Lab 5 Report

Producer Consumer Web App

Prepared by: Saifullah Musaad 21010651

Project on GitHub: https://github.com/Saifullah-1/Producer-Consumer.git

Executive Summary:

An assembly line that produces different products consists of different processing machines Ms that are responsible for processing the product at different stages and queue Qs to handle product movement between different processing stages. In this assignment, we will develop a simulation program to simulate this production line as a queuing network.

Project Goals:

1. Design an object-oriented queueing simulation program.
2. Draw a UML class diagram that represents your model.
3. Apply different design patterns such as concurrency DP, snapshot DP, and observer DP.

Project Team Members:

Member 1:	Saifullah Musaad	21010651
Member 2:	Ahmed Ashraf	21010040
Member 3:	Ahmed Osama	21010037
Member 4:	Rawan Mohamed	21010547

Steps to run the app:

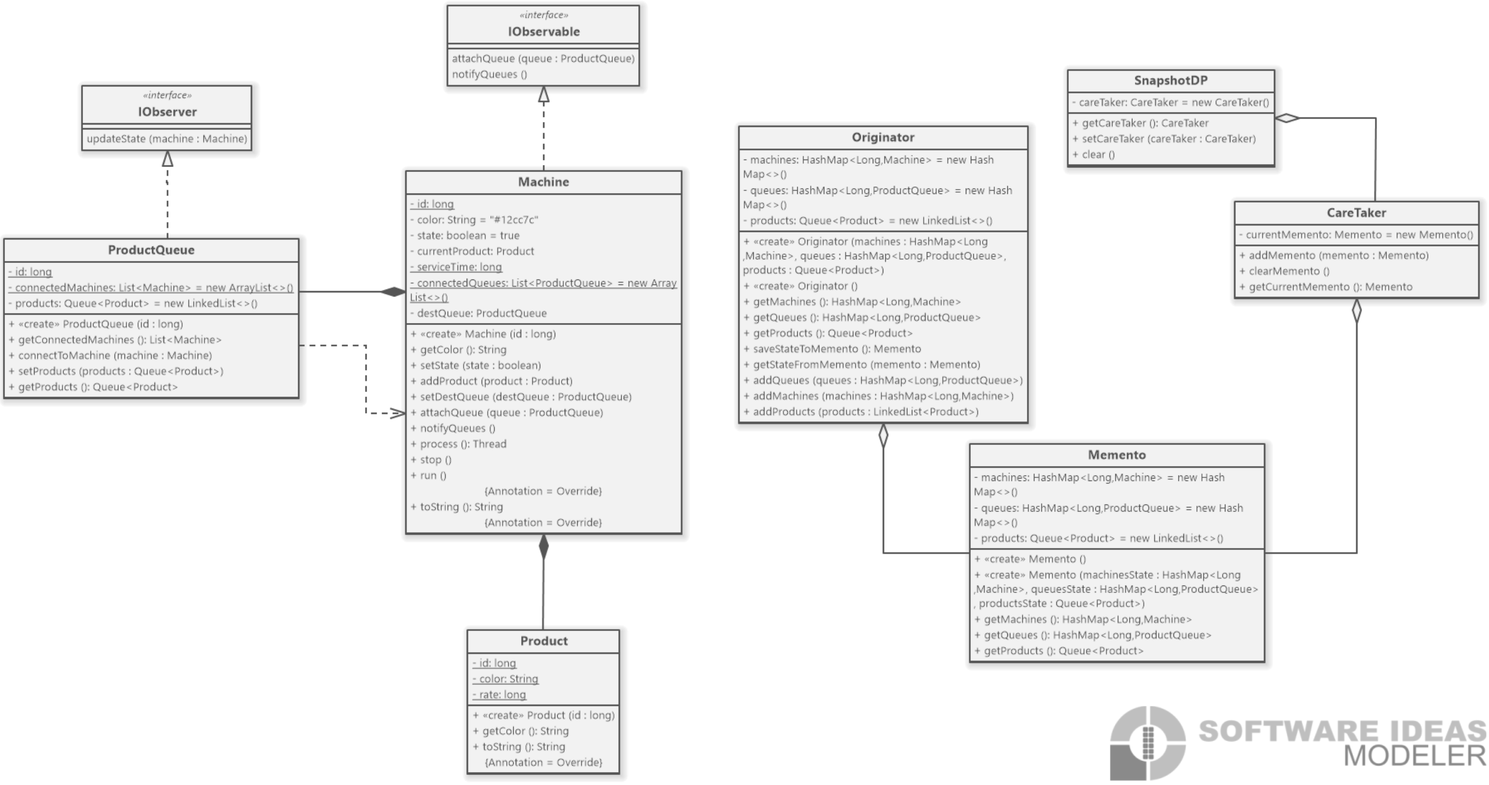
- Installing npm package in the Frontend folder >> **npm install**

