		AP CS Java
	Th	AP Style FRQ
	F	Lesson 7: Inheritance
Mar. 2	M	Lesson 8: Inheritance Overriding Methods
	T	
	W	
	Th	Assignment 3: Ultimate Frisbee
	F	Assignment 3: Ultimate Frisbee
Mar. 9	M	Quiz 2
	T	Lesson 10: Is-a Has-a Relationships
	W	
	Th	Lesson 11: Interfaces
	F	Assignment 4: Fraction Comparable
Mar. 16	M	Assignment 4: Fraction Comparable
	T	Lesson 12: Wrapper Classes
	W	
	Th	Exam 2, Lesson 13: Algorithms
	F	Lesson 14: Linear Search
Mar. 23	M	Spring Break: Lab 2: Elevens
Mar. 30	M	Quiz 3, Assignment 5: Game Wheel
	T	Assignment 5: Game Wheel
	W	Lesson 15: Selection Sort

AP FRQ: Word Sandwich

- on the paper write (by hand) your code from the FRQ
- Let's grade together

GOAL no. 1: Familarize ourselves with AP Board grading

RUBRIC

Part A:	`makeSandwich`	**3 points**

+1 returns substrings consisting of a "first" and "last" part of `str1`. Substrings can overlap by 1 or 1 letter may be missed.

+1 correctly returns first `n` characters and remaining characters of `str1`

+1 combines the two substrings of `str1` with `str2` in-between

Part B:	`allSandwiches`	**6 points**

+1 creates and returns an array

+1 array returned is of correct length

+1 loops over each element of array/through each relevant letter of `str1`

+2 calls `makeSandwich` with correct arguments

- +1 `int` argument is off by no more than 1 (e.g. `k` used instead of `k + 1`)
- +1 all arguments are correct

+1 stores each new `String` created by `makeSandwich` in the correct position in the result array

2/18/2020

AP-Style FRQ Word Sandwich

Part A

```
private String makeSandwich(String str1, String str2,
                            int n)
{
   return str1.substring(0,n) + str2 + str1.substring(n);
}
```

n → end

from 0 → n

Cut this from the word

Part B

```
private String[] allSandwiches(String str1, String str2)
{
   String[] result = new String[str1.length()-1];
   for (int k=0; k< result.length; k++)
   {
      result[k] = makeSandwich(str1, str2, k+1);
   }
return result;
}
```

k	Result	(k+1)	Arrays start @ 0
0			

```
Word Sandwich

Part A:

private String makeSandwich(String str1, String str2, int n)
{
 return str1.substring(0,n) + str2 + str1.substring(n);
}

Part B:

private String[] allSandwiches(String str1, String str2)
{
 String[] result = new String[str1.length() - 1];

 for (int k = 0; k < result.length; k++)
 {
 result [k] = makeSandwich(str1, str2, k + 1);
 }

return result;
}
```

AP FRQ: High Scores

Rubric

Part A:	`moveUp`	**3 points**

+1 retrieves the value of the `highScore` of the Player at index `pos` of `scoreboard` using the method `getHighScore()`

+1 compares the `highScore` of the Player at index pos to the `highScore` of Players at smaller indices

+1 inserts the `Player` previously at index `pos` of `scoreboard` into the correct place in `scoreboard` (i.e. before the first player with a lower score)

Part B:	`newFirstRow`	**6 points**

+1 uses `getName` method for elements on `scoreboard` and compares these to `name` argument

+1 if a `Player` on `scoreboard` has a matching name, changes the score by calling `updatescore` with the `score` value

+1 calls `moveUp` on the index of the modified `Player` object

+1 if no matching name is found, creates new `Player` object at the end of scoreboard

+1 calls `moveUp` on the last index of `scoreboard` if a new `Player` is added

+1 returns the correct boolean value (i.e. true if

2/10/2020

AP-Style FRQ: High Scores

Part A:

```
private void moveUp(int pos)
{   int k=0;
    while (k<pos && scoreboard.get(k).getHighScore() >
                    scoreboard.get(pos).getHighScore())
        k++;
    scoreboard.add(k, Scoreboard.remove(pos));
}
```

