Playing Well - A Deck of Cards

Due Tuesday, September 13 at 8 a.m.

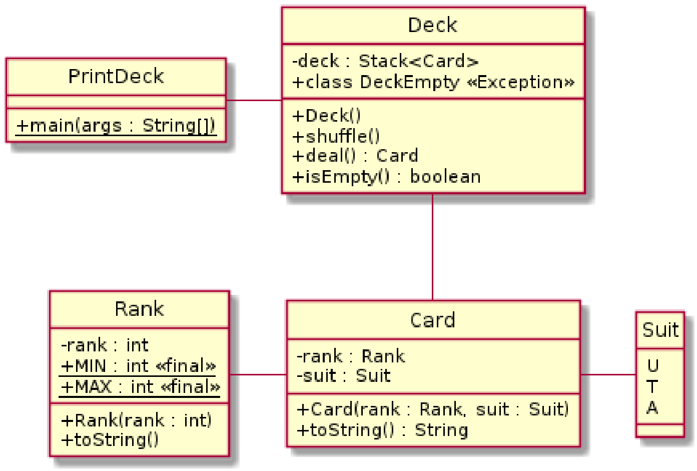
CSE 1325 - Fall 2022 - Homework #3 - Revision 1 - 1

Assignment Background

Having written a basic class in Java, it's time to spread your wings into multiple classes – a Deck of Cards with 3 suits (U, T, and A), and ranks from 0 to 9. We'll also practice some basics: enums, simple containers from the Java Class Library, exceptions, and (at the bonus level) regression tests.

Full Credit

In your git-managed cse1325 directory **cse1325/P03/full_credit** (capitalization matters!), implement the following class diagram. (Don't panic. They're short classes!)

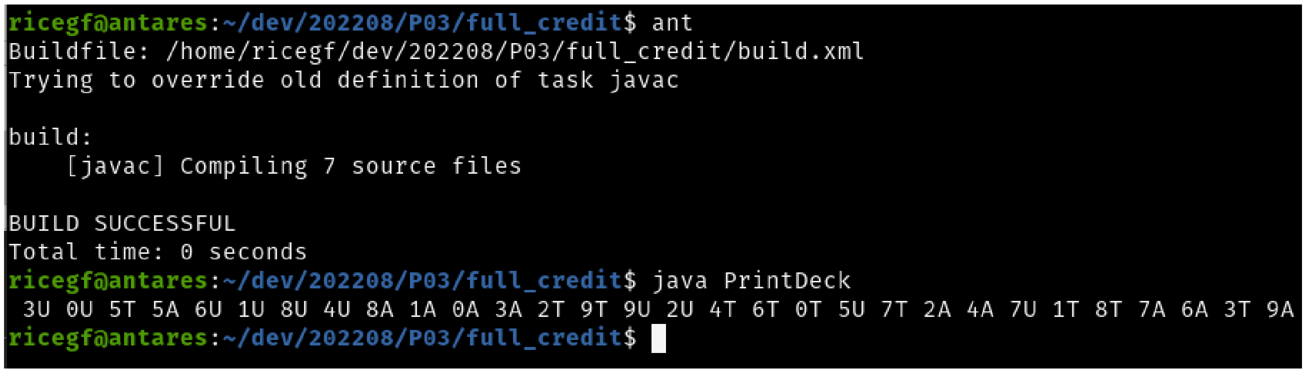


- In **Suit.java**, write an enum with U, T, and A. This could be a single line of code. Don't overthink it.
- In **Rank.java**, write a class that encapsulate an int between 0 and 9. Add public static final ints MIN (0) and MAX (9) to declare this range. In the constructor, initialize the `rank` field to the parameter. For now, the toString method can simply return the string representation of the `rank` field (we'll get fancier in the Bonus). This is again a very simple class. Don't try to make it hard.
- In **Card.java**, write a class that encapsulates a Suit and a Rank. The constructor assigns its parameters to the corresponding fields. The toString returns the `rank` field first, then the `suit`, with no space. For example, `3U` or `0T` or `9A`.
- **In Deck.java, things get more interesting.**
    - The `deck` field is a java.util.Stack. This is similar to an ArrayList, but includes a `push` method (which adds its parameter to the top of the stack), a `pop` method (which removes and returns the top element from the stack), an `empty` method (which returns true if the stack has no elements and false if it does), and a `size` method (which returns the number of elements currently on the stack). See the Stack documentation for details.
    - The constructor should instance one of *every possible card* (`0U` to `9A`, 30 cards in all), pushing each onto the `deck` field. Think 2 nested for loops here.

