CS 5004: Object Oriented Design and Analysis

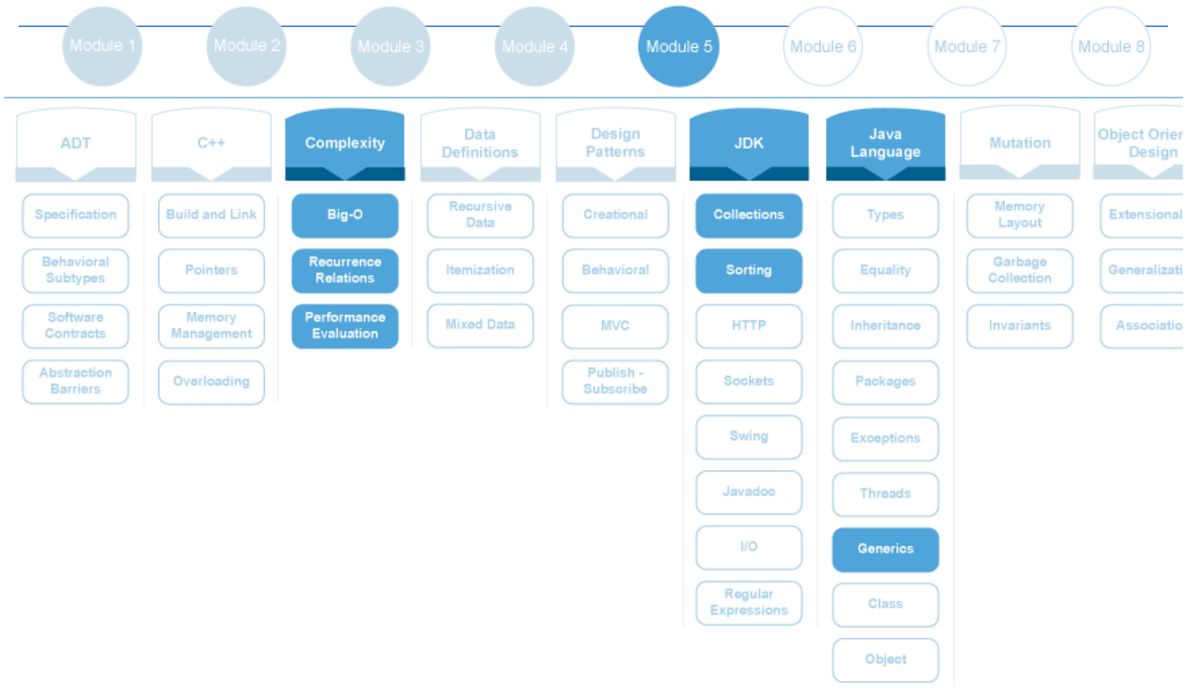
Northeastern University
College of Computer and Information Science

HOME
SYLLABUS
CODEWALKS
SCHEDULE
MODULES
0
1
2
3
4
5
6
7
8
9
10

PIAZZA

- Overview
- Course Map
- Objectives

In this module we take a look at Java's Collections library discussing sets, maps, queues, and other commonly used data structures. We discuss our previous implementations, and look at the JDK's implementations.



1. Given a custom Map implementation, refactor the code to use the JDK standard Map.
2. Given a custom Queue implementation, refactor the code to use the JDK standard Queue.
3. Given a custom Set implementation, refactor the code to use the JDK standard Set.
4. Given a Collection, use the JDK Comparator to sort the elements.
5. Given a sorting specification, order a collection of data by using the Java standard library.
6. Given a Java class, write down the time complexity for a specific method using big-O notation.
7. Given a Java class, write down the time complexity for a specific method using big-Omega notation.
8. Given the complexity analysis of a method, interpret the performance.
9. Given a Java class with a method whose implementation uses recursion, write down the recurrence relation.
10. Given a set of Java classes, evaluate their performance by writing performance tests.
11. Given a Java class, use JUnit to write performance tests.
12. Given a set of Java classes with performance tests, plot their performance results.

- Reading
- Assignment 7

Java Tutorial: Collections

- DUE DATE: Thursday, March 22th @11:59am

Submission Criteria

Repository Contents

Your repositories must contain only the following:

1. pom.xml file and
2. src folder with appropriate sub-folders with your code and test code only.

