PARTY SIMULATOR PROJECT REPORT

Beyza Sungar, Mustafa Eren Tuğcu
200316032, 200316035

ABSTRACT This report analyzes a Java program simulating a party scenario where guests consume food and drinks from shared trays while a waiter refills them. The simulation involves multiple threads representing guests and a waiter, along with shared resources such as food trays for börek, cake, and drinks. Guests have limits on consumption, and trays have fixed capacities. Through this simulation, the interaction between multiple threads sharing resources is illustrated, along with the importance of synchronization to maintain program correctness.

I. INTRODUCTION

The provided Java program simulates a party scenario where guests and a waiter interact in a multi-threaded environment. In this simulation, guests consume food and drinks from shared trays, while the waiter continuously refills the trays as needed. The program represents a practical example of concurrent programming, demonstrating synchronization and coordination between threads to manage shared resources effectively.

The simulation involves several key components: PartySimulation orchestrates the overall scenario, initializing shared resources and starting guest and waiter threads. FoodTray manages the current capacity of items and provides methods for taking items and refilling trays. Guests have predefined consumption limits for each item, and they consume items from the trays until they reach their limits. The waiter checks for empty trays and refills them until all items are consumed.

This report will provide a detailed analysis of the program's design, functionality, and behavior. It will discuss the design overview, simulation details, and conclude with insights into multi-threading concepts demonstrated in the simulation.

II. METHODS

A. DESIGN OVERVIEW

The program is structured around four main classes: PartySimulation, FoodTray, Guest, and Waiter.

PartySimulation:

- Initializes shared resources (food trays).
- Starts guest and waiter threads.
- Manages thread synchronization using join().

FoodTray:

- Manages item quantities on trays.
- Provides synchronized methods for taking items and refilling trays.
- Handles thread synchronization using wait() and notifyAll().

Guest:

- Represents individual guests at the party.
- Consumes items from trays until reaching consumption limits.
- Interacts with FoodTray for item consumption and waiting for refills.

Waiter:

- Monitors trays for empty status.
- Refills trays when empty.
- Interacts with FoodTray for tray refilling.

B. IMPLEMENTATION ANALYSIS

PartySimulation Implementation:

- Initializes shared resources and starts threads.
- Ensures all threads are properly joined to end the simulation.

FoodTray Implementation:

- Manages current item quantities on trays.
- Provides synchronized methods for thread-safe access.
- Implements wait() and notifyAll() for thread synchronization.

Guest Implementation:

- Consumes items from trays.
- Waits for refills if a tray is empty.
- Simulates consumption with a sleep period.

Waiter Implementation:

- Monitors trays and refills them when empty.
- Refills trays based on remaining quantities of items.

C. FUNCTIONALITY ANALYSIS

Guest Behavior: Guests consume items from trays until reaching consumption limits, waiting for refills if necessary.

Waiter Behavior: The waiter continuously checks for empty trays and refills them until all items are consumed.

Synchronization: Thread synchronization ensures proper coordination between guests and the waiter, preventing race conditions and maintaining data integrity.

III.REFERENCES

- Java documentation: https://docs.oracle.com/en/java/
- Silberschatz, A., Galvin P.B., & Gagne, G.. Operating Systems Concepts of Essencials(Second Edition). John Wiley & Sons, Inc. ISBN 978-1-118-80492-6.

IV. CONCLUSION

The Party Simulation program demonstrates effective multi-threading concepts in Java. It illustrates synchronization and coordination between threads to manage shared resources in a concurrent environment. The analysis provides insights into how each class contributes to the functionality of the simulation and how thread synchronization is managed to ensure correct behavior.