- The `shuffle` method should mix up the order of the cards in the `deck`. Happily, Java can shuffle the contents of any of its collections using the `java.util.Collections.shuffle` static method. See the shuffle documentation for details.
- The `deal` method should verify that the `deck` is not empty - if it is, throw a DeckEmpty exception (which you must declare - I extended `IndexOutOfBoundsException`, but you may extend whatever make sense to you). If not empty, pop a Card from the `deck` and return it.
- The `isEmpty` method is true if the `deck` is empty, false otherwise.

- In **PrintDeck.java**, write a `main` method to instance and shuffle a `Deck` and print out each `Card` it contains to the console, space separated. You cannot access `Deck.deck` from `main` - it's *encapsulated*. Use `Deck`'s methods instead!

Here's an example of the suggested solution's output. The order of your cards will vary.



```
ricegf@antares:~/dev/202208/P03/full_credit$ ant
Buildfile: /home/ricegf/dev/202208/P03/full_credit/build.xml
Trying to override old definition of task javac

build:
    [javac] Compiling 7 source files

BUILD SUCCESSFUL
Total time: 0 seconds
ricegf@antares:~/dev/202208/P03/full_credit$ java PrintDeck
 3U 0U 5T 5A 6U 1U 8U 4U 8A 1A 0A 3A 2T 9T 9U 2U 4T 6T 0T 5U 7T 2A 4A 7U 1T 8T 7A 6A 3T 9A
ricegf@antares:~/dev/202208/P03/full_credit$ 
```

Include a working build.xml file so that the `ant` command will build your code (this was provided to you in lecture). Add, commit, and push all files to your *private* cse1325 GitHub repository.

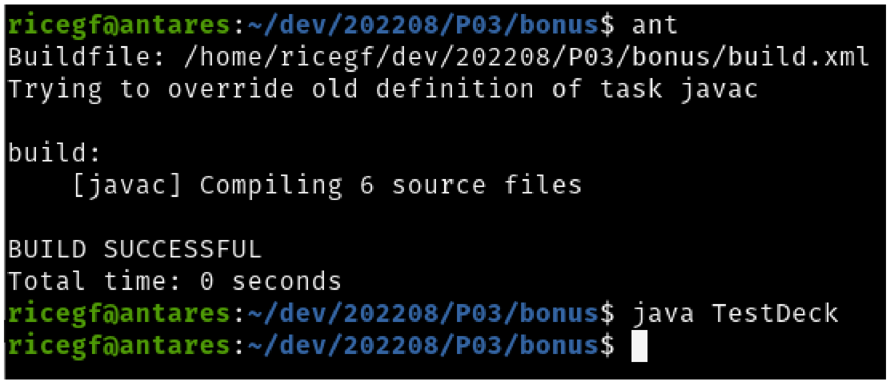
Bonus

In your git-managed cse1325 directory **cse1325/P03/bonus**, add `TestDeck.java` with a main method that tests your Deck class. Implement the following test vectors just as we discussed in Lecture 05. You will receive up to 5 bonus points for each of these 3 tests, for a possible maximum of 15 bonus points.

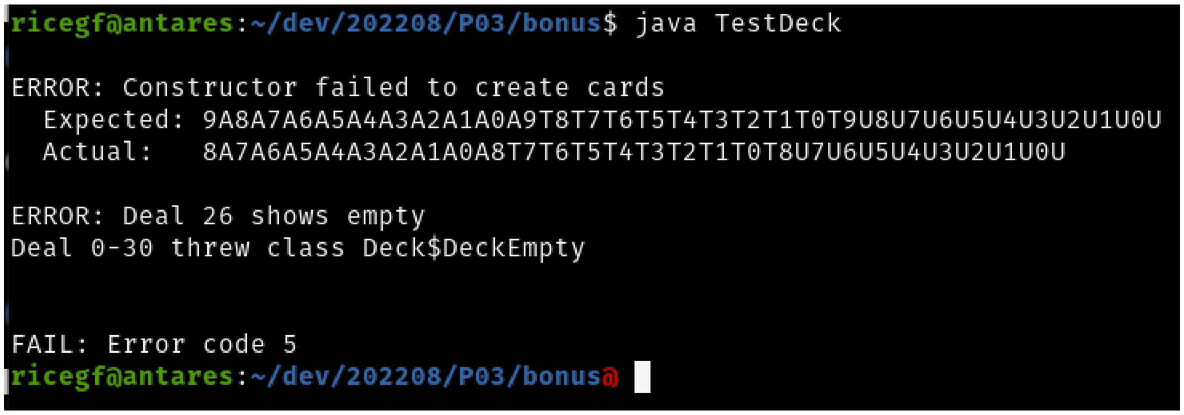
- Instance a Deck object and verify that every card is in it exactly once. Since the cards are not yet shuffled, you can check each popped card in the order expected. Or, create a String containing all of the card's toString representations and compare to an expected String. This tests both the constructor and the `deal` method.
- Instance and shuffle two decks, and verify that the cards are popped in different orders. This is a very simple test of the `shuffle` method.
- Instance a deck and deal 29 cards, verifying that the `isEmpty` method is false after each deal. The deal the 30th card and verify that isEmpty is true. Finally, enter a try/catch clause and try to deal a 31st card, verifying that a Deck.DeckEmpty exception is thrown. This tests both isEmpty and protection from dealing from an empty deck.

