Assignment 4

CS 1027
Computer Science Fundamentals II

Due date: April 6 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

- Building binary trees using queues
- Using and traversing tree structures
- Recursion
- Inheritance and dynamic binding
- Algorithm design

Introduction

We recently saw the 2022 Winter Olympics in which skaters, skiers, snowboarders, bobsledders, and many other winter Olympians from around the world competed in one of the most highly regarded sporting tournaments. We now introduce to you the Winter Javalympics in which we will push our limits and show our unwavering work ethic as we implement a program that simulates a skier coming down a treacherous hill.

A hill is represented as a binary tree and the skier starts at the root of the tree and has to quickly decide which way to ski next – left or right – based on what they see ahead in each of the segments below. In some cases, only one direction will be accessible and when the skier reaches the bottom, i.e. there are no further places to ski below, they are finished the run. Note that there are 3 types of hill segments: jump segment (with a specified height), slalom segment (with a specified direction), and regular segments that do not contain anything special. The slalom segments have a direction of either "L" for leeward, which means it is sheltered from the weather and therefore safe, or "W" for windward, which means it takes the brunt of the terrible mountainous weather and is unsafe. The windward slaloms should be avoided unless it is absolutely necessary to take them.

The skier must decide the best path to take down the hill but each hill is unknown to the skier until they begin down the frigid slopes. Thus, the skier has to decide the next direction to go, one step at a time. To clarify, a skier cannot pre-plan the whole route before starting. It has to be determined as they are skiing down the snowy hill. The "best" direction to go at each segment is based on several criteria to ensure the skier is showing off their skills optimally to yield the highest score possible. The requirements for choosing the next path are as follows:

Assignment 4

CS 1027
Computer Science Fundamentals II

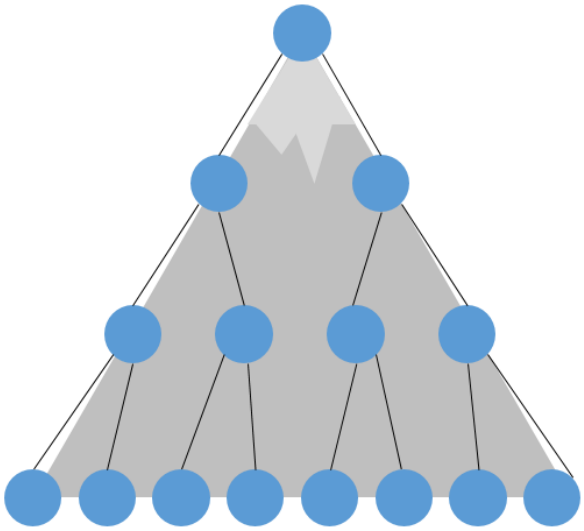


Figure 1. A diagram illustrating a tree structure to represent a ski hill with various segments. Note that this diagram is not part of the program – there is no visual aspect in this program.

- If there are no subsequent paths, they are finished
- If there is exactly one subsequent path, they must take that path regardless of what it contains or does not contain
- If there are two subsequent paths,
    - If both contain jumps, choose the one with the greater height or go right if their heights are equal
    - If exactly one contains a jump, take that path
    - If both contain slaloms, go to the one whose direction is leeward (**you can safely assume** that any time there are two slaloms from the same segment, one is leeward and one is windward – never two of the same direction)
    - If exactly one contains a slalom and the other is a regular segment, go to the slalom ONLY if its direction is leeward, otherwise go to the regular segment
    - If both are regular segments, choose right as the default direction

NOTE: all left and right directions mentioned here are from the perspective of us looking at our computer screen, not from the perspective of the skier.

Provided files

The following is a list of files provided to you for this assignment. **Do not alter these files**.

- SkiSegment.java
- JumpSegment.java (subclass of SkiSegment.java)
- SlalomSegment.java (subclass of SkiSegment.java)

- TestTreeBuilder.java (to test your TreeBuilder class)
- TestSki.java (to test your Ski class)
- and several more classes for trees, unorderedlists, queues, etc.

Classes to Implement

For this assignment, you must implement two Java classes: **TreeBuilder** and **Ski**. Follow the guidelines for each one below.

In both these classes, you can implement more private (helper) methods, if you want to, but you may not implement more public methods. You may not add instance variables other than the ones specified below nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

TreeBuilder.java

This class is used to create the binary trees for the program. They must be created using a queue-based approach, very similar to the level-order traversal that uses queues, but here the tree is being built rather than traversed. This class must use generics to build trees storing data of any type.

The class should not have any instance variables and only needs one method. Note that this class does not require a constructor since there is nothing to initialize. Java has a "default constructor" when none are explicitly provided, so it knows how to initialize the object when you use the "new" keyword. The only required method in this class is:

- public LinkedBinaryTree<T> buildTree (T[] data) – takes in a T array input parameter which contains the set of values that will be inserted in the new tree's nodes. Note that some elements of the array may be null to represent that there is no tree node in that position. Within this method, use two queues, dataQueue of type LinkedQueue<T> and parentQueue of type LinkedQueue<BinaryTreeNode<T>>, to build the tree using a level-order-esque approach. The summarized algorithm steps are listed below.

