Ton Duc Thang University
Faculty of Information Technology

Assignment

COURSE: **DATA STRUCTURES AND ALGORITHMS**

CODE: **504008**

TOPIC: Student Management using AVL tree

(Students carefully read all instructions before conducting)

I. Problem statement

A university manages students by IDs and needs to perform adding, deleting, and searching students efficiently based on IDs. Because there are risks in adding and deleting, the university requires to **undo** and **redo** functionalities. In addition, there is a functionality to transform keys from student ID to GPA.

Required functionalities for the system are listed below

- Building an AVL tree to store student information where keys are students' IDs.
- Implementing actions of adding, deleting, and searching for students in the AVL tree.
- Implementing actions of undoing and redoing.
- Traversing students regarding the level-order
- Building an AVL tree where keys are students' GPAs.

II. Resources

The attached source code includes:

- Data files, input files and result files
  - *testcase* folder consists of 07 testcases from *testcase1.txt* to *testcase7.txt*.
  - *expected_output* folder consists of 07 outputs from *output1.txt* to *output7.txt*, which are expected results corresponding to 07 testcases above.
  - *list_student.txt*: information of 100 students for testcases.
  - *information.txt*: descriptions of actions in testcase files.
- Source code files:
  - *Main.java*: initializing objects, reading testcase files and list_student.txt, invoking designated methods, and writing results to files.
  - *Student.java* and *Node.java*: **Student** and **Node** classes respectively. **STUDENTS DO NOT MODIFY THESE FILES.**

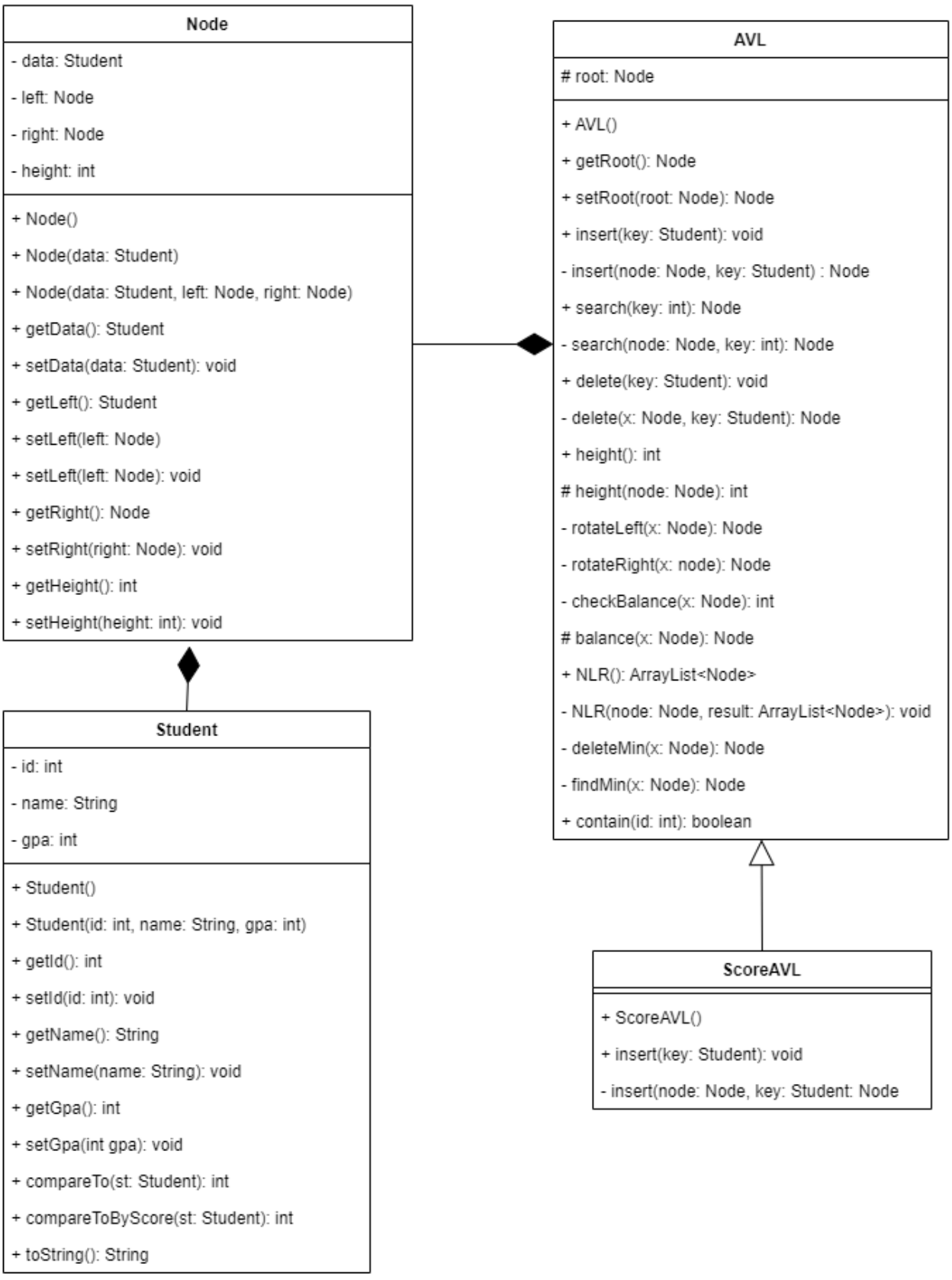
- *AVL.java*: **AVL** class, there exists several supporting methods and ones to build up an AVL tree. Students complete non-implemented methods in this file.
- *StudentManagement.java*: **StudentManagement** class with an attribute of *AVL tree* for storing students. Students complete designated methods in AVL class in advance and the others in this file.
- *ScoreAVL.java*: **ScoreAVL** class, used for Task 6. Students complete this file regarding requirements of Task 6.

