# SNS COLLEGE OF TECHNOLOGY

**Kurumbapalayam (Po), Coimbatore – 641 107**

**AN AUTONOMOUS INSTITUTION**

**Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai**

**DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY**

## 23CSB201-OBJECT ORIENTED PROGRAMMING

# UNIT IV - EXCEPTION AND MULTITHREADING

## TOPIC 8 - Thread synchronization,

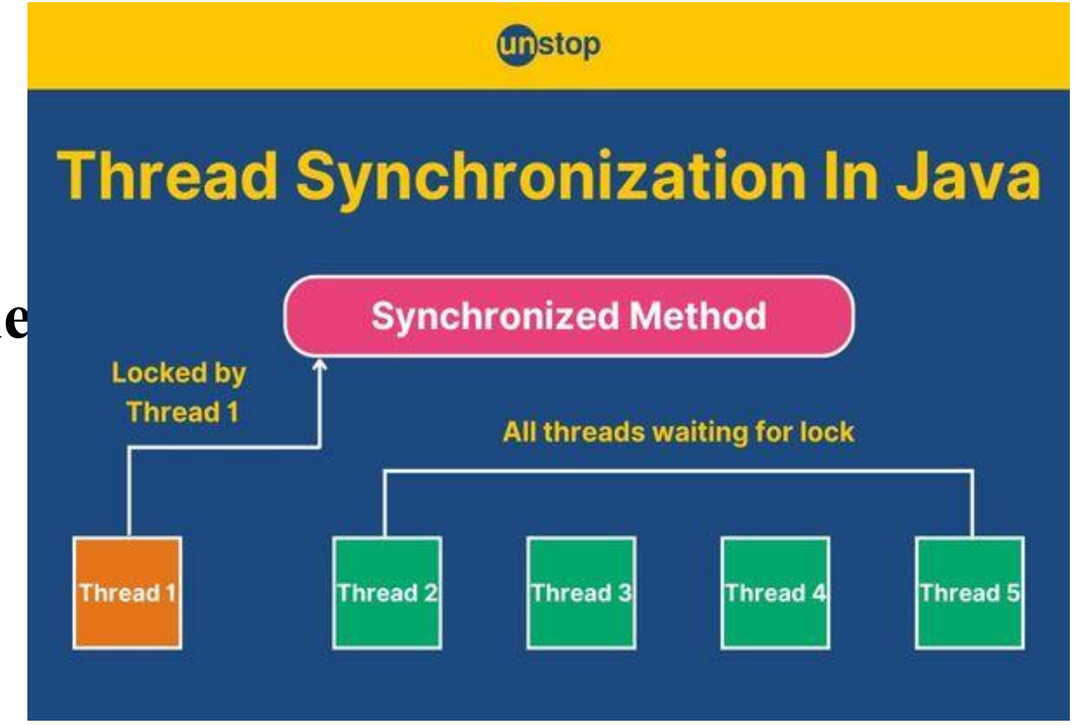
# 1. Thread Synchronization

## ✓ What is Synchronization?

**Thread Synchronization** is a mechanism used to **control access to shared resources** by multiple threads.

It ensures that:

- Only **one thread accesses critical data at a time**
- Prevents **data inconsistency**



