CSD 3464 – ASSIGNMENT 05 (Question 21)

Overview:

The following question is an **extension** to *Absolute Java* (6th Ed.)'s Chapter 06 Programming Project Q21 (pg. 425). Please follow the instructions included in **this** document and implement the following Java files:

- ⇒ HighScoreDriver.java *(Contains a `main()` method)*
- ⇒ HighScoreList.java
- ⇒ Player.java (Completed for you!)
- ⇒ Name.java (Completed for you!)

The above classes/files should both be inside a package called *q21*.

Instructions:

Start by reviewing the `Player` and `Name` classes provided. You will use these classes to complete this assignment question. You should not need to modify these classes in anyway.

`Players` have both a `name` and `score` field. The `name` is an instance of the `Name` class and `score` is a positive `int` value. The `Player` class also comes with a set of accessor and mutator methods as well as predefined `equals` and `toString` methods you may find useful throughout this question.

The `HighScoreList` class is used to hold up to 10 `Players` and their high scores in the field `scoreList`. The class only has one constructor with no parameters that creates an empty list of `Players` of size 10 and initializes the `numPlayers` fields to 0. This constructor has been provided to you in full.

Initially upon the creation of a `HighScoreList` object the `scoreList` field contains 10 `null` values in the array. Additionally, `HighScoreList`'s field `numPlayers` is set to 0 as there are no `Player` object's in the array yet.

null	null	null	null	null	null	null	null	null	null

Your first task is to implement a `public void` method called `addPlayer` inside of the `HighScoreList` class that takes a `Player`'s `name` as `Sting` and their `score` as an int and adds the player to the `scoreList` array.

- The `addPlayer` method should add a `Player` object, with specified `name` and `score`, to the **correct** location of the `scoreList` array. The `scoreList` should have the `Player` with the highest `score` at index 0 with `Players` with decreasing `scores` decrease from there.

- To accomplish this loop over each index of the `scoreList` and check if the value of the `Player` object's `score` you wish to add is greater than that `Player` at the current index.
  - If the `score` is greater, you should insert the parameter `Player` object at the current index and shift over all current `Players` object from this index on over by 1; for example, assuming the below `ScoreList`

"Aaron"	"Joe"	"Jill"	"Sam"	"Terry"	"Frank"	"Terry"	"Erica"	null	null
100	98	90	71	55	45	40	32		

And wishing to add a `Player` object with `name` of `"Alice"` with a `score` of `50` the `ScoreList` should be updated to:

"Aaron"	"Joe"	"Jill"	"Sam"	"Terry"	"Alice"	"Frank"	"Terry"	"Erica"	null
100	98	90	71	55	50	45	40	32	

- If a `Player` is inserted into `scoreList` be sure to increment the `numPlayers` field by `1`, unless the `numPlayers` is already 10 as this represents a full `HighScoreList`.
  - If the `scoreList` is full (with 10 `Players`) and you insert a `Player`, the lowest score `Player` (at index 9) would be dropped from `scoreList`

Next you are required to implement a `public void` method named `removePlayer` that takes a `String` and removes **all** `Players` from the `ScoreList` who have the same `name`.

- You should use `Name`'s `equals` method to compare if two `Player` objects have the same `names`
  - **HINT:** To access `Player`'s name use its `getName()` method. You may call `Name`'s `equals()` on the object it returns
- If you find a `Player` object to remove from `scoreList` you can replace that object with the value `null`. For example, if you wanted to remove `"Terry"` from the below array

"Aaron"	"Joe"	"Jill"	"Sam"	"Terry"	"Frank"	"Terry"	"Erica"	null	null

you should end up with the updated array:

"Aaron"	"Joe"	"Jill"	"Sam"	"Frank"	"Erica"	null	null	null	null

- **HINT:** In the above we removed `"Terry"` twice. After matching the first `"Terry"` add all `Player` objects from that point, which are not `"Terry"`, into a local array inside the method.

"Frank"	"Erica"	null	null	null	null	null	null	null

- After the construction of the above local array you can insert each of its non-`null` elements into `scoreList` from the index of the first `"Terry"` and ensure every element beyond that is `null`.

"Aaron"	"Joe"	"Jill"	"Sam"	"Frank"	"Erica"	null	null	null	null

- **REMEMBER: Each time** a `Player` object is removed from the array remember to decrement the field `numPlayers` by 1.

Furthermore, the class `HighScoreList` must also include the method `public toString()` which returns a `String` containing all the `Player`'s `name` and their `score` as a single `String` object. Each line in the returned `String` object should be in the format:

"#. Name: `name` Score: `score`\n"

- `#` above should be replaced with the value 1 to `numPlayers` based on their position in `scoreList`
- `name` above should be in the format returned by the `Name` class' `toString` method
- `score` is the value stored in the field score

The last method that needs to be implemented in the `HighScoreList` class is called `findPlayerScore` that takes a `String name` and returns the `score` of that `Player` matching the provided `name`. If more than one match exists return the first match, if no matches exist then return the value `-1`.

- Your first step should be to convert the `String name` to a `Name` object
- After which you should use the `Name's equals` method to check if two `Player` objects have the same `Name`.

You are also required to write a `static` class called `HighScoreDriver` in another file that contains a `main()`. The purpose of this class is to test the functionality of your `HighScoreList` class. This class has a lot completed for you, including the generation of a menu and returning a valid input selection from that menu. What you need to finish is the series of //TODO blocks inside of the `switch` structure to make the program execute properly.

- The `'A'` selection should prompt the user for a player's full name (separated by spaces) and positive integer value for score. With these values a `Player` should be added to `highScores` by calling one of the `HighScoreList` methods you developed.

- The `'B'` selection should print all the players and their scores to their console.
  - **Hint:** Use the `HighScoreList`'s `toString()` method

- The `'C'` selection should prompt the user for a player's full name (separated by spaces). With this name generate a `Name` object and call the appropriate method from `HighScoreList` to find the score of the player with that name, if any.
  - If no player exists display the message: "`Sorry the player, you are searching for does not exist!`"

- The `'D'` selection should prompt the user for a player's full name (separated by spaces).

With this name generate a `Name` object and call the appropriate method from `HighScoreList` to remove all `Player`s with that name from the local `highScores` object.

- The `'E'` selection should print the following message to the console:
  `"Thank-you for using our high score application! Good-bye."`