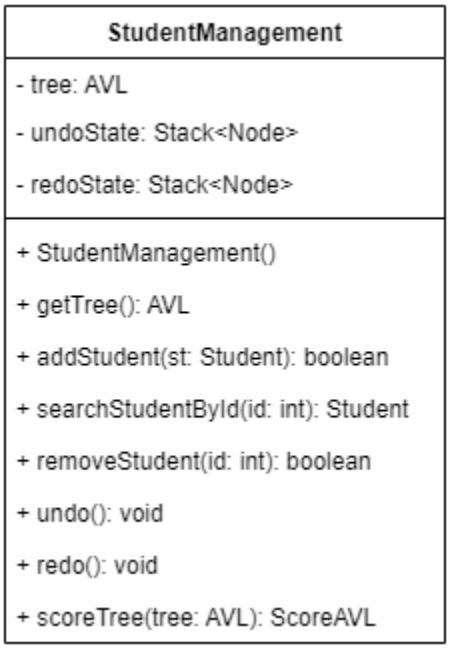
III. Assignment Procedure

- Students download and decompress resources.
- Carefully read task descriptions and provided source code files to understand file structures and actions in testcase files.
- Files *testcase1.txt* to *testcase6.txt* are corresponding to 6 tasks, *testcase7.txt* is a combination of 6 files before.
- Students are able to use *testcase7.txt* when fulfilling 6 tasks before. Testcases for scoring are mostly similar to *testcase7.txt*.
- Naming conventions
  - Task 1: YC1
  - Task 2: YC2
  - Task 3: YC3
  - Task 4: YC4
  - Task 5: YC5
  - Task 6: YC6
- For YC1, YC2, and YC3, students build up the **AVL** tree by completing designated methods in *AVL.java* and then ones corresponding to each task *StudentManagement.java*.
- For YC4, YC5, and YC6, students conduct in **StudentManagement** class. For YC4 and YC5, students must be careful because they affect previous. For YC6, students complete missing methods in *AVLScore.java*.
- After finishing **StudentManagement** class, students compile and execute the program using the **main** method in *Main.java*. Students change arguments regarding to testcases and compare outputs to ones in *expected_output* folder.
- For tasks that students are not able to complete, do not delete related methods and ensure the program can be executed successfully given the main method in Main.java.
- Google Drive folder of the assignment also consists of *version.txt*, students regularly check for updates via this file. If there exist any updates, then carefully read the descriptions and download the newest resources. *Version.txt* provides update contents and dates in case the assignment description is not clear.

