```
package stackadt;

public interface MyStack<T> {
    void push(T item) throws StackOverflowException;
    T pop() throws StackUnderflowException;
    boolean isEmpty();
    boolean isFull();
    void display();
}

package stackadt;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class StackAdt {

    private static PrintWriter logWriter;

    public static void main(String[] args) {
        try {
            logWriter = new PrintWriter(new FileWriter("stack_log.txt", true)); //
append mode
        } catch (IOException e) {
            System.out.println("Failed to open log file: " + e.getMessage());
            return;
        }

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter initial stack size: ");
        logWriter.println("Enter initial stack size: ");
        int initialSize = sc.nextInt();
        sc.nextLine(); // consume newline
        logWriter.println(initialSize);
        logWriter.flush();

        StackArray<String> stack = new StackArray<>(initialSize);

        int choice;
        do {
            System.out.println("\n--- Stack Menu ---");
            System.out.println("1. Push  2. Pop  3. Display  4. Exit");

            System.out.print("Choice: ");
            logWriter.println("Choice: ");
            choice = sc.nextInt();
            sc.nextLine();
            logWriter.println(choice);
            logWriter.flush();

            try {
                switch (choice) {
                    case 1 -> {
                        System.out.print("Enter value to push: ");
                        logWriter.println("Enter value to push: ");
                        String value = sc.nextLine();
                        logWriter.println(value);
                        logWriter.println("Pushed " + value);
                        logWriter.flush();

                        stack.push(value);
                    }
                    case 2 -> {
                        String popped = stack.pop();
                        logWriter.println("Popped " + popped);
                        logWriter.flush();
                    }
                    case 3 -> {
                        stack.display();
                        logWriter.println("Displayed stack: " + stack.toString());
                        logWriter.flush();
                    }
                    case 4 -> {
                        logWriter.println("exit ...");
                        logWriter.flush();
                    }
                    default -> {
                        System.out.println("Invalid choice. Try again.");
                        logWriter.println("Invalid choice: " + choice);
                        logWriter.flush();
                    }
                }
            } catch (Exception e) {
                logWriter.println("Exception: " + e.getMessage());
                logWriter.flush();
            }

        } while (choice != 4);

        sc.close();
        logWriter.close();
    }
}
```

```
package stackadt;
```

```
import java.util.Arrays;

class StackOverflowException extends Exception {
    public StackOverflowException(String message) {
        super(message);
    }
}

class StackUnderflowException extends Exception {
    public StackUnderflowException(String message) {
        super(message);
    }
}

public class StackArray<T> implements MyStack<T> {
    private Object[] stack;
    private int top;
    private int capacity;
    private final int MAX_CAPACITY = 100;

    public StackArray(int initialCapacity) {
        if (initialCapacity > MAX_CAPACITY) {
            throw new IllegalArgumentException("Initial capacity cannot exceed " +
MAX_CAPACITY);
        }
        this.stack = new Object[initialCapacity];
        this.capacity = initialCapacity;
        this.top = -1;
    }

    @Override
    public void push(T item) throws StackOverflowException {
        if (isFull()) {
            throw new StackOverflowException("Cannot push: Stack reached maximum
capacity (" + MAX_CAPACITY + ")");
        }
        if (top + 1 == capacity) {
            resize();
        }
        stack[++top] = item;
    }

    @Override
    @SuppressWarnings("unchecked")
    public T pop() throws StackUnderflowException {
        if (isEmpty()) {
            throw new StackUnderflowException("Cannot pop from empty stack.");
        }
        T item = (T) stack[top];
        stack[top--] = null;
        return item;
    }

    @Override
    public boolean isEmpty() {
        return top == -1;
    }

    @Override
    public boolean isFull() {
        return capacity == MAX_CAPACITY && top + 1 == MAX_CAPACITY;
    }

    private void resize() {
        int newCapacity = Math.min(capacity * 2, MAX_CAPACITY);
        stack = Arrays.copyOf(stack, newCapacity);
        capacity = newCapacity;
        System.out.println("Stack resized to capacity: " + newCapacity);
    }

    @Override
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
            return;
        }
        System.out.print("Stack elements (top to bottom): [ ");
        for (int i = top; i >= 0; i--) {
            System.out.print(stack[i]);
            if (i != 0) System.out.print(", ");
        }
        System.out.println(" ]");
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("[ ");
        for (int i = top; i >= 0; i--) {
            sb.append(stack[i]);
            if (i != 0) sb.append(", ");
        }
        sb.append(" ]");
        return sb.toString();
    }
}
```

```
Enter initial stack size:
3
Choice:
```

```
1
Enter value to push:
erf
Pushed erf
Choice:
2
Popped erf
Choice:
1
Enter value to push:
erf3
Pushed erf3
Choice:
3
Displayed stack: [ erf3 ]
Choice:
4
exit ...
```