

HW4: Let's Play Poker



Warm-Up Question

How would you access the number '7' in this array?

```
int array[] = {1, 3, 5, 7, 9};
```

Searching through an array

- Start from index 0, stored in “i”
- Increment, aka add 1, to “i” every time the loop finishes
- Stop when we have finished going through the entire array
 - Aka when “i” reaches the size of the array

```
boolean containsZero = false;  
for (int i = 0; i < a.length; i++) {  
    if (a[i] == 0) {  
        containsZero = true;  
    }  
}
```



Ending Loops Early

- Break: Leaves loop, loop stops running
- Continue: Leaves current iteration of loop, starts next iteration of loop
- Return: Leaves entire METHOD, loop stops running, method completes

<pre>while(true) { break; System.out.println("Loop"); } System.out.println("End");</pre> Loop finishes. "End" is printed.	<pre>while(true) { continue; System.out.println("Loop"); } System.out.println("End");</pre> Next iteration of loop runs. Nothing is printed.	<pre>while(true) { return; System.out.println("Loop"); } System.out.println("End");</pre> Loop and method finish. Nothing is printed.
--	---	--

Reading input

- Use Scanner class from `java.util.Scanner`

Methods that might come in handy:

- `hasNext()`
 - `hasNextInt()`
 - `next()`
 - `nextInt()`
-
- `next()` - grabs next “word” (separated by space)
 - `nextLine()` - grabs next “line” (separated by newline)

Rest of Scanner documentation: <https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>



A Greeting Program

```
Scanner scnr = new Scanner(System.in);
int age = 0;
String name = "";

while (scnr.hasNext()) {
    if (scnr.hasNextInt()) {
        age = scnr.nextInt();
    } else {
        name = scnr.next();
    }
}
System.out.println("Hello, " + name + ". You are " + age + " years old.");
```

This program will read in two arguments in (a name and age), and print “Hello, <name>. You are <age> years old.” The two arguments can be in any order.

Creating and using methods

- A java method is a collection of statements that are grouped together to perform an operation.

```
/* Takes in two integers, a and b, performs some  
calculation, and returns the result */
```

```
public int methodName(int a, int b) {  
    //body that does calculation  
}
```

- public: can be seen/accessed from other classes (use this)
- int: return type, gives integer back to method that called it
- methodName: name of the method
- int a, int b: what you need to give it for it to work

Example: printSum

- Say we want a method to print two numbers added together
- Let's make a few design choices

Name: printSum

Parameters: two integers to add

Returns: nothing, because the purpose of this method is to print

```
public void printSum(int a, int b) {...}
```

- Now we know how the method works, let's implement it

```
public void printSum(int a, int b) {  
    System.out.println(a+b);  
}
```

Poker hands

The meat of the program will be recognizing the potential poker hands you can make with 5 cards.

Your program should have methods that will loop through the arrays representing each player's hand, and check if a certain hand can be made. EX: checkPair(), checkStraight(), etc.

Here is a list of poker hands ----->

Note: You will not be required to implement a check for High Card.

ROYAL FLUSH This hand contains five cards in sequence, all of the same suit.	A ♥	K ♥	Q ♥	J ♥	10 ♥
STRAIGHT FLUSH This hand contains five cards in sequence, all of the same suit.	8 ♠	7 ♠	6 ♠	5 ♠	4 ♠
4 OF A KIND This hand contains all four cards of one rank and any other unmatched card.	5 ♦	5 ♠	5 ♥	5 ♣	3 ♥
FULL HOUSE This hand contains three matching cards of one rank and two matching cards of another rank.	K ♥	K ♦	K ♠	5 ♥	5 ♣
FLUSH This hand contains all five cards are of the same suit, but not in sequence.	K ♠	J ♠	9 ♠	7 ♠	3 ♠
STRAIGHT This hand contains five cards of sequential rank in at least two different suits.	Q ♠	J ♦	10 ♣	9 ♠	8 ♥
3 OF A KIND This hand contains three cards of the same rank, with two cards not of this rank nor the same as each other.	Q ♠	Q ♥	Q ♦	5 ♠	9 ♣
2 PAIR This hand contains two cards of the same rank, plus two cards of another rank.	K ♥	K ♠	J ♠	J ♦	9 ♦
1 PAIR This hand contains two cards of one rank, plus three cards which are not of this rank nor the same.	A ♠	A ♦	9 ♥	6 ♠	4 ♦
HIGH CARD made of any five cards not meeting any of the above requirements.	A ♦	7 ♥	5 ♣	3 ♦	2 ♠

<http://www.wsop.com/how-to-play-poker/images/how-to-ranking.jpg>

Note about Straight Flush and Straight

- Straight Flushes and Straights can CIRCLE AROUND back to Ace
- Ex. “ACE, 10, Jack, Queen, King” is legal
- CANNOT circle around farther
- Ex. “ACE, 2, Jack, Queen, King” is not legal
- “ACE, 2, 3, 4, 5” is still a valid straight