IV. Descriptions of Student, Node, ScoreAVL, and StudentManagement classes

- Descriptions of class attributes and methods are as below.
  - **Student** (***read-only, do not modify this file***):
    - *id*: student ID
    - *name*: full name
    - *gpa*: grade point average
    - **compareTo(Student st):** compare two students based on IDs
    - **compareToByScore(Student st):** compare two students based on GPAs.
    - **toString():** returns string with student information.
  - **Node** (***read-only, do not modify this file***):
    - *data:* a ***Student*** object
    - *left*, *right:* left and right children
    - *height:* height of the node
  - **AVL**:
    - Create an AVL tree using student IDs as keys.
    - *root:* the root node
    - Several provided methods, including **height**, **rotateRight**, **rotateLeft**, **checkBalance**, **balance**, **NLR**, etc.
  - **ScoreAVL**:
    - A child class of the AVL class, is used to create an AVL tree with keys as GPAs.


Node
- data: Student
- left: Node
- right: Node
- height: int
+ Node()
+ Node(data: Student)
+ Node(data: Student, left: Node, right: Node)
+ getData(): Student
+ setData(data: Student): void
+ getLeft(): Student
+ setLeft(left: Node)
+ setLeft(left: Node): void
+ getRight(): Node
+ setRight(right: Node): void
+ getHeight(): int
+ setHeight(height: int): void
AVL
root: Node
+ AVL()
+ getRoot(): Node
+ setRoot(root: Node): Node
+ insert(key: Student): void
- insert(node: Node, key: Student) : Node
+ search(key: int): Node
- search(node: Node, key: int): Node
+ delete(key: Student): void
- delete(x: Node, key: Student): Node
+ height(): int
# height(node: Node): int
- rotateLeft(x: Node): Node
- rotateRight(x: node): Node
- checkBalance(x: Node): int
balance(x: Node): Node
+ NLR(): ArrayList<Node>
- NLR(node: Node, result: ArrayList<Node>): void
- deleteMin(x: Node): Node
- findMin(x: Node): Node
+ contain(id: int): boolean
Student
- id: int
- name: String
- gpa: int
+ Student()
+ Student(id: int, name: String, gpa: int)
+ getId(): int
+ setId(id: int): void
+ getName(): String
+ setName(name: String): void
+ getGpa(): int
+ setGpa(int gpa): void
+ compareTo(st: Student): int
+ compareToByScore(st: Student): int
+ toString(): String
ScoreAVL
+ ScoreAVL()
+ insert(key: Student): void
- insert(node: Node, key: Student: Node



- Descriptions of class attributes and methods are as below.
  - **StudentManagement**:
    - *tree*: an ***AVL*** object whose keys are student IDs.
    - *undoState*: state to be undone
    - *redoState*: state to be redone
    - **addStudent(Student st):** add a student to the *tree* (Task 1).
    - **searchStudentById(int id):** search for a student in the *tree* given his **id** (Task 2).
    - **removeStudent(int id):** delete a student given his **id** (Task 3).
    - **undo():** perform undoing (Task 4)
    - **redo():** perform redoing (Task 5)
    - **scoreTree(AVL tree):** create an AVL tree whose keys are student GPAs from an AVL tree whose keys are student IDs (Task 6).

