PSLG Week 6

Ben & Amy

ICTLC Online QR Code:

DLSH

Github

Agenda

Lists

Stacks

Queues

Lists : what they do?

A list is an indexed data structure

(i.e. elements are accessed by calling an index that stores them).

Unlike data structures such as arrays Lists have dynamic sizes and can grow with data as opposed to a new structure needing to be made on addition of new data passed the maximum capacity.

Examples of list data structures are :

ArrayLists (dynamically resizable arrays, commonly found in languages like Java)

Linked Lists (where elements are linked together via pointers, allowing efficient insertions and deletions)

Syntax : ArrayLists

Import:

```
import java.util.ArrayList;
```

Initialisation:

```
Arraylist<Type> name = new ArrayList<>();
```

Important methods:

```
name.add(element);

name.get(index);
```

Problem time !!! (Oh naur)

Create a java program that stores module codes for lectures

e.g. “CS4019”, “MA4119”,”LA4392”

When the user runs the program they will be presented with 3 options:
Add, Display, Quit.

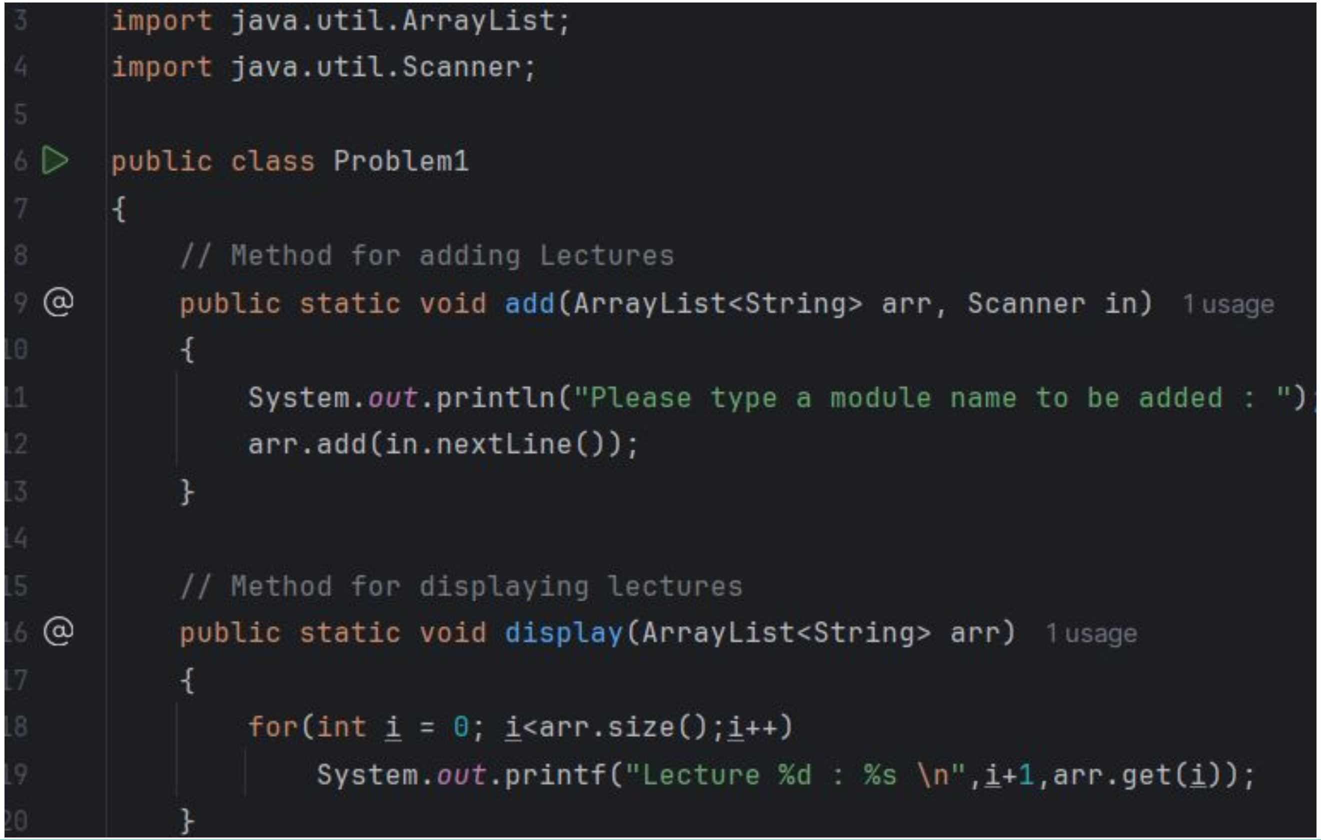
The system will then read a scanner input of their option and based on the read input will either:

