



Synchronization in Java

Last Updated : 04 Jan, 2025

In [Multithreading](#), **Synchronization** is crucial for ensuring that multiple threads operate safely on shared resources. Without **Synchronization**, data inconsistency or corruption can occur when multiple threads try to access and modify shared variables simultaneously. In Java, it is a mechanism that ensures that only one thread can access a resource at any given time. This process helps prevent issues such as data inconsistency and [race conditions](#) when multiple threads interact with shared resources.

Example: Below is the Java Program to demonstrate synchronization.

```
// Java Program to demonstrate synchronization in Java
class Counter {
    private int c = 0; // Shared variable

    // Synchronized method to increment counter
    public synchronized void inc() {
        c++;
    }

    // Synchronized method to get counter value
    public synchronized int get() {
        return c;
    }
}

public class Geeks {
    public static void main(String[] args) {
        Counter cnt = new Counter(); // Shared resource

        // Thread 1 to increment counter
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                cnt.inc();
            }
        });

        // Thread 2 to increment counter
        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                cnt.inc();
            }
        });
    }
}
```



```
// Wait for threads to finish
try {
    t1.join();
    t2.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}

// Print final counter value
System.out.println("Counter: " + cnt.get());
}
```

Output

Counter: 2000

Explanation: Two threads, **t1** and **t2**, increment the shared counter variable concurrently. The **inc()** and **get()** methods are synchronized, meaning only one thread can execute these methods at a time, preventing race conditions. The program ensures that the final value of the counter is consistent and correctly updated by both threads.

Synchronized Blocks in Java

Java provides a way to create threads and synchronise their tasks using **synchronized blocks**.

A **synchronized block** in Java is synchronized on some object. Synchronized blocks in Java are marked with the **synchronized** keyword. All synchronized blocks synchronize on the same object and can only have one thread executed inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block. If you want to master concurrency and understand how to avoid common pitfalls,

General Form of Synchronized Block

```
synchronized(sync_object)
{
    // Access shared variables and other
    // shared resources
}
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

This synchronization is implemented in [Java](#) with a concept called [monitors](#) or [locks](#). Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.

Example: Below is an example of synchronization using Synchronized Blocks

```
// Java Program to demonstrate synchronization block in Java

class Counter {
    private int c = 0; // Shared variable

    // Method with synchronization block
    public void inc() {
        synchronized(this) { // Synchronize only this block
            c++;
        }
    }

    // Method to get counter value
    public int get() {
        return c;
    }
}

public class Geeks {
    public static void main(String[] args) {
        Counter cnt = new Counter(); // Shared resource

        // Thread 1 to increment counter
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                cnt.inc();
            }
        });

        // Thread 2 to increment counter
        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                cnt.inc();
            }
        });

        // Start both threads
        t1.start();
        t2.start();

        // Wait for threads to finish
        try {
            t1.join();
            t2.join();
        } catch (InterruptedException e) {
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

```
        System.out.println("Counter: " + cnt.get());  
    }  
}
```

Output

Counter: 2000

Types of Synchronization

There are two synchronizations in Java mentioned below:

1. Process Synchronization
2. Thread Synchronization

1. Process Synchronization in Java

Process Synchronization is a technique used to coordinate the execution of multiple processes. It ensures that the shared resources are safe and in order.

Example: Here is a popular example of Process Synchronization in Java

```
// Java Program to demonstrate Process Synchronization  
class BankAccount {  
    private int balance  
        = 1000; // Shared resource (bank balance)  
  
    // Synchronized method for deposit operation  
    public synchronized void deposit(int amount)  
    {  
        balance += amount;  
        System.out.println("Deposited: " + amount  
            + ", Balance: " + balance);  
    }  
  
    // Synchronized method for withdrawal operation  
    public synchronized void withdraw(int amount)  
    {  
        if (balance >= amount) {  
            balance -= amount;  
            System.out.println("Withdrawn: " + amount  
                + ", Balance: " + balance);  
        }  
        else {  
            System.out.println(  
                "Insufficient balance to withdraw: "  
            );  
        }  
    }  
}
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

```

    }

    public class Geeks {
        public static void main(String[] args)
        {
            BankAccount account
                = new BankAccount(); // Shared resource

            // Thread 1 to deposit money into the account
            Thread t1 = new Thread(() -> {
                for (int i = 0; i < 3; i++) {
                    account.deposit(200);
                    try {
                        Thread.sleep(50); // Simulate some delay
                    }
                    catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });

            // Thread 2 to withdraw money from the account
            Thread t2 = new Thread(() -> {
                for (int i = 0; i < 3; i++) {
                    account.withdraw(100);
                    try {
                        Thread.sleep(
                            100); // Simulate some delay
                    }
                    catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });

            // Start both threads
            t1.start();
            t2.start();

            // Wait for threads to finish
            try {
                t1.join();
                t2.join();
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }

            // Print final balance
            System.out.println("Final Balance: "
                               + account.getBalance());
        }
    }

```

Output

Deposited: 200 Balance: 1200

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Deposited: 200, Balance: 1400

Withdrawn: 100, Balance: 1300

Final Balance: 1300

Explanation: It demonstrates process synchronization using a bank account with deposit and withdrawal operations. Two threads, one for depositing and one for withdrawing, perform operations on the shared account. The methods **deposit()** and **withdraw()** are synchronized to ensure thread safety, preventing race conditions when both threads access the balance simultaneously. This ensures accurate updates to the account balance.

2. Thread Synchronization in Java

Thread Synchronization is used to coordinate and ordering of the execution of the threads in a multi-threaded program. There are two types of thread synchronization are mentioned below:

- [Mutual Exclusive](#)
- Cooperation ([Inter-thread communication in Java](#))

Example: Java Program to demonstrate thread synchronization for Ticket Booking System

