**Lab extension 1: Blackjack**

You are going to write a blackjack game. Now, for simplicity's sake, we're going to cheat a bit for now: we're not going to use a fair deck, as that would require keeping track of which cards have already been played. So really, don't bet against the house on this game. We're going to assume an infinite deck, for starters.

Leaving aside our simplifications, here's a refresher as to the rules of Blackjack (or, if you will, a really simplified version of Blackjack):

1. The player is dealt two cards.
2. The player's object is to get as high a score as possible, without exceeding 21 ("busting").
3. Numbered cards count as their natural value.
4. Jack, Queen, and King ("face cards") count as 10.
5. Aces are valued at 1 or 11 according to the player's preference.
6. If the hand value exceeds 21 points, it busts.

Simplifications:

1. For this game, we won't display or represent actual cards, but just their scores: values between 1 and 11.
2. We are not going to worry about the fact that once you've drawn all four '2' cards, the probability of drawing another '2' is zero. In our case, the dealer is cheating with an infinite deck.
3. We are not going to worry about keeping track of what hand the player already has, just what their total score is.
4. At some tables, the player is asked what value he or she wants for an ace; at others, the dealer chooses the "best" choice for them. Our program should always ask, even if the choice would be obvious (leading to a score of 21, or leading to a "bust").

Initially, the game should present the player with a *two-card hand*: the sum of two random values, each ranging from 1 to 10. Why 1 to 10, and not 1 to 11? Because if the player gets an ace, he or she gets to *choose* whether or not it should count as 1 or 11. If your program (as the dealer) ever deals a 1, it should *ask* the user whether or not to add 10 to the total.

The astute reader may notice that while there are 4 ways to get a card valued at 2 (one for each suit), and 4 ways to get a card valued at 4 (one for each suit), there are *16* ways to get a card valued at 10 (for each suit, a 10, Jack, Queen, or King). We will ignore this for now, because we're playing with a simplified (and infinite) deck.

Now, as mentioned, the player is dealt two cards, and should be *asked* if any 1 should be turned into an 11.

The player should then be told his or her score, and so begins a *loop*:

The player is asked whether to *hit* or *stick*. Hit means the program should deal the player one more card; stick means the player is satisfied with that score. If the player types *hit*, the program should deal another card. The program should continue asking *hit* or *stick* until the sum of the card values (including aces valued at 11) hits 21; if it equals 21, the player wins! If, however, it exceeds, 21, the player loses, and should be insulted accordingly!

**Helpful hints**

Use the standard input library to read numbers and strings from the user. The API for this library is provided on p.127 of your text. To use it, download [StdIn.java](http://introcs.cs.princeton.edu/java/15inout/StdIn.java) into the same directory as your program.

You cannot compare Strings (or any objects) with the == operator and get the results you probably expect -- two Strings that contain exactly the same characters will not necessarily be "==" to each other. To compare Strings to see if they contain the same (or similar) characters, you need to use certain String methods. The String method [equals](http://download.oracle.com/javase/1.4.2/docs/api/java/lang/String.html#equals(java.lang.Object)) returns a boolean result, True if the Strings contain the same characters, False if not. For example:

if ( command.equals( "quit" ) ) {

// This code will be executed if String command contains

// 'q', 'u', 'i', and finally 't'

}