Advanced Programming

Week 4

Multithreading

- Thread Pools
- Thread Synchronization
- Deadlock

Tilahun Melak(PhD)

October, 2025

Objectives

At the end of this lecture, students will be able to:

- Explain the concept of a Thread Pool
- Describe Thread Synchronization techniques
- Implement Thread Synchronization in practice
- Discuss causes of Thread Deadlock

Thread Pools

- **Java Thread Pool**
  - A thread pool is a collection of worker threads that are kept ready to perform tasks and can be reused multiple times.
  - A fixed number of threads are created in the pool.
  - When a task is submitted, one thread from the pool is assigned to execute it.
  - After completing the task, the thread returns to the pool and becomes available for reuse.
- **Executor Interface**
  - Provides a higher-level mechanism for managing the execution of Runnable tasks.
  - An Executor object creates and manages a thread pool to run these tasks efficiently.

Oracle. (n.d.). Class Executors. In Java Platform, Standard Edition 8 API Specification. Oracle.

Thread Pools Cntd…

- Executor advantages over creating threads manually
  - **Thread reuse:** reuse existing threads to eliminate new thread overhead
  - **Better performance:** saves time because there is no need to create new thread.
- Executor method *execute()* accepts a Runnable as an argument
  - Assigns each Runnable it receives to one of the available threads in the thread pool
  - If none available, creates a new thread or waits for a thread to become available

Oracle. (n.d.). Class Executors. In Java Platform, Standard Edition 8 API Specification. Oracle.

Thread Pools Cntd…

- Interface *java.util.concurrent.ExecutorService*
  - Extends Executor
  - Declares methods for managing the life cycle of an Executor
  - Can be instantiated by using static method *newFixedThreadPool()* of class *java.util.concurrent.Executors*

Oracle. (n.d.). ExecutorService (Java SE 8 & Java SE 11 platform API specifications).

Thread Pools Cntd…

- methods
  - *execute()* returns immediately from each invocation
  - *shutdown()* notifies the ExecutorService to stop accepting new tasks, but continues executing tasks that have already been submitted

    ……

    ExecutorService es = Executors. newFixedThreadPooll();

    es.execute(new Task());

    es.execute(new Task());

    ……

Oracle. (n.d.). ExecutorService (Java SE 8 & Java SE 11 platform API specifications).

Thread Synchronization

- Synchronization in java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- Other threads wait
- A shared resource may be corrupted if it is accessed simultaneously by multiple threads.

Schildt, H. (2023). Java: The Complete Reference (12th ed.). McGraw-Hill Education.

Thread Synchronization Cntd…

- Java provides built-in **monitors** to implement synchronization.
- **Every object has a monitor and a monitor lock.**
- A monitor ensures that its object's lock is **held by only one thread at a time.**
- Monitors can be used to **enforce mutual exclusion**.
- To enforce mutual exclusion:
  - A thread must acquire the lock before performing its operation.
  - Other threads attempting to access the same lock are blocked until the lock is released.

Schildt, H. (2023). Java: The Complete Reference (12th ed.). McGraw-Hill Education.

Thread Synchronization Cntd…

- synchronized statement
  - Enforces mutual exclusion on a block of code

    synchronized (object){

    statements

    } // end synchronized statement
  - *object* is the object whose monitor lock will be acquired (normally *this)*
- A synchronized method is equivalent to a synchronized statement that encloses the entire body of a method

Oracle. (n.d.). The synchronized statement. In The Java™ Tutorials: Concurrency. Oracle.

Thread Synchronization Cntd…

- Why use Synchronization ?
  - The synchronization is mainly used to:
    - Prevent thread interference.
    - Prevent consistency problem.
- Types of Synchronization
  - There are two types of synchronization
    - Process Synchronization
    - Thread Synchronization
- In this lecture, we will focus only on thread synchronization.

Oracle. (n.d.). Synchronization. In Java documentation. Oracle.

Thread Synchronization Cntd…

- Types of Thread Synchronization in Java:
  - **Mutual Exclusion** (to prevent thread interference and data inconsistency)
    - *Synchronized method*
    - *Synchronized block*
  - **Cooperation** (inter-thread communication using wait(), notify(), notifyAll())

Oracle. (n.d.). Concurrency (The Java™ Tutorials > Essential Java Classes > Concurrency). Oracle.

