# cis112-week10: Binary Search Tree (BST)

*v2025-05-04*

**Content**

# Introduction

This week we cover

- Binary Search Tree (BST)
- Visitor pattern

## Web resources

- Data Structures
- Abstract Data Type (ADT)
- Tree
- Graph Theory

### Definitions

**Def.** Let $x$ be a node in a binary tree. $x$ is said to satisfy *Binary Search Tree Property*

1. If $y$ is a node in the left subtree of $x$, then $y.key \leq x.key$.
2. If $y$ is a node in the right subtree of $x$, then $y.key \geq x.key$.

**Def.** A *Binary Search Tree* is a binary tree that satisfies the *Binary Search Tree Property*.

### Library `LibBST`

`LibBST` in `theory`, has all the methods of `LibTree` of the previous week. See "G1. eclipse compare" in "Goal". In addition to that, it has

```java
traverseInOrder(NodeBSTInterface<T> node, VisitorInterface<T> v)
```

method which enables visitor usage.

# Visitor Pattern

1. `VisitorInterface` is used for visitor pattern

   ```java
   public interface VisitorInterface<T> {

       void visit(NodeBSTInterface<T> node);

   }
   ```

2. A class implementing `VisitorInterface` can be used as visitor.

   For example, the following class is a visitor.

   ```java
   public class VisitorPrintShort<T> implements VisitorInterface<T> {

       @Override
       public void visit(NodeBSTInterface<T> node) {
           Student student = (Student) node.data();
           System.out.printf("l=%s n=%s g=%4.2f\n" //
                     , student.getLastName() //
                     , student.getName() //
                     , student.getGpa() //
           );
       }

   }
   ```

3. Once constructed, a visitor object can be applied to every node in the tree. For example, the following code visits each node in the tree inorder:

   ```java
   public static <T> void traverseInOrder(
       NodeBSTInterface<T> node
       , VisitorInterface<T> v
   ) {
       if (node == null) {
           return;
       }
       traverseInOrder(node.left(), v);
       v.visit(node);
       traverseInOrder(node.right(),v);
   }
   ```

# Goal

In `Student_Test`

- `getArrayStudent` generates a number of students, `studentNO`, and returns them in a `Student`-array `arrStudent`.
- Because of `Student` does not support `Comparable` interface, we cannot create a BST using `Student` instances in `arrStudent` directly. We can convert our `Student` instance to `StudentComparedByGPA`, which has a `compareTo` method by means of GPAs.
- `populateBSTByGPA` gets student array and populates a BST, then returns the tree.
- `populateBSTByName` does the same thing. The difference is the `compareTo` method. The nodes are compared by GPA in `populateBSTByGPA` and by name in `populateBSTByName`. To do that `Student` is extended to `StudentByGPA` and `StudentByName`. Similarly, `StudentByName` defines its `compareTo` method in terms of `LastName` and `Name`.

# G0. Fill StudentInfo

1. Fill your data in `StudentInfo`.

# G1. eclipse compare

Eclipse has a facility to compare two files and highlight the differences.

1. Download and import the previous week `cis112_week09`.
2. Select `cis112_week09.theory.LibTree`.
3. Press `Cntr` and select `cis112_week10.theory.LibBST`.
4. While both `LibTree` and `LibBST` are selected, right click to get context menu.
5. In the menu, select `Compare With` > `Each Other`.
6. Scroll down and understand the visualization.

# G2. Comparable Support

1. Uncomment the following in `Student_Test`

   ```java
   //      // sort by GPA
   //      bstByGPA.plot();
   //      printByGPA(bstByGPA);
   ```

2. Complete `compareTo` method, in `StudentComparedByGPA`. Use the following definition.

   **Definition.** Let $x$ and $y$ be two objects of type `StudentComparedByGPA`. $x$ is *smaller than* $y$, i.e., $x < y$, if GPA of $x$ is smaller than that of $y$.

3. `populateBSTByGPA` method, in `Student_Test`, takes `arrStudent`, converts `Student` to `StudentComparedByGPA`. Then populates a BST with `StudentComparedByGPA` objects. Finally, returns the BST.

# G3. Visitor Pattern

- `VisitorInterface` interface in `theory`, requires `visit` method which takes a node as a parameter and does some operation on it.
- `VisitorPrintShort` implements `VisitorInterface`. Its `visit` method prints the information of the node.
- In `Student_Test`, `printByGPA` passes `VisitorPrintShort` to each node by means of `traverseInOrder` method.

  **Q.** Do you see any pattern in the output of `printByGPA`?

## StudentComparedByName

1. Uncomment the following in `Student_Test`

   ```java
   //      // sort by name
   //      bstByName.plot();
   //      printByName(bstByName);
   ```

2. Complete `compareTo` method, in `StudentComparedByName`. Use the following definition.

   **Definition.** Let $x$ and $y$ be two objects of type `StudentComparedByGPA`. $x$ is *smaller than* $y$, i.e., $x < y$, iff

   1. LastName of $x$ is smaller than that of $y$.
   2. Name of $x$ is smaller than that of $y$ when it is the case that lastNames are the same.

3. In `Student_Test`, `printByName` passes `VisitorPrintShort` to each node by means of `traverseInOrder` method.

   **Q.** Do you see any pattern in the output of `printByName`?

   **Q.** Can you explain the results of `printByGPA` and `printByName`?

# Challenge

# C1. More Visitors

1. Make a copy of `Student_Test` as `Visitor_Test`. Add necessary stuff to test the following.

# C2. Attendance

1. Uncomment the following in `Visitor_Test`

   ```java
   //      // attendance
   //      attendance(bstByGPA);
   ```

2. Complete `VisitorAttendance`, which counts the number of students.

   **Hint.** Use `VisitorPrintShort` and `printByGPA` pair as example.

# C3. Students with GPA less than a given limit

1. Uncomment the following in `Visitor_Test`

   ```java
   //      // gpaLess
   //      gpaLess(bstByGPA, 2.00f);
   ```

2. Complete `VisitorGPALess`, which counts the number of students with GPA less than given limit.

# C4. Average

1. Uncomment the following in `Visitor_Test`

   ```java
   //      // average
   //      average(bstByGPA);
   ```

2. Complete `VisitorGPAAverage`, which calculates the average of GPAs.