```java
import java.io.*;
import java.util.Scanner;

public class StackAdt {
    public static void main(String[] args) throws Exception {
        // Setup logging
        FileWriter fw = new FileWriter("log.txt", true); // append mode
        PrintWriter logWriter = new PrintWriter(fw, true);
        PrintStream originalOut = System.out;

        // Redirect System.out to write both to console and log
        PrintStream dualOut = new PrintStream(new OutputStream() {
            @Override
            public void write(int b) throws IOException {
                originalOut.write(b);
                logWriter.write(b);
            }
        });
        System.setOut(dualOut);

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter initial stack size: ");
        int initialSize = scanner.nextInt();
        logWriter.println("INPUT: Enter initial stack size: " + initialSize);
        scanner.nextLine(); // Consume newline

        StackArray stack = new StackArray(initialSize);

        while (true) {
            System.out.println("\n--- Stack Menu ---");
            System.out.println("1. Push  2. Pop  3. Display  4. Exit");
            System.out.print("Choice: ");
            int choice = scanner.nextInt();
            logWriter.println("INPUT: Choice: " + choice);
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter value to push: ");
                    String value = scanner.nextLine();
                    logWriter.println("INPUT: Enter value to push: " + value);
                    try {
                        stack.push(value);
                    } catch (StackOverflowException e) {
                        System.out.println("Exception: " + e.getMessage());
                    }
                    break;

                case 2:
                    try {
                        stack.pop();
                    } catch (StackUnderflowException e) {
                        System.out.println("Exception: " + e.getMessage());
                    }
                    break;

               case 3:
                    stack.display();
                    break;
                case 4:
                    System.out.println("Program exiting...");
                    scanner.close();
                    logWriter.close();
                    return;

                default:
                    System.out.println("Invalid choice.");
            }
        }
    }
}
```

```
public class StackArray implements MyStack {
    private Object[] stack;
    private int top;
    private int capacity;
    private static final int MAX_CAPACITY = 100;

