King Saud University
College of Computer and Information Sciences

Department of Computer Science

CSC 212 Data Structures Project Report – 2nd Semester 2024-2025

Developing a Photo Management Application

Authors

Name	ID	List of all methods implemented by each student
عبدالعزيز فهد البراك	**444101921**	**Class Album and AlbumInvertedIndex**
سعود خالد المطيري	**444102176**	**Class BST And Photo And Ll**
محمد عبدالله آل رشود	**444101749**	**Class PhotoManager and InvIndexPhotoManager**

1. Introduction

This report explains the creation of a photo management application. The goal of the app is to help users organize and quickly access their photos by using descriptive tags. Users can build albums that collect photos based on chosen tag conditions.

In this report, we discuss:

- How the main classes (Photo, PhotoManager, Album) were built.
- How photo search was optimized using an inverted index.
- The performance evaluation of the system using Big-Oh analysis.

2. Specification

ADT Album: Specification

Operations:

1. **Method** Album (String name, String condition, PhotoManager manager)
   requires: none. **input:** name, condition, manager
   results: Creates a new album with the given name, condition, and manager. **output:** none.
2. **Method** getName ()
   requires: none. **input:** none.
   results: Returns the name of the album. **output:** String.
3. **Method** getCondition ()
   requires: none. **input:** none.
   results: Returns the condition of the album. **output:** String.
4. **Method** getManager ()
   requires: none. **input:** none.
   results: Returns the PhotoManager of the album. **output:** PhotoManager.
5. **Method** getPhotos ()
   requires: none. **input:** none.
   results: Returns the list of photos that match the album's condition. **output:** LinkedList<Photo>.
6. **Method** getNbComps ()
   requires: none. **input:** none.
   results: Returns the number of comparisons done in getPhotos(). **output:** int.

ADT AlbumInvertedIndex: Specification

Operations:

1. **Method** AlbumInvertedIndex (String name, String condition, InvIndexPhotoManager mgr)
   requires: none. **input:** name, condition, mgr.
   results: Creates an AlbumInvertedIndex with the given name, condition, and manager. **output:** none.
2. **Method** getPhotos ()
   requires: none. **input:** none.
   results: Returns the list of photos that match the album's condition using inverted index.
   output: LinkedList<Photo>.

ADT BST: Specification

Operations:

1. **Method** BST ()
   requires: none. **input:** none.
   results: Creates an empty binary search tree. **output:** none.
2. **Method** empty ()
   requires: none. **input:** none.
   results: Checks if the tree is empty. **output:** boolean.
3. **Method** full ()
   requires: none. **input:** none.
   results: Checks if the tree is full (always false). **output:** boolean.
4. **Method** retrieve ()
   requires: tree is not empty. **input:** none.
   results: Returns the data stored in the current node. **output:** T.
5. **Method** findKey (String k)
   requires: none. **input:** key k.
   results: Finds the node with key k and sets it as current. **output:** boolean.
6. **Method** insert (String k, T value)
   requires: key k is not already in the tree. **input:** key k, value.
   results: Inserts a new node with key k and value into the tree. **output:** boolean.
7. **Method** remove_key (String k)
   requires: none. **input:** key k.
   results: Removes the node with key k from the tree. **output:** boolean.

ADT BSTNode: Specification

Operations:

1. **Method** BSTNode (String k, T value)
   requires: none. **input:** key k, value.
   results: Creates a new node with the given key and value, with null left and right. **output:** none.
2. **Method** BSTNode (String k, T value, BSTNode<T> l, BSTNode<T> r)
   requires: none. **input:** key k, value, left node l, right node r.
   results: Creates a new node with the given key, value, left, and right children. **output:** none.