Code Criteria

1. Your solution for each problem should be in its own package. The package names should follow this naming convention `edu.neu.ccs.cs5004.assignmentN.problemM` where you replace `N` with the assignment number and `M` with the problem number, e.g., all your code for problem 1 for this assignment must be in a package named `edu.neu.ccs.cs5004.assignment7.problem1`.
2. Your project should successfully build using maven generating all the default reports.
3. Your Javadoc generation should complete with no errors or warnings.
4. Your Checkstyle report must have no violations.
5. Your JaCoCo report must indicate 75% code coverage per package for "Branches" **and** "Instructions" or more.
6. Your FindBugs report must have no violations (except for violations categorized as PERFORMANCE or SECURITY, you can ignore those).
7. Your PMD report should have no violations.

Problem 1

Design and implement in Java an Integer Binary Tree (IBT) **and** an Integer Binary Search Tree (IBST) whose nodes can hold `Integer` objects.

An IBT is one of :

1. empty
2. a node that contains:
    1. an integer value inside the node
    2. a left subtree that is itself an IBT,
    3. a right subtree that is iteself an IBT

An IBT must not contain any duplicate integer values.

An IBST is similar to an IBT with the following extra restrictions:

1. every node in an IBST contains:
    1. an integer value inside the node
    2. a left subtree that is itself an IBST, **and**, all the integer values in the left-subtree are **smaller** than the value of this node

3. a right subtree that is iteself an IBST, **and**, all the integer values in the right-subtree are **larger** than the value of this node.

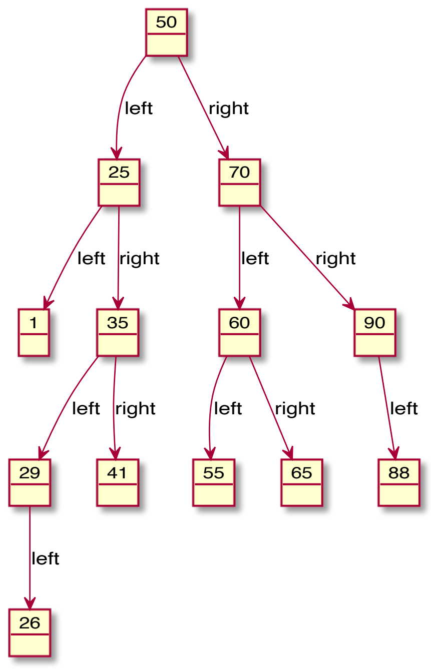
Your design and implementation should support at a minimum the following operations on a IBT:

- create an empty IBT
- add a given integer to the IBT. The operation returns the IBT with the node added. You are free to chose where to add the new node in the IBT.
- the ability to check if the tree is the empty tree
- the ability to know if a given Integer is already stored in the IBT inside a node

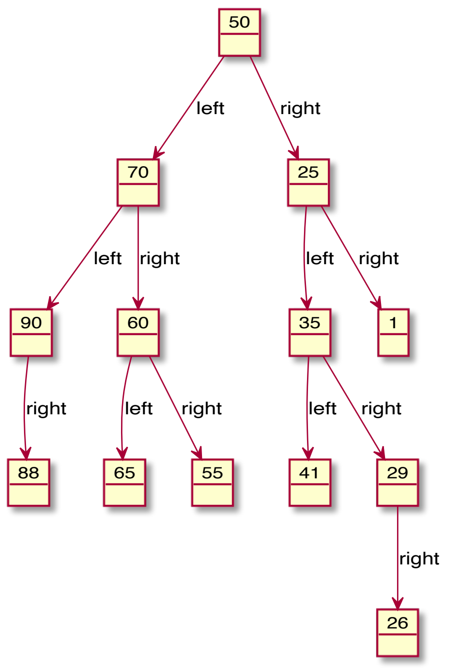
Your design and implementation should support at a minimum the following operations on a IBST:

- create an empty IBST
- add a given integer value to an IBST. The operation should return a new IBST that should contain a new node with the new integer value passed as an argument. Your operation **must** return a valid IBST.
- the ability to check if the tree is the empty tree
- the ability to know if a given Integer is already stored in the IBST inside a node
- add the operation `deleteNode` to IBST; the operation takes in an `Integer`, `element`, and if there is a node with the value `element` in the IBST then remove that node from the IBST and return the new IBST, else if `element` is not in the IBST return the IBST unchanged. The IBST returned must be a valid IBST with all nodes connected as a tree (no dead nodes/leafs).
- add the operation `mirror` to IBST; the operation returns an IBT which is the mirror image of the current IBST. Here is an example (empty nodes are not shown in the example diagrams)

IBST Example



IBST Example's Mirror Image



Finally you should design the following operation for both IBTs **and** IBSTs

1. an operation called `isBst()` that returns true when called on an instance that satisfies the conditions for an IBST, and false otherwise. Calling `isBst()` on the return values of the operation `mirror` should return false. Calling `isBst()` on the return values of the other operations that return an IBST should return true.

