Cross off the steps you already completed. Check which remaining ones you complete in class now.

- Understand the rules of the game if it were played on a physical board. Note that the checker to be moved is specified by the number THAT IS WRITTEN ON THE CHECKER.

- Code the class and main method for a Java application.

- The most important things are the VARIABLES. Declare static variables to represent the board and the checkers on it. The count of checkers on the board is useful to easily tell when the game is over. A Scanner will be used for all the input. The first thing main does is prompt for number of checkers to start with. So, the code to start with is:

```java
import java.util.Scanner;
public class OneDimCheckers {
    static int board[] = new int[10];
    static int nPieces;
    static Scanner sc = new Scanner(System.in);
    public static void main(String a[]) {
        System.out.print("Number of checkers to start with [2-9]:");
        nPieces = sc.nextInt();
        System.out.println(nPieces);
    }
}
```

The last statement is temporary. It's there to test the inputting of the value for nPieces.

- [OMIT THIS FOR NOW IF YOU HAVEN'T DONE IT] Add a (while) loop to repeatedly test (nPieces < 2 || nPieces > 9) and if so, print a message, repeat the prompt and use sc to input another integer for nPieces.

- Code a static method say "printBoard" to print the integers in the board array with spaces in between. Code this first so you can test future code. Replace the last statement in main above with a call to this method to partially test the board printing.

- Code a static method named say "fill" to fill the board with pieces 1, 2, ..., nPieces. Code the call to it in main. Then, code a call to printBoard (Printing the board after filling will test your fill method and is a requirement in the specification. Test your program so far for 2, 9 and a number between 2 and 9.

- (a) Code a static method named say "makeOneMove" with a dummy body. For now, make the dummy body unconditionally print a greeting and subtract 1 from nPieces. (b) Next, code the while loop in main that will control the calling of makeOneMove and then printing the board and repeat while nPieces is still > 1. When nPieces goes down to 1, the "End of Game." message should be printed and main should return. The purpose of the dummy behavior in makeOneMove is to test this loop in main.
Add the code to `makeOneMove` to prompt for a number and letter and then use `sc` twice to input them into local variables. Also, add the code to reprint the number and letter on a line by itself with no space in between. This is required by the specification and it will enable you to test the input of the number and letter. You could and should test this with the existing `main` without changing `main`.

Think of the simple strategy to use a loop to see if the number appears in the board. (See the assignment and tip document.) Decide whether to implement this search loop inside `makeOneMove` or to create a new static method for this purpose. Then, code the loop. Add a temporary printing operation to test the search. Make `makeOneMove` (a) invoke the search loop and (b) print Bad Move if the number does not appear in the board.

You can now work a little at a time to complete and perfect the project. The position P (0-9) where the checker that the user wants to move has been determined. The R or L direction has been inputted. What is relevant for (a) telling whether the user specified a move that fits the rules, and (b) if so, exactly which and how array values are changed, are:

- Does board[P] contain 0 or not?
- If the direction is R, is P+1 a legal array index (i.e., >= 0 and <= 9, but only one of these must be tested.)? If so, does board[P+1] contain 0, or not?
- In one of the cases when the direction is R, is P+2 a legal array index? If so, does board[P+2] contain a piece or not?
- Analogous numbers as array indexes when the direction is L.
- What are the subscripts (among P-2, P-1, P, P+1 or P+2) of array element(s) to be set to zero or to be assigned the value from board[P].

Another incremental option: First implement and test the game logic ignoring the need to detect if an index is a legal array index. Of course, when an illegal array index is used for an array access, the program will crash. But testing and debugging can be done with positions near the center. Second, add the conditionals to test when certain array indexes are illegal.