# Application Program Development

APD545

Instructor: Maryam Sepehrinour
Email: Maryam.Sepehrinour@SenecaPolytechnic.ca

Seneca

## Outcomes

Understanding of Threads.

Understanding of Java Threads.

Understanding of Thread Lifecycle.

Examples with Java Threads and it’s management.

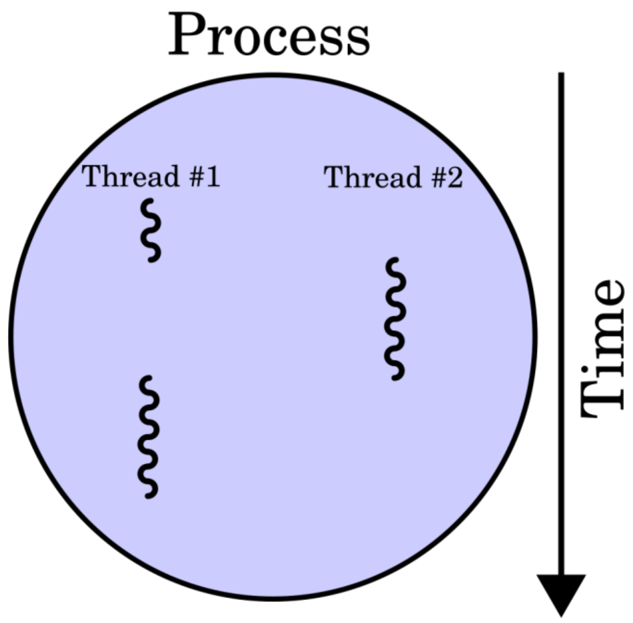
## Motivation

- Single-threaded applications can lead to poor responsiveness where lengthy activities must complete before others can begin.
- Performance is poor as the development times goes up with this.
- Poor utilization of CPU resources.
- Maintenance time cost is higher.
- Poor utilization of cache storage by not utilizing the resources properly.

## Definitions

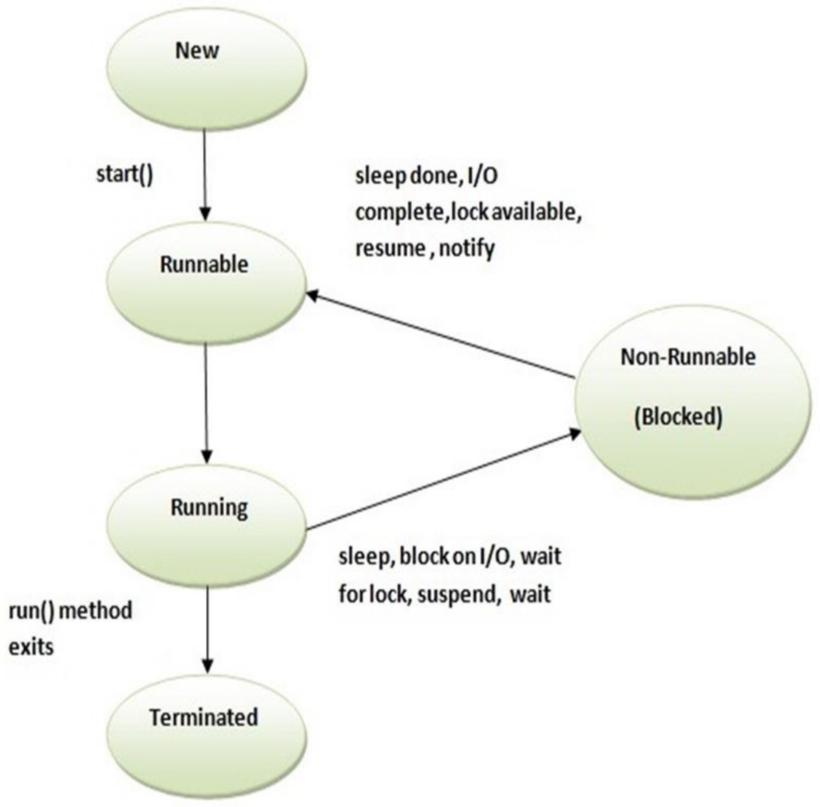
- a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. In many cases, a thread is a component of a process.

- The multiple threads of a given process may be executed concurrently (via multithreading capabilities), sharing resources such as memory, while different processes do not share these resources.



