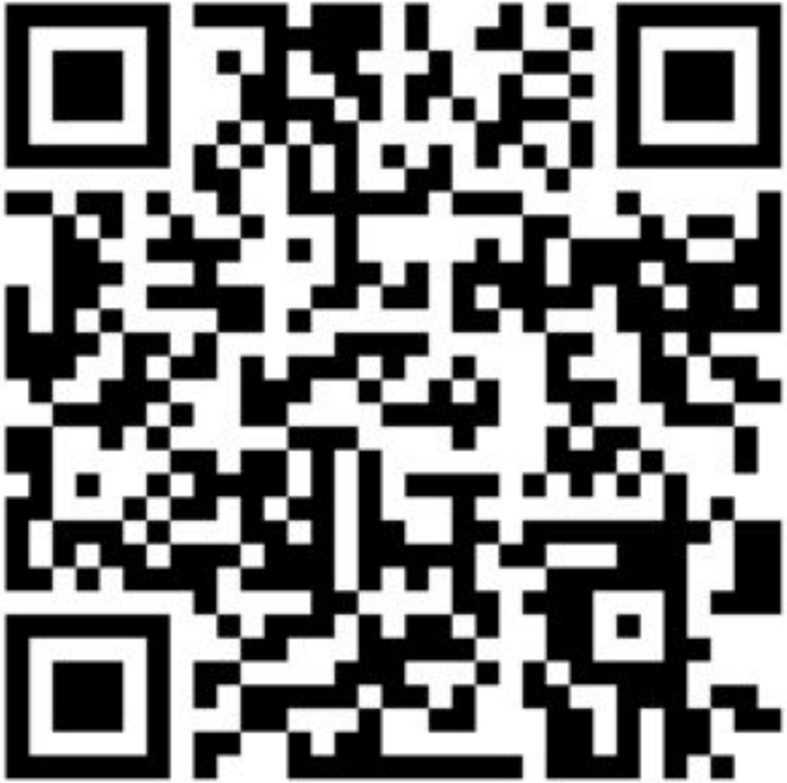
PSLG Week07

Ran by Amy and Ben

ICTLC Online QR Code:



Github

Agenda

- Lists
- Stacks
- Queues

Lists

- Indexed data structure (elements are accessed by calling the index that stores them).
- Have dynamic sizes and can grow with data.
  - Examples:
    - Linked Lists : Elements linked together via pointers.
    - Array Lists : Dynamically resizable arrays, commonly found in java.

ArrayList Syntax

Import:

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

Initialisation:

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

Important methods:

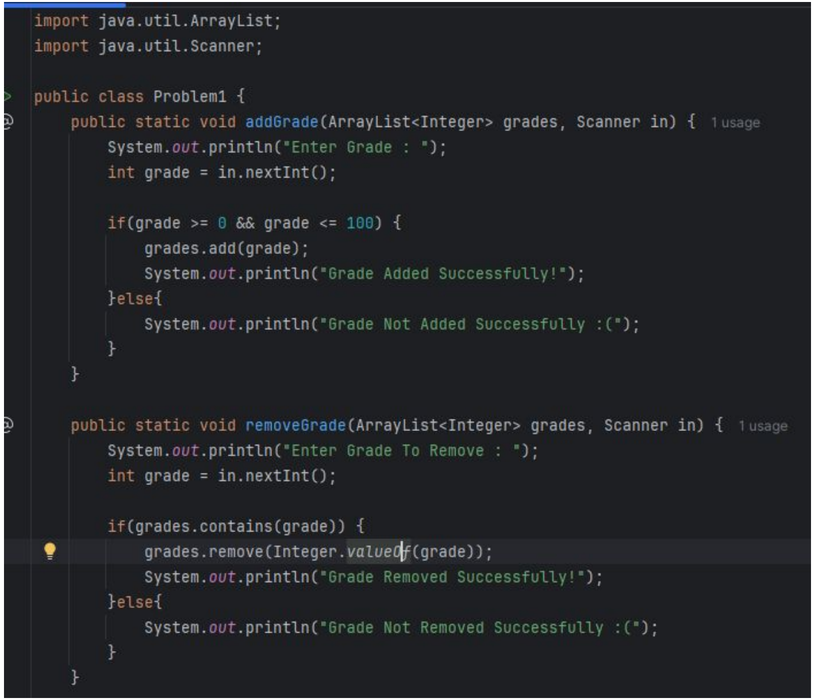
```
name.add(element);

name.get(index);
```

