What we will cover:
Palindromes
Selection Sort

Palindromes
Palindromes are words that are the same both backwards and forwards. Words like eye, mom, dad, and racecar are all palindromes. Before we solve this recursively, we must solve it iteratively. To test a string, we need to introduce a string-related function, `charAt()`. Remember that strings are simply arrays of characters. We can use the string function `charAt()` to directly access a position in the string. If we pass the method 0, it will return the first letter, if we pass it `stringName.length() – 1`, it will give us the last letter. Recall that you can gain the length of a string by using the `length()` method, you must have the empty parentheses.

```java
String test = "peppermint";
System.out.println(test.charAt(0));
System.out.println(test.charAt(test.length() - 1));
```

Output:
```
p
t
```

Why would we bother accessing single characters? char is a primitive data type, so we can compare items that are characters using the `==` operator. To solve this problem iteratively, we should look at the first and last letter of the string, and if they are equal, compare the second and next to last, and so on until we have no more letters, or we find a set that don’t match. Let’s create a method to test this for us.

```java
public static boolean isPalindrome(String s) {
    // base case, a 1 character string is a palindrome
    if (s.length() <= 1) {
        return true;
    } else {
        // loop through the string and compare elements
        for (int low = 0, high = s.length() - 1; low <= high; low++, high--) {
            if (s.charAt(low) != s.charAt(high)) {
                return false;
            }
        }
    }
    return true;
}
```

Class Exercise 48

We can now convert this to a recursive method using the same procedure from the last lecture.

```java
public static boolean isPalindrome(String s, int low, int high) {
    if (low >= high) {
        return true;
    } else if (s.charAt(low) != s.charAt(high)) {
        return false;
    } else {
        isPalindrome(s, low + 1, high - 1);
    }
    return true;
}
```
This version uses the two pointers to keep track of where we are in the string. Once our pointers have crossed (think binary search), we know we can quit examining. Otherwise, we adjust the pointers and call \texttt{isPalindrome()} again. A common technique with recursive methods is to call the recursive method with a convenience method. We could create an overloaded \texttt{isPalindrome()} method with no parameters that simply calls the above method with the proper parameters.

\begin{verbatim}
public static boolean isPalindrome(String s) {
    return isPalindrome(s, 0, s.length() - 1);
}
\end{verbatim}

This allows us to call the method and only pass it a string. We do not have to worry about calculating the indices in the call to the method. The method with the extra parameters is known as a recursive helper method.

\textbf{Selection Sort}

We can also create a version of selection sort that runs recursively. Remember that selection sort finds the smallest number in a array and swaps it with the first number. It then considers the first element sorted and works on the rest of the array until all numbers have been sorted.

\begin{verbatim}
public static void selectionSort(double[] data) {
    selectionSort(data, 0, data.length - 1);
}
\end{verbatim}

\begin{verbatim}
public static void selectionSort(double[] data, int low, int high) {
    if (low < high) {
        int minIndex = low;
        double min = data[low];
        for (int i = low + 1; i <= high; i++) {
            if (data[i] < min) {
                min = data[i];
                minIndex = i;
            }
        }
        data[minIndex] = data[low];
        data[low] = min;
        selectionSort(data, low + 1, high);
    }
}
\end{verbatim}