# Threads in Java

- Java provides concurrency available to you via language and API's
- The life cycle of the thread in java is controlled by JVM.
- The java thread states are as follows:
  - New
  - Runnable
  - Running
  - Non-Runnable (Blocked)
  - Terminated


New
start()
Runnable
sleep done, I/O complete, lock available, resume, notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

## Threads in Java

- New
  - The thread is in new state if you create an instance of Thread class but before the
  - invocation of start() method.
- Runnable
  - The thread is in runnable state after invocation of start() method, but the thread
  - scheduler has not selected it to be the running thread.
- Running
  - The thread is in running state if the thread scheduler has selected it.
- Non-Runnable (Blocked)
  - This is the state when the thread is still alive, but is currently not eligible to run.
- Terminated
  - A thread is in terminated or dead state when its run() method exits.

## Defining a Thread

- There are two ways to create a thread:
  - Extend Thread Class:
  - public class MyThread extends Thread {
  - public void run () {
  - }
  - }
  - One must override run() method.
- Create a Runnable Object (Implementing the Interface):
  - public class MyRunnable implements Runnable {
  - public void run() {
  - }
  - }
  - One must implement run() method.

Create an object of your subclass:

```
        Runnable r = new MyRunnable();
```

Construct a Thread object from the runnable object:

```
        Thread t = new Thread(r);
```

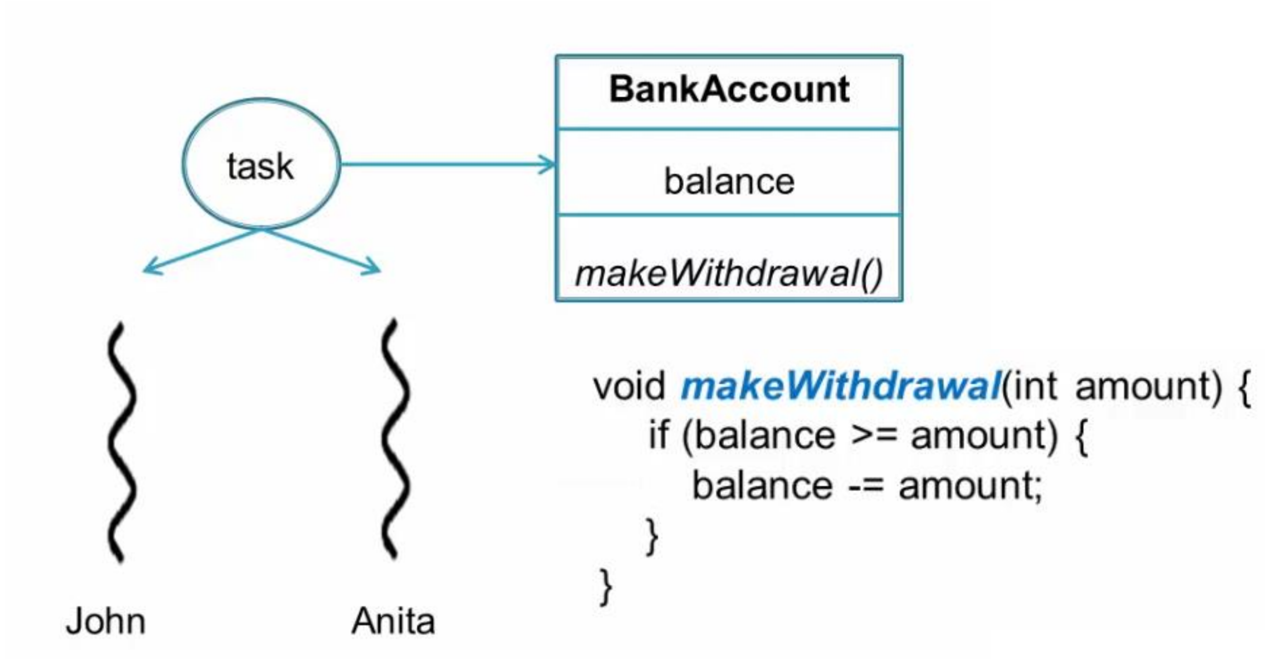
Call the start method to start the thread:

```
        t.start();
```

Seneca

## Thread Synchronization

- A shared resource may be corrupted if it is accessed simultaneously by multiple threads. For example, two unsynchronized threads accessing the same bank account may cause conflict.



