New University logo

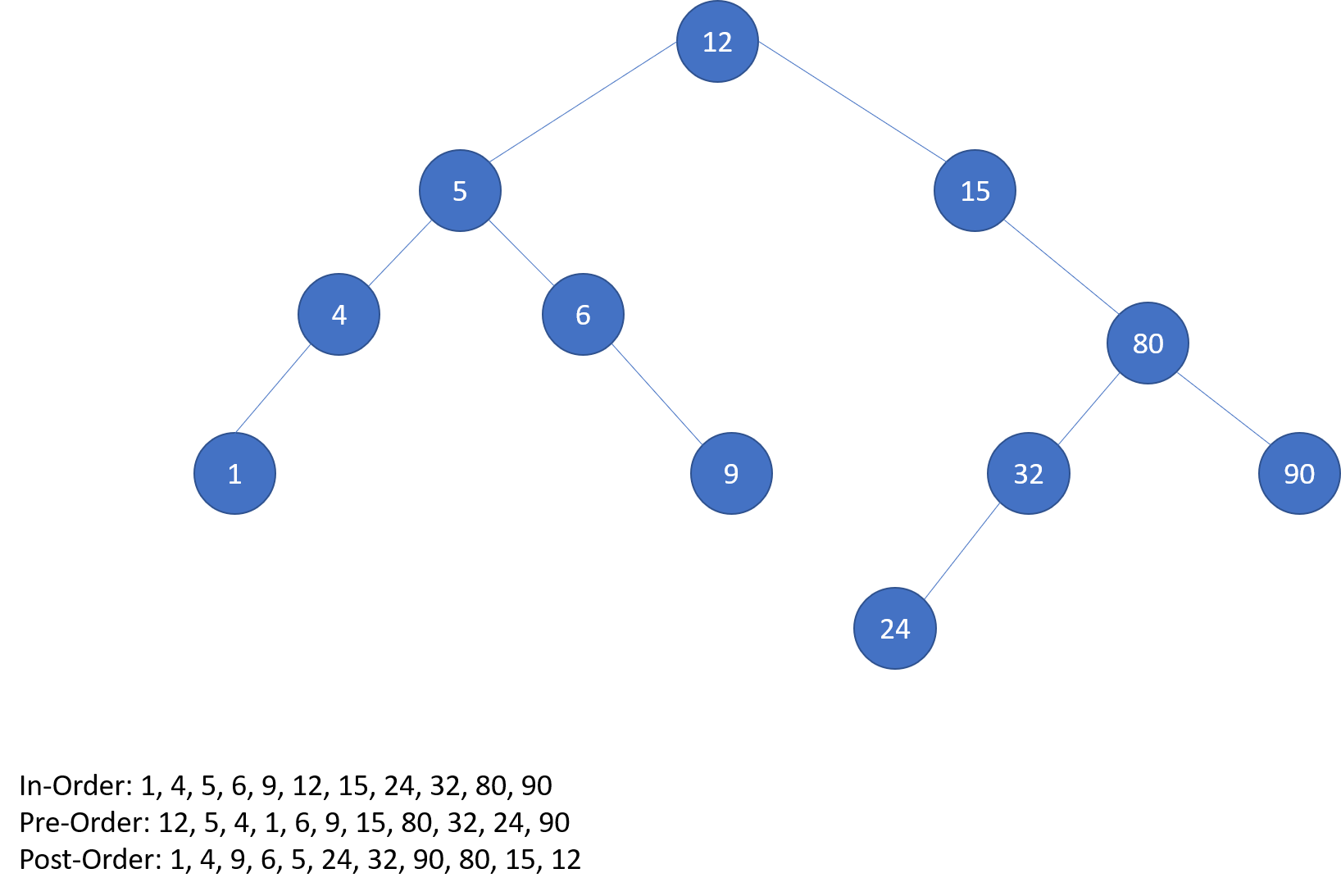

|  |  |  |  |
| --- | --- | --- | --- |
| Academic Year | 2023 | | |
| Semester | Fall | Winter | Summer |
| Course Code - Name | CSCI 2010U – Data Structures | | |
| Instructor | Dr. Razi Iqbal | | |
| Assessment | Mock Final Exam |  | |

**Question 1**

Provided the following numbers, construct a Binary Search Tree. Once constructed, write down the Pre-Order, In-Order and Post-Order for the tree.

