Application of Abstract Data Types (ADT) and Stack in Student Management

Phạm Hiếu Trung

Abstract Data Types (ADT)

- Definition: ADTs represent how data is defined and the operations allowed on that data, without worrying about the implementation.
- Example: Stack, Queue, List are examples of ADTs.
- Key Principle: Abstraction, hiding the internal details of data representation.

ADVANTAGES OF ADTS

- Separation of concerns: Data and operations are separated from implementation.
- Modularity: Easy to modify and extend.
- Reusability: ADTs can be reused across different applications.

Stack - Definition and Characteristics

•Definition: A Stack is an ADT that follows the Last In, First Out (LIFO) principle.
Operations:
Push: Add an element to the top.
Pop: Remove an element from the top.
Peek: View the top element without removing it.

Use Cases of Stack in Software Development

•Memory management: Stack is used to manage function calls (call stack).

• Undo operations: Used in text editors and other software to keep track of actions.

• Expression evaluation: Used in arithmetic and logical expression parsing.

Problem Breakdown

Input: Student ID, Name, Score.
Process: Add, edit, delete, and manage students in a stack.
Output: Display student information and ranking.

How the Stack Solves the Problem

- Stack Operations: Each action (Add, Edit, Delete) can be represented as a stack operation.

- LIFO Logic: The most recently added student is the first to be processed.

- Data Security: Helps in managing memory efficiently, reducing overhead.

Code Walkthrough - Part 2: Stack Implementation

Stack Class: Implements the basic operations of a stack.
Operations:
push: Adds a student to the stack.
pop: Removes the top student.
peek: Views the top student without removing.
isEmpty: Checks if the stack is empty.

Code Walkthrough - Part 3: Node Class

- Node Structure: Each Node holds a reference to a Student object and the next node in the stack.
- Linked List: Stack is implemented using a linked list of nodes.

Code Walkthrough - Part 4: Main Program

Initialize Stack: Create a new stack of students.

Sample Operations: Add students, display the stack, pop students, and test stack functionality.

Demo: Running the program to show stack operations.

Stack Operations in the Student Management System

Adding a student: Using the push operation.

Removing a student: Using the pop operation.

Peeking at the last student added: Using the peek operation.

Ranking System Based on Student Score

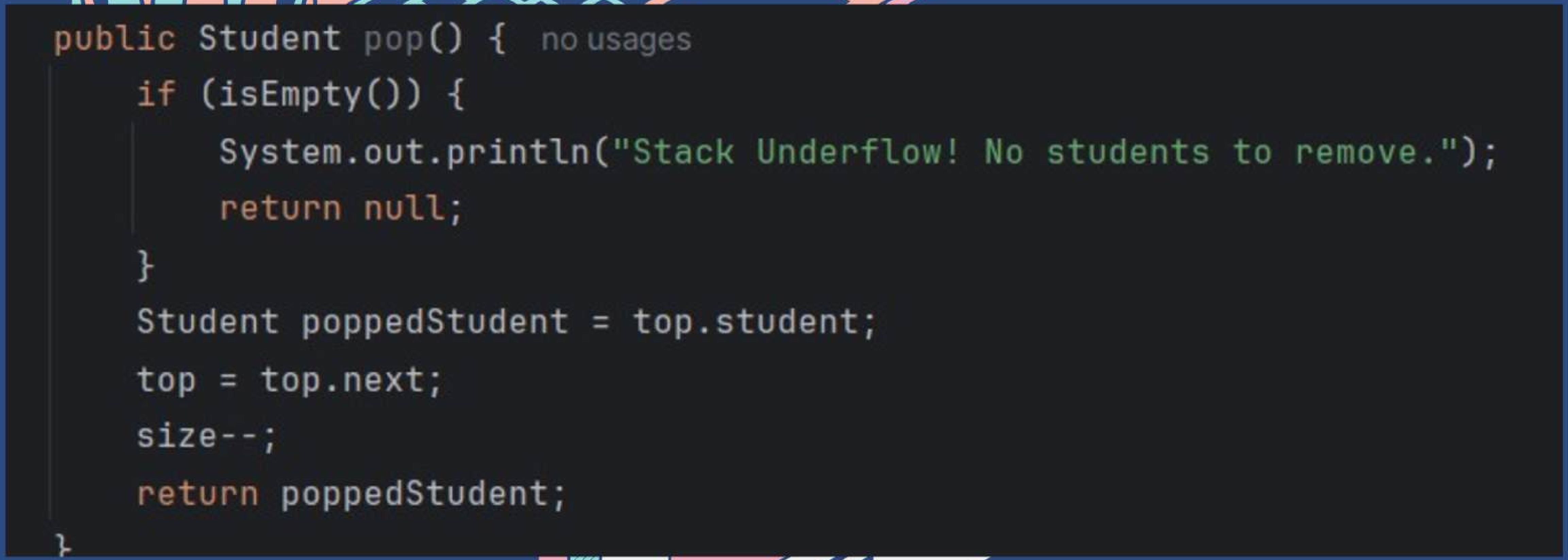
- Score Ranges: [0 – 5.0)
- Fail [5.0 – 6.5)
- Average [6.5 – 7.5)
- Good [7.5 – 9.0)
- Very Good [9.0 – 10.0]
- Excellent Assign Ranking: Based on the student’s score, the program assigns a rank.