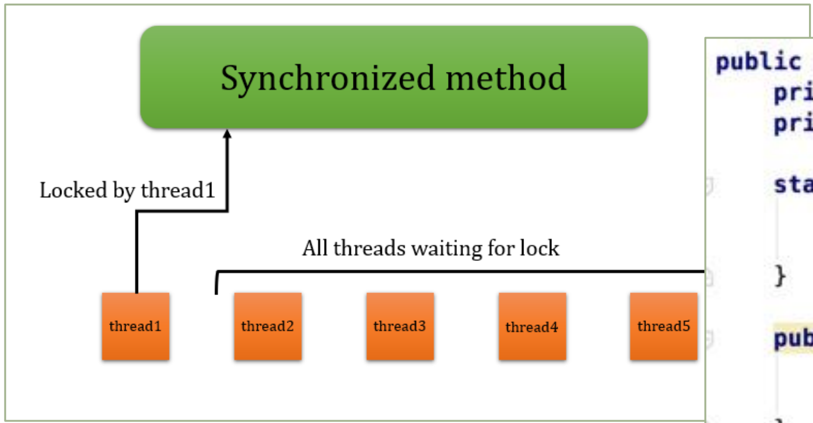
```
void makeWithdrawal(int amount) {
    if (balance >= amount) {
        balance -= amount;
    }
}
```

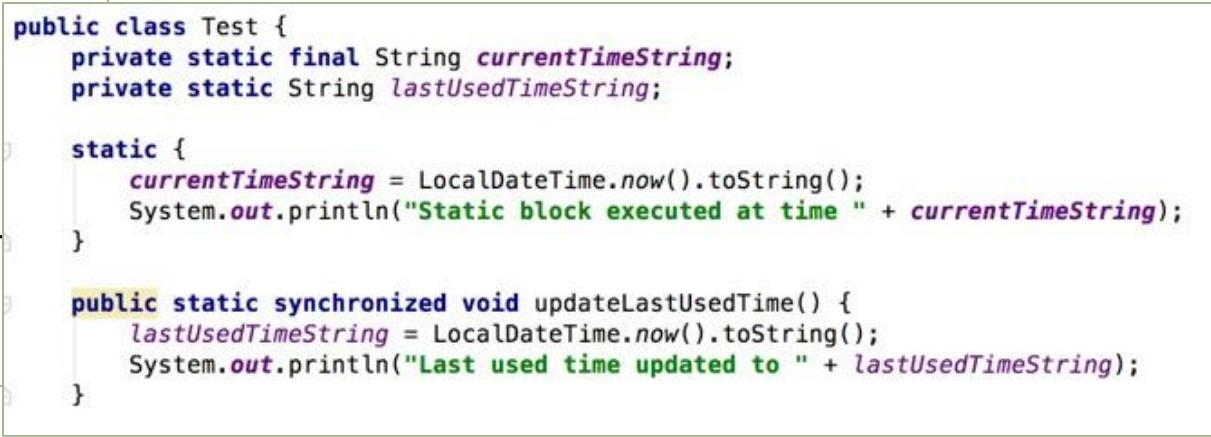
## Synchronization Concept

- Synchronization is built around the concept known as the intrinsic lock
- Every object has an intrinsic lock associated with it
- A thread that needs access to an object's fields has to acquire the object's intrinsic lock
- A thread has to release the intrinsic lock when it's done with an object
- A thread is said to own the intrinsic lock since acquires until releases the object's intrinsic lock
- Any other thread will block when it attempts to acquire the object's  intrinsic lock, if the lock is owned by another thread

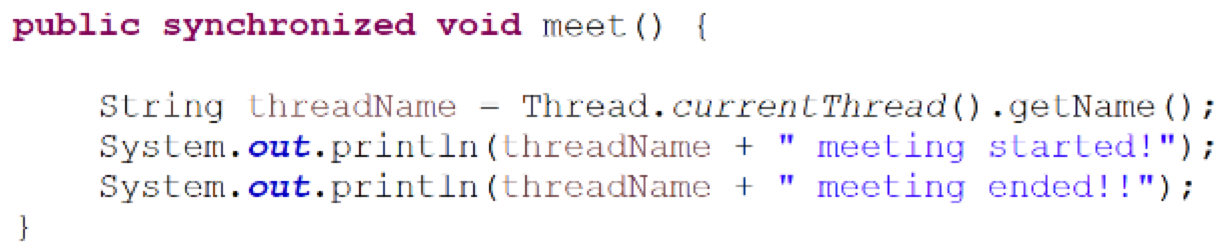
## Types of Synchronization

- There are two types of synchronization
  - Process Synchronization.
  - Thread Synchronization.
    - Mutual Exclusive (keep threads from interfering with one another while sharing data. This can be done by three ways in java)
      - Synchronized method.
      - Synchronized block.
      - static synchronization.
    - Cooperation (Inter-thread communication in java)