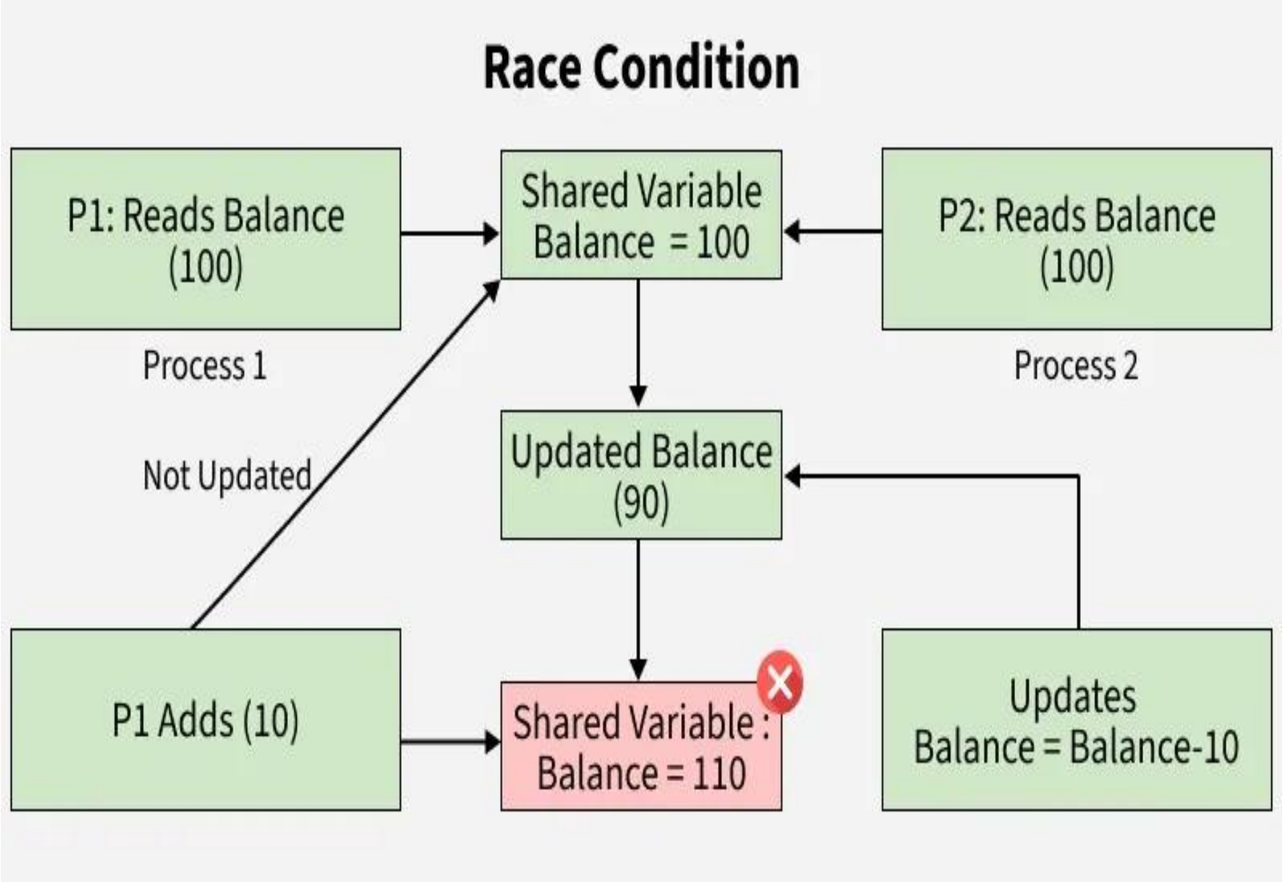
**Problem Without Synchronization (Race Condition)**

When multiple threads access the same data:

```
class Counter {
    int count = 0;
    void increment() {
        count++; // not safe
    }
}
```

If two threads run increment() at the same time:

- Output may be incorrect
- This problem is called **Race Condition**

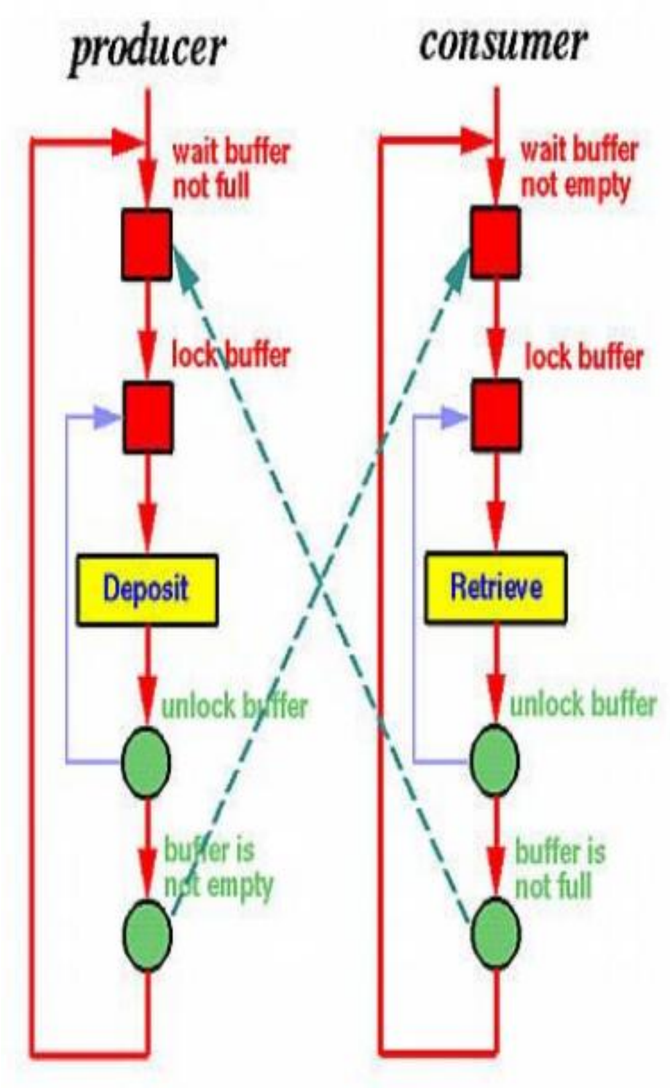



# Solution: Synchronization

## ✓ 1. Synchronized Method

```
class Counter {
    int count = 0;
    synchronized void increment() {
        count++;
    }
}
```

- Only one thread can execute this method at a time



## ✅ 2. Synchronized Block

```
class Counter {
    int count = 0;
    void increment() {
        synchronized(this) {
            count++;
        }
    }
}
```

- Used when only part of code needs synchronization

## Example Program

```
class Test {
    int count = 0;
    synchronized void increment() {
        count++;
    }
}
class MyThread extends Thread {
    Test t;
  MyThread(Test t) {
        this.t = t;
    }

    public void run() {
        for(int i=0; i<1000; i++) {
            t.increment();
        }
    }
}
public class Main {
public static void main(String[] args) throws Exception {
        Test obj = new Test();

        MyThread t1 = new MyThread(obj);
        MyThread t2 = new MyThread(obj);

        t1.start();
        t2.start();

        t1.join();
        t2.join();

        System.out.println(obj.count);
    }
}
```

