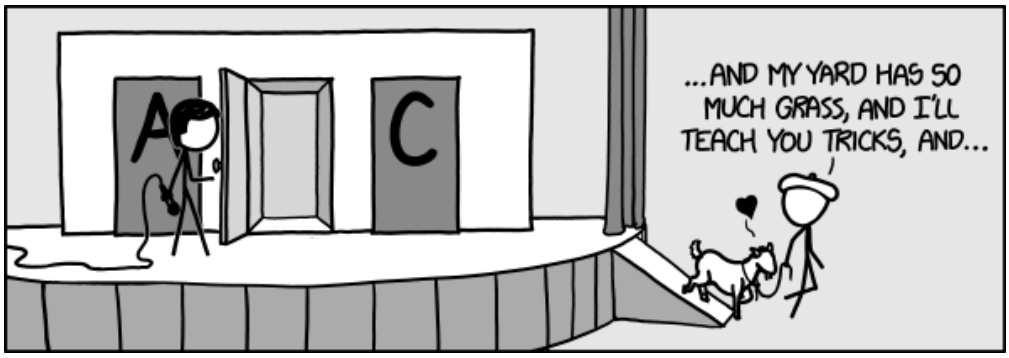
# CS62 - Flippy Card

MONTY HALL




http://www.xkcd.com/1282/

## Java Practice

To continue to give you practice with the syntax of Java, for the first part of this assignment, you're going to write a few stand-alone methods. Fill in the three functions below in the `WarmUp` class.

1. Write a method called `countHearts` that takes an array of `Card`s as a parameter and counts how many of the cards have the suit "hearts".

   For example, if we ran the following code:

   ```
   Card[] cards = new Card[4];

   cards[0] = new Card(1, "clubs");
   cards[1] = new Card(1, "hearts");
   cards[2] = new Card(3, "hearts");
   cards[3] = new Card(2, "diamonds");

   System.out.println(countHearts(cards));
   ```

   it would print 2.

2. Write a method called `addArraysSameLength` that takes two arrays of `double`s as parameters. The method should make a new array where the $i$th entry in this array represents the sum of the $i$th entries of the two input arrays and return this new array. You may assume that the arrays have the same length.

   For example, if we ran the following code:

```
double[] array1 = {1, 2, 3, 4, 5};

double[] answer = addArrays(array1, array1);
System.out.println(Arrays.toString(answer));
```

it would print `[2.0, 4.0, 6.0, 8.0, 10.0]`.

*Aside:* There is a class called `Arrays`[1] that has a number useful static methods for processing arrays. `Arrays.toString()` is a helpful method that brings out the contents of the array since the default behavior when you try and print an array just prints the memory location.

3. Write a method called `reverseArray` that takes an array of `String`s as a parameter and reverses the order of the array. It should NOT return a new array. Instead, it should mutate (i.e. change) the array that was input.

   For example, if we ran the following code:

```
String[] words = {"I", "love", "my", "CS", "classes", "!"};

System.out.println("Before: " + Arrays.toString(words));
reverseArray(words);
System.out.println("After: " + Arrays.toString(words));
```

   it would print:

```
Before: [I, love, my, CS, classes, !]
After: [!, classes, CS, my, love, I]
```

4. *OPTIONAL:* If you want some extra practice (NOT REQUIRED), try writing another function called `addArrays` that adds two arrays as described above, but removes the requirement that they be the same length. It should produce an array the size of the longest input array. If there isn't a corresponding entry in one of the arrays because the other is longer, then it should just use the value of the longer array.

   For example, if we ran the following code:

```
double[] array1 = {1, 2, 3, 4, 5};
double[] array2 = {10, 10};

double[] answer = addArrays(array1, array2);
System.out.println(Arrays.toString(answer));
```

   it would print `[11.0, 12.0, 3.0, 4.0, 5.0]`.

[1]The full documentation for the `Array` class is at: `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html`

## Flippy Card: The Game

For the rest of the assignment, we're going to write a game I call "Flippy Card"[2]. When the game starts, you deal five cards face down. Your goal is to achieve as high a score as possible. Your score only includes cards that are *face up*. Red cards (hearts and diamonds) award positive points, while black cards (clubs and spades) award negative points. Cards 2-10 have points worth their face value, Jack, Queen and King are 10, and Ace is 11.

The game is played by flipping cards (either from face down to face up or from face up to face down). As you play, you are told your total score (i.e. total of the face up cards) AND the total score of the cards that are face down. The challenge is that you only get a fixed number of flips and then the game is over.

Here is an example of the output from playing the game:

```
FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 0
Face down total: -5
Number of flips left: 5
Pick a card to flip between 1 and 5 (-1 to end game): 1

Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 4
Pick a card to flip between 1 and 5 (-1 to end game): 2

Queen of hearts | Jack of clubs | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 0
Face down total: -5
Number of flips left: 3
Pick a card to flip between 1 and 5 (-1 to end game): 2

Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 2
Pick a card to flip between 1 and 5 (-1 to end game): 10

10 is not a valid card
Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 2
```

[2]If you have a better name, please, please let me know :)

```
Pick a card to flip between 1 and 5 (-1 to end game): 0

0 is not a valid card
Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | FACE-DOWN
Face up total: 10
Face down total: -15
Number of flips left: 2
Pick a card to flip between 1 and 5 (-1 to end game): 5

Queen of hearts | FACE-DOWN | FACE-DOWN | FACE-DOWN | 3 of hearts
Face up total: 13
Face down total: -18
Number of flips left: 1
Pick a card to flip between 1 and 5 (-1 to end game): -1


----------------------
Your score: 13
Best possible score: 15
```

At each point where the program asks the user to pick a card, the game waits for user input.

## Code

We'll implement the game using four classes, two of which you've seen already.

- `Card` We'll be using the `Card` class from class. There are a number of ways to adapt this class for our use, but we'll keep it simple and add a few extra methods directly to this class. **There are two methods that you need to fill in the details for this class.**
- `CardDealer` This is the class that we looked at in class/lab.
- `FlippyCards` This is the class that keeps track of the state of the game. Virtually all of the methods that you'll be implementing are here, though you will use the two previous classes to help you. Make sure you understand what each method is supposed to do.
- `FlippyCardGame` This class is where the game interface is implemented. It gets the data from the user, update the board, and repeats the process.

## How to proceed

You may fill in the details of the classes however you'd like, but here's what I would recommend.

- Add the two methods to the `Card` class. Add a `main` method to this class and *test them to make sure they work.*
- Read through the two flippy card classes. Make sure you understand what all of the methods in `FlippyCards` are supposed to do and that you understand how they're used ini the `FlippyCardGame` class.
- Implement methods ini the `FlippyCards` class incrementally. Write one method and then test it! To test it, again, add a `main` method and write a small test or two to make sure it behaves like you'd expect. If you try and implement all of the methods and then test it by simply running `FlippyCardGame`, 1) it's very unlikely you'll get it perfect the first time and then 2) it's going to be very hard to figure out where the problem is.
- Once you have things working, play the game! Then, like always, document your code :)