    public StackArray(int initialCapacity) {
        if (initialCapacity <= 0 || initialCapacity > MAX_CAPACITY) {
            throw new IllegalArgumentException("Initial capacity must be between 1 and " +
MAX_CAPACITY);
        }
        stack = new Object[initialCapacity];
        capacity = initialCapacity;
        top = -1;
    }

    @Override
    public void push(Object item) throws StackOverflowException {
        if (isFull()) {
            if (capacity >= MAX_CAPACITY) {
                throw new StackOverflowException("Cannot push: Stack has reached max capacity
(" + MAX_CAPACITY + ").");
            } else {
                resize();
                System.out.println("Stack resized to capacity: " + capacity);
            }
        }
        stack[++top] = item;
        System.out.println(item + " pushed to stack.");
    }

    @Override
    public Object pop() throws StackUnderflowException {
        if (isEmpty()) {
            throw new StackUnderflowException("Cannot pop from empty stack.");
        }
        Object item = stack[top--];
        System.out.println(item + " popped from stack.");
        return item;
    }

    @Override
    public void display() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
        } else {
            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 boolean isEmpty() {
       return top == -1;
    }

    @Override
```

```java
    public boolean isFull() {
        return top == capacity - 1;
    }

    private void resize() {
        int newCapacity = Math.min(capacity * 2, MAX_CAPACITY);
        Object[] newStack = new Object[newCapacity];
        System.arraycopy(stack, 0, newStack, 0, capacity);
        stack = newStack;
        capacity = newCapacity;
    }

    @Override
    public String toString() {
        if (isEmpty()) return "[ ]";
        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();
    }
}
```

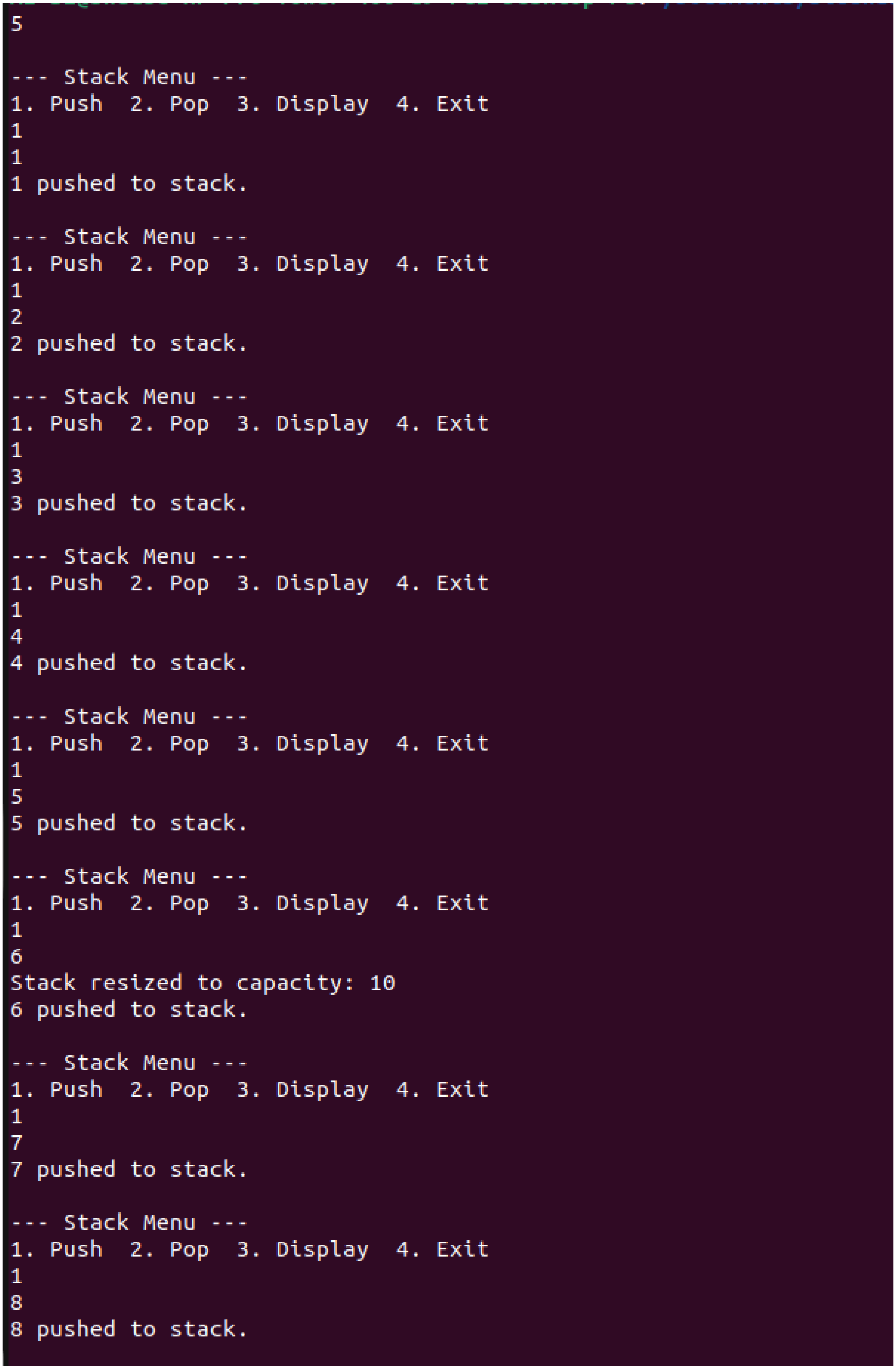
```java
public interface MyStack {
    void push(Object item) throws StackOverflowException;
    Object pop() throws StackUnderflowException;
    void display();
    boolean isEmpty();
    boolean isFull();
}
```

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

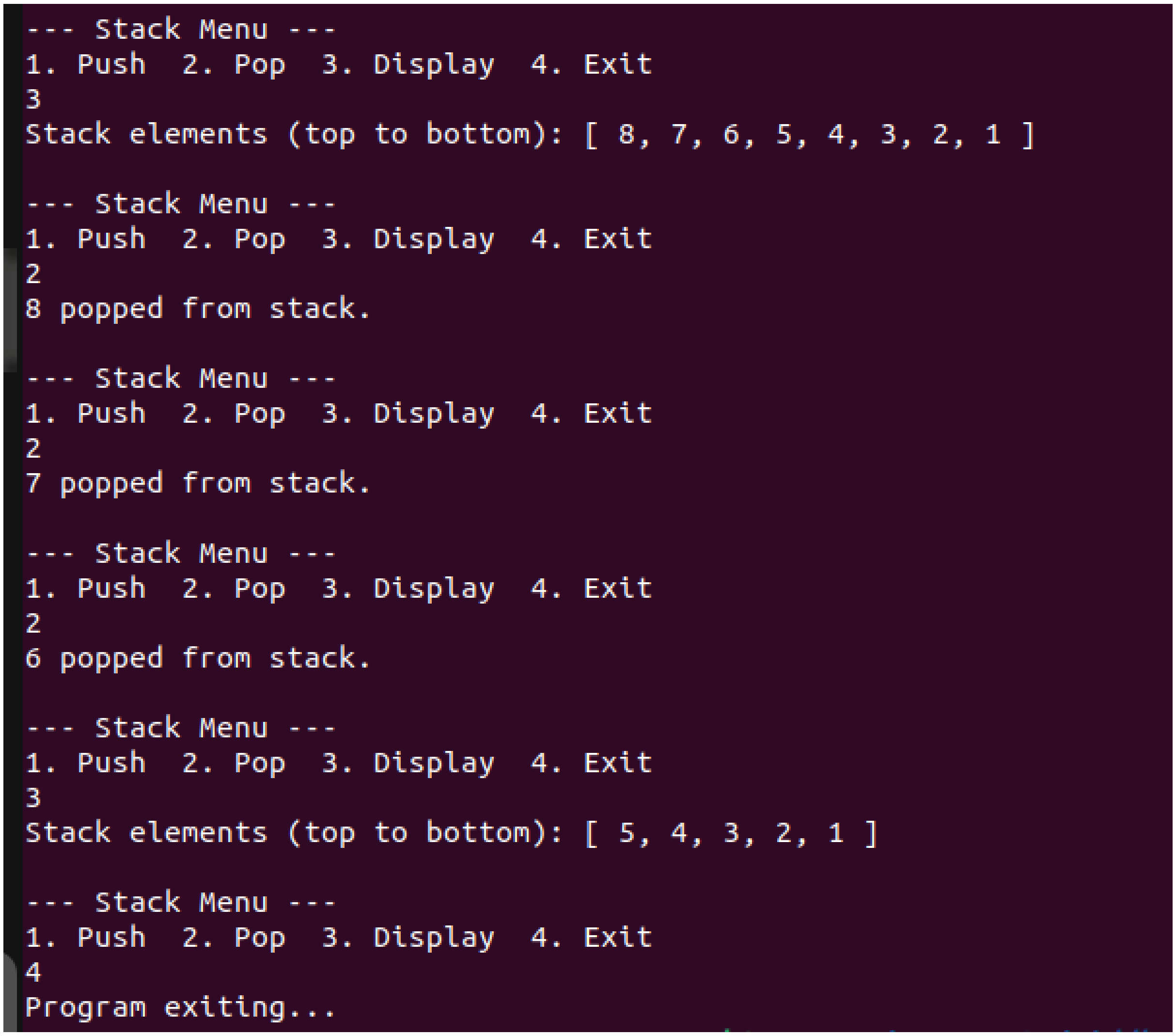
```java
public class StackUnderflowException extends Exception {
    public StackUnderflowException(String message) {
        super(message);
    }
}
```

```
Enter initial stack size: INPUT: Enter initial stack size: 1

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Apple
Apple pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Bannana