```
// Java Program to demonstrate thread synchronization for Ticket Booking System
class TicketBooking {
    private int availableTickets = 10; // Shared resource (available tickets)

    // Synchronized method for booking tickets
    public synchronized void bookTicket(int tickets) {
        if (availableTickets >= tickets) {
            availableTickets -= tickets;
            System.out.println("Booked " + tickets + " tickets, Remaining tickets: " + availableTickets);
        } else {
            System.out.println("Not enough tickets available to book " + tickets);
        }
    }

    public int getAvailableTickets() {
        return availableTickets;
    }
}

public class Geeks {
    public static void main(String[] args) {
```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

```

        booking.bookTicket(2); // Trying to book 2 tickets each time
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

});

// Thread 2 to book tickets
Thread t2 = new Thread(() -> {
    for (int i = 0; i < 2; i++) {
        booking.bookTicket(3); // Trying to book 3 tickets each time
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
});

// Start both threads
t1.start();
t2.start();

// Wait for threads to finish
try {
    t1.join();
    t2.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}

// Print final remaining tickets
System.out.println("Final Available Tickets: " +
booking.getAvailableTickets());
}
}

```

Output

```

Booked 2 tickets, Remaining tickets: 8
Booked 3 tickets, Remaining tickets: 5
Booked 3 tickets, Remaining tickets: 2
Booked 2 tickets, Remaining tickets: 0
Final Available Tickets: 0

```

Explanation: Here the TicketBooking class contains a **synchronized** method **bookTicket()**, which ensures that only one thread can book tickets at a time, preventing race conditions and overbooking. Each thread attempts to book a set number of tickets in a loop, with thread synchronization ensuring that the

Mutual Exclusive helps keep threads from interfering with one another while sharing data. There are three types of Mutual Exclusive mentioned below:

- Synchronized method.
- Synchronized block.
- Static synchronization.

Example: Below is the implementation of the Java Synchronization

```
// A Java program to demonstrate working of synchronized.
import java.io.*;
// A Class used to send a message
class Sender {
    public void send(String msg)
    {
        System.out.println("Sending " + msg); // Changed to print without new
line
        try {
            Thread.sleep(100);
        }
        catch (Exception e) {
            System.out.println("Thread interrupted.");
        }
        System.out.println(msg + "Sent"); // Improved output format
    }
}

// Class for sending a message using Threads
class ThreadedSend extends Thread {
    private String msg;
    Sender sender;

    // Receives a message object and a string message to be sent
    ThreadedSend(String m, Sender obj)
    {
        msg = m;
        sender = obj;
    }

    public void run()
    {
        // Only one thread can send a message at a time.
        synchronized (sender)
        {
            // Synchronizing the send object
            sender.send(msg);
        }
    }
}

// Driver class
class Geeks {
    public static void main(String args[])
    {

```

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).


```

        S1.start();
        S2.start();

        // Wait for threads to end
        try {
            S1.join();
            S2.join();
        }
        catch (Exception e) {
            System.out.println("Interrupted");
        }
    }
}

```

Output

```

Sending Hi
Hi Sent
Sending Bye
Bye Sent

```

Explanation: In the above example, we choose to synchronize the Sender object inside the run() method of the ThreadedSend class. Alternately, we could define the **whole send() block as synchronized**, producing the same result. Then we don't have to synchronize the Message object inside the run() method in the ThreadedSend class.

We do not always have to synchronize a whole method. Sometimes it is preferable to **synchronize only part of a method**. Java synchronized blocks inside methods make this possible.

Example: Below is the Java program shows the synchronized method using an anonymous class

```

// Java Pogram to synchronized method by
// using an anonymous class
import java.io.*;

class Test {
    synchronized void test_func(int n)
    {
        // synchronized method
        for (int i = 1; i <= 3; i++) {
            System.out.println(n + i);
            try {
                Thread.sleep(100);
            }
            catch (Exception e) {

```



We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

```
// Driver Class
public class Geeks {
    // Main function
    public static void main(String args[])
    {
        // only one object
        final Test t = new Test();

        Thread a = new Thread() {
            public void run() { t.test_func(15); }
        };

        Thread b = new Thread() {
            public void run() { t.test_func(30); }
        };

        a.start();
        b.start();
    }
}
```

Output

16
17
18
31
32
33

Explanation: Here the Test class has a synchronized method test_func() that prints a sequence of numbers with a slight delay, ensuring thread safety when accessed by multiple threads. Two threads are created using anonymous classes, each calling the test_func() method with different values. The synchronized keyword ensures that only one thread can execute the method at a time.

[Comment](#)[More info](#)[Advertise with us](#)

Next Article

Importance of Thread
Synchronization in Java

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Java Tutorial

Java is one of the most popular and widely used programming languages. Used to develop mobile apps, desktop apps, web apps, web servers, games, and enterprise-level systems. Java was invented by James...

4 min read

Java Overview

Java Basics

Java Flow Control

Java Methods


Java Arrays

Java Strings

Java OOPs Concepts

Java Interfaces

Java Collections

 **GeeksforGeeks**
Sanchhaya Education Private Limited
Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate
Tower, Sector- 136, Noida, Uttar Pradesh
(201305)

Registered Address:
K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).



Advertise with us

Company

About Us
Legal
Privacy Policy
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

DSA

Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Web Technologies

HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Computer Science

Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

Languages

Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Python Tutorial

Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

DevOps

Git
Linux
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

Interview Preparation

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).

Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects

@GeeksforGeeks, Sanchhaya Education Private Limited, All rights reserved

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our [Cookie Policy](#) & [Privacy Policy](#).