Lab 1 - Part 1 - Design Document

Overview and Objectives:

The objective of this project is to develop a socket-based client-server application for Asterix's online Toy Store offering two exclusive products: Tux and Whale stuffed animals. The server component is designed to process queries about the availability and pricing of these toys using a custom thread pool mechanism to efficiently manage incoming requests. This thread pool, which is configurable at start-up, ensures that a static number of threads are always ready to handle client queries, thereby optimizing resource utilization and response times.

The server listens on a specified network port, accepts client requests over socket connections, and delegates the requests to idle threads in the thread pool for processing. Each request includes the method name "Query" and the toy's name as arguments, with the server responding with the item's price if available in stock, -1 if the item is not found, and 0 if the item is out of stock.

On the client side, a simple for-loop mechanism is used to generate sequential query requests to the server. The design supports the use of multiple client processes to issue concurrent requests, demonstrating the thread pool's capability to handle simultaneous queries effectively. The core of the assignment involves implementing the server's thread pool, including the creation and synchronization of threads, managing the request queue, and ensuring smooth communication between the client and server over sockets. This project aims to showcase efficient thread management, socket programming, and synchronization techniques in a real-world application scenario.

Design:

Request Routing:

A client-server architecture where each client request is handled using a distinct socket connection. After the server responds to a request, the connection is terminated. For any new request from the client, a fresh socket connection is established, ensuring that each interaction is isolated and managed independently. Thread-per-request model was used.

1. **Server Acceptance**: Upon accepting a connection, the server does not process the request directly. Instead, it creates a task representing the client's request and adds it to the task queue using the ClientHandler interface.
2. **Task Queue**: This queue holds pending tasks. Its design ensures that incoming requests are managed in an orderly fashion, awaiting processing by available threads in the thread pool.
3. **Thread Pool Assignment**: The thread pool, consisting of pre-initialized worker threads, monitors the task queue. When tasks are present, threads are notified of a task and an idle thread retrieves a task from the queue.
4. **Request Processing**: The worker thread processes the task, which involves querying the in-memory catalog and preparing the response based on the specified operation (e.g., "Query").
5. **Response Return**: Once processed, the response is sent back to the client through the same connection, handled by the worker thread.

This model enhances the application's ability to handle multiple simultaneous requests.

We used PrintWriter and BufferedReader for input and output Streams. Initially we used DataInputStream and DataOutputStream, but due to restriction in size of data transferable by these, changed it to PrintWriter and BufferedReader. Used only synchronised operations for locking mechanisms as they were the most simple primitives.

There are 4 Components in this implementation: Server, ClientHandler, ThreadPool and Client.

The Server.java file outlines the server-side architecture for Asterix's online Toy Store, focusing on handling client requests via a socket-based communication. It incorporates a ServerSocket to listen for incoming connections and uses a

custom ThreadPool for managing concurrent requests efficiently. The server maintains an in-memory toy catalog, with each toy identified by a name, price, and stock level.

The server receives a request like (Query, itemName) and sends a response -1, 0, price, -2.

Response codes:

-1 = Item does not exist in the store

0 = Item exists but is not in stock

price = Item is in stock and price is sent

-2 = The query method is not valid.

> toyQuery method: Takes itemName as input and checks the catalog for the item and outputs one of the above response codes (-1, 0, price of the item). This method has the synchronised keyword attached to it so that race conditions are avoided. In this case where only Query method (read only) is used, race conditions don’t occur. But in general when shared data structures are used they oocur.

> Toy class: This class is used to create instances of different toys in the catalog. Each toy has a name, price and stock.

The ClientHandler.java file is responsible for handling individual client connections to the server. Upon accepting a connection, it processes client requests by reading from the input stream, executing the specified query (currently, only "Query" operations are supported, with a fallback for unrecognized commands), and sending back the response. This class plays a important role in the server's architecture by managing the lifecycle of each client connection, from parsing the request to returning the query result and closing the connection. It demonstrates the practical use of sockets for two-way communication between client and server.

> Uses the toyQuery method from server and issues it’s own error “-2” to inform the client of invalid query (like “Buy” or “Send”).

The ThreadPool.java file defines a custom thread pool for managing concurrent execution of tasks in the Asterix's online Toy Store application. The ThreadPool class initializes with a specified number of worker threads, maintains a queue for tasks, and facilitates adding tasks to this queue for execution. Worker threads continuously poll the task queue and execute tasks as they become available. The design ensures efficient task management and resource utilization by dynamically adjusting active threads based on the workload. This component is critical for handling multiple client requests simultaneously, showcasing an effective use of Java's concurrency features to achieve scalable server behavior.

> addTask method: Takes an instance of ClientHandler as task and adds it to taskQueue and notifies all the threads to pick up the task.

> getTask method: Every thread keeps checking for a task in the taskList and goes into waiting when the queue is empty. Once the threads know of a task, they remove the first task from the queue and execute it.

> stop method: Stops the thread pool.

> WorkerThread class: This class implements the basic threads for the threadpool. It has an instance of the threadpool to get the next task to run. And a variable isRunning to check if the current thread is running or not.

The Client.java file implements the client-side functionality for the Asterix's online Toy Store, using socket-based communication to interact with the server. The client constructs a query message, sends it to the server, and processes the response, which includes displaying the item's price, stock availability, or error messages based on the server's reply. It supports sending multiple randomized queries in a loop, measuring response times for performance analysis, and displays basic query statistics like average response time, and maximum and minimum request times. The client sends a request like (Query, itemName) to the server and receives a response -1, 0, price, -2.

> ClientQuery method: Takes input as (Query, itemName) and sends a request to the server and displays the response.

Citations:

1. https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html
2. Course on Java Multithreading : https://learning.oreilly.com/course/java-multithreading-and/9781804619377/