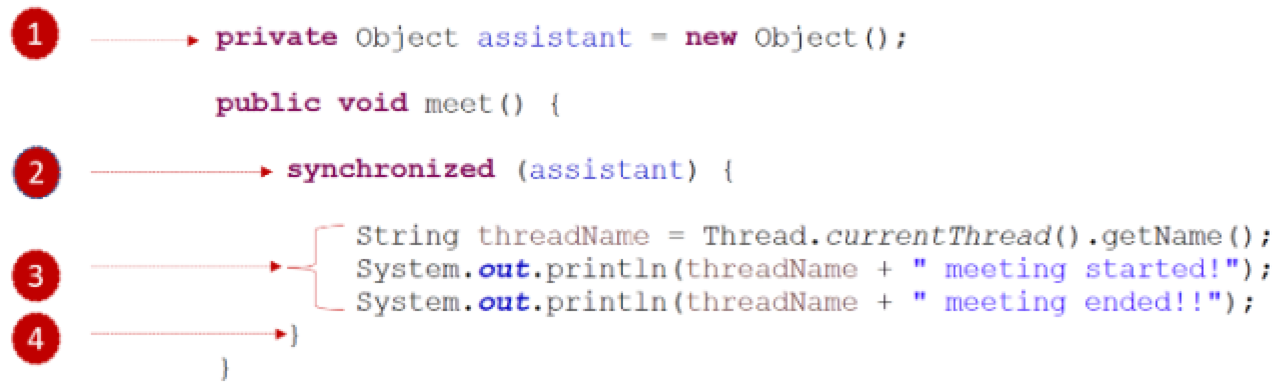


```
public class Test {
    private static final String currentTimeString;
    private static String lastUsedTimeString;

    static {
        currentTimeString = LocalDateTime.now().toString();
        System.out.println("Static block executed at time " + currentTimeString);
    }

    public static synchronized void updateLastUsedTime() {
        lastUsedTimeString = LocalDateTime.now().toString();
        System.out.println("Last used time updated to " + lastUsedTimeString);
    }
```



```
public synchronized void meet() {

    String threadName = Thread.currentThread().getName();
    System.out.println(threadName + " meeting started!");
    System.out.println(threadName + " meeting ended!!");
}
```

*Fig: Synchronized method*

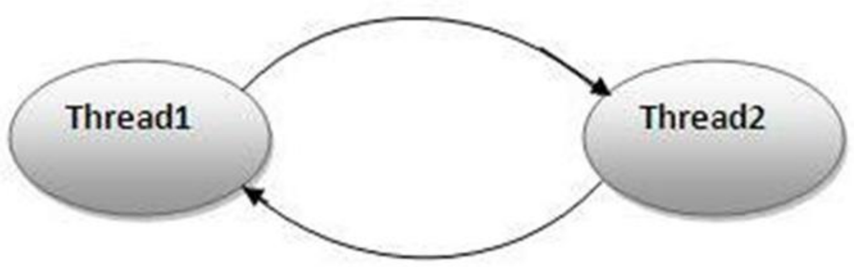


```
private Object assistant = new Object();

public void meet() {

    synchronized (assistant) {

        String threadName = Thread.currentThread().getName();
        System.out.println(threadName + " meeting started!");
        System.out.println(threadName + " meeting ended!!");
    }
}
```

