Jonathan Simonin

Professor Wei

CSE 2100

30 October 2016

# 24 Programming Assignment – User Input

## Programming Design:

The overall design of this program is more involved than initially it seems. There were several things I had to do differently than the last assignment, as well as add some more methods. In order to do this, I did the following:

1. Create methods for taking user input and restarting the game as well as making an introduction.
2. When we ask for input we do a check to see if it is valid, i.e. if it contains any obscure letters like P or S, or the numerical values are greater than 13 or less than 1 we will ask again.
3. Creating all possible operations and putting it in a string array as a list
   a. This generates all 64 combinations of operators we can use
4. Creating all possible hand combinations with the use of 2 methods that can swap the numbers in an array.
   a. We create 24 total combinations to evaluate by doing this – use of recursion.
   b. Next we will perform the evaluation of the swapped or original hand given.
5. When we perform the computation, we take our operation list and cut it up into a character array, this way we can use each set of 3 easier in our postfix notation.
   a. Before we step inside the computation method, we put in a parameter for which infix notation we are testing and then go through each tree to find the solutions possible.
   b. We also pass in an array which is the current hand we test
   c. Then we perform the method called PuzzleSolve
6. In our PuzzleSolve method we take in our parameters and keep track of the amount of solutions we have found.
   a. We create a temporary binary tree and create each individual tree depending on the computation we want to perform by the infix parameter
   b. After each tree is made we perform the method called Checking and see if it falls between our parameters for error due to rounding. If we found a solution increase the count, if we end up with no solutions we tell the user and ask for input again or to quit.
7. In the method checking we do the following steps:

a. Pass in a tree with an initial position and go down to the bottom of the tree
b. Depending on what we see, we will create evaluations by storing the numbers and if we end going back up the tree we will perform the operation.

I made this program to work in unison so each method depended on another in some way. I thought of it as making a video game, as I have experience with, and I made it try to play with fluidity and a nice experience. When debugging this program I could tell if something wasn't working write or if a certain method wasn't being executed because I would create print lines to see if it would follow through with a certain condition and then execute the next method.

**Tradeoffs:**

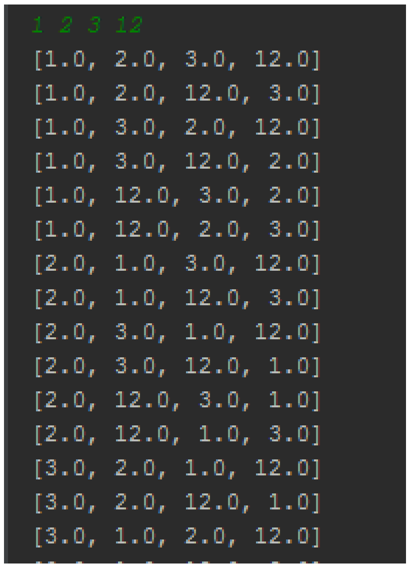
Throughout this assignment I had considered many things. One of the major things I took into consideration and actually used was taking my code from the last assignment and pasting it into this assignment and throwing out what I knew I could not use – which was at least half. The other bits, such as creating all the operation lists and general formation of adding things to a stack (which I later changed to a tree) could be usable. The rest I got rid of and started by adding the user input and experience. Another tradeoff I considered was doing brute force calculations with the tree but I knew that would be much more inefficient and would create many more lines of confusing code than using recursion to do what I wanted – recursion was essentially the key to solving this.

**Extensions**

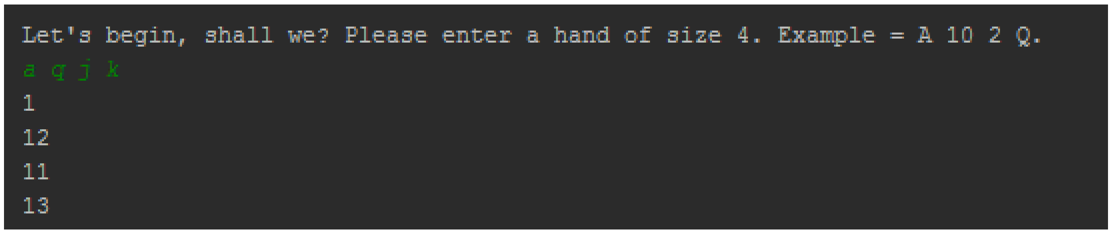
If I were to have any extensions for this I would want to create a graphical interface which I would do using Unity, a game development engine, so it would look nicer while doing the same functions. Another thing I would have fixed or addressed is the way the tree can be added to, I felt like it was very monotonous and inefficient. Moreover, I would try to create a better way of printing out the expressions. Currently I have it so that the method prints out the resultant with the next number its evaluating with – which is partly incorrect but I am not sure how to do it the way the assignment requires.