Problem 1 🗿

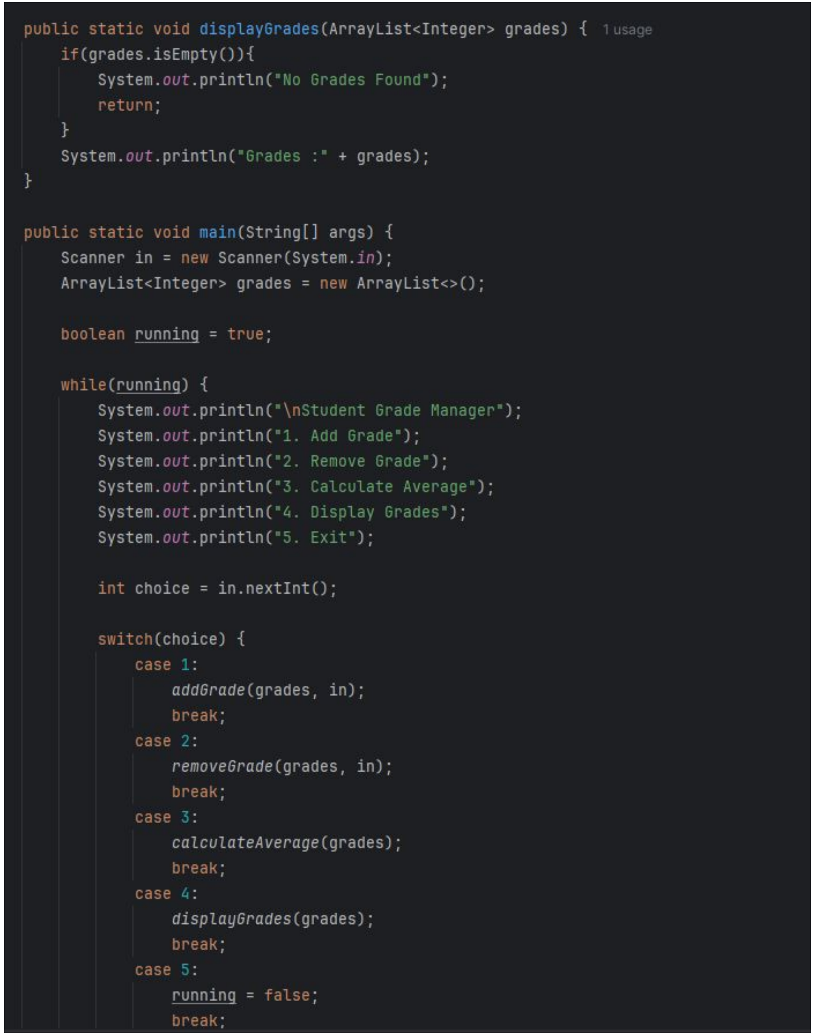
Create a program that manages students grades. The program should be able to handle:

1. Add a students grade(Integer between 0 & 100).
2. Remove a specific grade(if it exists in the list).
3. Display all grades in the list.

