JAVA Programming

Course Instructor: **Dr. N Nandini Devi**

School of Computer Science

UPES Dehradun

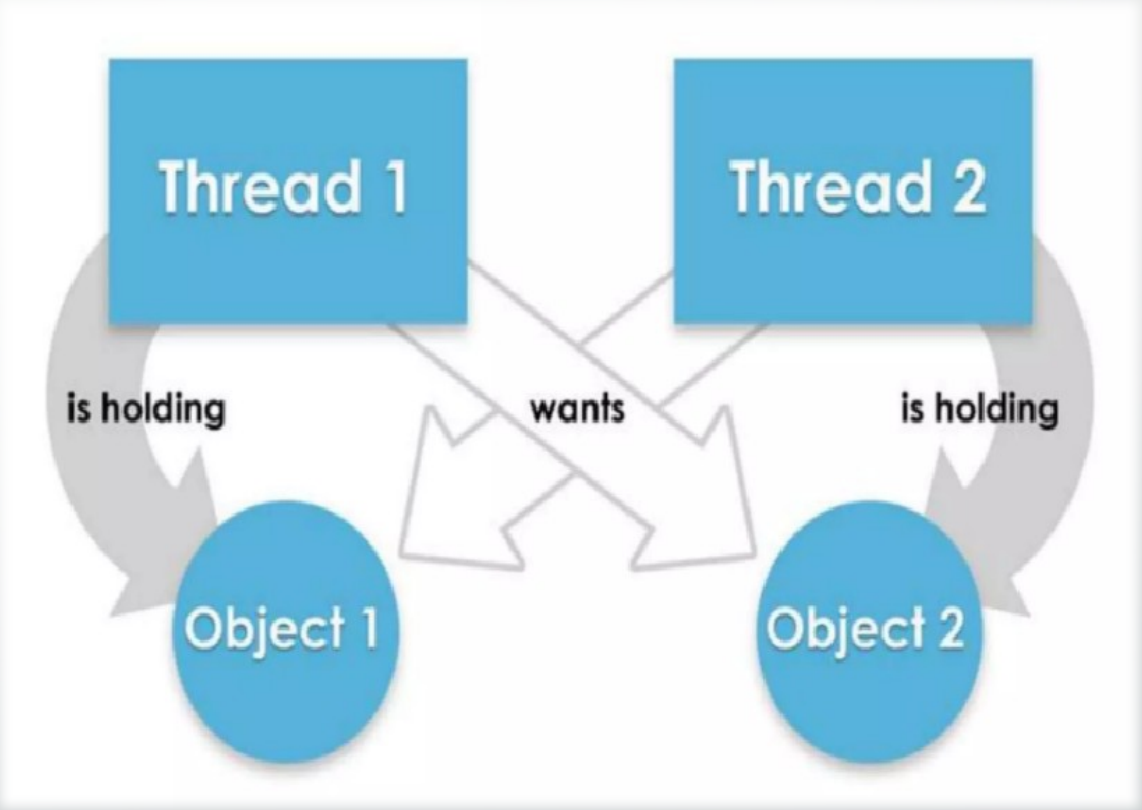
TOPICs to be discussed

- Deadlock Scenario
- Inter-Thread Communication
  - wait()
  - notify()
  - notifyAll()
- Producer-Consumer Scenario

Let’s START ...!!!

Deadlock Scenario

- A deadlock occurs in **Java** (and other programming languages) when two or more threads are waiting for each other to release locks on resources, causing both threads to be stuck indefinitely.



Deadlock (Example)

```
class BankAccount {
   private final String accountName;   private float balance;
   public BankAccount(String name){   accountName = name; balance = 0;   }
   public synchronized void deposit(float amount){
           balance += amount;
           System.out.println(accountName + ": Deposited " + amount + ". New Balance: " +
balance);
   }
   public synchronized void withdraw(float amount){
      balance -= amount;
      System.out.println(accountName + ": Withdrew " + amount + ". New Balance: " + balance);
   }
   public synchronized void transfer(float amount, BankAccount target){
      System.out.println("Lock acquired on " + accountName + " by " +

                  Thread.currentThread().getName());
      withdraw(amount);
      synchronized(target){
         System.out.println("Lock acquired on " + target.accountName + " by " +

                           Thread.currentThread().getName());
         target.deposit(amount);
         System.out.println("Transferred " + amount + " from " + accountName + " to " +

                                                      target.accountName);
```

Deadlock (Example)

```
class MoneyTransfer implements Runnable {
   private BankAccount source, target;
   private float amount;
   public MoneyTransfer(BankAccount source, BankAccount target, float amount){
      this.source = source;  this.target = target;  this.amount = amount;
   }
   public void run(){
      source.transfer(amount, target);
   }
}
class MainClass{
   public static void main(String[] args) {
       BankAccount Account1 = new BankAccount("AliceAccount");
       BankAccount Account2 = new BankAccount("BobAccount");

       Runnable transaction1 = new MoneyTransfer(Account1, Account2, 1200);
       Thread t1 = new Thread(transaction1);
       t1.start();

       Runnable transaction2 = new MoneyTransfer(Account2, Account1, 700);
       Thread t2 = new Thread(transaction2);
       t2.start();
   }
}
```

Deadlock (Example)

- **What will be the output of the given program?**

Output:

Lock acquired on AliceAccount by Thread-0

Lock acquired on BobAccount by Thread-1

AliceAccount: Withdrew 1200.0. New Balance: -1200.0

BobAccount: Withdrew 700.0. New Balance: -700.0

Program halts at this point…

Inter-Thread Communication

- In a **multi-threaded environment**, threads often share resources, which can lead to situations where threads need to coordinate with each other.
- **Inter-thread communication** in **Java** allows threads to wait for certain conditions to be met before proceeding, ensuring safe and efficient access to shared resources.