V. Descriptions of text data files

- *information.txt* consists of actions and their order, for example
  - 0: add a student to the AVL tree
  - 1: write down the result of NLR traversal in the current tree
  - 2: search for a student given his ID
  - 3: delete a student given his ID
  - 4: perform undoing

- 5: perform redoing
- 6: returns an AVL tree whose keys are student GPAs from the current tree whose keys are student IDs

- Input files for execution include *list_student.txt* and *testcase**X**.txt* with ***X*** as a testcase order. There 07 testcases in **testcase** folder.
- The structure of *list_student.txt* is as below:

Student ID, Full name, GPA

list_student.txt
```
0,Hugo Clark,4
1,Anita Mansell,62
2,Teddy Neal,30
3,Anaiya Cobb,7
4,Maximilian Mcclain,18
5,Prisha Clemons,71
```

- *testcast**X**.txt* consists of actions regarding to the format below:

Action-order <space> [***id*** argument if exist]

testcase > testcase7.txt
```
8   0 68
9   0 6
10  0 12
11  1
12  2 68
13  3 12
14  4
15  1
16  2 12
17  3 98
18  1
19  0 93
20  0 33
21  0 32
22  4
23  4
24  4
25  4
26  1
27  5
28  5
29  1
30  0 48
```

Explanation for line 8 to 15 above:

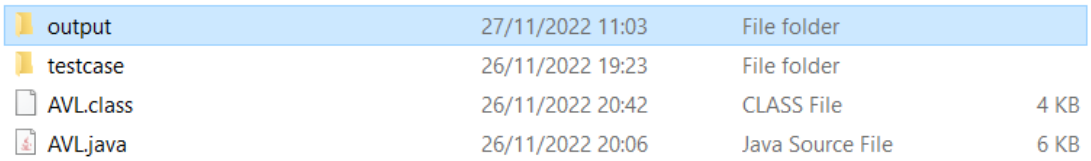
    - Add a student whose ID is 68
    - Add a student whose ID is 6
    - Add a student whose ID is 12
    - Write down the result of NLR traversal in the current tree
    - Find and print information of the student whose ID is 68
    - Delete the student whose ID is 12
    - Undo
    - Write down the result of NLR traversal in the current tree
- Each testcase may consist of many actions. To support students' work, testcases from 1 to 6 consist of only action kinds as below.
    - *testcase1.txt*: adding, deleting, and NLR traversing only.
    - *testcase2.txt*: adding and searching only.
    - *testcase3.txt*: adding, deleting, and NLR traversing only.
    - *testcase4.txt*: adding, deleting, NLR traversing, and undoing.
    - *testcase5.txt*: adding, deleting, NLR traversing, undoing, and redoing.
    - *testcase6.txt*: building up GPA-based AVL tree only.
    - *testcase7.txt*: many kinds of actions (high-score testcases are similar to *testcase 7*)
- The *expected_output* folder consists of 07 results files corresponding to 07 testcases.

Notice:

- Students may add extra data and testcases to evaluate the program in many cases; however, you have to ensure the consistent structure of data files as above.
- Students carefully read the **main** method to find out approaches to complete tasks in the designated order.
- Students may add extra actions in the **main** method to verify completed methods; however, you have to ensure **the program can be executed successfully with the original main method**.
- Students may add extra methods to support your work; however, new methods must be declared and implemented in files for submission and the submission can be executed successfully with the given *Main.java*.
- DO NOT MODIFY **CLASS AND METHOD NAMES**, STRICTLY FOLLOW THE GIVEN CLASS DIAGRAM.
- Students do not modify files not attached in the submission.

VI. Execution instructions

- The **main** method in *Main.java*, is going with supporting methods for reading files and performing actions. Students strictly complete designated methods so that the **main** method reads files and invokes corresponding methods correctly.
- To evaluate tasks, students must fully complete methods following the requirement, after compiling successfully, create a folder named **output** in the same folder of source code files.



- The **main** method contains pre-defined paths to read *testcase**X**.txt* from **testcase** folder and write to *output*X.*txt* in **output** folder.
- Students execute the program using the command below to evaluate

  java Main X

  in which *X* is the testcase order (*testcase**X**.txt*) that students desire to use.
  For example, students want to use testcase 1, then run this command

  java Main 1

  the result is written to *output1.txt* in **output** folder.
- After evaluating using given testcases, students may create your own testcases and use them for evaluation.

VII. Tasks

Students are **not allowed to use additional libraries** except the given ones in source code files.