NOTE: all IBSTs *are* IBTs but **not** all IBTs are IBSTs. Your solution **must** exploit this property.

Problem 2

In Problem 1, you designed an Integer Binary Tree (IBT) and Integer Binary Search Tree (IBST). In this problem, you want to reuse that code, and extend it by developing the following features/operations for your IBST.

1. Develop an iterator that iterates over your IBST using a pre-order tree walk.
2. Develop an iterator that iterates over your IBST using a post-order tree walk.
3. Develop an iterator that iterates over your IBST using a in-order tree walk.

Your IBST should provide three separate methods, one for each iterator so that clients can call the method that returns the iterator that they prefer.

Problem 3

In this problem, you are **required** to use the types in `java.util` in your solutions.

A hiring company would like to design a skill index for all of their job applicants. A skill index is a data collection that contains applicants' IDs as keys, and values are lists of skills and keywords that applicants listed on their online job application. Here is a toy-example of a skill index.

Applicant ID.	Skills
112233	"Java", "Python", "OOD", "Go", "ASP.NET"
223456	"Java", "JavaScript", "security", "r", "Scala"
357911	"AI", "Java", "Python", "assembly", "AWS"
481216	"hardware design", "Java", "C#",
714210	"Java", "Python", "Ruby", "Cloud computing"

Their current program analyzes all of the received job applications, and generates `java.util.Map<K,V>` that contains aggregated information about job applicants. In this Map, keys are the reported skills, e.g., "Java", and values are lists of IDs belonging to those employees who have reported possessing the skill. Here is an example of what the map contains once the program processes applications for a day.

Skill	Applicants
Java	112233, 223456, 357911, 481216, 714210
Python	1122233, 357911, 714210
OOD	112233
Go	112233
ASP.NET	112233
JavaScript	223456
security	223456
r	223456
Scala	223456
AI	357911
assembly	357911
AWS	357911
hardware d	481216
C#	481216
Ruby	714210
Cloud comp	714210

The hiring company would like to hire you to design a program that, given the `java.util.Map<;K,V>` that they already have, returns the desired skill index as a new Map. So given the preceding example, your program should produce a `java.util.Map<;K,V>` that holds the data in the first table.

Problem 4

Your are in the final stages of a job interview with a company providing video streaming services (for example, Netflix, Hulu, HBO, Comcast), and to get that sweet, satisfying job offer, you are asked to build a ``media library'', a data collection to manage the company's collection of movies and TV series. Here is their description the required media library.

The media library is a collection of movies and TV series. There doesn't exist a prescribed order for that library, but for each movie or TV series we would like to store the following information.

- The movie's or TV series' alias. This is a short name that we want to give to the movie/TV series. We would like the media library to have unique aliases. So if we try to add a new movie or a series, and use an alias that is already in the media library, then we should be told that our alias is already used, and that we should come up with a new alias.
- The movie's or TV series' title.
- The movie's or TV series' release year as a four digit number.
- The movie's or TV series' directors, there can be one or more. A director is typically a person's name.
- The movie's or TV series' main actors, there can be one or more. For each actor we provide their name.

Once we populate the media library, we would like to be able to perform the following tasks:

- Given a director's name, we should be able to get all the movies and TV series' that they directed. If there is more than one movie directed, the results should be sorted by the movies' and TV series' release years. Most recent media content first.
- We should be able to add a new movie or TV series to the media library. Please see our earlier comment on aliases.
- We should be also able to request a movie/TV series for streaming. More specifically, we should be able to provide the movie' or TV series' alias, and the media library should add one to the number of times that the movie/TV series was streamed.
- We should be able to ask the media library for the streaming statistics of some movie/TV series. More specifically, we should be able to provide the movie' or TV series' alias, and the media library should return the number of times that the movie was streamed.
- Finally, we should be able to ask the media library to return the most streamed movie or TV series (the most streamed media content). If there are more than one movie or TV series, then return any one of them.

To land a job with flying colors, for each of the operations on the media library, the company would like you to add the runtime and space complexity of the operation in your documentation.