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

interface MyStack<T> {
    String push(T item) throws Overflow;
    T pop() throws Underflow;
    boolean isEmpty();
    boolean isFull();
    String display();
}

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

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

class StackArray<T> implements MyStack<T> {
    private Object[] array;
    private int top;
    private int capacity;
    private int maxcapacity = 10;

    public StackArray(int initialCapacity) {
        if (initialCapacity <= 0 || initialCapacity > maxcapacity) {
            throw new IllegalArgumentException("Invalid initial capacity");
        }
        array = new Object[initialCapacity];
        capacity = initialCapacity;
        top = -1;
    }

    @Override
    public String push(T item) throws Overflow {
        String s="";
        if (isFull()) {
            if (capacity == maxcapacity) {
                throw new Overflow("Stack overflow: Maximum capacity reached.");
            }
            s = resize();
        }
        array[++top] = item;
        System.out.println(item + " pushed to stack.");
        return s+"\n"+item+" pushed to stack.";
    }

    @Override
    public T pop() throws Underflow {
        if (isEmpty()) {
            throw new Underflow("Cannot pop from empty stack.");
        }
        return (T) array[top--];
    }
```

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

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

    private String resize() {
        int newCapacity = Math.min(capacity * 2, maxcapacity);
        Object[] newArray = new Object[newCapacity];
        System.arraycopy(array, 0, newArray, 0, capacity);
        capacity = newCapacity;
        array = newArray;
        String s = "Stack resized to capacity: " + capacity;
        System.out.println(s);
        return s;
    }

    @Override
    public String display() {
        if (isEmpty()) {
            return "Stack is empty.";
        }
        String result = "Stack elements (top to bottom): [ ";
        for (int i = top; i >= 0; i--) {
            result += array[i];
            if (i > 0) {
                result += ", ";
            }
        }
        result += " ]";
        return result;
    }

}

public class StackAdt {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            FileWriter fw = new FileWriter("log_stack.txt", true);
            PrintWriter logWriter = new PrintWriter(fw, true);

            System.out.print("Enter initial stack size: ");
            int size = sc.nextInt();
            logWriter.println("INPUT initial size: " + size);

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

            int choice;
            do {
                System.out.println("\n--- Stack Menu ---");
                System.out.println("1. Push");
                System.out.println("2. Pop");
                System.out.println("3. Display");
                System.out.println("4. Exit");
                System.out.print("Choice: ");
                choice = sc.nextInt();
```

```
                sc.nextLine();
                logWriter.println("\n--- Stack Menu ---");
                logWriter.println("1. Push");
                logWriter.println("2. Pop");
                logWriter.println("3. Display");
                logWriter.println("4. Exit");
                logWriter.println("Choice: "+choice);

                try {
                    switch (choice) {
                        case 1:
                            System.out.print("Enter value to push: ");
                            String value = sc.nextLine();
                            logWriter.println("Enter value to push: "+value);
                            String p = stack.push(value);
                            logWriter.println(p);
                            break;
                        case 2:
                            String popped = stack.pop();
                            System.out.println(popped + " popped from stack.");
                            logWriter.println(popped + " popped from stack.");
                            break;
                        case 3:
                            String s = stack.display();
                            System.out.println(s);
                            logWriter.println(s);
                            break;
                        case 4:
                            System.out.println("Program exiting...");
                            logWriter.println("Program exiting...");
                            break;
                        default:
                            System.out.println("Invalid choice.");
                            logWriter.println("Invalid choice.");
                    }
                } catch (Overflow | Underflow e) {
                    System.out.println("Exception: " + e.getMessage());
                    logWriter.println("Exception: " + e.getMessage());
                }
            } while (choice != 4);

            sc.close();
            logWriter.close();
        } catch (IOException e) {
            System.err.println("Error opening log file: " + e.getMessage());
        }
    }
}
```