**Labsheet 9:**

Implement java program demonstrate the concept of producer-consumer synchronization using the Q, Producer, and Consumer classes, and what is the purpose of the synchronized, wait, and notify methods in achieving this synchronization?`

**Aim:**

The aim of the provided Java program is to demonstrate a solution to the producer-consumer problem using multithreading and synchronized methods. In this scenario, multiple producer and consumer threads are sharing a common resource (an instance of class Q), and the goal is to ensure that producers and consumers coordinate in a way that prevents issues.

**package** Labsheet9;

**import** java.io.\*;

**class** Q {

**int** n;

**boolean** valueSet = **false**;

**synchronized** **int** get() {

**while**(!valueSet)

**try** {

wait();

} **catch**(InterruptedException e) {

System.***out***.println("InterruptedException caught");

}

System.***out***.println("Got: " + n);

valueSet = **false**;

notify();

**return** n;

}

**synchronized** **void** put(**int** n) {

**while**(valueSet)

**try** {

wait();

} **catch**(InterruptedException e) {

System.***out***.println("InterruptedException caught");

}

**this**.n = n;

valueSet = **true**;

System.***out***.println("Put: " + n);

notify();

}

}

**class** Producer **implements** Runnable {

Q q;

Producer(Q q) {

**this**.q = q;

**new** Thread(**this**, "Producer").start();

}

**public** **void** run() {

**int** i = 0;

**while**(**true**) {

q.put(i++);

}

}

}

**class** Consumer **implements** Runnable {

Q q;

Consumer(Q q) {

**this**.q = q;

**new** Thread(**this**, "Consumer").start();

}

**public** **void** run() {

**while**(**true**) {

q.get();

}

}

}

**class** PCFixed {

**public** **static** **void** main(String args[]) {

Q q = **new** Q();

**new** Producer(q);

**new** Consumer(q);

System.***out***.println("Press Control-C to stop.");

}

}

**Algorithm**:

1. Create a shared class **Q** that has two synchronized methods: **put** and **get**.
   * In the **put** method, producers add data to the shared resource, and if the resource is full, they wait for consumers to consume.
   * In the **get** method, consumers retrieve data from the shared resource, and if the resource is empty, they wait for producers to produce data.
   * Use a **valueSet** boolean to keep track of whether data is available.
   * Use the **wait()** and **notify()** methods for synchronization.
2. Create two classes, **Producer** and **Consumer**, each implementing the **Runnable** interface.
   * The **Producer** class continuously adds data to the shared resource by calling the **q.put(i++)** method.
   * The **Consumer** class continuously retrieves data from the shared resource by calling the **q.get()** method.
3. In the **main** method of the **PCFixed** class:
   * Create an instance of the shared resource class **Q**.
   * Create instances of the **Producer** and **Consumer** classes, passing the shared resource **Q** as a parameter.
   * Start the producer and consumer threads.
   * The program runs indefinitely until manually stopped.

**Program Explanation**: The provided Java program demonstrates the producer-consumer problem solution with synchronization:

* The **Q** class represents the shared resource. It uses synchronized methods **put** and **get** to ensure safe access.
* Producers (class **Producer**) continuously produce data and add it to the shared resource.
* Consumers (class **Consumer**) continuously consume data from the shared resource.
* The **main** method creates instances of the shared resource and both producer and consumer threads.
* The program starts the threads and runs indefinitely.
* Producers and consumers coordinate with each other through synchronization to prevent issues such as data corruption and race conditions.

The program provides a simple example of how multithreading and synchronization can be used to solve the producer-consumer problem, where multiple threads share a common resource with well-defined access rules.

**Top of Form**

**Detailed Program Explanation**

1. **Q Class**:
   * **n**: An integer variable used to store the shared data.
   * **valueSet**: A boolean flag that indicates whether the shared data has been produced.
   * **synchronized int get()**: A synchronized method that allows a consumer to get the data. If **valueSet** is **false** (no data is available), the consumer will wait until data is produced (using the **wait()** method). Once data is available, it prints the data, sets **valueSet** to **false**, and notifies the producer (using the **notify()** method).
   * **synchronized void put(int n)**: A synchronized method that allows a producer to put data. If **valueSet** is **true** (data is already available), the producer will wait until the consumer consumes the data. Once the data is consumed, it stores the new data in **n**, sets **valueSet** to **true**, and notifies the consumer.
2. **Producer Class**:
   * This class implements the **Runnable** interface and represents the producer thread.
   * In the constructor, it receives an instance of the **Q** class and starts a new thread.
   * In the **run()** method, it keeps producing data by calling the **q.put(i++)** method, where **q** is the shared queue.
3. **Consumer Class**:
   * This class also implements the **Runnable** interface and represents the consumer thread.
   * In the constructor, it receives an instance of the **Q** class and starts a new thread.
   * In the **run()** method, it keeps consuming data by calling the **q.get()** method, where **q** is the shared queue.
4. **PCFixed Class**:
   * This is the main class that sets up the producer-consumer scenario.
   * It creates an instance of the **Q** class, which represents the shared data queue.
   * It then creates one producer and one consumer thread, passing the shared **Q** instance to both.
   * The program will keep running indefinitely until you manually stop it.
5. **Thread Synchronization**:
   * The **synchronized** keyword is used to synchronize the methods of the **Q** class, ensuring that only one thread can execute them at a time.
   * The **wait()** and **notify()** methods are used to handle synchronization between the producer and consumer. The producer waits if data is not consumed, and the consumer waits if data is not produced.