Ask for a module code to be added

Display all module codes

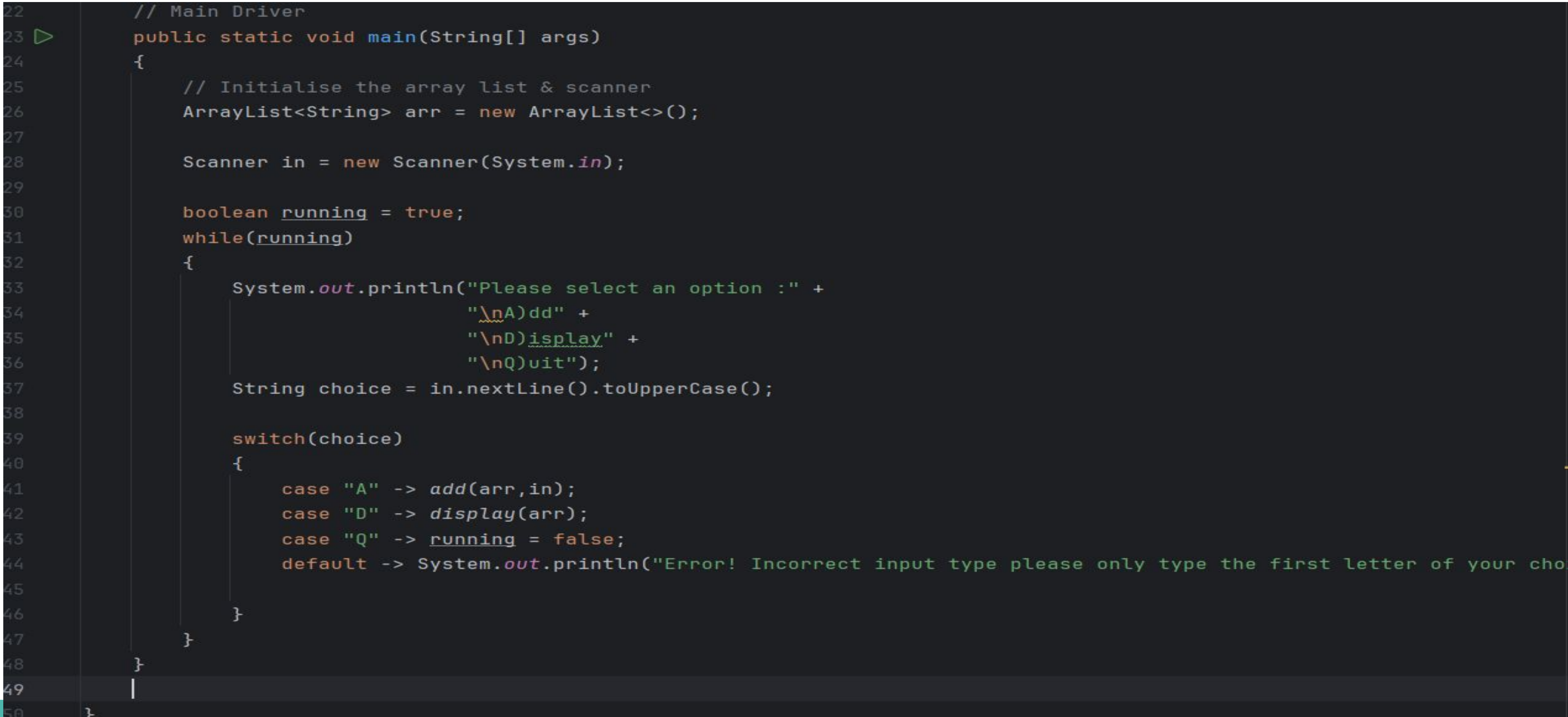
Or quit the program.