Mutual Exclusive

- Mutual Exclusion in Java
  - **Purpose:** Prevents threads from interfering with each other when accessing shared data.
  - **Two ways to achieve mutual exclusion**:
    - Synchronized method – locks the entire method.
    - Synchronized block – locks only a specific section of code.

Oracle. (n.d.). Concurrency (The Java™ Tutorials > Essential Java Classes > Concurrency). Oracle.

Mutual Exclusive Cntd…

- Concept of Locks in Java
  - Synchronization relies on **locks (monitors).**
  - Every object in Java has an associated lock.
  - A thread must **acquire the object's lock** before accessing shared fields and **release** it when done.
  - Java provides additional lock implementations in the package java.util.concurrent.
  - locks for more advanced control (e.g., ReentrantLock).

Oracle. (n.d.). Concurrency (The Java™ Tutorials > Essential Java Classes > Concurrency). Oracle.

Mutual Exclusive Cntd…

Synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Syntax:

```
synchronized void display (String msg)
```

Oracle. (n.d.). Java tutorials: Synchronized methods. Oracle.

Example: Java Synchronized Method

```
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}
```

Example: Java Synchronized Method Cntd…

```
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
```

Example: Java Synchronized Method Cntd..

```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

Example: Java Synchronized Method Cntd…

```
public class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Mutual Exclusive Cntd…

Synchronized block

- Synchronized block can be used to perform synchronization on any specific resource of the method. Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.
- If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Syntax to use synchronized block:

```
synchronized (object reference expression) {
  //code block
}
```

Oracle. (n.d.). Java tutorials: Synchronized methods. Oracle.

Example: Java Synchronized Block

```
class TableTwo{

 void printTable(int n){
   synchronized(this){//synchronized block
     for(int i=1;i<=5;i++){
      System.out.println(n*i);
      try{
       Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
     }
   }
 }//end of the method
}
```

Example: Java Synchronized Block Cntd…

```
class MyThreadOne extends Thread{
Table t;
MyThreadOne(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
```

Example: Java Synchronized Block Cntd…

```
class MyThreadTwo extends Thread{
Table t;
MyThreadTwo(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

Example: Java Synchronized Block Cntd…

```
public class TestSynchronizedBlock{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Inter-thread communication

- Inter-thread communication allows synchronized threads to communicate with each other.
- It enables one thread to pause execution in its critical section, while another thread is allowed to enter and execute in the same critical section.
- This mechanism is supported by the following methods of the Object class:

    1) wait()
    2) notify()
    3) notifyAll()

Oracle. (n.d.). *Java inter-thread communication and synchronization*. Oracle Java Documentation.

Inter-thread communication Cntd…

1) **wait() method**

- Causes the current thread to release the lock and enter a waiting state.
- The thread remains waiting until:
  - another thread calls notify() or
  - notifyAll(), ora specified timeout has elapsed.
- Must be called from within a synchronized method or synchronized block; otherwise, it throws IllegalMonitorStateException.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Inter-thread Communication Cntd…

2) notify() method

- Wakes up a single thread that is waiting on this object's monitor.

  If many threads are waiting on this object, one of them is chosen to be awakened.

- **Syntax:** public final void notify()

3) notifyAll() method

- Wakes up all threads that are waiting on this object's monitor.
- **Syntax:** public final void notifyAll()

Oracle. (n.d.). *Object (Java Platform SE 8)*

Example: Inter-thread Communication

```
class Customer{
int amount=10000;

synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
```

Example: Inter-thread Communication Cntd…

```
synchronized void deposit(int amount){
System.out.println("going to deposit…");
this.amount+=amount;
System.out.println("deposit completed… ");
notify();
}
}
```