Output will be correct due to synchronization

## Advantages of Synchronization

✓ Prevents data inconsistency

✓ Avoids race conditions

✓ Ensures thread safety

## Disadvantages

✗ Slows down performance

✗ Can cause deadlock

✗ Complex to manage

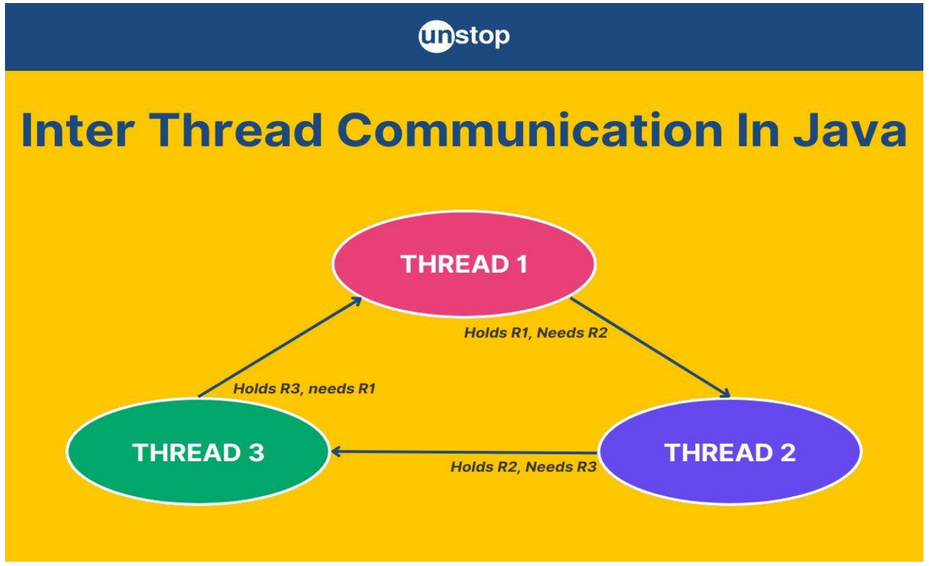
## 2. Inter-Thread Communication

### ✓ What is Inter-thread Communication?

It is a mechanism where **threads communicate with each other** to coordinate execution.

Used when:

- One thread depends on another thread's result



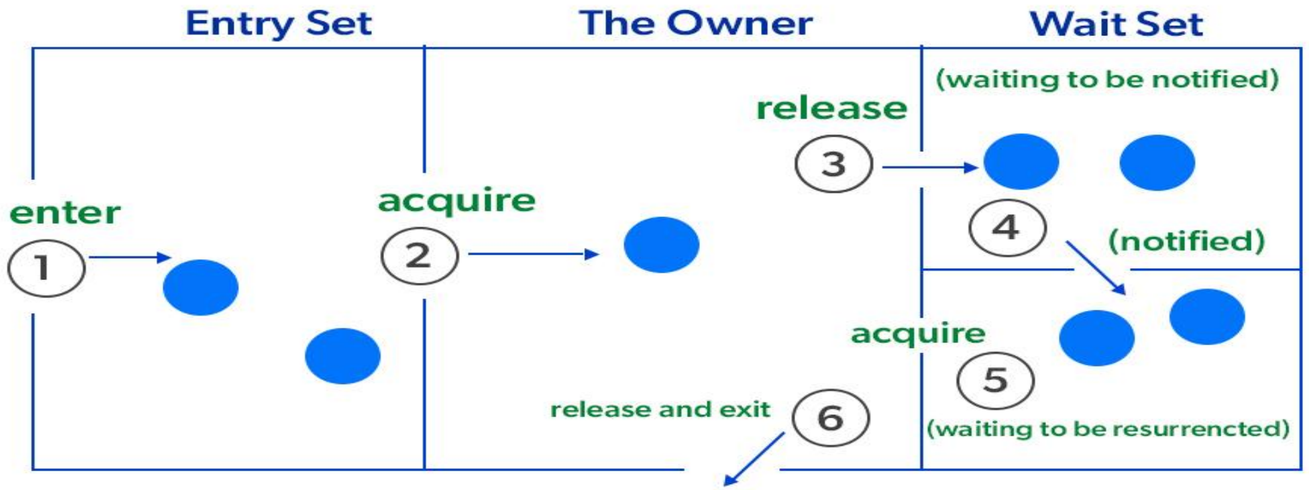
# Methods Used

All belong to Object class:

| Method | Description |
|---|---|
| wait() | Thread waits |
| notify() | Wakes one thread |
| notifyAll() | Wakes all threads |

**How It Works**

- wait() → releases lock and waits
- notify() → wakes one waiting thread
- notifyAll() → wakes all waiting threads



## Example: Producer-Consumer Problem

```
class Shared {
    int data;
    boolean hasData = false;
    synchronized void produce(int value) throws Exception {
        while(hasData) {
            wait();
        }
        data = value;
        hasData = true;
        System.out.println("Produced: " + data);
        notify();
    }
    synchronized void consume() throws Exception {
        while(!hasData) {
            wait();
        }
        System.out.println("Consumed: " + data);
        hasData = false;
        notify();
    }
}
class Producer extends Thread {
    Shared s;
    Producer(Shared s) {
        this.s = s;
    }
  public void run() {
        try {
            for(int i=1; i<=5; i++) {
                s.produce(i);
            }
        } catch(Exception e) {}
    }
}

class Consumer extends Thread {
    Shared s;

    Consumer(Shared s) {
        this.s = s;
    }

    public void run() {
        try {
            for(int i=1; i<=5; i++) {
```

```
        } catch(Exception e) {}
    }
}
public class Main {
    public static void main(String[] args) {
        Shared s = new Shared();
        Producer p = new Producer(s);
        Consumer c = new Consumer(s);
        p.start();
        c.start();
    }
}
```

**Output (Example)**

Produced: 1

Consumed: 1

Produced: 2

Consumed: 2

...