**Test Cases:**

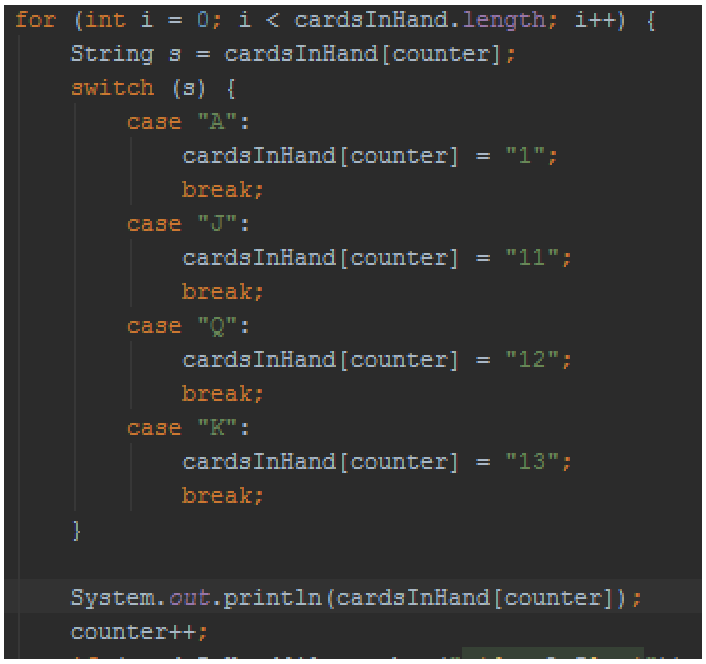
Throughout the assignment I created test cases to see how the program would work with each method – generating all 24 combinations of the hand, all 64 operator combos, printing out the right solutions, having no repeated combinations, making sure user input was working correctly, resetting the game, getting user input, and seeing what the user input received was. Since it is harder to explain with words, screenshots of what I tested seems more appropriate. On the output document, you will see some of the error finding my program does, however I noticed that the calculations printed out twice for some reason. Note* Check the output document, it shows more test cases being tested not shown in this document.



```
1 2 3 12
[1.0, 2.0, 3.0, 12.0]
[1.0, 2.0, 12.0, 3.0]
[1.0, 3.0, 2.0, 12.0]
[1.0, 3.0, 12.0, 2.0]
[1.0, 12.0, 3.0, 2.0]
[1.0, 12.0, 2.0, 3.0]
[2.0, 1.0, 3.0, 12.0]
[2.0, 1.0, 12.0, 3.0]
[2.0, 3.0, 1.0, 12.0]
[2.0, 3.0, 12.0, 1.0]
[2.0, 12.0, 3.0, 1.0]
[2.0, 12.0, 1.0, 3.0]
[3.0, 2.0, 1.0, 12.0]
[3.0, 2.0, 12.0, 1.0]
[3.0, 1.0, 2.0, 12.0]
```