Students conduct the assignment in Java 11 or Java 8. Students are not allowed to use *var* data type. Submissions are judged in Java 11 and students take your own responsibility for errors caused by Java version differences.

In given source code files, file reading and writing instructions are pre-defined, hence students do not need to implement by yourselves nor modify them to avoid mistakes.

1. TASK 1 (2.0 points)

Students implement the method

private Node insert(Node node, Student key)

to add a new student to the AVL tree using recursion. Note that the tree must be balanced after the action (Students may use supporting methods provided in **AVL** class).

After that, for **StudentManagement** class**,** implement the method:

public boolean addStudent(Student st)

to add a new student to the *tree*. If the provided student object has a non-existing ID, then return *true*; otherwise, do not add the object and return *false*.

Main class is provided with actions of reading data files, invoking the method above, and writing down the full names of students in the *tree* in NLR traversal order. Students compile and run

java Main 1

The program reads *testcase1.txt* and writes to *output1.txt* in the **output** folder. Students compare this file *output1.txt* in the **expected_output** folder to evaluate your work.

Note: This is the task that students must complete correctly to gain scores from later tasks.

2. TASK 2 (1.0 point)

In **AVL** class, students implement the method

private Node search(Node x, int key)

to find a node given a **key**. For **StudentManagement** class, students complete the method below:

public Student searchStudentById(int id)

to find a student given his **id** as the argument. If there exists a student with the given id, then return the student object; otherwise, return *null*.

Main class is provided with actions of reading data files, invoking the method above, and writing down information about the found student. Students compile and run

java Main 2

The program reads *testcase2.txt* and writes to *output2.txt* in the **output** folder. Students compare this file *output2.txt* in the **expected_output** folder to evaluate your work.

3. TASK 3 (2.0 points)

In **AVL** class, students implement the method

private Node delete(Node x, Student key)

to delete a node given its **key**. Note that students must rebalance the tree after deleting (Students may use supporting methods provided in **AVL** class).

For **StudentManagement** class, students implement the method below:

public boolean removeStudent(int id)

to delete a student given his **id** as the argument. If there exists the student, then delete and return *true*; otherwise, do not delete and return *false*.

Main class is provided with actions of reading data files, invoking the method above, and writing files. Students compile and run

java Main 3

The program reads *testcase3.txt* and writes to *output3.txt* in the **output** folder. Students compare this file *output3.txt* in the **expected_output** folder to evaluate your work.

4. TASK 4 (2.0 points)

Implement the method below

public void undo()

to roll back the last action. **Undo** action makes effect for **adding** and **deleting** actions only.

For example, a list of 08 actions as below

1. Add a student whose ID is 68
2. Add a student whose ID is 72
3. Delete the student whose ID is 68
4. Write down the result of NLR traversal in the current tree
5. Add a student whose ID is 12
6. Undo
7. Undo
8. Write down the result of NLR traversal in the current tree

At step (4), the tree currently consists of the node with the key 72 only. After steps (6) and (7), the result of NLR traversal is the same as the one after step (2). *Explanation*: step (6) rolls back the action at step (5), and step (7) rolls back the action at step (3).

Hint: To accomplish this task, for adding and deleting actions, students clone a new tree from the current tree and make use of the provided Stack<Node> undoState to store the root of the cloned tree. When undoing, students need to pop the stack and assign the output node to the root of the current tree, thus restoring the state of the tree.

Main class is provided with actions of reading data files, invoking the method above, and writing files. Students compile and run

java Main 4

The program reads *testcase4.txt* and writes to *output4.txt* in the **output** folder. Students compare this file *output4.txt* in the **expected_output** folder to evaluate your work.

5. TASK 5 (1.0 point)

Implement the method below

public void redo()

to roll back the last **undo** action. **Redo** actions make an effect for **undo** ones only, if there do not exist **undo** actions, then **redo** ones take no effect. The two kinds of actions can be interleaved; however, the number of redo actions is based on the ones of undo before. When performing a new action that is not **undo**, then **redo** action takes no effect.