12, 5, 6, 15, 80, 32, 4, 9, 24, 90, 1

**Answer:**

****

**Question 2**

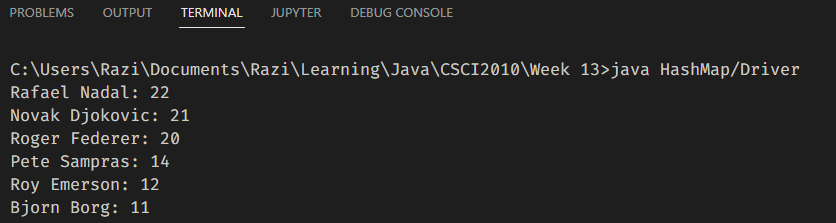
In this exercise, you are provided a list of Tennis Players along with their number of Grand Slam wins of all time and you are required to use a HashMap to list them from highest to lowest wins along with classifying them according to their wins. Please refer to the expected output of the program for more details.

You are provided with Driver class that has a main method which generates the list of players and passes it to a function public static String**[]** sortByWins**(**String**[]** players**,** Integer**[]** wins**).** This function returns a **String[]** which **main** method uses to display the output.The Code for **Driver** class with **main** method is provided below.

Your job is to complete public static String**[]** sortByWins**(**String**[]** players**,** Integer**[]** wins**)** method. Please note the following:

* Use HashMap to write this function and make sure to return a **String[]** that shows the name of the player along with number of wins.
* The function should return a **String[]** in a way that it is sorted by number of wins in descending order.
* The function takes 2 arrays; a **String[]** for player names and **Integer[]** for number of wins. The size of both the arrays will always be same.
* **String[]** always corresponds to **Integer[]**. It means that **String[0] = Integer[0]** and so on. Hence, names of the players would always correspond to wins at the same index.

Below is the expected output of the program:



Below is the Driver class with main function.

package HashMap**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**HashMap**;**

public class Driver **{**

public static String**[]** sortByWins**(**String**[]** players**,** Integer**[]** wins**)**

**{**

// Fill in this function

**}**

public static void main**(**String**[]** args**)** **{**

String**[]** players **=** **new** String**[]{**

"Pete Sampras"**,** "Novak Djokovic"**,** "Roger Federer"**,** "Roy Emerson"**,** "Rafael Nadal"**,** "Bjorn Borg"

**};**

Integer**[]** wins **=** **new** Integer**[]{**

14**,** 21**,** 20**,** 12**,** 22**,** 11

**};**

**for** **(**String result**:** sortByWins**(**players**,** wins**))**

**{**

System**.**out**.**println**(**result**);**

**}**

**}**

**}**

**Answer:**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**HashMap**;**

public class Driver **{**

public static String**[]** sortByWins**(**String**[]** players**,** Integer**[]** wins**)**

**{**

HashMap**<**Integer**,** String**>** map **=** **new** HashMap**<>();**

**for** **(**int i**=**0**;** i**<**players**.**length**;** i**++)**

**{**

map**.**put**(**wins**[**i**],** players**[**i**]);**

**}**

Arrays**.**sort**(**wins**,** Collections**.**reverseOrder**());**

String**[]** result **=** **new** String**[**wins**.**length**];**

**for** **(**int i**=**0**;** i**<**wins**.**length**;** i**++)**

**{**

result**[**i**]** **=** map**.**get**(**wins**[**i**])** **+** ": " **+** wins**[**i**];**

**}**

**return** result**;**

**}**

public static void main**(**String**[]** args**)** **{**

String**[]** players **=** **new** String**[]{**

"Pete Sampras"**,** "Novak Djokovic"**,** "Roger Federer"**,** "Roy Emerson"**,** "Rafael Nadal"**,** "Bjorn Borg"