Solution

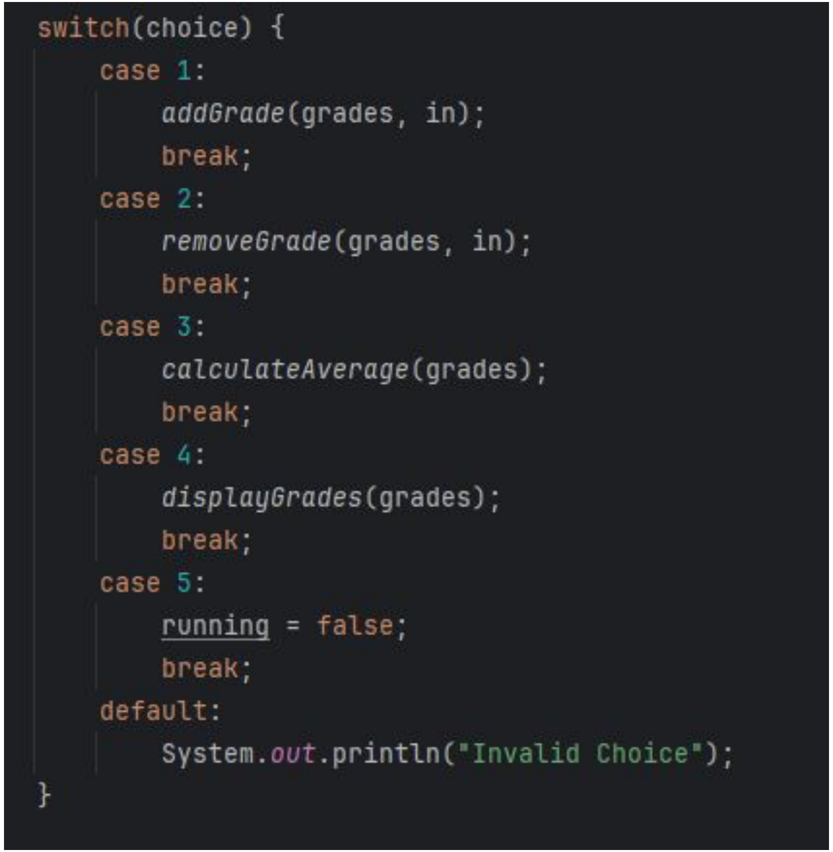


```java
import java.util.ArrayList;
import java.util.Scanner;

public class Problem1 {
    public static void addGrade(ArrayList<Integer> grades, Scanner in) {  1 usage
        System.out.println("Enter Grade : ");
        int grade = in.nextInt();

        if(grade >= 0 && grade <= 100) {
            grades.add(grade);
            System.out.println("Grade Added Successfully!");
        }else{
            System.out.println("Grade Not Added Successfully :(");
        }
    }

    public static void removeGrade(ArrayList<Integer> grades, Scanner in) {  1 usage
        System.out.println("Enter Grade To Remove : ");
        int grade = in.nextInt();

        if(grades.contains(grade)) {
            grades.remove(Integer.valueOf(grade));
            System.out.println("Grade Removed Successfully!");
        }else{
            System.out.println("Grade Not Removed Successfully :(");
        }
    }
```



```java
public static void displayGrades(ArrayList<Integer> grades) {  1 usage
    if(grades.isEmpty()){
        System.out.println("No Grades Found");
        return;
    }
    System.out.println("Grades :" + grades);
}

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    ArrayList<Integer> grades = new ArrayList<>();

    boolean running = true;

    while(running) {
        System.out.println("\nStudent Grade Manager");
        System.out.println("1. Add Grade");
        System.out.println("2. Remove Grade");
        System.out.println("3. Calculate Average");
        System.out.println("4. Display Grades");
        System.out.println("5. Exit");

        int choice = in.nextInt();

        switch(choice) {
            case 1:
                addGrade(grades, in);
                break;
            case 2:
                removeGrade(grades, in);
                break;
            case 3:
                calculateAverage(grades);
                break;
            case 4:
                displayGrades(grades);
                break;
            case 5:
                running = false;
                break;
```

Solution(continued)



```
switch(choice) {
    case 1:
        addGrade(grades, in);
        break;
    case 2:
        removeGrade(grades, in);
        break;
    case 3:
        calculateAverage(grades);
        break;
    case 4:
        displayGrades(grades);
        break;
    case 5:
        running = false;
        break;
    default:
        System.out.println("Invalid Choice");
}
```

Stacks

Stacks are a data structure with two main operations :

- push() → pushes the element on top of the stack
- pop() → pops the element off the stack

Stacks have restricted access known as First in, Last out (FILO), meaning elements in a stack can only be accessed by popping them.

