Data Structures
313 QC-CUNY
Surya Teja Gavva

Programming for Data Structures

1. Interfaces and Classes

Objective

Familiarize yourself with the concepts of interfaces, classes, inheritance, and polymorphism in Java by designing a basic Library Management System.

Description

You're tasked with designing a basic system for managing books in a library. The system needs to allow you to perform operations like adding books, checking out books, and returning books.

Classes & Interfaces

1. **Item Interface:** Represents any item that can be checked out from the library.
    - `checkout()`: Mark the item as checked out.
    - `returnItem()`: Mark the item as available.
    - `isAvailable()`: Check if the item is available for checkout.
2. **Book Class:** This will implement the `Item` interface. Attributes should include:
    - `title`
    - `author`
    - `ISBN`
    - `isCheckedOut`
3. **Library Class:** Represents the library which holds a collection of books.
    - `addBook(Book book)`: Adds a book to the library.

- `checkoutBook(String ISBN)`: Checkout a book based on its ISBN.
- `returnBook(String ISBN)`: Return a book to the library.
- `findBook(String ISBN)`: Returns a `Book` based on its ISBN.

Guidelines

1. Start by defining the `Item` interface with its methods.
2. Create the `Book` class that implements the `Item` interface.
3. Make sure the `Book` class appropriately updates the `isCheckedOut` status when checked out and returned.
4. Implement the `Library` class, which should maintain a collection (like an `ArrayList`) of `Book` objects.
5. In the `Library` class, implement the `addBook`, `checkoutBook`, `returnBook`, and `findBook` methods. Make sure that when a book is checked out, it's marked as such, and when it's returned, it's marked as available.

Extensions

1. Add an `User` class with attributes like `name`, `userID`, and a list of `checkedOutItems`. Modify the system to track which user has checked out a particular book.
2. Implement a `Magazine` class that also implements the `Item` interface. Add functionalities in the `Library` class to manage magazines.
3. Add exception handling. For instance, throw and catch appropriate exceptions if someone tries to check out a book that's already checked out or return a book that isn't checked out.

2. Queues

Description

You're tasked with simulating a basic print queue system for a small office. In this office, print jobs are queued up one after the other in a first-come-first-served manner.

Task Requirements

1. Create a `PrintJob` class with attributes:
   - `documentName`: The name of the document to be printed.
   - `pages`: The number of pages in the document.
2. Create a `PrintQueue` class that simulates the queue of print jobs. This class should have:
   - A queue to store the print jobs (`Queue<PrintJob>`).
   - A method to add a job to the print queue: `enqueue(PrintJob job)`.
   - A method to process and remove the next job in the queue (simulate printing): `dequeue()`.
   - A method to peek at the next job without removing it: `peek()`.
   - A method to check if the print queue is empty: `isEmpty()`.

Guidelines

1. Start by defining the `PrintJob` class. This is a simple class with attributes and their getter methods.
2. Proceed to create the `PrintQueue` class. Utilize Java's built-in `LinkedList` as the underlying data structure for your queue:

```
1     Queue<PrintJob> queue = new LinkedList<>();
2
```

3. Implement the methods in the `PrintQueue` class. Make sure to display relevant messages when a job is enqueued, dequeued, or when trying to dequeue from an empty queue.
4. In the `main` method, create a small simulation by:
   - Adding multiple print jobs to the queue.
   - Processing (printing) them in order.
   - Checking the status of the queue after each operation.

3. List ADT

1. Given an array of sorted integers (increasing order), and integer S, find if there are two elements that sum to S.

2. Write a program with a recursive algorithm to reverse a singly linked list.

3. Implement the iterative algorithm to reverse a linked list, (The three pointers algorithm discussed in class)

4. Write a program to convert a Prefix expression to a Postfix expression.

5. A palindrome is a phrase that reads the same forward and backward (examples: 'racecar', 'radar', 'noon', or 'rats live on no evil star'). By extension we call every string a palindrome that reads the same from left to right and from right to left. Develop a recursive algorithm that takes as input a string and decides whether the string is a palindrome. 1. Write down your algorithm in pseudocode. 2. Implement your algorithm in the `PalindromeChecker` class.