**};**

Integer**[]** wins **=** **new** Integer**[]{**

14**,** 21**,** 20**,** 12**,** 22**,** 11

**};**

**for** **(**String result**:** sortByWins**(**players**,** wins**))**

**{**

System**.**out**.**println**(**result**);**

**}**

**}**

**}**

**Question 3**

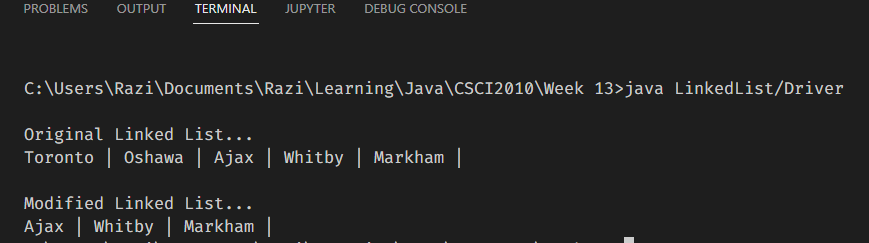
In this exercise, you are provided a list of City names in the form of a Linked List. You are required to remove all the city names that have an ‘O’ or ‘o’ in their names. Please refer to the expected output of the program for more details.

You are provided with Driver class that has a **main** method which generates the Linked List with city names and passes the **head** of this Linked List to a function public static Node removeCitiesWithO**(**Node head**).** This function is expected to remove all the nodes of this Linked List where the city name contains an **‘O’** oran **‘o’** in it.This function returns a **head** of the modified Linked List.Please note that you are not required to create a new Linked List. You have to make changes to the original Linked List.The **main** method uses a public static void printList**(**Node head**)** function to display the Linked List.You are required to write thisprintListfunction as well.The Code for **Driver** class with **main** method is provided below.

Please note the following:

* Create **Node** class as necessary.
* You are not allowed to use any other data structure than Linked List. Also, please do not use built-in Linked List functions.
* You are free to use **String** library functions in Java.
* Make sure your function works for an empty Linked List and if there is only one element in the Linked List that requires deletion. Also, keep an eye on the case where head needs to be deleted and there are still other nodes in the Linked List.

Below is the expected output of the program:



Below is the Driver class with main function.

package LinkedList**;**

public class Driver **{**

public static Node removeCitiesWithO**(**Node head**)**

**{**

**// Fill in this function**

**}**

public static void printList**(**Node head**)**

**{**

**// Fill in this function**

**}**

public static void main**(**String**[]** args**)**

**{**

Node head **=** **new** Node**(**"Toronto"**);**

head**.**next **=** **new** Node**(**"Oshawa"**);**

head**.**next**.**next **=** **new** Node**(**"Ajax"**);**

head**.**next**.**next**.**next **=** **new** Node**(**"Whitby"**);**

head**.**next**.**next**.**next**.**next **=** **new** Node**(**"Markham"**);**

System**.**out**.**println**(**"\nOriginal Linked List..."**);**

printList**(**head**);**

System**.**out**.**println**(**"\n\nModified Linked List..."**);**

printList**(**removeCitiesWithO**(**head**));**

**}**

**}**

**Answer:**

class Node

**{**

String value**;**

Node next**;**

Node**()**

**{**

**this.**value **=** ""**;**

**}**

Node**(**String value**)**

**{**

**this.**value **=** value**;**

**}**

**}**

public class Driver **{**

public static Node removeCitiesWithO**(**Node head**)**

**{**

**if** **(**head **==** **null)**

**{**

**return** **null;**

**}**

Node temp **=** **new** Node**();**

temp**.**next **=** head**;**

Node currentNode **=** temp**;**

**while** **(**currentNode**.**next **!=** **null)**

**{**

**if** **(**currentNode**.**next**.**value**.**contains**(**"O"**)** **||** currentNode**.**next**.**value**.**contains**(**"o"**))**

**{**

currentNode**.**next **=** currentNode**.**next**.**next**;**

**}**

**else**

**{**

currentNode **=** currentNode**.**next**;**

**}**

**}**

**return** temp**.**next**;**

**}**

public static void printList**(**Node head**)**

**{**

Node currentNode **=** head**;**

**while** **(**currentNode **!=** **null)**

**{**

System**.**out**.**print**(**currentNode**.**value **+** " |"**);**

currentNode **=** currentNode**.**next**;**

**}**

**}**

public static void main**(**String**[]** args**)**

**{**

Node head **=** **new** Node**(**"Toronto"**);**

head**.**next **=** **new** Node**(**"Oshawa"**);**

head**.**next**.**next **=** **new** Node**(**"Ajax"**);**

head**.**next**.**next**.**next **=** **new** Node**(**"Whitby"**);**

head**.**next**.**next**.**next**.**next **=** **new** Node**(**"Markham"**);**

System**.**out**.**println**(**"\nOriginal Linked List..."**);**

printList**(**head**);**

System**.**out**.**println**(**"\n\nModified Linked List..."**);**

printList**(**removeCitiesWithO**(**head**));**

**}**

**}**

**Question 4**

In this exercise, you are provided two Binary Trees where each node of the tree is a Student class object containing name and marks as data members. These students have taken a quiz in which they could not perform very well so the professor decided to give them another chance to update their marks by providing them another quiz.