**Important Rules**

✓ wait(), notify(), notifyAll() must be used inside **synchronized block/method**

✓ Always use **loop (while)** instead of if for safety

✓ Avoid deadlock situations

**3. Synchronization vs Inter-thread Communication**

| Feature | Synchronization | Inter-thread Communication |
|---|---|---|
| Purpose | Control access | Communication |
| Mechanism | synchronized keyword | wait(), notify() |
| Focus | Data safety | Coordination |

## 4. Real-Life Example

- Synchronization → ATM transactions (only one user at a time)
- Inter-thread → Producer (chef) & Consumer (customer)

## 5. Key Exam Points

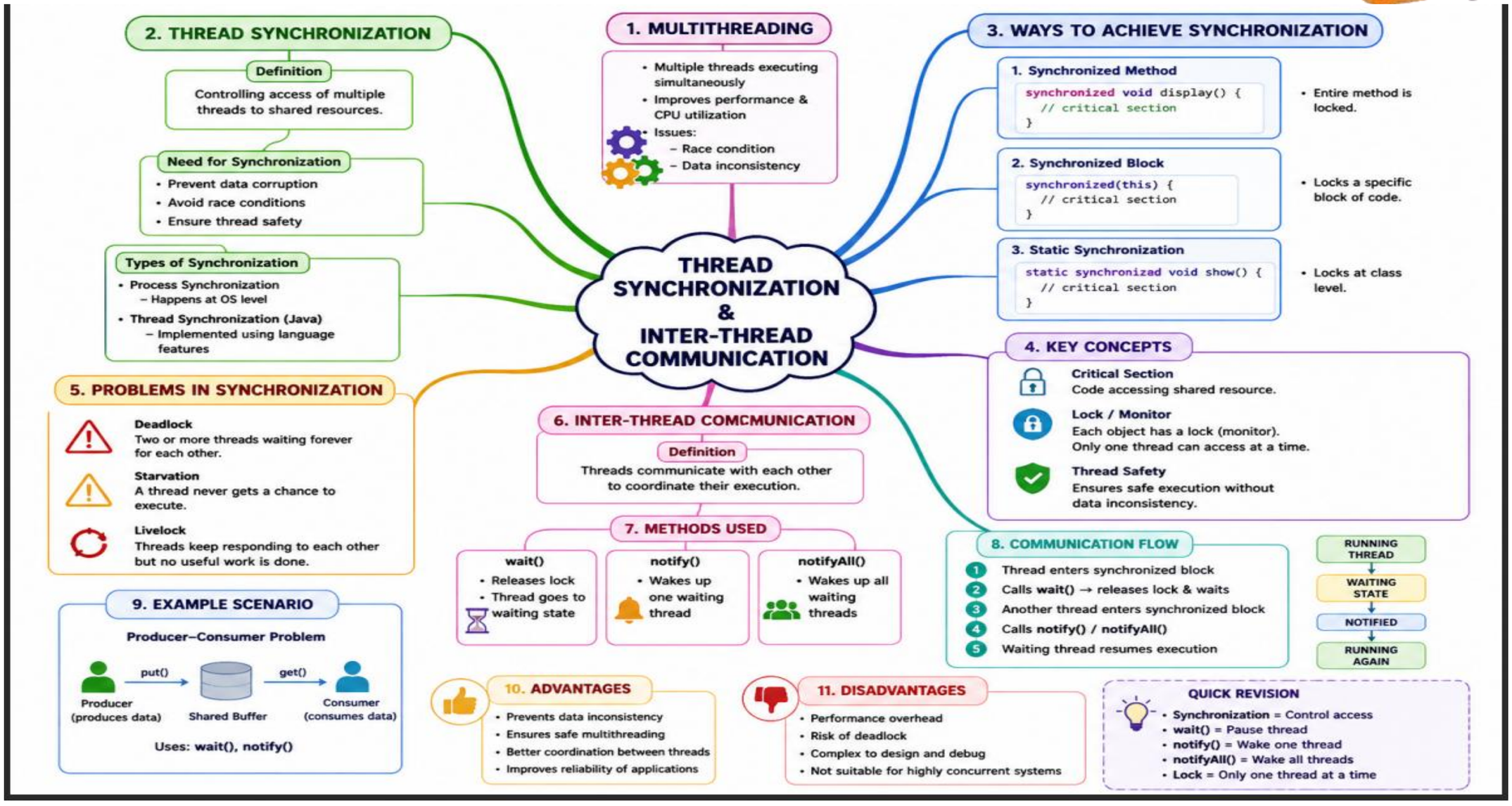
- Synchronization avoids **race condition**
- Use synchronized keyword
- Inter-thread uses wait(), notify()
- Must be used inside synchronized context