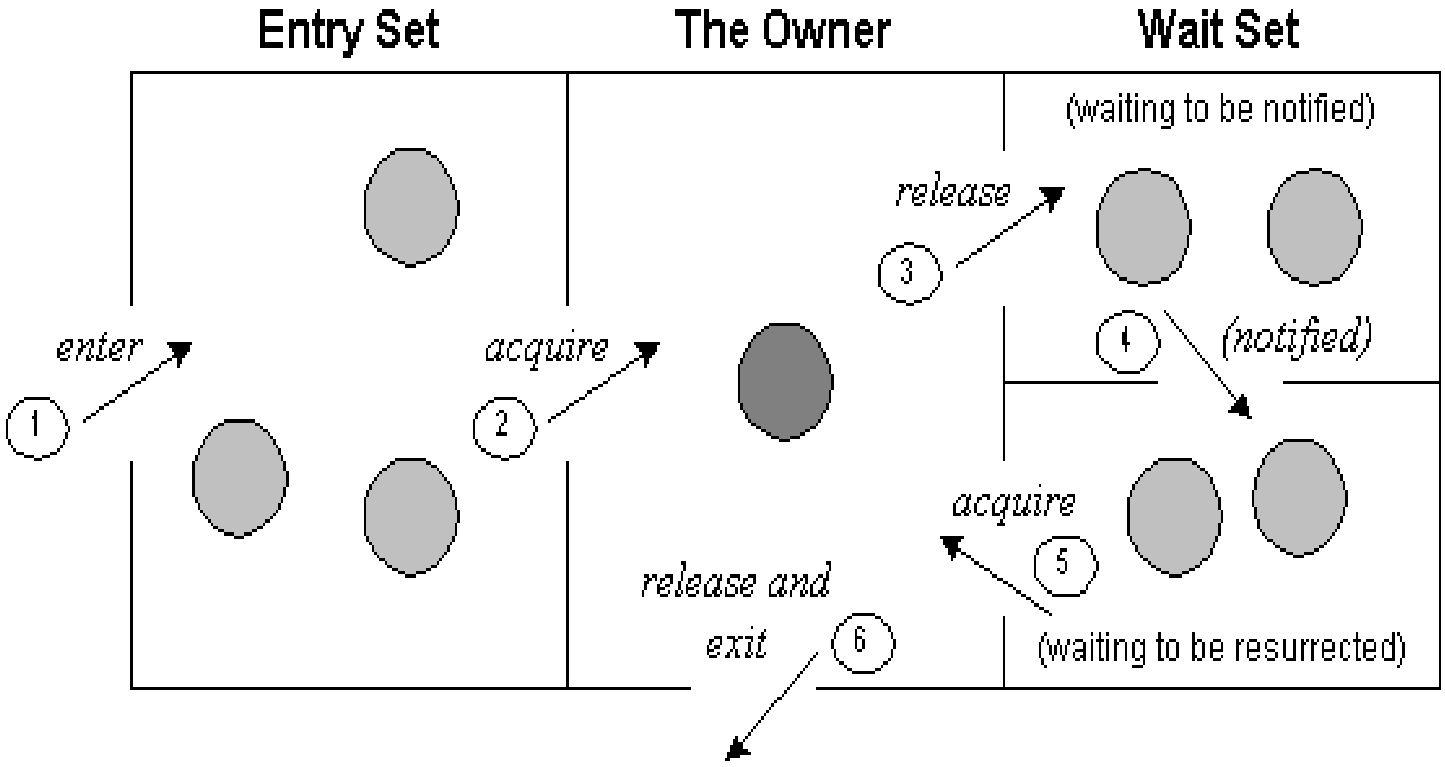
Example: Inter-thread Communication Cntd…

```
class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();


}}
```

Inter-thread communication

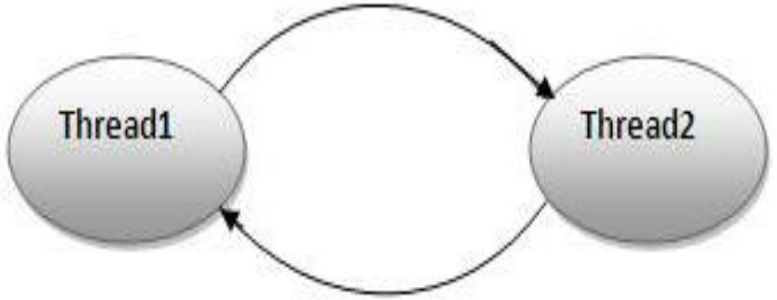
The process of inter-thread communication



Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Deadlock

- Deadlock in java is a part of multithreading.
- Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.
- Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- Multithreaded producer/consumer relationship
  - Producer thread generates data and places it in a shared object called a buffer
  - Consumer thread reads data from the buffer
- Operations on the buffer data shared by a producer and a consumer are state dependent
  - Should proceed only if the buffer is in the correct state
  - If in a not-full state, the producer may produce
  - If in a not-empty state, the consumer may consume
- Must synchronize access to ensure that data is written to the buffer or read from the buffer only if the buffer is in the proper state

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- Can implement a shared using the synchronized keyword and Object methods wait, notify and notifyAll
  - can be used with conditions to make threads wait when they cannot perform their tasks
- A thread that cannot continue with its task until some condition is satisfied can call Object method wait
  - releases the monitor lock on the object
  - thread waits in the waiting state while the other threads try to enter the object's synchronized statement(s) or method(s)

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- A thread that completes or satisfies the condition on which another thread may be waiting can call Object method notify
  - allows a waiting thread to transition to the runnable state
  - the thread that was transitioned can attempt to reacquire the monitor lock
- If a thread calls notifyAll, all the threads waiting for the monitor lock become eligible to reacquire the lock

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- Bounded buffer is used to minimize the amount of waiting time for threads that share resources and operate at the same average speeds
- If the buffer is full, the producer should wait until a consumer consumed a value to free an element in the buffer.
- If the buffer is empty at any given time, a consumer thread must wait until the producer produces another value.
- ArrayBlockingQueue is a bounded buffer that handles all of the synchronization details for you

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- Give programmers more precise control over thread synchronization.
- Any object can contain a reference to an object that implements the java.util.concurrent.locks .Lock interface.
- A thread calls the Lock's lock method to acquire the lock.
- Once a lock has been obtained by one thread, the Lock object will not allow another thread to obtain the Lock until the first thread releases the Lock (by calling the Lock's unlock method).

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- All other threads attempting to obtain that Lock on a locked object are placed in the waiting state
- Class java.util.concurrent.locks.ReentrantLock is a basic implementation of the Lock interface.
- ReentrantLock constructor takes a boolean argument that specifies whether the lock has a fairness policy
- If true, the ReentrantLock's fairness policy is "the longest-waiting thread will acquire the lock when it is available"

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- If false, there is no guarantee as to which waiting thread will acquire the lock.

- A thread that owns a Lock and determines that it cannot continue with its task until some condition is satisfied can wait on a condition object
- Lock objects allow you to explicitly declare the condition objects on which a thread may need to wait
- Condition objects
  - Associated with a specific Lock
  - Created by calling a Lock's new Condition method

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- To wait on a Condition object, call the Condition ’s await method
  - immediately releases the associated Lock and places the thread in the waiting state for that Condition
- Another thread can call Condition method signal to allow a thread in that Condition’s waiting state to return to the runnable state
  - Default implementation of Condition signals the longest-waiting thread

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- Condition method signalAll transitions all the threads waiting for that condition to the runnable state
- When finished with a shared object, thread must call unlock to release the Lock
- Lock and Condition may be preferable to using the synchronized keyword
- Lock objects allow you to interrupt waiting threads or to specify a timeout for waiting to acquire a lock

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Producer/Consumer Relationship without Synchronization

- Lock object is not constrained to be acquired and released in the same block of code
- Condition objects can be used to specify multiple conditions on which threads may wait
- Possible to indicate to waiting threads that a specific condition object is now true

Oracle. (n.d.). *Object (Java Platform SE 8) – wait() method*. Oracle.

Summary

- In today's lecture we have discussed about;
  - Thread Pool
  - Thread Synchronization
  - Deadlock
  - Examples

References

- Oracle. (n.d.). Class Executors. In Java Platform, Standard Edition 8 API Specification. Oracle.
- Oracle. (n.d.). ExecutorService (Java SE 8 & Java SE 11 platform API specifications).
- Schildt, H. (2023). Java: The Complete Reference (12th ed.). McGraw-Hill Education.
- Oracle. (n.d.). *The synchronized statement*. In *The Java™ Tutorials: Concurrency*. Oracle.
- Oracle. (n.d.). Concurrency (The Java™ Tutorials > Essential Java Classes > Concurrency). Oracle.
- Oracle. (n.d.). *Java inter-thread communication and synchronization*. Oracle Java Documentation.
- Oracle. (n.d.). Object (Java Platform SE 8) – wait() method. Oracle.