You are required to create a **Student** class with **String** name and **int** marks data members, a parameterized constructor and a **toString** method.

You are provided with a **Driver** class that has a **main** method which generates two binary trees assuming each one is the performance of a student for each quiz. Each node of each binary tree is a Student class object. The trees are designed in a way that they are binary search trees. For example, in the figure below, Steve, Bill, Jeff and Elon appeared for the quiz and got marks for that quiz and tree is built using BST rules. Similarly, in Quiz 2, Steve, Bill, Lizzy, Elon and Sheena appeared for the quiz and got marks for that quiz and tree is built using BST rules. A Final Result tree is built based on the maximum marks of the students from both the quizzes. For example, Steve got more marks in Quiz 2 than Quiz 1, his marks from Quiz 2 should be considered in the final result. Same goes for Bill. Elon got same marks in both the quizzes so his marks won’t change in the final result. However, since Jeff did not appear for Quiz 2, his marks from Quiz 1 are considered in the final result. Similarly, Lizzy and Sheena appeared only in Quiz 2 so their marks from Quiz 2 will be considered in the final result.

|  |  |
| --- | --- |
| **Quiz 1** | **Quiz 2** |
|  |  |
| **Final Result** | |
|  | |

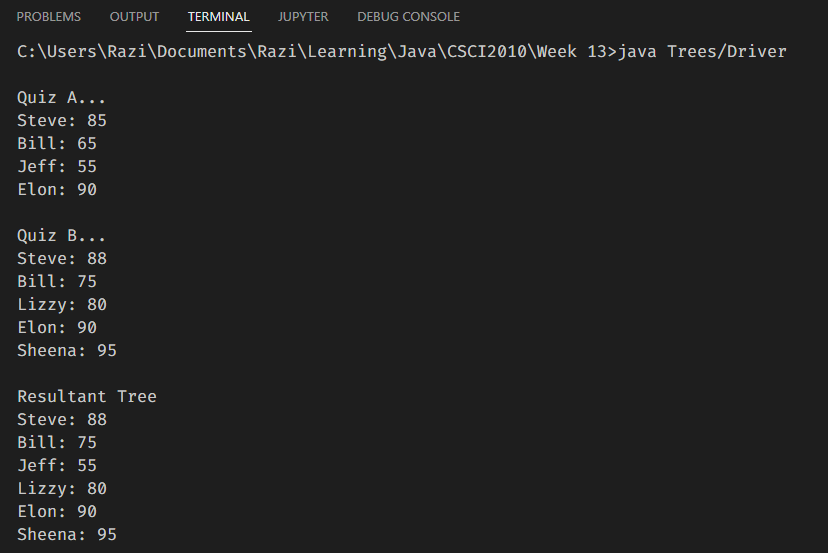
You are required to write a public static Node updateMarks**(**Node root\_A**,** Node root\_B**)** function that takes in both the trees as a parameters of the function and generates a resultant tree such that the maximum marks of the students from each quiz are considered. Please refer to the screenshot above. Also, please note that you are not required to create a new tree and hence should be updating one of the two trees as a final result to avoid using extra space in the RAM.

The **main** method uses a public static void preOrder**(**Node root**)** function to display the trees.You are required to write thispreOrder function as well.

Please note the following:

* Create **Node** class as necessary.
* You are not allowed to use any other data structure than Tree.
* You are free to use **String** library functions in Java.
* Do not forget to write Time and Space complexity of both the functions.