Stack Syntax

Import

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

Initialisation

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

Useful Methods

```
name.push(item);

name.pop();
```

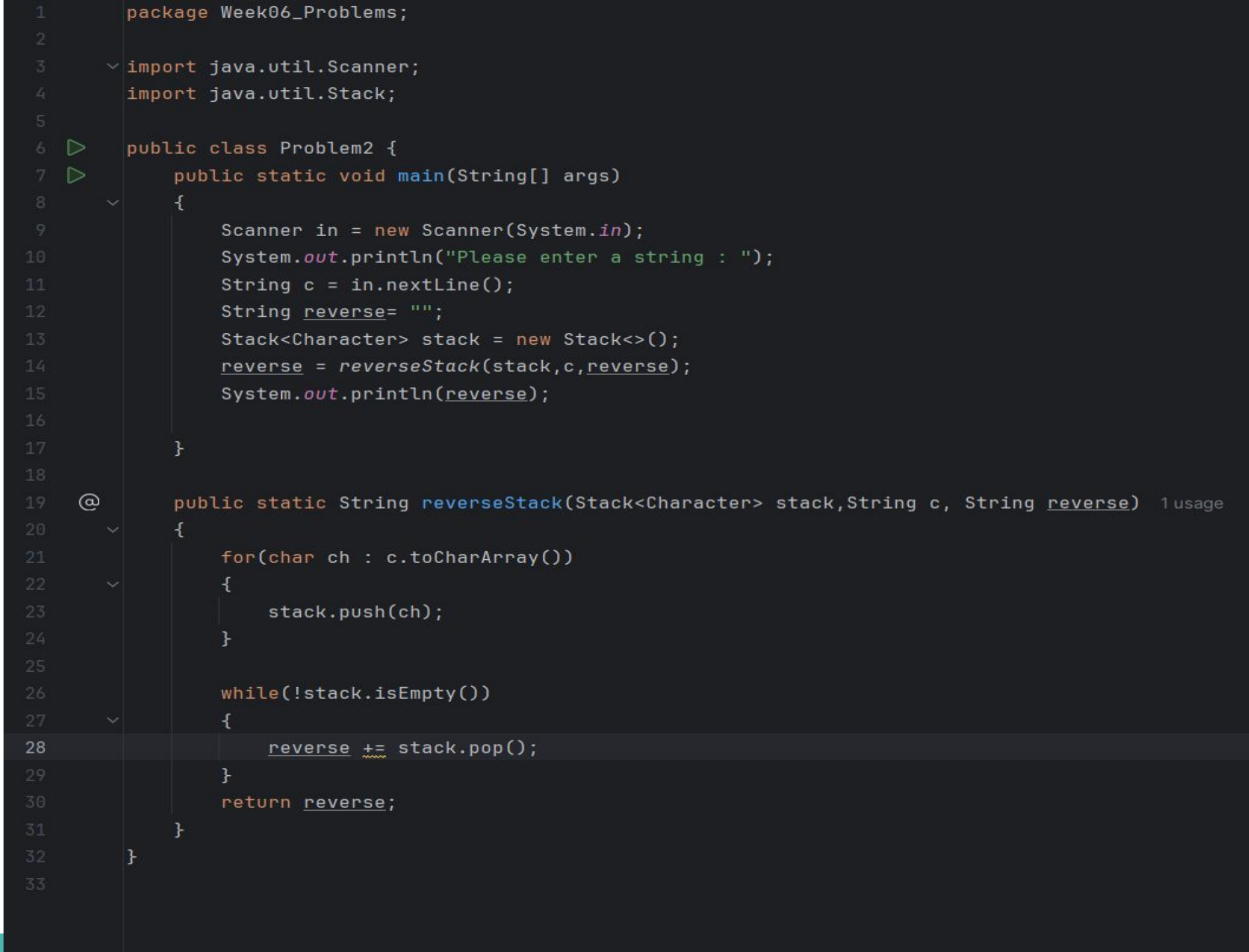
Problem 2 😎

Create a program that utilises a stack to reverse a given string.