Solution Part 1 : Add and Display methods



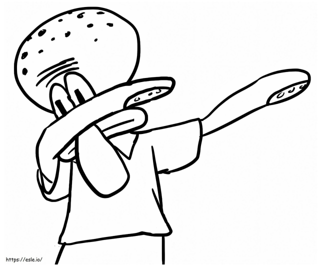
```
import java.util.ArrayList;
import java.util.Scanner;

public class Problem1
{
    // Method for adding Lectures
    public static void add(ArrayList<String> arr, Scanner in)  1 usage
    {
        System.out.println("Please type a module name to be added : ")
        arr.add(in.nextLine());
    }

    // Method for displaying lectures
    public static void display(ArrayList<String> arr)  1 usage
    {
        for(int i = 0; i<arr.size();i++)
            System.out.printf("Lecture %d : %s \n",i+1,arr.get(i));
    }
```

Solution Part 2 : Main Driver



```java
    // Main Driver
    public static void main(String[] args)
    {
        // Initialise the array list & scanner
        ArrayList<String> arr = new ArrayList<>();

        Scanner in = new Scanner(System.in);

        boolean running = true;
        while(running)
        {
            System.out.println("Please select an option :" +
                                "\nA)dd" +
                                "\nD)isplay" +
                                "\nQ)uit");
            String choice = in.nextLine().toUpperCase();

            switch(choice)
            {
                case "A" -> add(arr,in);
                case "D" -> display(arr);
                case "Q" -> running = false;
                default -> System.out.println("Error! Incorrect input type please only type the first letter of your cho

            }
        }
    }

}
```

Stacks and Queues



A stack is a data structure with two main operations

push() # pushes the element onto the top of the stack

pop() # pops the element off the stack

this restriction on the operations on how elements can be accessed is also known as a First in first out approach (Or FILO) and means that elements in a stack can only be accessed by popping them.

Due to these restrictions insertion and deletion in a stack is O(1) but searching elements is at worst O(N)

Stack Syntax

Import

```
import java.util.Stack;
```

Initialisation

```
Stack<Type> name = new Stack<>();
```

Useful Methods

```
name.push(item);

name.pop();
```

Stack problem

Write a Java program that checks if a given string is a palindrome using a stack.

Input:

A single string S (consisting of lowercase English letters).