Remember that if all tests pass, TestDeck should print *nothing*! For each failure, print what behavior was expected and then what behavior the Deck class actually exhibited, or else the exception generated.

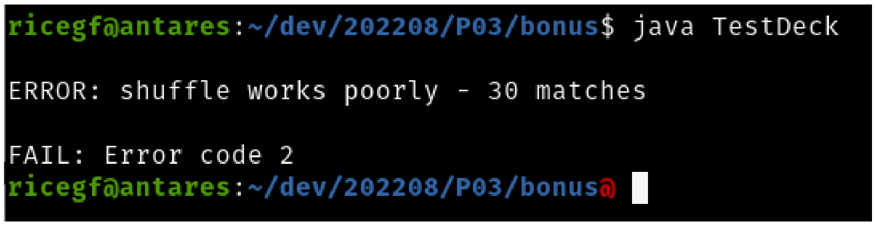
Here are example tests. The first run is with the working class, thus no output. Then I broke the Deck class in various ways to demonstrate how errors would be reported. You may but are not required to include screenshots in your GitHub repository for this assignment. Your error messages do NOT need to match these!



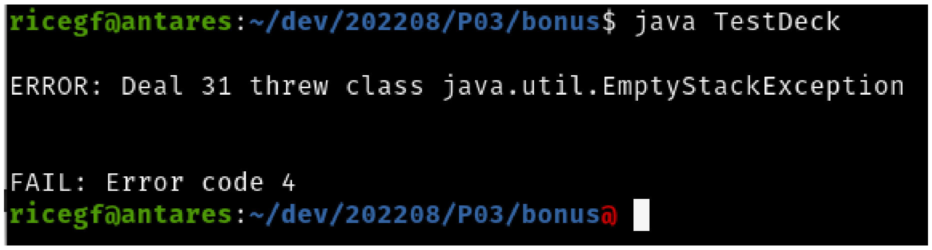
```
ricegf@antares:~/dev/202208/P03/bonus$ ant
Buildfile: /home/ricegf/dev/202208/P03/bonus/build.xml
Trying to override old definition of task javac

build:
    [javac] Compiling 6 source files

BUILD SUCCESSFUL
Total time: 0 seconds
ricegf@antares:~/dev/202208/P03/bonus$ java TestDeck
ricegf@antares:~/dev/202208/P03/bonus$
```



```
ricegf@antares:~/dev/202208/P03/bonus$ java TestDeck

ERROR: Constructor failed to create cards
  Expected: 9A8A7A6A5A4A3A2A1A0A9T8T7T6T5T4T3T2T1T0T9U8U7U6U5U4U3U2U1U0U
  Actual:   8A7A6A5A4A3A2A1A0A8T7T6T5T4T3T2T1T0T8U7U6U5U4U3U2U1U0U

ERROR: Deal 26 shows empty
Deal 0-30 threw class Deck$DeckEmpty


FAIL: Error code 5
ricegf@antares:~/dev/202208/P03/bonus@
```



```
ricegf@antares:~/dev/202208/P03/bonus$ java TestDeck

ERROR: shuffle works poorly - 30 matches

FAIL: Error code 2
ricegf@antares:~/dev/202208/P03/bonus@
```



```
ricegf@antares:~/dev/202208/P03/bonus$ java TestDeck

ERROR: Deal 31 threw class java.util.EmptyStackException


FAIL: Error code 4
ricegf@antares:~/dev/202208/P03/bonus@
```

