Lab 08 Computer Science 1027B Summer 2025

Introduction

The purpose of these labs is to provide practice problems for you to get used to the concepts being covered in class. It is very important that you do these labs on your own without relying on available AI tools. Questions on exams are often very similar to exercises presented in these labs, so understanding them and solving these problems gives you a huge advantage for the exams. Make sure that you complete both the code as well as provide written answers to the blue questions within the boxes.

When you have concluded this lab, you should submit the following files to OWL Brightspace:

- BalancedParentheses.java
- CustomerServiceCenter.java
- Exercise03.txt
- BrowserNavigation.java
- written_answers.txt (containing answers to all questions with blue text inside of the boxes)

Topics Covered

- Queues
- Stacks

Exercise 01

Write a class **BalancedParentheses** that has a method

public static boolean areParenthesesBalanced(String expression)

That uses a stack to check if a string has balanced parentheses (including '(' and ')', '{' and '}', and '[' and ']').

```
import java.util.Stack;

public class BalancedParentheses {
    public static boolean areParenthesesBalanced(String expression) {
        // your code here
    }

    public static void main(String[] args) {
        String expression1 = "{[()]}";
        String expression2 = "{[(])}";
        String expression3 = "{{[[(())]]}}";

        System.out.println("Expression 1: " + expression1 + " is balanced: " + areParenthesesBalanced(expression1));
        System.out.println("Expression 2: " + expression2 + " is balanced: " + areParenthesesBalanced(expression2));
        System.out.println("Expression 3: " + expression3 + " is balanced: " + areParenthesesBalanced(expression3));
    }
}
```

Q 01.1

Is using a stack efficient for this task? Why or why not?

Lab 08 Computer Science 1027B Summer 2025

Exercise 02

You are tasked with simulating a simple customer service center using a queue in Java. The customer service centre processes customer requests in the order that they arrive (first-come, first-served). You need to implement the following functionalities:

1. Add a customer request to the queue
2. Process the next customer request
3. Display the current queue (showing all customer requests currently waiting to be processed)

Implement a **CustomerServiceCenter** class with the following public methods:

- **void addRequest(String request)**: Adds a new customer request to the queue.
- **String processNextRequest()**: Processes the next customer request in the queue and returns it. If the queue is empty, return a message indicating that there are no requests to process.
- **void displayQueue()**: Displays all the customer requests currently in the queue.

An example usage could look like:

```
public class Main {
    public static void main(String[] args) {
        CustomerServiceCenter serviceCenter = new CustomerServiceCenter();

        serviceCenter.addRequest("Request 1: Password reset");
        serviceCenter.addRequest("Request 2: Account activation");
        serviceCenter.addRequest("Request 3: Technical support");

        System.out.println("Current Queue:");
        serviceCenter.displayQueue();

        System.out.println("Processing next request: " + serviceCenter.processNextRequest());
        System.out.println("Processing next request: " + serviceCenter.processNextRequest());

        System.out.println("Current Queue:");
        serviceCenter.displayQueue();

        System.out.println("Processing next request: " + serviceCenter.processNextRequest());
        System.out.println("Processing next request: " + serviceCenter.processNextRequest());

        System.out.println("Current Queue:");
        serviceCenter.displayQueue();
    }
}
```

In which the output would look like:

```
Current Queue:
Request 1: Password reset
Request 2: Account activation
Request 3: Technical support
Processing next request: Request 1: Password reset
Processing next request: Request 2: Account activation
Current Queue:
Request 3: Technical support
Processing next request: Request 3: Technical support
Processing next request: No requests to process
Current Queue:
```

Exercise 03

Add the following values to a queue and a stack (using add for the queue and push for the stack):

1, 5, 9, 10, 100, 21

Now, remove two elements (using poll for the queue and pop for the stack)

What are the elements remaining in the queue and the stack, respectively?

Q 03.1

Explain how you would implement a stack that supports an additional operation getMin() which returns the minimum element in the stack. What data structures would you use and why?

Exercise 04

Implement a simple web browser's back and forward navigation functionality using stacks. When a user visits a new webpage, it is added to the current page stack. When the user clicks the "back" button, the current page is pushed onto the forward stack and the last page from the current page stack is displayed. When the user clicks the "forward" button, the last page from the forward stack is moved back to the current page stack and displayed.

implement a **BrowserNavigation** class with the following functionalities:

- Visit a new page: Adds the new page to the current stack and clears the forward stack.
- Back: Moves the current page to the forward stack and displays the last page from the current stack.
- Forward: Moves the last page from the forward stack to the current stack and displays it.
- Display Current Page: Displays the current page.

- **void visit(String url)**: Visits a new page with the given URL.
- **String back()**: Moves to the previous page and returns the URL of the current page. If there is no previous page, return a message indicating so.
- **String forward()**: Moves to the next page and returns the URL of the current page. If there is no next page, return a message indicating so.
- **String getCurrentPage()**: Returns the URL of the current page.

An example usage could look like:

```java
public class Main {
    public static void main(String[] args) {
        BrowserNavigation browser = new BrowserNavigation();

        browser.visit("google.com");
        browser.visit("stackoverflow.com");
        browser.visit("github.com");

        System.out.println("Current Page: " + browser.getCurrentPage()); // output: github.com

        System.out.println("Back: " + browser.back()); // output: stackoverflow.com
        System.out.println("Back: " + browser.back()); // output: google.com

        System.out.println("Forward: " + browser.forward()); // output: stackoverflow.com
        System.out.println("Visit new page");
        browser.visit("oracle.com");
        System.out.println("Current Page: " + browser.getCurrentPage()); // output: oracle.com

        System.out.println("Forward: " + browser.forward()); // output: No forward pages
        System.out.println("Back: " + browser.back()); // output: google.com
    }
}
```

In which the output would look like:

```
Current Page: github.com
Back: stackoverflow.com
Back: google.com
Forward: stackoverflow.com
Visit new page
Current Page: oracle.com
Forward: No forward pages
Back: google.com
```

Here is a template to get you started:

```java
import java.util.Stack;

public class BrowserNavigation {
    private Stack<String> currentPageStack;
    private Stack<String> forwardStack;
    private String currentPage;

    public BrowserNavigation() {
        currentPageStack = new Stack<>();
        forwardStack = new Stack<>();
    }
```

```java
    // visit a new page
    public void visit(String url) {
        // your code here
    }

    // go back to the previous page
    public String back() {
        // your code here
    }

    // go forward to the next page
    public String forward() {
        // your code here
    }

    // get the current page
    public String getCurrentPage() {
        // your code here
    }

    public static void main(String[] args) {
        BrowserNavigation browser = new BrowserNavigation();

        browser.visit("google.com");
        browser.visit("stackoverflow.com");
        browser.visit("github.com");

        System.out.println("Current Page: " + browser.getCurrentPage()); // output: github.com

        System.out.println("Back: " + browser.back()); // output: stackoverflow.com
        System.out.println("Back: " + browser.back()); // output: google.com

        System.out.println("Forward: " + browser.forward()); // output: stackoverflow.com
        System.out.println("Visit new page");
        browser.visit("oracle.com");
        System.out.println("Current Page: " + browser.getCurrentPage()); // output: oracle.com

        System.out.println("Forward: " + browser.forward()); // output: No forward pages
        System.out.println("Back: " + browser.back()); // output: stackoverflow.com
    }
}
```