ITCS 209 Object Oriented Programming	Name:	Lab Score	Challenge Bonus	
	ID:			
	Section:			

Lab13: Sorting

You are to create a program of **Sorting.java** (not provided) that will read a text file, sort the words (delimited by white space characters), then print out the sorted words. Your class should have at least three methods.

1. The read method: the ***static*** method will accept a file name as a parameter, read the textual content in a given file, and return these words in either an array or a list of strings.

2. The print method: the ***static void*** method will accept an array or a list of strings (as compatible with the first method) and print out all the words separated by commas in one line.

3. The sort method: the ***static void*** method should accept an array or a list of strings (as returned by the read method), then sort it in **descending** order (b comes before a) using one of the sorting algorithms discussed in class (i.e., Selection, Bubble, Insertion, or MergeSort). **After each pass, call the second method to print out the immediate result.** You must implement the sorting algorithm from scratch (i.e. You may not use Arrays.sort() and Collections.sort()). Be prepared to explain (in **English**) how your sorting algorithm works from the output.

Example output (Insertion Sort):

```
Original: [ink, steak, shark, lunch, brick, cheque, software, ignore, start, wood]
Pass 1: [steak, ink, shark, lunch, brick, cheque, software, ignore, start, wood]
Pass 2: [steak, shark, ink, lunch, brick, cheque, software, ignore, start, wood]
Pass 3: [steak, shark, lunch, ink, brick, cheque, software, ignore, start, wood]
Pass 4: [steak, shark, lunch, ink, brick, cheque, software, ignore, start, wood]
Pass 5: [steak, shark, lunch, ink, cheque, brick, software, ignore, start, wood]
Pass 6: [steak, software, shark, lunch, ink, cheque, brick, ignore, start, wood]
Pass 7: [steak, software, shark, lunch, ink, ignore, cheque, brick, start, wood]
Pass 8: [steak, start, software, shark, lunch, ink, ignore, cheque, brick, wood]
Pass 9: [wood, steak, start, software, shark, lunch, ink, ignore, cheque, brick]
```

Hint: str1.compareTo(str2) returns **0** if `str1` is equal to `str2`; a **negative value** if `str1` is lexicographically less than `str2`; and a **positive value** if `str1` is lexicographically greater than `str2`.

Finally, your **main method** should demonstrate how the above three methods work. You can use the provided text file as an input or create your own text file (make sure it contains at least ten words).

Note: The Movie.java & SoritngMovie.java will be used in the challenge section only.

Challenge Bonus (Optional):

Your task is to sort the movies in **ascending** order based on multiple criteria, the title ('a' comes before 'b'), released year (year 2012 comes before 2019), and movie ID (mid 2 comes before 3), respectively. If both movies have the same titles, their released years will be checked. If, again, their release years are equal, their movie ID will be compared.

== unsorted movie list ==	== sorted movie list (ascending) ==
[mid:1 |The Intern |2009]	[mid:7 |American Ultra |2019]
[mid:2 |The Gift |2009]	[mid:5 |Pasolini |2012]
[mid:3 |The Lost Room |2009]	[mid:8 |Sweet Red Bean Paste |2019]
[mid:4 |The Gift |2012]	[mid:2 |The Gift |2009]
[mid:5 |Pasolini |2012]	[mid:4 |The Gift |2012]
[mid:6 |The Intern |2009]	[mid:1 |The Intern |2009]
[mid:7 |American Ultra |2019]	[mid:6 |The Intern |2009]
[mid:8 |Sweet Red Bean Paste |2019]	[mid:3 |The Lost Room |2009]

As shown in the sorted list above, the movie mid 7 comes before mid 5 because "American Ultra" is lexicographically less than "Pasolini." The movie mid 2 and mid 4 have the same title, but mid 2 is released before mid 4, so mid 2 comes before mid 4. The movie mid 1 and mid 6 have the same title and released year, so we have to compare their mids. As a result, mid 1 comes before mid 6 because the value of 1 is smaller than the value of 6.

The `Movie` class implementing the `Comparable` interface is provided. You have to complete the unimplemented method **`int compareTo(Movie m).`**

For this method, `m1.compareTo(m2)` **returns 0** if m1 is equal to m2 (i.e., they have the same title, release year, and mid); **return a negative** value if m1 comes before m2, and **return a positive** value if m1 comes after m2.

Write a **sorting algorithm** in `SortingMovie.java` class and display the sorted movie list in ascending order.