- Open **VueJs** on server >> **npm run serve**

- In *Controller.java* write in *@CrossOrigin* the server that **VueJs** running on

- Run *BackendApplication.java* file

UML Diagram:



«interface»
IObservable
attachQueue (queue : ProductQueue)
notifyQueues ()
«interface»
IObserver
updateState (machine : Machine)
ProductQueue
- id: long
- connectedMachines: List<Machine> = new ArrayList<>()
- products: Queue<Product> = new LinkedList<>()
+ «create» ProductQueue (id : long)
+ getConnectedMachines (): List<Machine>
+ connectToMachine (machine : Machine)
+ setProducts (products : Queue<Product>)
+ getProducts (): Queue<Product>
Machine
- id: long
- color: String = "#12cc7c"
- state: boolean = true
- currentProduct: Product
- serviceTime: long
- connectedQueues: List<ProductQueue> = new ArrayList<>()
- destQueue: ProductQueue
+ «create» Machine (id : long)
+ getColor (): String
+ setState (state : boolean)
+ addProduct (product : Product)
+ setDestQueue (destQueue : ProductQueue)
+ attachQueue (queue : ProductQueue)
+ notifyQueues ()
+ process (): Thread
+ stop ()
+ run ()
{Annotation = Override}
+ toString (): String
{Annotation = Override}
Product
- id: long
- color: String
- rate: long
+ «create» Product (id : long)
+ getColor (): String
+ toString (): String
{Annotation = Override}
SnapshotDP
- careTaker: CareTaker = new CareTaker()
+ getCareTaker (): CareTaker
+ setCareTaker (careTaker : CareTaker)
+ clear ()
Originator
- machines: HashMap<Long,Machine> = new HashMap<>()
- queues: HashMap<Long,ProductQueue> = new HashMap<>()
- products: Queue<Product> = new LinkedList<>()
+ «create» Originator (machines : HashMap<Long,Machine>, queues : HashMap<Long,ProductQueue>, products : Queue<Product>)
+ «create» Originator ()
+ getMachines (): HashMap<Long,Machine>
+ getQueues (): HashMap<Long,ProductQueue>
+ getProducts (): Queue<Product>
+ saveStateToMemento (): Memento
+ getStateFromMemento (memento : Memento)
+ addQueues (queues : HashMap<Long,ProductQueue>)
+ addMachines (machines : HashMap<Long,Machine>)
+ addProducts (products : LinkedList<Product>)
CareTaker
- currentMemento: Memento = new Memento()
+ addMemento (memento : Memento)
+ clearMemento ()
+ getCurrentMemento (): Memento
Memento
- machines: HashMap<Long,Machine> = new HashMap<>()
- queues: HashMap<Long,ProductQueue> = new HashMap<>()
- products: Queue<Product> = new LinkedList<>()
+ «create» Memento ()
+ «create» Memento (machinesState : HashMap<Long,Machine>, queuesState : HashMap<Long,ProductQueue>, productsState : Queue<Product>)
+ getMachines (): HashMap<Long,Machine>
+ getQueues (): HashMap<Long,ProductQueue>
+ getProducts (): Queue<Product>

How Design Patterns was applied:

Snapshot: To apply the Snapshot (Memento) pattern, we created a “Memento” class to store your system's state and used an “Originator” to save and restore states. Utilize a “CareTaker” to manage and retrieve multiple states when needed.