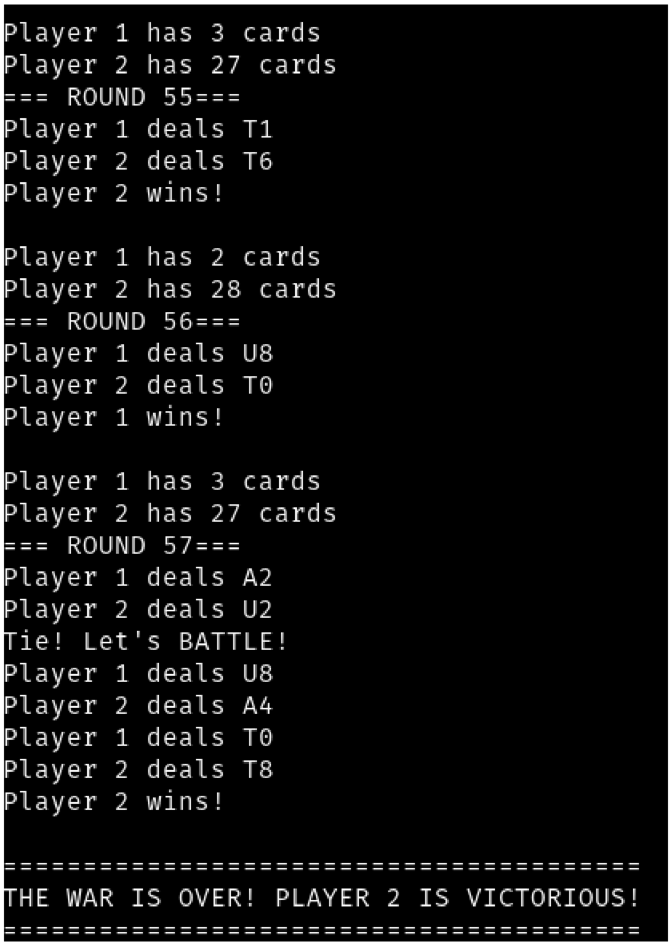
Include a working build.xml file so that the `ant` command will build your code (this was provided to you in lecture). Add, commit, and push all files to your *private* cse1325 GitHub repository.

Extreme Bonus

In your git-managed cse1325 directory **cse1325/P03/extreme_bonus**, write a simple game or more serious program of your choice using class `Deck`. You may make any changes or enhancements to the classes developed for the Bonus level, add any classes you need, and use any members of the standard Java library you need.

The suggested solution implements the card game `War`, adapting the rules from a standard 52 card deck to our 30 card UTA deck. But you may write any game you like! We will grade very gently.

Here are the rules for War: https://bicyclecards.com/how-to-play/war/ As a kid, we always turned over 4 cards each for a war rather than 1 card each. If a player had fewer than 4 cards left, that's how many we turned over. If no cards were left, the cardless player lost. So that's how I wrote the suggested solution! (Note: You'll sometimes hit an "infinite loop" of card battles, so every 4051st battle I flip the result so that the *lowest* rank wins. The nice thing about games is that you can make up new rules as needed!)



```
Player 1 has 3 cards
Player 2 has 27 cards
=== ROUND 55===
Player 1 deals T1
Player 2 deals T6
Player 2 wins!

Player 1 has 2 cards
Player 2 has 28 cards
=== ROUND 56===
Player 1 deals U8
Player 2 deals T0
Player 1 wins!

Player 1 has 3 cards
Player 2 has 27 cards
=== ROUND 57===
Player 1 deals A2
Player 2 deals U2
Tie! Let's BATTLE!
Player 1 deals U8
Player 2 deals A4
Player 1 deals T0
Player 2 deals T8
Player 2 wins!

========================================
THE WAR IS OVER! PLAYER 2 IS VICTORIOUS!
========================================
```

Other games you may be able to adapt would be Blackjack, Go Fish, Uno, or the various forms of Poker and Solitaire. The game show "Card Sharks" also had a high / low game that could be adapted to use our `Deck`.

On a more serious note, a common memory challenge deals the cards in an array (say, alternating rows of 7 and 8 cards) face down, then allows the player to select in pairs until the ranks matched and the pair removed. The object is to remove all cards in as few moves as possible.

The deck could also be used to construct a math drill for children, with the rank of the first card being the operation to perform (say, A is add, U is subtract, and T is multiply). So the correct answer for 4U 3A would be 4-3 or 1. Or you could replace the ranks with +, -, x, and ÷ to be even more clear.