ADT InvIndexPhotoManager: Specification

Operations:

1. **Method** InvIndexPhotoManager ()
   requires: none. **input:** none.
   results: Creates an inverted index photo manager with an empty index and base manager.
   output: none.
2. **Method** addPhoto (Photo p)
   requires: none. **input:** photo p.
   results: Adds photo p to the base manager and updates the index. **output:** none.
3. **Method** deletePhoto (String path)
   requires: none. **input:** photo path.
   results: Deletes the photo with the given path from the base manager and updates the index.
   output: none.
4. **Method** getPhotos ()
   requires: none. **input:** none.
   results: Returns the index of photos (BST of photo lists). **output:** BST<LinkedList<Photo>>.

ADT LinkedList: Specification

Operations:

1. **Method** LinkedList ()
   requires: none. **input:** none.
   results: Creates an empty linked list. **output:** none.
2. **Method** empty ()
   requires: none. **input:** none.
   results: Checks if the list is empty. **output:** boolean.
3. **Method** last ()
   requires: list is not empty. **input:** none.
   results: Checks if current is at the last node. **output:** boolean.

4. **Method** full ()
   requires: none. **input:** none.
   results: Checks if the list is full (always false). **output:** boolean.
5. **Method** findFirst ()
   requires: list is not empty. **input:** none.
   results: Sets current to the first node. **output:** none.
6. **Method** findNext ()
   requires: current is not at the last node. **input:** none.
   results: Moves current to the next node. **output:** none.
7. **Method** retrieve ()
   requires: list is not empty. **input:** none.
   results: Returns the element at the current node. **output:** T.
8. **Method** update (T e)
   requires: list is not empty. **input:** element e.
   results: Updates the current node with e. **output:** none.
9. **Method** insert (T e)
   requires: none. **input:** element e.
   results: Inserts e after the current node and moves current to the new node. **output:** none.
10. **Method** remove ()
    requires: list is not empty. **input:** none.
    results: Removes the current node. **output:** none.
11. **Method** display ()
    requires: none. **input:** none.
    results: Displays the contents of the list. **output:** none

ADT List: Specification

Operations:

1. **Method** findFirst ()
   requires: list is not empty. **input:** none.
   results: Sets current to the first element.
   output: none.
2. **Method** findNext ()
   requires: current is not at the last element. **input:** none.
   results: Moves current to the next element.
   output: none.
3. **Method** retrieve ()
   requires: list is not empty. **input:** none.
   results: Returns the element at the current position.
   output: T.
4. **Method** update (T e)
   requires: list is not empty. **input:** element e.
   results: Updates the current element with e.
   output: none.

5. **Method** insert (T e)
   requires: list is not full. **input:** element e.
   results: Inserts element e after the current position.
   output: none.
6. **Method** remove ()
   requires: list is not empty. **input:** none.
   results: Removes the current element.
   output: none.
7. **Method** full ()
   requires: none. **input:** none.
   results: Checks if the list is full.
   output: boolean.
8. **Method** empty ()
   requires: none. **input:** none.
   results: Checks if the list is empty.
   output: boolean.
9. **Method** last ()
   requires: list is not empty. **input:** none.
   results: Checks if current is at the last element.
   output: boolean.

ADT Node: Specification

Operations:

1. **Method** Node (T value)
   requires: none. **input:** value.
   results: Creates a node with the given value and sets next to null. **output:** none.

ADT Photo: Specification

Operations:

1. **Method** Photo (String path, LinkedList<String> tags)
   requires: none. **input:** path, tags.
   results: Creates a photo with the given path and tags. **output:** none.
2. **Method** getPath ()
   requires: none. **input:** none.
   results: Returns the path of the photo. **output:** String.

3. **Method** getTags ()
   requires: none. **input:** none.
   results: Returns the tags of the photo. **output:** LinkedList<String>.
4. **Method** displayPhoto ()
   requires: none. **input:** none.
   results: Displays the photo's path and its tags. **output:** none.