Hint: To accomplish the task, for undo actions, students clone a new tree from the current one and use the provided Stack<Node> redoState to store the root of the cloned tree. When redoing, students need to pop the stack and assign the output node to the root of the current tree, thus restoring the state of the tree

Main class is provided with actions of reading data files, invoking the method above, and writing files. Students compile and run

java Main 5

The program reads *testcase5.txt* and writes to *output5.txt* in the **output** folder. Students compare this file *output5.txt* in the **expected_output** folder to evaluate your work.

6. TASK 6 (1.0 point)

ScoreAVL class is a child class of **AVL** class and it is used to store an **AVL** tree whose keys are students' GPAs.

In **ScoreAVL** class, students complete the method below

private Node insert(Node x, Student key)

to add a new node to the tree.

After that, for **StudentManagement** class**,** students implement the method below:

public ScoreAVL scoreTree(AVL tree)

to build up and return an AVL tree whose keys are GPAs from an AVL tree whose keys are students' IDs, argument *tree*. Student objects are added to the GPA-based AVL in level-order

traversal of the argument *tree*. Testcases for task 6 do not contain students with duplicated GPAs and thus students do not have to handle duplication.

Main class is provided with actions of reading data files, invoking the method above, and writing files. Students compile and run

java Main 6

The program reads *testcase6.txt* and writes to *output6.txt* in the **output** folder. Students compare this file *output6.txt* in the **expected_output** folder to evaluate your work.

7. TASK 7 (1.0 point)

In the progress of conducting the 6 tasks above, students must ensure the program can be executed successfully given testcases with different kinds of actions. *testcase7.txt* is an example of a case with many actions at once.

Main class is provided with actions of reading data files, invoking the method above, and writing files. Students compile and run

java Main 7

The program reads *testcase7.txt* and writes to *output7.txt* in the **output** folder. Students compare this file *output7.txt* in the **expected_output** folder to evaluate your work.

Students are highly recommended to create own testcases with many different kinds of actions to evaluate your work.

VIII. Notices for submission

- If students are not able to accomplish a task, then remain in the original state of the method. DO NOT DELETE THE METHOD OF THE TASK because it causes compile errors for **main** method. Students must test whether your implementation can be compiled and executed successfully with the original **main** method before making the submission.
- Do not use absolute paths in your work.
- All output files, *output**X**.txt for X = {1,2,3,4,5,6,7}*, are written in the **output** folder which is located in the same folder as the given file source code files.
- For students that use IDEs (Eclipse, Netbeans, etc.), you must ensure the program can be compiled and executed command prompt, **do not use packages**, and *output**X**.txt* must be located in **output** folder. Paths defined in **Main.java** are compatible to command prompt.
- An example structure of the assignment organized correctly:

output	26/11/2022 19:45	File folder	
testcase	26/11/2022 19:23	File folder	
AVL.class	26/11/2022 20:42	CLASS File	4 KB
AVL.java	26/11/2022 20:06	Java Source File	6 KB
information.txt	26/11/2022 19:48	Text Document	1 KB
list_student.txt	09/11/2022 18:36	Text Document	3 KB
Main.class	26/11/2022 20:42	CLASS File	5 KB
Main.java	26/11/2022 20:41	Java Source File	5 KB
Node.class	26/11/2022 20:42	CLASS File	2 KB
Node.java	21/11/2022 00:25	Java Source File	2 KB
ScoreAVL.class	26/11/2022 20:42	CLASS File	1 KB
ScoreAVL.java	26/11/2022 19:43	Java Source File	1 KB
Student.class	26/11/2022 20:42	CLASS File	2 KB
Student.java	26/11/2022 20:25	Java Source File	2 KB
StudentManagement.class	26/11/2022 20:42	CLASS File	3 KB
StudentManagement.java	26/11/2022 20:42	Java Source File	4 KB