Part B:

```
public boolean newScore (string name, int score)
    {
        for ( int k=0; k< scoreboard.size(); k++ )
        {
            if( scoreboard.get(k).getName().equals (name))
            {
                scoreboard.get(k).updateScore (score);
                moveUp(k);
                return false;
            }
        }
        scoreboard.add( new Player (name, score));
        moveUp(scoreboard.size() - 1);
        return true;
}
```

```
High Scores

Part A:

private void moveUp(int pos)
{
  int k = 0;
  while (k < pos && scoreboard.get(k).getHighScore() >
         scoreboard.get(pos).getHighScore())
    k++;
  scoreboard.add(k, scoreboard.remove(pos));
}


Part B:

public boolean newScore(String name, int score)
{
  for (int k = 0; k < scoreboard.size(); k++)
  {
    if(scoreboard.get(k).getName().equals(name))
    {
      scoreboard.get(k).updateScore(score);
      moveUp(k);
      return false;
    }
  }

  scoreboard.add(new Player(name, score));
  moveUp(scoreboard.size()-1);
  return true;
}

```

AP : FRQ

ROUTE

Rubric

Rubric: AP FRQ Practice - Route

Part A:	`getTotalDistance`	**4 points**

+1 Creates and returns `double` value

+1 Iterates through all elements but one in `waypoints` (no bounds errors)

+1 Uses `get` to return two adjacent elements in `waypoints`

+1 Correctly calls the `distTo` function on each pair of adjacent elements and adds this to the returned total

Part B:	`shortestDistance`	**5 points**

+1 Initializes `double` variable with appropriate value (i.e. a distance between two waypoints or maximum possible `double` value) and returns this at the end

+1 Uses nested loops in an attempt to iterate through each pair of elements in `waypoints`

+1 Iterates through <u>every</u> pair of elements in `waypoints`, but <u>without</u> any element being selected with itself as a pair.

+1 Correctly calls the `distTo` function on pairs of elements from `waypoints`

+1 Changes the value of the returned variable for each shorter distance found

Route.txt

```
Route

Part A:

public double getTotalDistance()
{
  double totalDist = 0;
  for (int k = 0; k < waypoints.size() - 1; k++)
  {
    totalDist += waypoints.get(k).distTo(waypoints.get(k + 1));
  }
  return totalDist;
}

Part B:

public double shortestDistance()
{
  double shortest = waypoints.get(0).distTo(waypoints.get(1));
  for(int j = 0; j < waypoints.size() - 1; j++)
  {
    for(int k = j + 1; k < waypoints.size(); k++)
    {
      double dist = waypoints.get(j).distTo(waypoints.get(k));
      if (dist < shortest)
      {
        shortest = dist;
      }
    }
  }
  return shortest;
}
```

AP FRQ: Reverser

Rubric

Rubric: AP FRQ Practice - Reverser

Part A:	`reverseWord`	**2 points**

+1 Iterates through all valid indices of characters in `word` and gets substrings consisting of each of these characters

+1 Builds a `String` using these letters concatenated in reverse order to appearance in `word` and returns this `String`

Part B:	`reverseAllWords`	**7 points**

+1 Finds index of first space in `sentence`

+1 Splits sentence into two parts, one part being the sentence from index of first space (+1) to the end of the sentence

+1 Calls `reverseWord` on substring of `sentence` from start up to the index of first space

+1 Uses recursion or iteration to repeat procedure above on the remaining part of the sentence. Reversed word is cocatenated to the end of the first part of the sentence each time

+1 Reverses final word of sentence (i.e. when index of space is 0)

+1 Returns completed `String` with all words reversed (may include additional or missing spaces or missing period)

+1 Ensures there are spaces between each adjacent pair of reversed words, a period at the end, and no additional spaces (e.g. before the start, at end, before period)

```
Reverser

Part A:
public static String reverseWord(String word)
{
  String result = "";
  for(int k = word.length(); k > 0; k--)
  {
    result += word.substring(k - 1,k);
  }
  return result;
}

Part B (iterative):

public static String reverseAllWords(String sentence)
{
  String firstPart = "";
  String lastPart = sentence.substring(0, sentence.length() - 1);
  int pos = lastPart.indexOf(" ");
  while (pos >=0)
  {
    firstPart += reverseWord(lastPart.substring(0, pos)) + " ";
    lastPart = lastPart.substring(pos + 1);
    pos = lastPart.indexOf(" ");
  }
  return firstPart + reverseWord(lastPart) + ".";
}

Part B (recursive):

public static String reverseAllWords(String sentence)
{
  int pos = sentence.indexOf(" ");
  if(pos < 0)
  {
    return reverseWord(sentence.substring(0, sentence.length() - 1))
    + ".";
  }
  else
  {
    String firstPart = reverseWord(sentence.substring(0, pos));
    String lastPart = sentence.substring(pos + 1);
    return firstPart + " " + reverseAllWords(lastPart);
  }
}
```