Observer: To implement the Observer pattern, we created an interface “IObservable” for subjects and “IObserver” for observers. In classes, we used these interfaces (“Machine” and “ProductQueue”) to establish a communication mechanism where observers are notified of changes in subjects.

Concurrency: Concurrency is applied using threads. Specifically, the Machine class implements the “Runnable” interface and starts a new thread in its process method. This allows multiple machines to run concurrently, each handling its processing independently. Additionally, synchronization mechanisms are used, such as synchronized methods in the “ProductQueue” class, to ensure thread safety when accessing shared data structures.

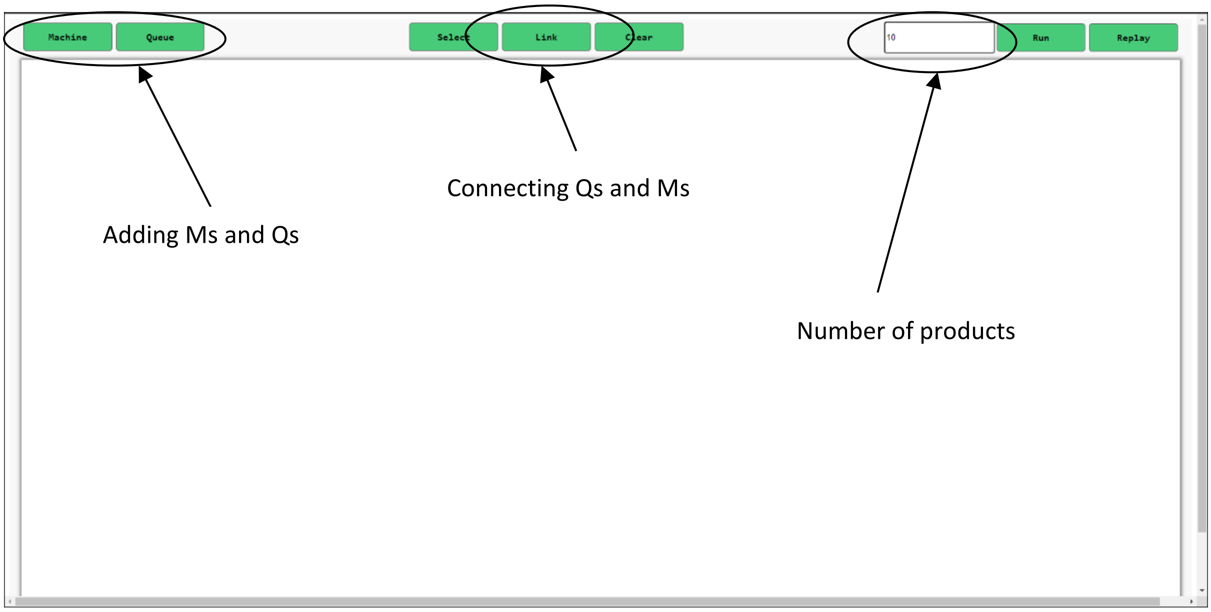
Design decisions:

- The maximum number of products is 100 because of the lack of colors.

- The range of service time is between 1, 30 seconds.

- User must add an output and input stream queue.