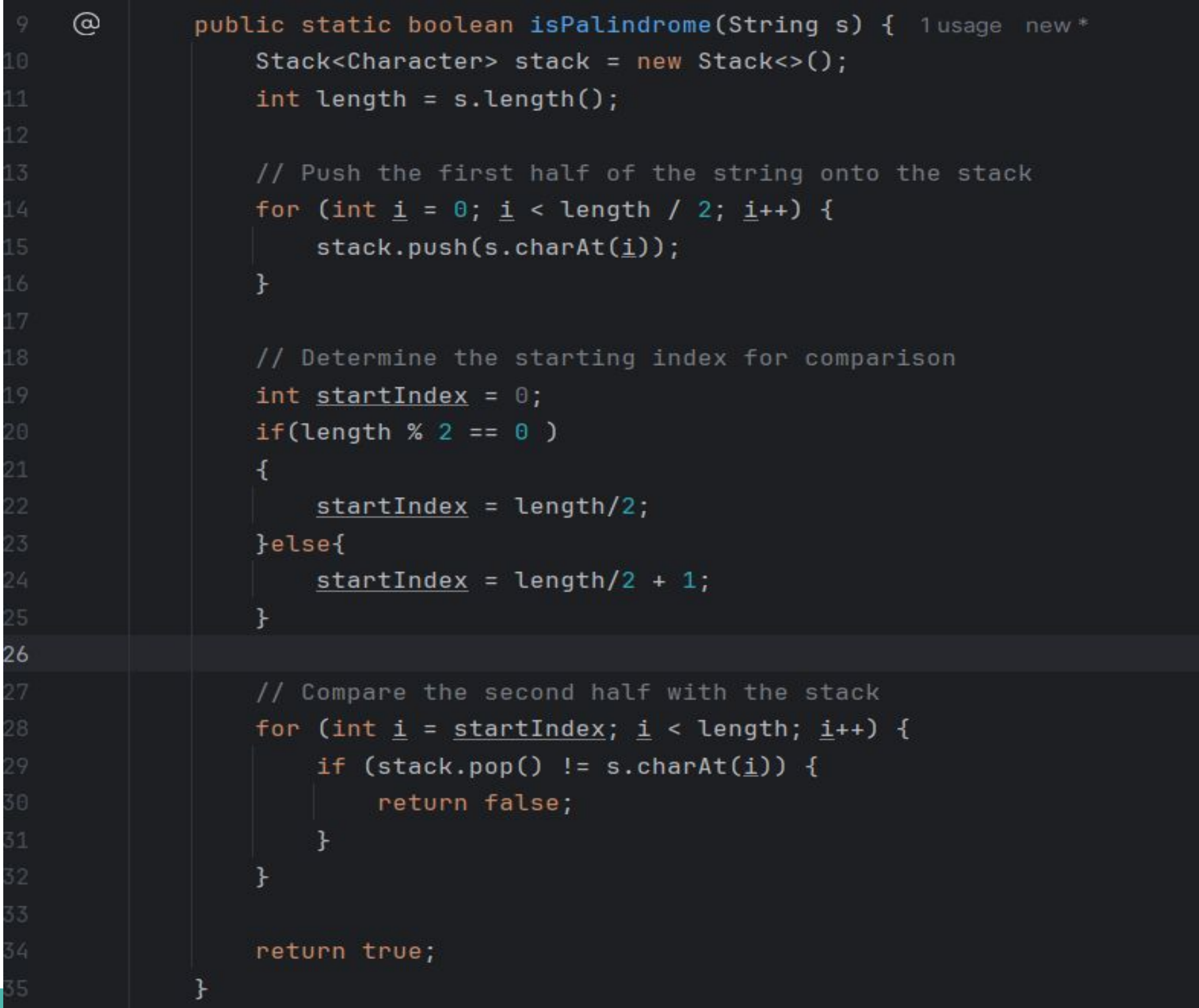
Output:

Print `"YES"` if the string is a palindrome, otherwise print `"NO"`.

Example:

```
Input: "racecar"

Output: "YES"
```

Solution



```java
    public static boolean isPalindrome(String s) {
        Stack<Character> stack = new Stack<>();
        int length = s.length();

        // Push the first half of the string onto the stack
        for (int i = 0; i < length / 2; i++) {
            stack.push(s.charAt(i));
        }

        // Determine the starting index for comparison
        int startIndex = 0;
        if(length % 2 == 0 )
        {
            startIndex = length/2;
        }else{
            startIndex = length/2 + 1;
        }

        // Compare the second half with the stack
        for (int i = startIndex; i < length; i++) {
            if (stack.pop() != s.charAt(i)) {
                return false;
            }
        }

        return true;
    }
```

Stacks and Queues

Queues are the opposite of stacks they operate off a first in first out system (FIFO)

Think of being in a queue in the shop (Painful I know)

it has similar key operations to a stack except they use different key words

e.g.