*Fig: Synchronized Block*

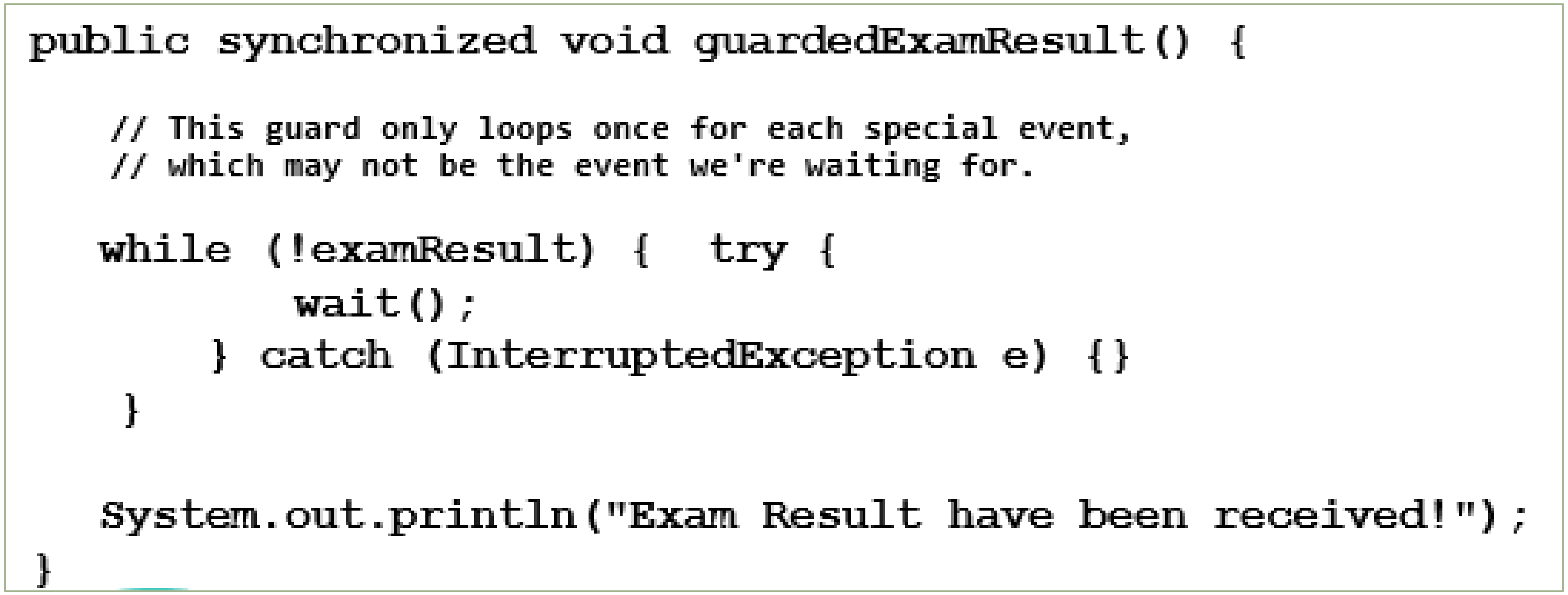
## Deadlock

- The threads t1 and t2 are blocked forever, waiting for each other - this  problem is defined as being a deadlock

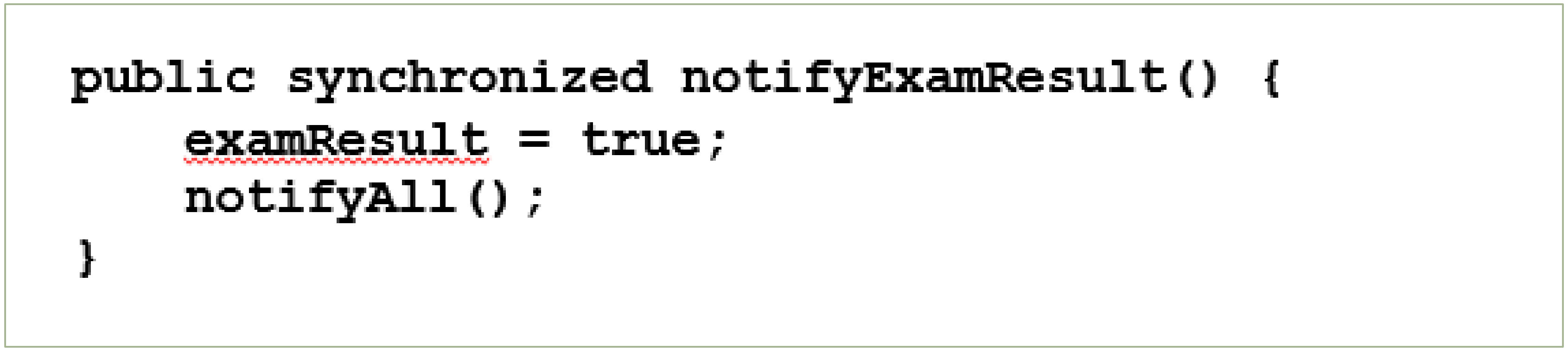


## Inter-Thread Communication

- Threads have to coordinate their actions (they must work together).
- The guarded block is the most common coordination idiom for threads coordination.
- The guarded block uses three methods from Object class:
  - wait()
    - Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
  - notify()
    - Wakes up a single thread
  - notifyAll()
    - Wakes up all threads



```
public synchronized void guardedExamResult() {

    // This guard only loops once for each special event,
    // which may not be the event we're waiting for.

   while (!examResult) {  try {
            wait();
        } catch (InterruptedException e) {}
    }

   System.out.println("Exam Result have been received!");
}
```



```
  public synchronized notifyExamResult() {
      examResult = true;
      notifyAll();
  }
```

Important note:

There is a second notification method, **notify**, which wakes up a single thread. The **notify**method doesn't allow you to specify the thread that is woken up.

## Thank you!

Seneca