## 6. Summary

👉Synchronization = safe data access

👉Inter-thread = communication between threads

👉Both are essential for multithreading


THREAD SYNCHRONIZATION & INTER-THREAD COMMUNICATION
1. MULTITHREADING
Multiple threads executing simultaneously
Improves performance & CPU utilization
Issues:
– Race condition
– Data inconsistency
2. THREAD SYNCHRONIZATION
Definition
Controlling access of multiple threads to shared resources.
Need for Synchronization
Prevent data corruption
Avoid race conditions
Ensure thread safety
Types of Synchronization
Process Synchronization
– Happens at OS level
Thread Synchronization (Java)
– Implemented using language features
3. WAYS TO ACHIEVE SYNCHRONIZATION
1. Synchronized Method
synchronized void display() {
// critical section
}
Entire method is locked.
2. Synchronized Block
synchronized(this) {
// critical section
}
Locks a specific block of code.
3. Static Synchronization
static synchronizad void show() {
// critical section
}
Locks at class level.
4. KEY CONCEPTS
Critical Section
Code accessing shared resource.
Lock / Monitor
Each object has a lock (monitor).
Only one thread can access at a time.
Thread Safety
Ensures safe execution without data inconsistency.
5. PROBLEMS IN SYNCHRONIZATION
Deadlock
Two or more threads waiting forever for each other.
Starvation
A thread never gets a chance to execute.
Livelock
Threads keep responding to each other but no useful work is done.
6. INTER-THREAD COMCMUNICATION
Definition
Threads communicate with each other to coordinate their execution.
7. METHODS USED
wait()
Releases lock
Thread goes to waiting state
notify()
Wakes up one waiting thread
notifyAll()
Wakes up all waiting threads
8. COMMUNICATION FLOW
1 Thread enters synchronized block
2 Calls wait() → releases lock & waits
3 Another thread enters synchronized block
4 Calls notify() / notifyAll()
5 Waiting thread resumes execution
RUNNING THREAD
WAITING STATE
NOTIFIED
RUNNING AGAIN
9. EXAMPLE SCENARIO
Producer–Consumer Problem
put()
get()
Producer (produces data)
Shared Buffer
Consumer (consumes data)
Uses: wait(), notify()
10. ADVANTAGES
Prevents data inconsistency
Ensures safe multithreading
Better coordination between threads
Improves reliability of applications
11. DISADVANTAGES
Performance overhead
Risk of deadlock
Complex to design and debug
Not suitable for highly concurrent systems
QUICK REVISION
Synchronization = Control access
wait() = Pause thread
notify() = Wake one thread
notifyAll() = Wake all threads
Lock = Only one thread at a time

# Assessment

1. What is multitasking?
2. How can you identify the process?
3. How do you see a thread?
4. What is Multithreading and How it is Different from Multitasking?

# THANK YOU