enqueue

dequeue

Queue Syntax

Import

```
import java.util.Queue;
```

Initialisation

```
Queue<Type> name = new LinkedList<>();
```

Useful Methods

```
name.offer(item);

name.poll();
```

Queue Problem

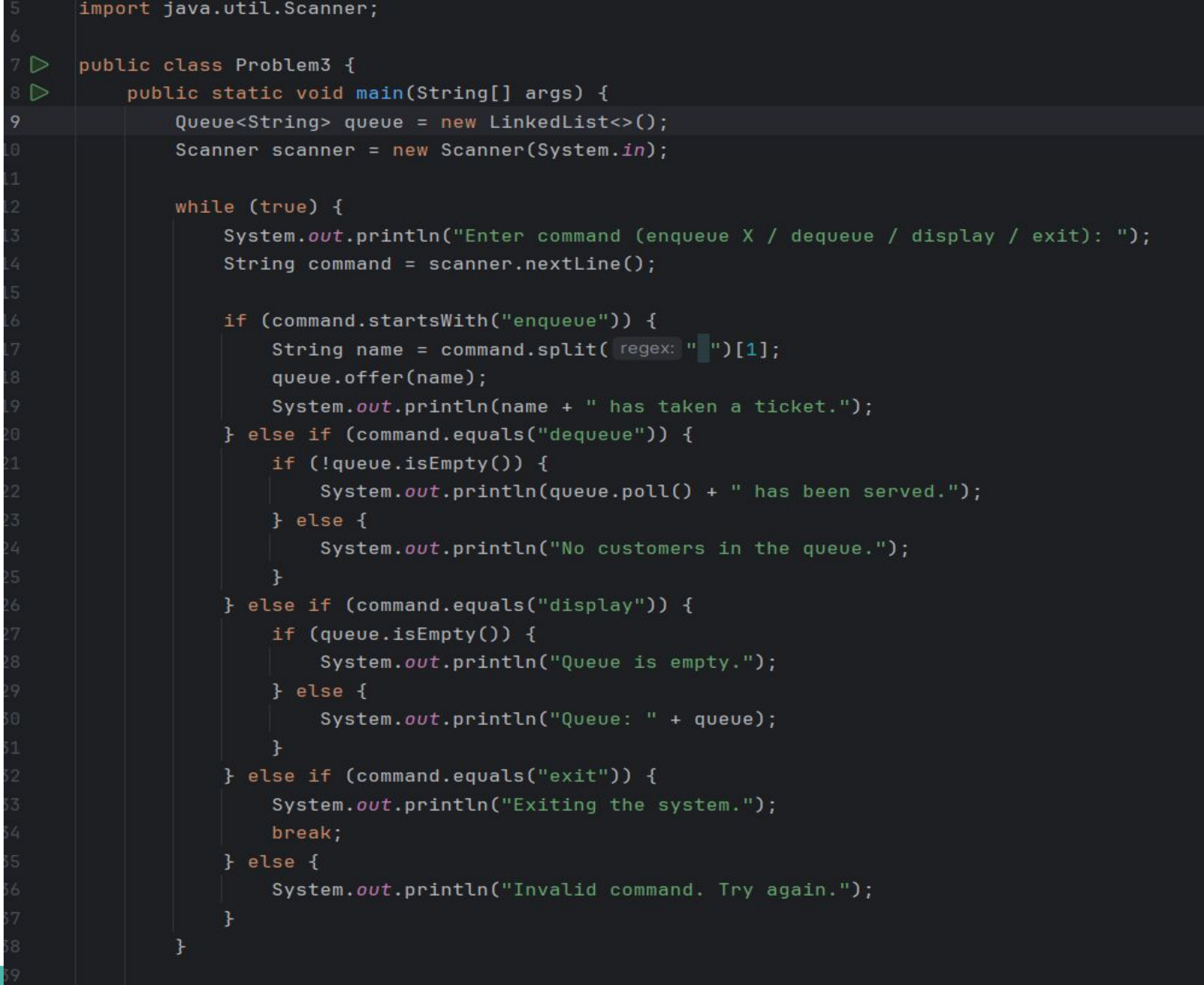
You are building a ticketing system where customers arrive in a queue to be served. Implement a Java program using a queue that:

1. Allows customers to take a ticket (enqueue).
2. Serves the customer at the front of the queue (dequeue).
3. Displays the current queue of customers.

Input & Operations:

- `"enqueue X"` → Adds customer `X` to the queue.
- `"dequeue"` → Removes the customer at the front of the queue.
- `"display"` → Shows the current queue from front to back.

Queue Solution



```java
import java.util.Scanner;

public class Problem3 {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("Enter command (enqueue X / dequeue / display / exit): ");
            String command = scanner.nextLine();

            if (command.startsWith("enqueue")) {
                String name = command.split( regex: " ")[1];
                queue.offer(name);
                System.out.println(name + " has taken a ticket.");
            } else if (command.equals("dequeue")) {
                if (!queue.isEmpty()) {
                    System.out.println(queue.poll() + " has been served.");
                } else {
                    System.out.println("No customers in the queue.");
                }
            } else if (command.equals("display")) {
                if (queue.isEmpty()) {
                    System.out.println("Queue is empty.");
                } else {
                    System.out.println("Queue: " + queue);
                }
            } else if (command.equals("exit")) {
                System.out.println("Exiting the system.");
                break;
            } else {
                System.out.println("Invalid command. Try again.");
            }
        }
```