ADT PhotoManager: Specification

Operations:

1. **Method** PhotoManager ()
   requires: none. **input:** none.
   results: Creates a new photo manager with an empty photo list. **output:** none.
2. **Method** getPhotos ()
   requires: none. **input:** none.
   results: Returns the list of photos. **output:** LinkedList<Photo>.
3. **Method** addPhoto (Photo p)
   requires: none. **input:** photo p.
   results: Adds photo p to the list. **output:** none.
4. **Method** deletePhoto (String path)
   requires: none. **input:** path of the photo.
   results: Deletes the photo with the given path from the list. **output:** none.

ADT Test: Specification

Operations:

1. **Method** main (String[] args)
   requires: none. **input:** command line arguments.
   results: Tests the photo and album functionalities: adds photos, creates albums, prints photo info, and deletes a photo. **output:** none.
2. **Method** toTagsLinkedList (String tags)
   requires: none. **input:** comma-separated string of tags.
   results: Converts the string into a LinkedList of tags. **output:** LinkedList<String>.
3. **Method** printTags (LinkedList<String> tags)
   requires: none. **input:** list of tags.
   results: Displays all tags from the list. **output:** none.
4. **Method** printPhotos (LinkedList<Photo> photos)
   requires: none. **input:** list of photos.
   results: Displays the paths of all photos in the list. **output:** none.

3. Design

The system is designed to manage photos tagged with keywords, allowing tag-based filtering and retrieval. Each Photo object stores a file path and a LinkedList<String> of tags. The PhotoManager class maintains a LinkedList<Photo> for basic photo storage and operations like add and delete.

To improve search performance, a tag-based inverted index is implemented via InvIndexPhotoManager, using a BST<LinkedList<Photo>>, where each key is a tag and the value is a list of photos containing that tag. This supports efficient retrieval for multi-tag queries using intersection of lists.

All data structures (LinkedList, BST) are implemented manually to avoid using Java's standard collections and reinforce algorithmic understanding. The design separates concerns between photo storage, indexing, and filtering logic to maintain modularity and ease of testing.

4. Implementation

The system was implemented in Java using manually built data structures. Photos are represented by objects that store a file path and a list of descriptive tags. These tags are stored using a custom singly linked list.

All photos are managed through a PhotoManager class, which maintains a linked list of photo objects. This class supports adding new photos and deleting them based on their file path.

To allow photo searches based on tags, the Album class implements a method that processes the condition string and searches through all stored photos. It checks if each photo contains all the tags mentioned in the condition. This approach uses linear search and is simple but not efficient for large datasets.

To improve performance, the InvIndexPhotoManager class builds an inverted index using a binary search tree. Each tag is stored as a key in the tree, and each key maps to a list of photos containing that tag. This structure allows the program to quickly find matching photos without checking every one manually.

Custom implementations of a linked list and binary search tree were used throughout the project to give full control over how data is stored, searched, and traversed.

5. Performance analysis

Performance Analysis for: **deletePhoto(String path)**

Big-O Complexity: O(n × m), where *n* is the number of photos in baseManager and *m* is the average number of tags per photo.

Performance Analysis for: getPhotos()
Big-O Complexity: O(1), where the method returns the BST index directly without performing any computation or traversal.

Performance Analysis for: addPhoto(Photo p)
Big-O Complexity: O(1), where the photo is inserted directly into the linked list without any traversal or condition checks.

Performance Analysis for: deletePhoto(String path)
Big-O Complexity: O(n), where *n* is the number of photos in the linked list, since it may need to search through the entire list to find the matching photo.

6. Conclusion

In this project, we developed a

photo management system that allows users to organize their photos using tags and create albums based on specific conditions. By using Linked Lists and BST, we made the process of adding, deleting, and searching for photos more efficient. The system now retrieves photos quickly, even as the number of photos grows.

This project improved our skills in using data structures and applying object-oriented programming concepts in practice. It also helped us gain experience in building organized and scalable systems. Overall, the project strengthened our technical and teamwork abilities, preparing us for future development tasks.