Stack resized to capacity: 2
Bannana pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Bannana, Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 4
Program exiting...
Enter initial stack size: INPUT: Enter initial stack size: 5

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Apple
Apple pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Banana
Banana pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Strawberry
Strawberry pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Pineapple
Pineapple pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 1
Enter value to push: INPUT: Enter value to push: Melon
Melon pushed to stack.
```

```
--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Melon, Pineapple, Strawberry, Banana, Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 2
Melon popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Pineapple, Strawberry, Banana, Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 2
Pineapple popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Strawberry, Banana, Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 2
Strawberry popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Banana, Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 2
Banana popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack elements (top to bottom): [ Apple ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 2
Apple popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 3
Stack is empty.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
Choice: INPUT: Choice: 4
Program exiting...
```



```
5

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
1
1 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
2
2 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
3
3 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
4
4 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
5
5 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
6
Stack resized to capacity: 10
6 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
7
7 pushed to stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
1
8
8 pushed to stack.
```



```
--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
3
Stack elements (top to bottom): [ 8, 7, 6, 5, 4, 3, 2, 1 ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
2
8 popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
2
7 popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
2
6 popped from stack.

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
3
Stack elements (top to bottom): [ 5, 4, 3, 2, 1 ]

--- Stack Menu ---
1. Push  2. Pop  3. Display  4. Exit
4
Program exiting...
```