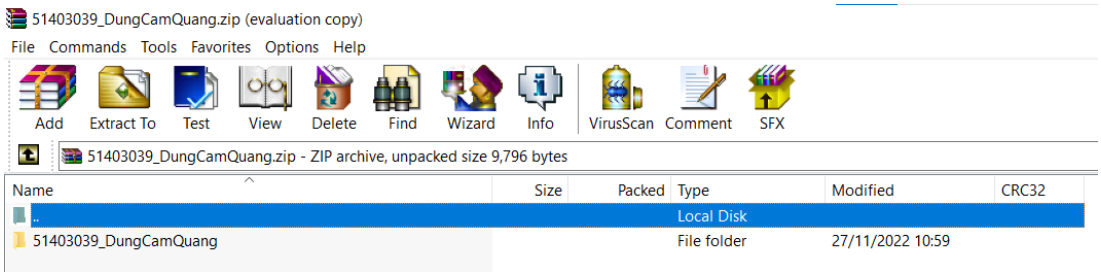
- In **output** folder:

Name	Date modified	Type	Size
output1.txt	26/11/2022 20:49	Text Document	1 KB
output2.txt	26/11/2022 20:49	Text Document	1 KB
output3.txt	26/11/2022 20:49	Text Document	1 KB
output4.txt	26/11/2022 20:49	Text Document	1 KB
output5.txt	26/11/2022 20:49	Text Document	1 KB
output6.txt	26/11/2022 20:49	Text Document	1 KB
output7.txt	26/11/2022 20:49	Text Document	2 KB

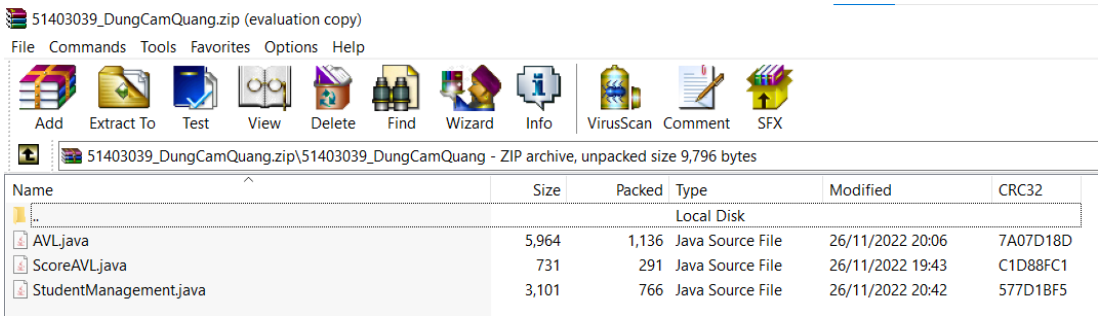
IX. Submission instructions

- Students submit ***AVL.java***, ***StudentManagement.java***, and ***ScoreAVL.java***, **DO NOT RENAME THE THREE DESIGNATED FILES AND DO NOT SUBMIT ANY OTHER FILES.**
- **Students copy the three designated files to a folder named StudentID_FullName** (FullName: no separators, no Vietnamese accents), then compress them as a zip file and submit by the deadline.
- For any careless mistakes, including but not limited to wrong folder names, not placing files under the designated folder, redundant files, etc. then students get **0.0** point.
- A correct submission is as below:
    - Submission file:

- In the compression:



- Bên trong thư mục:



X. Evaluation and policy

- Submissions are judged automatically using testcases (input and output file structures are the same as the descriptions in the assignment) and thus students take responsibility if not following the instructions or cause compile errors.
- Testcases for evaluation are in the same format as the ones in the assignment but have different contents. Students gain scores for each task if and only if the program passes all testcases of that task.
- Compile errors cause **0.0 points for the whole assignment**.
- **All submissions are carefully checked for plagiarism. Illegal actions including but not limited to copying source code on the internet, copying from other students', allowing other students to copy yours, etc. cause 0.0 point for Progress II and being banned for the final examination.**
- If the submission is suspected to plagiarism, then additional interviews may be conducted to verify.
- **Deadline: 23:00, Dec 13th, 2022.**

-- END --