Algorithm for building trees

initialize dataQueue with all elements to be added into the nodes in order: first enqueue data[0], then enqueue data[1], and so on.

initialize parentQueue as empty queue

create a LinkedBinaryTree object and set as its root a node storing the first element of dataQueue

enqueue the root node on parentQueue

while dataQueue has elements

a = dequeue from dataQueue

b = dequeue from dataQueue

parent = dequeue from parentQueue

if a is not null, add node storing a as left child of parent and enqueue on parentQueue

if b is not null, add node storing b as right child of parent and enqueue on parentQueue

return the LinkedBinaryTree object with the root node from above

Ski.java

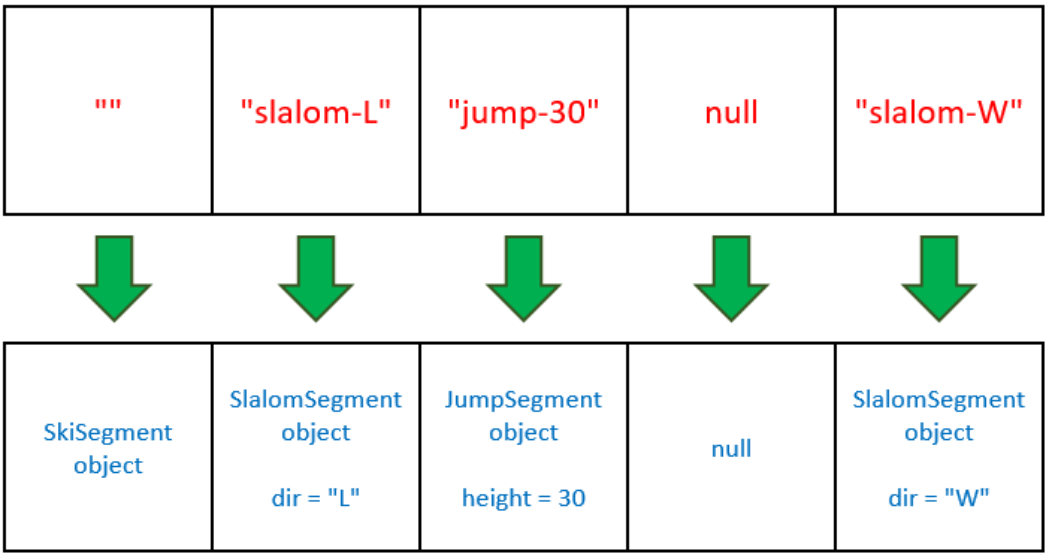
This class is the heart of the program in which the skier will go down the hill one step at a time based on the requirements explained in the Introduction.

The class must have one private instance variable:

- private BinaryTreeNode<SkiSegment> root;

The class must have the following public methods:

- public Ski(String[ ] data) [constructor]
  - Takes in a String array with data about the node types. The node types could be:
    - **null** → no child node there
    - **""** (empty string) → regular segment
    - **"jump-#"** → jump segment of height #, for example **"**jump-6**"** denotes a jump of height 6
    - **"slalom-L"** → slalom segment in the leeward (safe) direction
    - **"slalom-W"** → slalom segment in the windward (unsafe) direction
  - Create an array called segments of objects of the class SkiSegment that will contain the same information as the array parameter data, but with SkiSegment objects as shown in the diagram below (note that classes SlalomSegment and JumpSegment are subclasses of SkiSegment). To do this consider every element data[i]:
    - If data[i] contains the string **"**jump**",** then store in segments[i] a JumpSegment object: new JumpSegment(String.valueOf(i),data[i]); the parameter String.valueOf(i) will be used as the identified for this JumpSegment object
    - If data[i] contains the string **"**slalom**",** then store in segments[i] a SlalomSegment object: new SalomSegment(String.valueOf(i),data[i])
    - Otherwise, store in segments[i] a SkiSegment object: new SkiSegment(String.valueOf(i),data[i])



- Create a TreeBuilder object and use the buildTree method to construct the tree with the data in array segments from the previous step.
- Store the root of that tree (use method getRoot() from class LinkedBinaryTree to get the root of the tree) in the instance variable root.

- public BinaryTreeNode<SkiSegment> getRoot()
  Return the tree's root node.
- public void skiNextSegment (BinaryTreeNode<SkiSegment> node, ArrayUnorderedList<SkiSegment> sequence)
  - Add the data stored in parameter node (use method getData() from class BinaryTreeNode) to the end of the sequence so the whole path is recorded.
  - Determine the next node to access from the node passed as parameter (i.e. node.getLeft() or node.getRight()) based on the requirements explained in the Introduction.
  - This function **must** use recursion, i.e. call itself: skiNextSegment(next node, sequence), for continuing the path to the next node.
  - When you reach the bottom of the hill (i.e. a leaf node in the tree), simply do nothing to let the function end (this is the base case).
  - **Hint**: it is recommended that you use one or more private helper methods to help keep your code clean.

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?

- Does the code run properly on Gradescope (even if it runs on Eclipse, it is up to you to ensure it works on Gradescope to get the test marks)
- Does the program produces compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty of 5%
- Including a "package" line at the top of a file will receive a penalty of 5%

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Wednesday, April 6, 2022 at 11:55pm ET)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.
- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own test harness class to do this) to ensure that your code works for other dice combinations as well. We are providing you with some tests but we may use additional tests that you haven't seen before for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.

- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- TreeBuilder.java
- Ski.java

Grading Criteria

Total Marks: [20]

Functional Specifications:

[1] TreeBuilder.java

[3] Ski.java

[13] Passing Tests (some additional, hidden tests will be run on Gradescope)

Non-Functional Specifications:

[1] Meaningful variable names, private instance variables

[1] Code readability and indentation

[1] Code comments