

APPLICATION OF
ABSTRACT DATA
TYPES (ADT) AND
STACK IN STUDENT
HANAGEMENT



Start



- 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.



- 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.



- 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.



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



- 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 THRUGHENTATION



- 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.







- 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.



- 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.

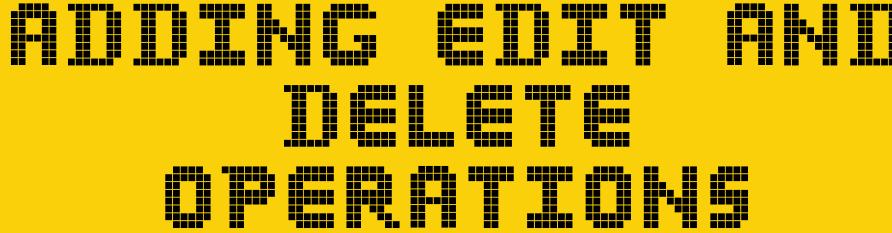


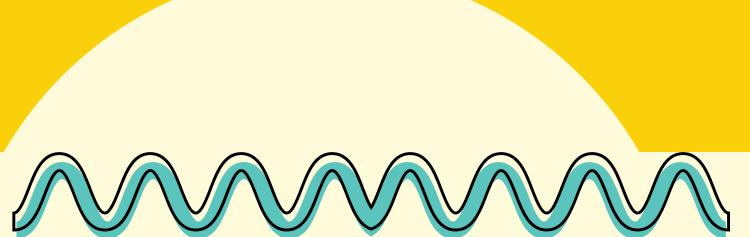
- Score Ranges:
- [0 5.0) Fail
- [5.0 6.5) Average
- [6.5 7.5) Good
- [7.5 9.0) Very Good
- [9.0 10.0] Excellent
- Assign Ranking: Based on the student's score, the program assigns a rank.





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



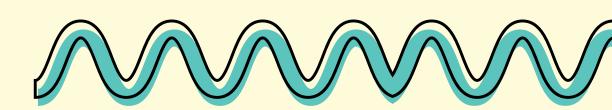


• Edit: Modify student details (ID, name, or score).

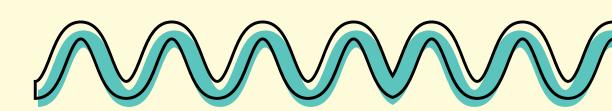
• Delete: Remove a student from the stack using the pop operation.

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

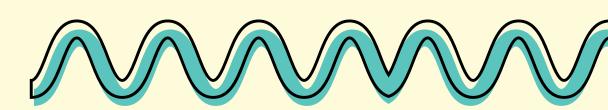
- Sorting: Implement a sorting algorithm (e.g., Bubble Sort) to order students by score.
- Searching: Linear search through the stack to find students by ID or name.



- 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.

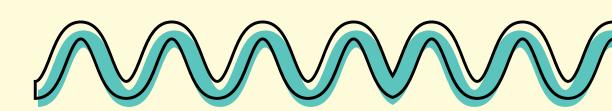


- Efficiency: Simple implementation, constanttime complexity for push and pop operations.
- Memory management: Effective for managing data in a LIFO structure.
- Flexibility: Can be adapted to various use cases like undo functions, recursion management, etc.



THE CURRED BY TO SAN THE PROBLEM OF THE PROBLEM OF

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



ROCHH INFROVENENT SUCCESTIONS (PART 1)

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

Security: Add authentication systems to protect student management.



- 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.

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

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

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

- 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.