The user must input a string and the program must output that string in reverse order.

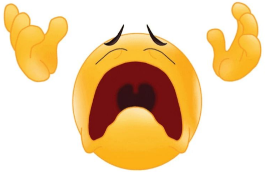
Eg. hello should be outputted by the program as olleh.

Solution



```java
package Week06_Problems;

import java.util.Scanner;
import java.util.Stack;

public class Problem2 {
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.println("Please enter a string : ");
        String c = in.nextLine();
        String reverse= "";
        Stack<Character> stack = new Stack<>();
        reverse = reverseStack(stack,c,reverse);
        System.out.println(reverse);

    }

    public static String reverseStack(Stack<Character> stack,String c, String reverse)
    {
        for(char ch : c.toCharArray())
        {
            stack.push(ch);
        }

        while(!stack.isEmpty())
        {
            reverse += stack.pop();
        }
        return reverse;
    }
}
```

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)

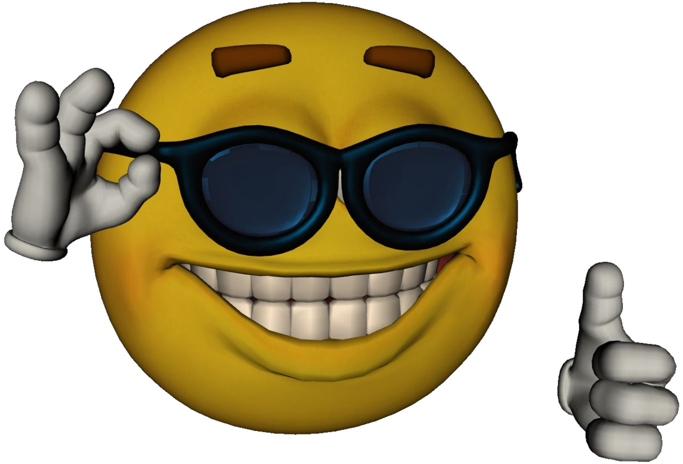
it has similar key operations to a stack except they use different key words

e.g.

enqueue

dequeue

Queues Syntax



Import

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

Initialisation

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

Useful Methods

```
name.offer(item);

name.poll();
```

PROBLEM ❗❗❗

Java Queue Madness: The Time-Traveling Breadline

Congratulations! You’ve just been hired as the lead software engineer for **QuantumBakery™,** a futuristic bakery that serves customers from **all points in time simultaneously**. However, due to a slight miscalculation in your time-traveling queue system, customers are **entering and leaving at completely unpredictable intervals**.

