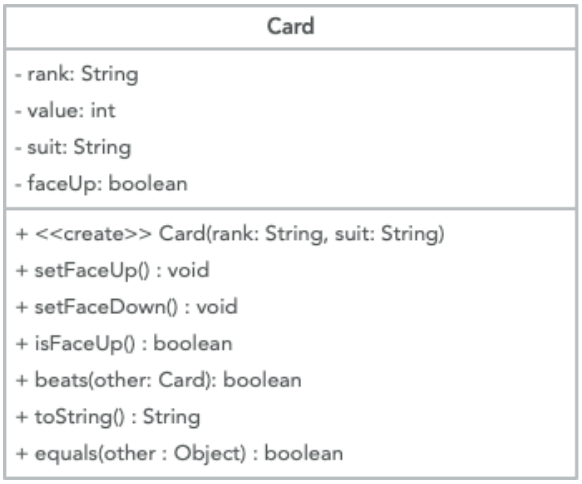
The game ends when the winner has all the cards in their pile, and the loser has none remaining.

2 Problem Solving (20%)

You will work in groups to answer the following questions on pen and paper. Make sure everyone in the group puts their name of the worksheet before handing it in.

1. Go to the website, `https://cardgames.io/war/`. Deal the cards and play through at least one complete round where your pile needs to be reshuffled, and you have encountered at least one War situation.

2. Recall from lecture that a *class* is the blueprint from which we can create an *object*. Objects can be manipulated and can interact with other objects. We can visually capture the classes and their relationships using the *class diagram* concept from the *Unified Modeling Language*, UML. Here is an example UML for a `Card` class.



Design the UML for the other classes that you will need to implement the full game.

(a) Read through the introduction again. The nouns for the various game components are classes. Create boxes for each class and write the name of the class in the top section of the box.

(b) An object is an *instance*, e.g. a variable, of a class type. The *state*, e.g. the data, is the unique values stored inside the object. Typically this data is declared *private*, e.g. a minus sign in UML, to support *encapsulation* and free other objects from having to worry about their internal representation. Capture the state of each class in the middle part of the boxes. If an object of one type *aggregates*, e.g. contains an object of another type, we capture that relationship using an arrow.

(c) The *behaviors*, e.g. methods of the object describe the ways in which the object can be manipulated. Try to capture as many behaviors as possible for your various classes. Think about the verbs that describe the ways in which an object can be manipulated. Also recall that typically you will need *accessors* and *mutators* to work with the object's state (next part). Typically, these methods will be *public*, e.g. a plus sign in UML, so that objects of other class types may interact. Capture the behaviors of each class in the bottom part of the boxes.

3. The most fundamental class for the game is the `Card` class Write the complete Java code for this class using `Card`'s UML from the previous question.

(a) For the constructor, you will need to associate a card's rank with its value. For example:
- The numbered cards should have the values `2-10`.
- The Jack has a value of `11`.
- The Queen has a value of `12`.
- The King has a value of `13`.
- The Ace has a value of `14`.

In addition, when a card is created, it should be face down.

(b) This card beats the other card only if its value is greater than the others.

```
Card card1 = new Card("3", "C");
Card card2 = new Card("Q", "H");
Card card3 = new Card("3", "D");
card1.beats(card2);    // false
card2.beats(card3);    // true
card1.beats(card3);    // false
```

(c) The `toString` method should override `Object`'s. It should return a string which contains the rank and suit.

```
@Override
public String toString() {
       // ...
}
// ..
Card card1 = new Card("3", "C");
Card card2 = new Card("Q", "H");
System.out.println(card1);     // "3C"
System.out.println(card2);     // "QH"
```

(d) The `equals` method should override `Object`'s and returns whether two cards have the same value, or not (regardless of suit). Pay attention that the argument, `other`, is of type `Object`, and not `Card`.

```
@Override
public boolean equals(Object other) {
       // ...
}

//...
Card card1 = new Card("7", "D");
Card card2 = new Card("7", "H");
Card card3 = new Card("A", "S");
card1.equals(card2);    // true
card1.equals(card2);    // false
card1.equals("7D");     // false  ("7D" is a String, not a Card)
```