EXAMPLE OF RANKING LOGIC

- **Student Data:**
- **ID: 1, Name: Hưng, Score: 7.2 (Rank: Good)**
- **ID: 2, Name: Trung , Score: 9.1 (Rank: Excellent)**
- **Ranking Output: Displays rank based on the predefined ranges.**

ADDING EDIT AND DELETE OPERATIONS

- Edit: Modify student details (ID, name, or score).
- Delete: Remove a student from the stack using



```java
public Student pop() {
    if (isEmpty()) {
        System.out.println("Stack Underflow! No students to remove.");
        return null;
    }
    Student poppedStudent = top.student;
    top = top.next;
    size--;
    return poppedStudent;
}
```

SORTING AND SEARCHING STUDENTS

Sorting: Implement a sorting algorithm (e.g.,Bubble Sort) to order students by score.
Searching: Linear search through the stack to find students by ID or name.

ALTERNATIVE ALGORITHM PROPOSAL

- Bubble Sort: Sort students by score.
- Alternative: Use Quick Sort or Merge Sort for better efficiency with large datasets.
- Evaluation: Comparing time complexity and efficiency.

ADVANTAGES OF USING STACK

- Efficiency: Simple implementation, constant- time complexity for push and pop operations.
- Memory management: Effective for managing data in a LIFO structure.
- Flexibility: Can be adapted to various use cases like undo functions, recursion management, etc.

PROGRAM EXECUTION DEMONSTRATION

- **Adding Students: Running the program to add students to the stack.**
- **Viewing Students: Demonstrating the push and display functionality.**

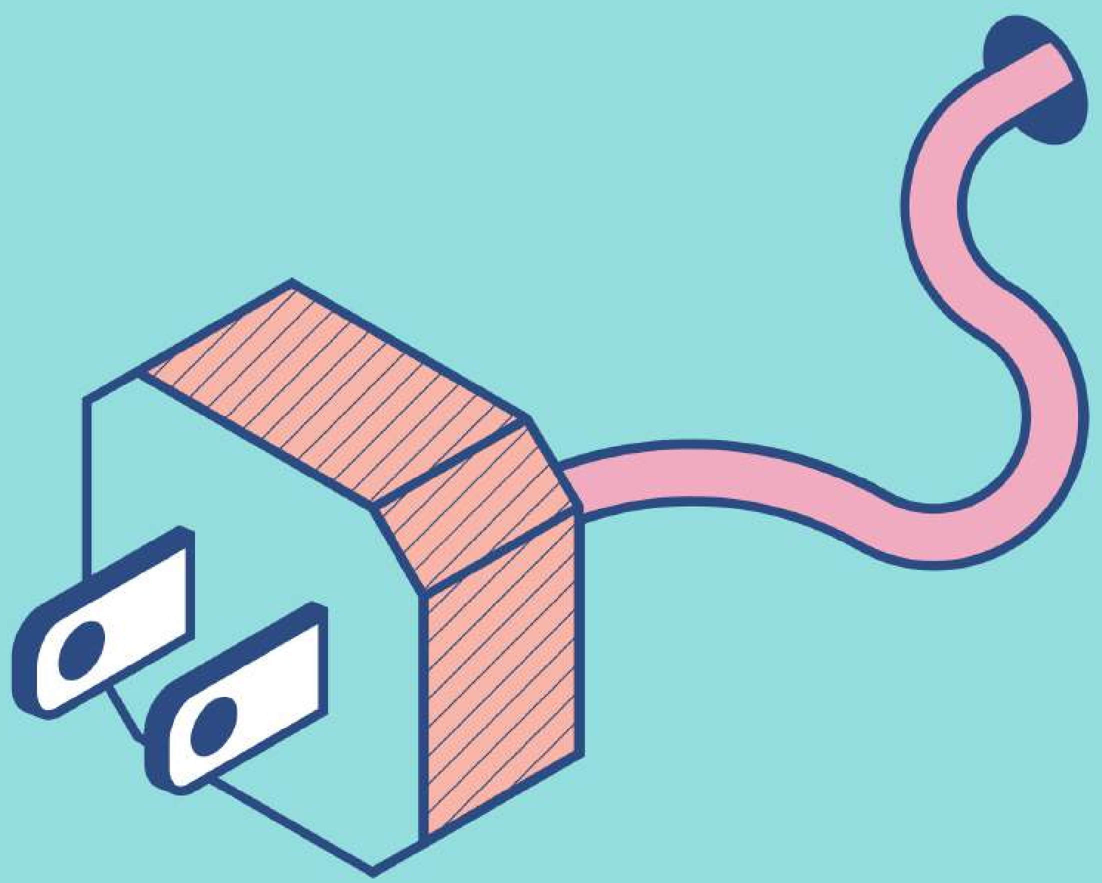
Program Improvement Suggestions (Part 1)

Code Optimization: Consider using a different data structure, such as a queue.