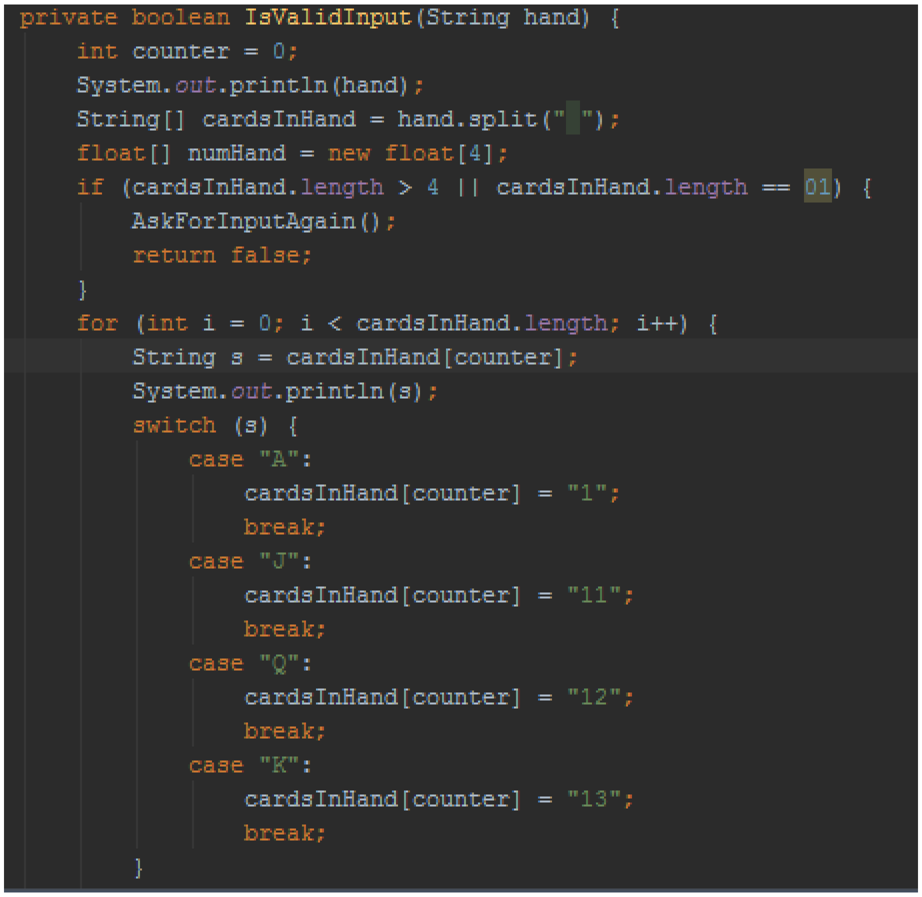


```
Let's begin, shall we? Please enter a hand of size 4. Example = A 10 2 Q.
a q j k
1
12
11
13
```

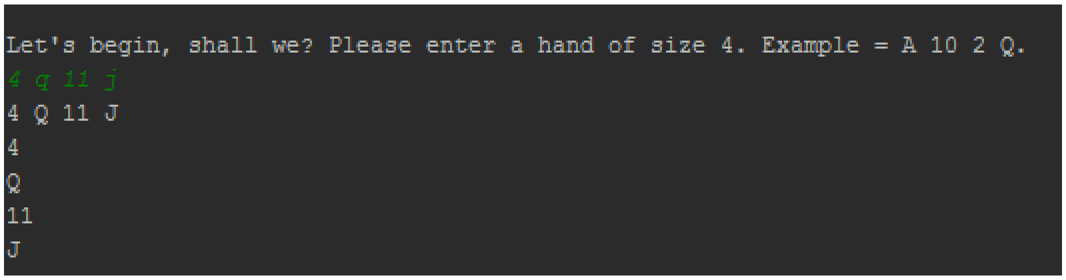


```
for (int i = 0; i < cardsInHand.length; i++) {
    String s = cardsInHand[counter];
    switch (s) {
        case "A":
            cardsInHand[counter] = "1";
            break;
        case "J":
            cardsInHand[counter] = "11";
            break;
        case "Q":
            cardsInHand[counter] = "12";
            break;
        case "K":
            cardsInHand[counter] = "13";
            break;
    }

    System.out.println(cardsInHand[counter]);
    counter++;
```



```
private boolean IsValidInput(String hand) {
    int counter = 0;
    System.out.println(hand);
    String[] cardsInHand = hand.split(" ");
    float[] numHand = new float[4];
    if (cardsInHand.length > 4 || cardsInHand.length == 01) {
        AskForInputAgain();
        return false;
    }
    for (int i = 0; i < cardsInHand.length; i++) {
        String s = cardsInHand[counter];
        System.out.println(s);
        switch (s) {
            case "A":
                cardsInHand[counter] = "1";
                break;
            case "J":
                cardsInHand[counter] = "11";
                break;
            case "Q":
                cardsInHand[counter] = "12";
                break;
            case "K":
                cardsInHand[counter] = "13";
                break;
        }
```



```
Let's begin, shall we? Please enter a hand of size 4. Example = A 10 2 Q.
4 q 11 j
4 Q 11 J
4
Q
11
J
```