More Notes about Checking Hands

- Keep in mind that some cases have already been checked for before
- Ex. Two Pairs, where both pairs are the same value. This should have been found in Four Of A Kind.
- Ex. Two Pairs, where one pair is part of a triple. This should have been found in Three of a Kind.
- Much less cases to test for for the “lower tier” hands, “upper” hands have filtered out some possibilities already
- You can write methods assuming certain conditions

Other Important Notes

- Static methods can only call other static methods and constructors!
- Ex. Given a method readInput() called from main(), readInput() must be static
- TL;DR: For this assignment, **every method should be static**

- You DO NOT have to check for wrong input - for this program the user is assumed to not be an idiot
- Cards WILL BE input in order
- This means cards of the same value will be next to each other in the array

- Your output should match EXACTLY - grading will be automated
- 1 should be output as A, 11 as J, 12 as Q, 13 as K

Steps to Completion

1. Read user input
2. Convert user input to value/suit arrays
3. Print value/suit arrays back to user
4. Check what the best hand is
5. Print the best hand
6. Repeat for player 2
7. Compare hands and print winner
8. MAKE SURE YOU FOLLOW THE STYLE GUIDELINES



1. Read user input

OPTION A:

- Grab the entire line of input in main, then pass it into a method to convert
- Create a new scanner = new Scanner(String input) to parse the String

OPTION B:

- Pass in the Scanner to a separate method
- Run nextInt() and next() on the Scanner that is passed in
- Recommended that in the method, the results are saved in a CLASS variable

In Code

```
// A
String input = scanner.nextLine();
parseInput(input);
public void parseInput(String input) {
    Scanner new_sclr = new Scanner(input);
    // read from new_sclr, save results to class var
}
```

```
// B
parseInput(scanner);
public void parseInput(Scanner scanner) {
    // read from scanner, save results to class var
}
```

2. Convert Input into Arrays

- To turn String into int: `Integer.parseInt(String string)`
 - EX: `int n = Integer.parseInt("7");` // stores value 7 into n
- To grab char from String: `string.charAt(int index)`
 - EX: `char c = ("hunter2").charAt(2)` // stores value 'n' into c
- Loop through the input five times, each time grabbing an integer and a char

```
// values and suits are arrays belonging to the Poker class
for (each of the 5 cards) {
    values[i] = next int from input
    suits[i] = next char from input
}
```

3. Print Values/Suits Back to User

- MUST BE SEPARATE METHOD
 - Make sure to print 1, 11, 12, 13 as A, J, Q, K accordingly
 - No spaces between values and suits
 - Spaces between different cards
-
- On examples given, there is STILL a space after the last card
 - Do NOT remove this last space



4. Check What the Best Hand Is

- EACH MUST BE ITS OWN METHOD (minus High Card aka no hands)
- Once you find a hand, no point checking the others
- Recommended each method returns a boolean, true if the hand exists, false otherwise
- Probably should not be taking any parameters - use the class arrays values/suits
- For most hands, you want to keep track of what the last card was, and how many times it has already appeared

CORE OF CHECKING FOR HANDS

/* returns boolean, true if hand exists, false if it doesn't */

```
public boolean checkXHand() {  
    int value = values[0];  
    int counter = 1;  
    for (int i = 1; i < values.length; i++) {  
        if (value != values[i]) {  
            value = values[i];  
            counter = 1;  
        }  
        else  
            counter++;  
    }  
    return false;  
}
```

Tips for Checking Hands

- Some hands are a combination of less advanced hands. Finish the less advanced hands first.
 - Remember that some cases have already been checked by “higher level” hands
 - You can return as soon as you know the hand exists
 - Ex. Once you find a pair for One Pair, you no longer have to check the rest
-
- Test EXTENSIVELY - many edge cases

5. Print the Best Hand

- Recommended that a separate method checks all the hands
- Recommended that this method returns a String, because result needed later

Ex.

```
public static String findBestHand() {  
    if (checkThisHand()) {  
        return "thisHand";  
    }  
    else if (checkThatHand()) {  
        return "thatHand";  
    }  
    else if ...  
}
```

6. Repeat for Player 2

- SAVE hand result for player 1 in a String somewhere
- Repeat steps 1-5, overwriting values/suits arrays



PLAYER 2
START!

7. Compare Hands

- Given the two Strings - the hands of player1 and player 2 - print a certain output
- HMM gee sounds like a method

Ex.

```
public void printWinner(String p1_hand, String p2_hand) {...}
```

- How do you determine which hand beats what?
- MY IMPLEMENTATION: Have a separate method “public int getRank(String hand)” that returns an integer representing the rank of the hand
- Ex. “Four of a Kind” returns 8, “One Pair” returns 2
- Print whichever hand is larger, or tie if they are equal

8. Style, Style, Style

- File, method, AND class headers
- Inline comments
- Logical blank line separation
- Lines LESS than 80 characters
- Consistent indentation of TWO SPACES per block
- NO hardcoded constants
- DESCRIPTIVE variables names



Questions