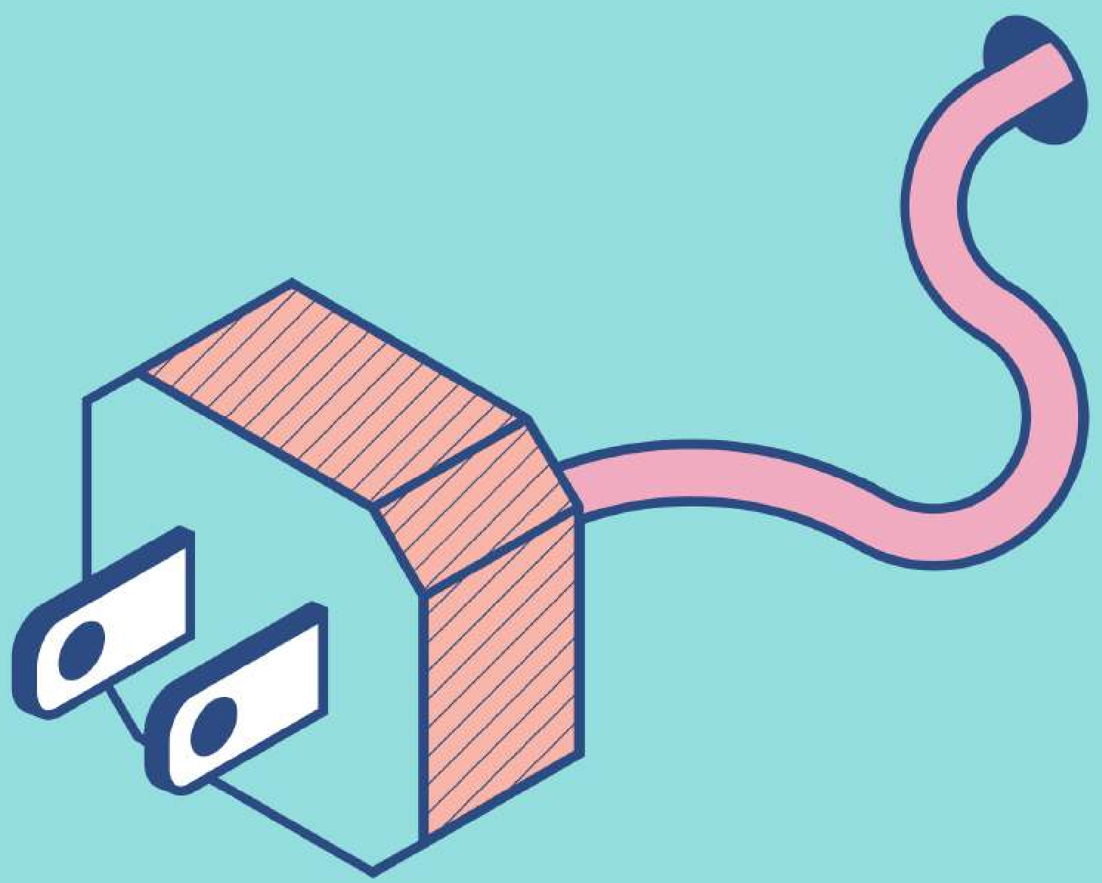
Security: Add authentication systems to protect student management.

Program Improvement Suggestions (Part 2)

•User Interface: Introduce a graphical user interface (GUI) for easier student management.
•Testing: Include unit tests to verify program reliability and functionality.



Applications of ADT and Stack in Other Fields



- Game Development: Using stack for move history and undo functionality.
- Computer Science: Stack is essential for algorithms and memory management.

Optimizing Stack for Large Applications

- Memory Optimization: Use efficient data structures for large-scale applications.
- Algorithm Improvement: Implement more optimized algorithms for student management.

Summary of ADT and Stack

- ADT: A powerful tool for managing data abstractly and efficiently.
- Stack: A key data structure with practical real-world applications.

Pros and Cons of Stack

- Pros: Simple, easy to implement, and effective for LIFO operations.
- Cons: Limited by the LIFO structure, harder to handle non-sequential data.

Comparing ADT with Other Data Structures

- ADT: Focuses on operations without worrying about implementation details.
- Comparison: Arrays (fixed size) vs Linked Lists (dynamic size).
- Example: ADT hides data representation, while data structures focus on implementation.

THANK YOU