PROBLEM ❗❗❗

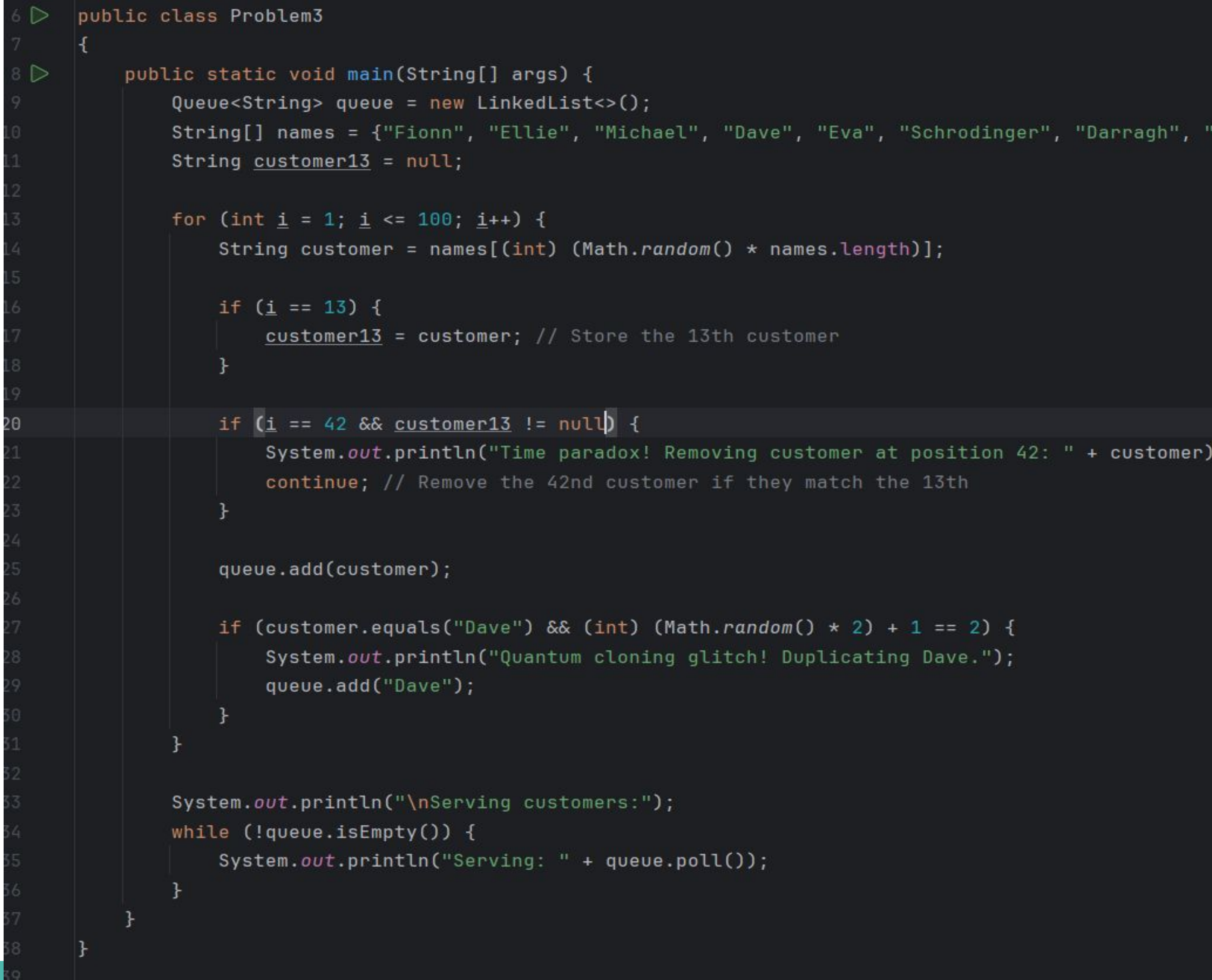
Your task is to implement a **QuantumQueue of type string** using Java's Queue interface that does the following:

1. Add **100 random customers** to the queue and serve them applying the following rules during adding them.

2. Every 42nd customer **is actually the same person as the 13th customer, due to a time paradox**, so they must be removed when they first appear.

3. If a customer's name is "**Dave**," he has a **50% chance of being instantly duplicated** in the queue because of a quantum cloning glitch.

Solution



```java
public class Problem3
{
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        String[] names = {"Fionn", "Ellie", "Michael", "Dave", "Eva", "Schrodinger", "Darragh", "
        String customer13 = null;

        for (int i = 1; i <= 100; i++) {
            String customer = names[(int) (Math.random() * names.length)];

            if (i == 13) {
                customer13 = customer; // Store the 13th customer
            }

            if (i == 42 && customer13 != null) {
                System.out.println("Time paradox! Removing customer at position 42: " + customer)
                continue; // Remove the 42nd customer if they match the 13th
            }

            queue.add(customer);

            if (customer.equals("Dave") && (int) (Math.random() * 2) + 1 == 2) {
                System.out.println("Quantum cloning glitch! Duplicating Dave.");
                queue.add("Dave");
            }
        }

        System.out.println("\nServing customers:");
        while (!queue.isEmpty()) {
            System.out.println("Serving: " + queue.poll());
        }
    }
}
```