Department of Artificial Intelligence and Machine Learning

Acharya Institute of Technology

Soladevanahalli,Bengaluru 560 107

Flipped class Presentation on

INTER-THREAD COMMUNICATION

Presented by

Group No	Name	USN
8	Namitha R	1AY24AI074
	TP Aswathi	1AY24AI109

Guide Name: Mr. Mohammed Tahir Mirji
Designation: Assistant Professor

ACHARYA

Dept of AI & ML

BACKGROUND OF TOPIC

ACHARYA

Dept of AI & ML

Background of the topic

In Java, multiple threads share the same memory space, which can cause synchronization and coordination issues. While synchronization controls access to shared resources, it does not manage execution order based on conditions. Inter-thread communication provides condition-based coordination using wait(), notify(), and notifyAll() methods. This mechanism prevents busy waiting and ensures efficient thread interaction.

Why this topic is important ?

Without communication:

- Threads may access shared resources incorrectly
- Causes race conditions
- Leads to data inconsistency

With communication:

- One thread waits
- Another thread notifies
- Execution becomes safe and controlled

Dept of AI & ML

wait( ) :

- wait() is a method in Java that causes the current thread to pause execution and release the object lock until another thread notifies it.

- In other words, when a thread invokes wait(), it temporarily stops running, gives up control of the shared object, and stays inactive until another thread signals it to continue execution.

A Simple Example:

```
class Restaurant {
   synchronized void orderFood() throws InterruptedException {
          System.out.println("Customer is waiting for food");
          wait();
          System.out.println("Customer got food");
   }

   synchronized void prepareFood() {
         System.out.println("Chef prepared food");
         notify();
    }
}
```

Dept of AI & ML

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

        Restaurant restaurant = new Restaurant();

        Thread customer = new Thread(() -> {
            try {
                restaurant.orderFood();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        Thread chef = new Thread(() -> {
            restaurant.prepareFood();
        });

        customer.start();
        chef.start();
    }
}
```

Output :

Customer is waiting for food
Chef prepared food
Customer got food