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

Lab14: Sorting

You are to create a program of **WordProcessor.java** to process on the list of words. This class will 1) read a text file, 2) print out the words, 3) sort the words (delimited by white space characters), and 3) remove duplicate words. The first two methods i.e., `readFile` and `printWords` are provided in the starter code. Your task is to implement the remaining two methods:

1. The sort method should be a **static void** method that accepts a list of strings as input and sorts it in **descending order** (with "b" coming before "a"). The method should use one of the sorting algorithms discussed in class, such as Selection, Bubble, Insertion, or MergeSort. **After each pass, the second method should be called to print out the immediate result**. It is important to note that you are not allowed to use Arrays.sort() or Collections.sort() and must implement the sorting algorithm from scratch. Test your program with the **Main.java** file and be prepared to explain (in **English**) how your sorting algorithm works based on the output.

Example output (Insertion Sort):

```
Original List:
[ink, data, ink, data, test, run, world, tech, mango]
-------------------------------------

Pass 1: [ink, data, ink, data, test, run, world, tech, mango]
Pass 2: [ink, ink, data, data, test, run, world, tech, mango]
Pass 3: [ink, ink, data, data, test, run, world, tech, mango]
Pass 4: [test, ink, ink, data, data, run, world, tech, mango]
Pass 5: [test, run, ink, ink, data, data, world, tech, mango]
Pass 6: [world, test, run, ink, ink, data, data, tech, mango]
Pass 7: [world, test, tech, run, ink, ink, data, data, mango]
Pass 8: [world, test, tech, run, mango, ink, ink, data, data]
```

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`.

2. The removeDuplicate method should be a **static void** method that accepts a list of strings as input and removes all duplicates from the list. For instance, if the list contains "ink sort make ink sort ink", then the method should change the list to "sort make ink". Note that the resulting list may not have the same ordering as the original list. One approach to implement this method is to sort the list of words first. Then, for each element in the list, look at its next neighbor to determine if it is present more than once. If so, remove it.

Example output :

```
Original List:
[ink, data, ink, data, test, run, world, tech, mango]
-------------------------------------

No Duplicate List:
[world, test, tech, run, mango, ink, data]
```

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: title ('a' comes before 'b'), released year (year 2012 comes before 2019), and movie ID (mid 2 comes before 3). If two movies have the same title, their released years will be compared. If their release years are equal, their movie ID will be used for comparison.

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

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