Below is the expected output of the program:



Below is the Driver class with main function.

package Trees**;**

public class Driver **{**

public static Node updateMarks**(**Node root\_A**,** Node root\_B**)**

**{**

**// Fill in this function**

**}**

public static void preOrder**(**Node root**)**

**{**

**// Fill in this function**

**}**

public static void main**(**String**[]** args**)** **{**

Node quiz\_A **=** **new** Node**(new** Student**(**"Steve"**,** 85**));**

quiz\_A**.**left **=** **new** Node**(new** Student**(**"Bill"**,** 65**));**

quiz\_A**.**right **=** **new** Node**(new** Student**(**"Elon"**,** 90**));**

quiz\_A**.**left**.**left **=** **new** Node**(new** Student**(**"Jeff"**,** 55**));**

Node quiz\_B **=** **new** Node**(new** Student**(**"Steve"**,** 88**));**

quiz\_B**.**left **=** **new** Node**(new** Student**(**"Bill"**,** 75**));**

quiz\_B**.**right **=** **new** Node**(new** Student**(**"Elon"**,** 90**));**

quiz\_B**.**left**.**right **=** **new** Node**(new** Student**(**"Lizzy"**,** 80**));**

quiz\_B**.**right**.**right **=** **new** Node**(new** Student**(**"Sheena"**,** 95**));**

System**.**out**.**println**(**"\nQuiz A..."**);**

preOrder**(**quiz\_A**);**

System**.**out**.**println**(**"\nQuiz B..."**);**

preOrder**(**quiz\_B**);**

System**.**out**.**println**(**"\nResultant Tree"**);**

preOrder**(**updateMarks**(**quiz\_A**,** quiz\_B**));**

**}**

**}**

**Answer:**

class Student**{**

String name**;**

int marks**;**

Student**(**String name**,** int marks**)**

**{**

**this.**name **=** name**;**

**this.**marks **=** marks**;**

**}**

public String toString**()**

**{**

**return** **this.**name **+** ": " **+** **this.**marks**;**

**}**

**}**

class Node

**{**

Student student**;**

Node left**;**

Node right**;**

Node**(**Student student**)**

**{**

**this.**student **=** student**;**

**}**

Node**(**Student student**,** Node left**,** Node right**)**

**{**

**this.**student **=** student**;**

**this.**left **=** left**;**

**this.**right **=**right**;**

**}**

**}**

public class Driver **{**

public static Node updateMarks**(**Node root\_A**,** Node root\_B**)**

**{**

**if** **(**root\_A **==** **null)**

**{**

**return** root\_B**;**

**}**

**if** **(**root\_B **==** **null)**

**{**

**return** root\_A**;**

**}**

**if** **(**root\_A**.**student**.**marks **<** root\_B**.**student**.**marks**)**

**{**

root\_A**.**student**.**marks **=** root\_B**.**student**.**marks**;**

**}**

root\_A**.**left **=** updateMarks**(**root\_A**.**left**,** root\_B**.**left**);**

root\_A**.**right **=** updateMarks**(**root\_A**.**right**,** root\_B**.**right**);**

**return** root\_A**;**

**}**

public static void preOrder**(**Node root**)**

**{**

**if** **(**root **==** **null)**

**{**

**return;**

**}**

System**.**out**.**println**(**root**.**student**.**toString**());**

preOrder**(**root**.**left**);**

preOrder**(**root**.**right**);**

**}**

public static void main**(**String**[]** args**)** **{**

Node quiz\_A **=** **new** Node**(new** Student**(**"Steve"**,** 85**));**

quiz\_A**.**left **=** **new** Node**(new** Student**(**"Bill"**,** 65**));**

quiz\_A**.**right **=** **new** Node**(new** Student**(**"Elon"**,** 90**));**

quiz\_A**.**left**.**left **=** **new** Node**(new** Student**(**"Jeff"**,** 55**));**

Node quiz\_B **=** **new** Node**(new** Student**(**"Steve"**,** 88**));**

quiz\_B**.**left **=** **new** Node**(new** Student**(**"Bill"**,** 75**));**

quiz\_B**.**right **=** **new** Node**(new** Student**(**"Elon"**,** 90**));**

quiz\_B**.**left**.**right **=** **new** Node**(new** Student**(**"Lizzy"**,** 80**));**

quiz\_B**.**right**.**right **=** **new** Node**(new** Student**(**"Sheena"**,** 95**));**

System**.**out**.**println**(**"\nQuiz A..."**);**

preOrder**(**quiz\_A**);**

System**.**out**.**println**(**"\nQuiz B..."**);**

preOrder**(**quiz\_B**);**

System**.**out**.**println**(**"\nResultant Tree"**);**

preOrder**(**updateMarks**(**quiz\_A**,** quiz\_B**));**

**}**

**}**