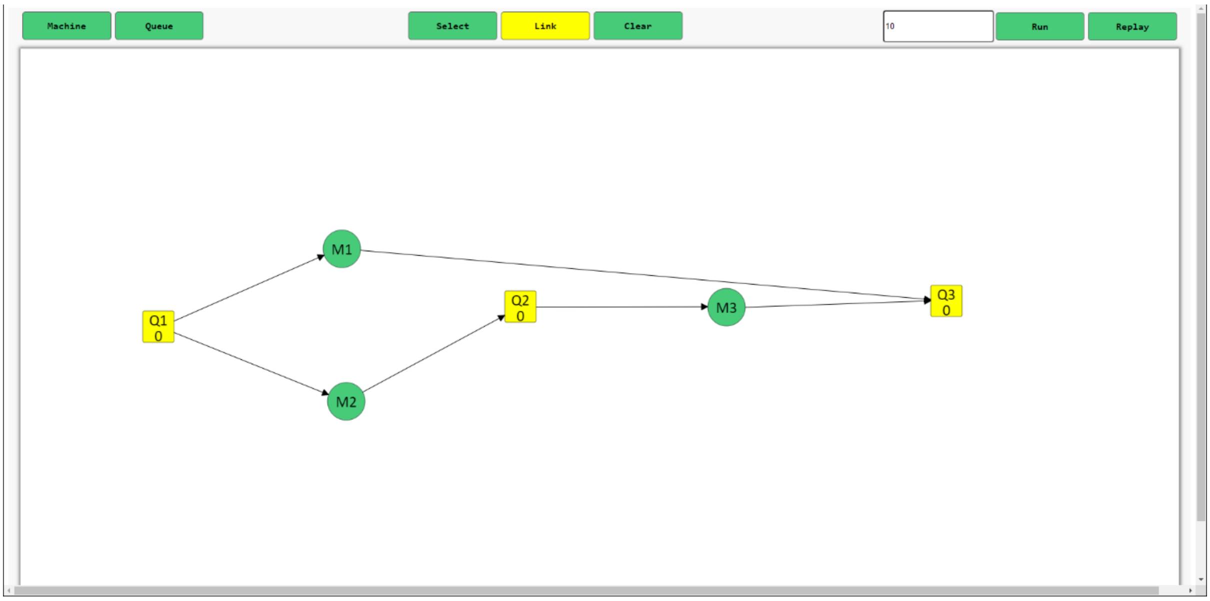
Snapshots of app UI and user guide:

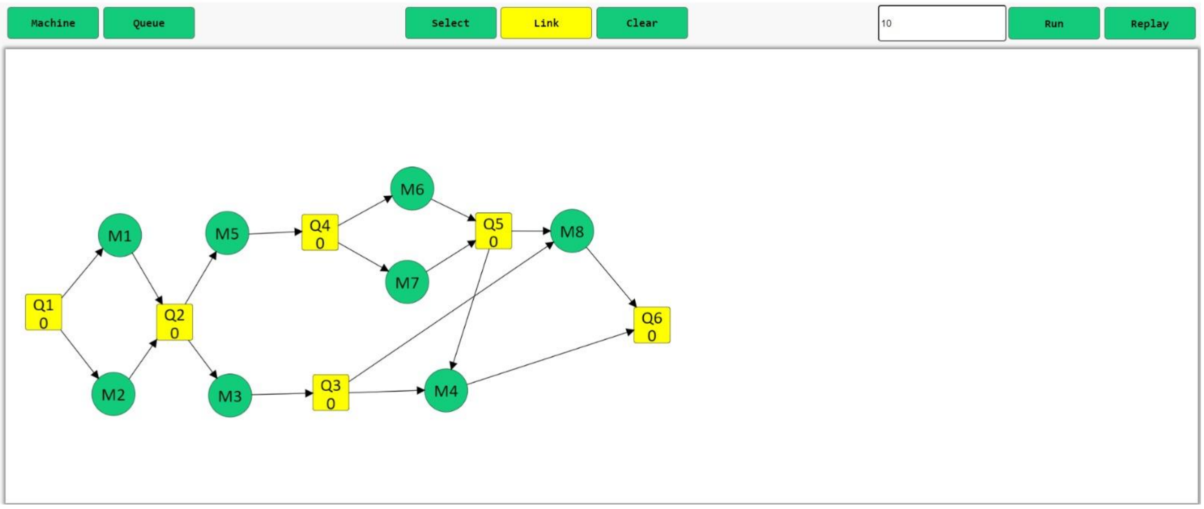


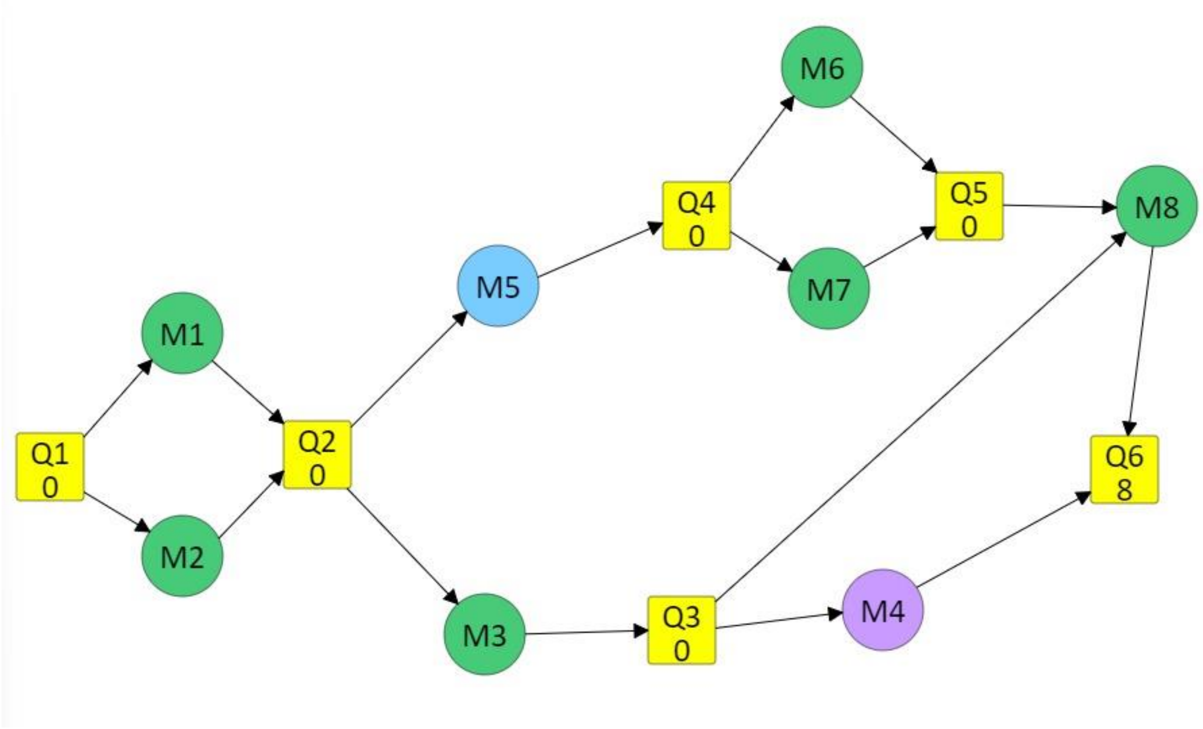
Adding Ms and Qs

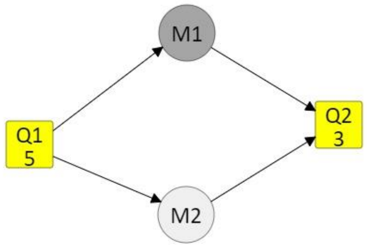
Connecting Qs and Ms

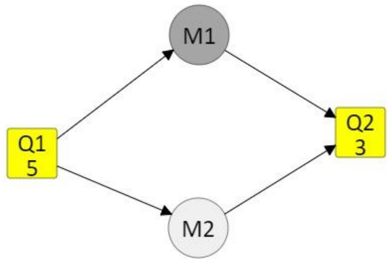
Number of products




Machine
Queue
Select
Link
Clear
10
Run
Replay
Q1 0
M1
M2
Q2 0
M5
M3
Q4 0
M6
M7
Q5 0
M8
Q3 0
M4
Q6 0


Q1 0
M1
M2
Q2 0
M5
M3
Q4 0
M6
M7
Q5 0
M8
Q3 0
M4
Q6 8


M1
Q1 5
Q2 3
M2


M1
Q1 5
Q2 3
M2