- **Java** provides three methods that allow threads to communicate and coordinate.
  - **wait()**
  - **notify()**
  - **notifyAll()**
- These methods are called on objects, not directly on threads, and they must be used within a **synchronized block** or a **synchronized method**.

wait()

- The **wait()** method is called on an object when a thread wants to pause execution until some specific condition is met.
- When a thread calls **wait()** on an object, it releases the lock on that object and enters the **waiting/blocked** state.
- It remains in this waiting state until either:
  - Another thread calls **notify()** or **notifyAll()** on the same object, or
  - The thread is interrupted.
- After being notified, the thread will re-acquire the lock on the object and resume execution.

Syntax:

```
public final void wait()
        throws
InterruptedException
```

Example:

```
synchronized(obj){
        obj.wait();
}
```

notify()

- The **notify()** method is called on an object to wake up a single thread that is waiting on that object's monitor (i.e., a thread that called **wait()** on the same object).
- If multiple threads are waiting, only one thread is chosen to wake up (typically chosen by the **thread scheduler**).
- The thread that gets notified will still need to acquire the lock before it can resume execution, so it may not immediately start running if another thread holds the lock.

Syntax:

```
public final void notify()
```

Example:

```
synchronized(obj){
        obj.notify();
}
```

notifyAll()

- The **notifyAll()** method is similar to **notify()**, but instead of waking up a single thread, it wakes up all threads that are waiting on the object's monitor.
- All waiting threads are moved to the **runnable** state, but only one of them will acquire the lock and continue execution. The rest will continue waiting until they can acquire the lock.

Syntax:

```
public final void notifyAll()
```

Example:

```
synchronized(obj){

obj.notifyAll();
}
```

Advantages of wait(), notify() and notifyAll()

- **Efficient Resource Management:**

  These methods prevent busy-waiting by allowing threads to sleep until they are needed, reducing CPU usage.

- **Coordination Between Threads:**

  These methods enable **complex thread coordination patterns**, such as producer-consumer, reader-writer, and more.

- **Avoiding Deadlock:**

  Using **notify()** or **notifyAll()** allows threads to avoid **deadlock** by ensuring that waiting threads are notified when the required condition is met.

Producer-Consumer Scenario

- The **Producer-Consumer scenario** is a classic example of **inter-thread communication** where:
  - A producer thread produces data and puts it into a shared buffer.
  - A consumer thread takes data from the buffer and processes it.
- The **producer** should wait if the buffer is full, and the **consumer** should wait if the buffer is empty.

Producer-Consumer Scenario (Example)

```
class SharedBuffer {
   private int data;    private boolean hasData = false;

   // Method for the producer to produce data
   public synchronized void produce(int value) throws InterruptedException{
        while(hasData){  wait(); }                    //If data is available, wait
        data = value;  hasData = true;
        System.out.println("Produced: " + data + " unit");
        notify();                                                    //Notify the consumer
that data is available
   }

   //Method for the consumer to consume data
   public synchronized int consume() throws InterruptedException{
        while(!hasData){  wait();  }                  //If no data is available, wait
        hasData = false;
        System.out.println("Consumed: " + data + " unit");
        notify();                                                    //Notify the producer
that space is available
        return data;
   }
```

Producer-Consumer Scenario (Example)

```
class ProducerConsumerExample {
   public static void main(String[] args) {
      SharedBuffer buffer = new SharedBuffer();
      //Producer thread
      Thread producer = new Thread(new Runnable(){
         public void run(){
               try{  for(int i=1; i<=5; i++){   buffer.produce(i); Thread.sleep(100);  }
                    }catch(InterruptedException e){   Thread.currentThread().interrupt();   }
         }
      });
      //Consumer thread
      Thread consumer = new Thread(new Runnable(){
         public void run(){
               try{  for (int i=1; i<=5; i++){  buffer.consume();  Thread.sleep(200);  }
               }catch(InterruptedException e){  Thread.currentThread().interrupt();    }
                  }
      });
      //Start threads
      producer.start();  consumer.start();
   }
}
```

Producer-Consumer Scenario (Example)

- **What will be the output of the given program?**

Output:

Produced: 1 unit

Consumed: 1 unit

Produced: 2 unit

Consumed: 2 unit

Produced: 3 unit

Consumed: 3 unit

Produced: 4 unit

Consumed: 4 unit

Produced: 5 unit

Consumed: 5 unit

Summary

Today, we learned about

- Deadlock Scenario
- Inter-Thread Communication (wait(), notify(), notifyAll() methods)
- Producer-Consumer Scenario


Thank You!