ENSF 607

2 - Multithreading (Part I)

What is Concurrency

- Real world systems components work in parallel.
  - Computer users can use their word processor while downloading music.
- The Ada programming language developed by US Department of Defense initially used concurrency in defense command and control systems.
- Both threads and processes are methods of parallelizing an application.

Processes vs. Threads

- Processes:
  - Execution units, that have self-contained environment, and have their own state information, memory spaces.
  - only interact with each other via **Inter Process Communication (IPC)** mechanisms (generally managed by the operating system).
  - Processes are often seen as synonymous with programs.

Processes vs. Threads

- Threads :
  - Threads run within a process
    - A single process may contain multiple threads
  - threads within a process share the same state and same memory space,
  - communicate with each other directly, because they share the same variables.
- *Asynchronous* threads run independently of each other.
- *Synchronous* threads can exchange messages with each other or wait for an action to occur in other thread(s).

Why Use Threads

- Make the UI more responsive:
  - Event-driven UI toolkits such as mouse click, have an event thread that processes UI. If in application, an event listener gets busy with a lengthy task, would not be able to respond to the other events
- Take advantage of multiprocessing
- Simplify modeling:
  - Using threads makes the simulation process simpler and closer to its real-world operation.
- Perform asynchronous or background processing.

Java Threads

Threads in Java

- Every Java program has **at least one thread**; **the main thread.**
- The JVM also creates other threads that are mostly invisible to the programmers:
  - garbage collection thread,
  - object finalization thread,
  - Etc..
- AWT and Swing, servlet, RMI also create threads.

Class Thread

- One way to create threads is to *extend* the Thread class and override the `run()` method.
- Class Thread belongs to `java.lang` package.
- Has several constructors

```
public Thread();
public Thread(String threadName);
Thread (Runnable target);
…
```

Very Simple Example

```
public class SimpleThread extends Thread {

  public void run() {
    System.out.println("A simple thread that does nothing!");
  }

  public static void main(String args[]) {
    Thread t = new SimpleThread();
    t.start();
  }
}
```

A Different Approach, Implementing Runnable

Implementing Runnable

- Another way to create threads in java is to implement the **`Runnable`** interface.
- This method of creating threads is more general
- If you need to inherit from another class such as when using the Applet or Frame classes, you can *implement* a Runnable interface instead, and write the required ***`run()`*** method.

Very Simple Runnable Class

```
public class SimpleThread implements Runnable {

  public void run() {
      System.out.println("A simple thread that does
      nothing!");
  }

   public static void main(String args[]) {
      Runnable r = new SimpleThread();
      Thread t = new Thread(r);
      t.start();
  }
}
```

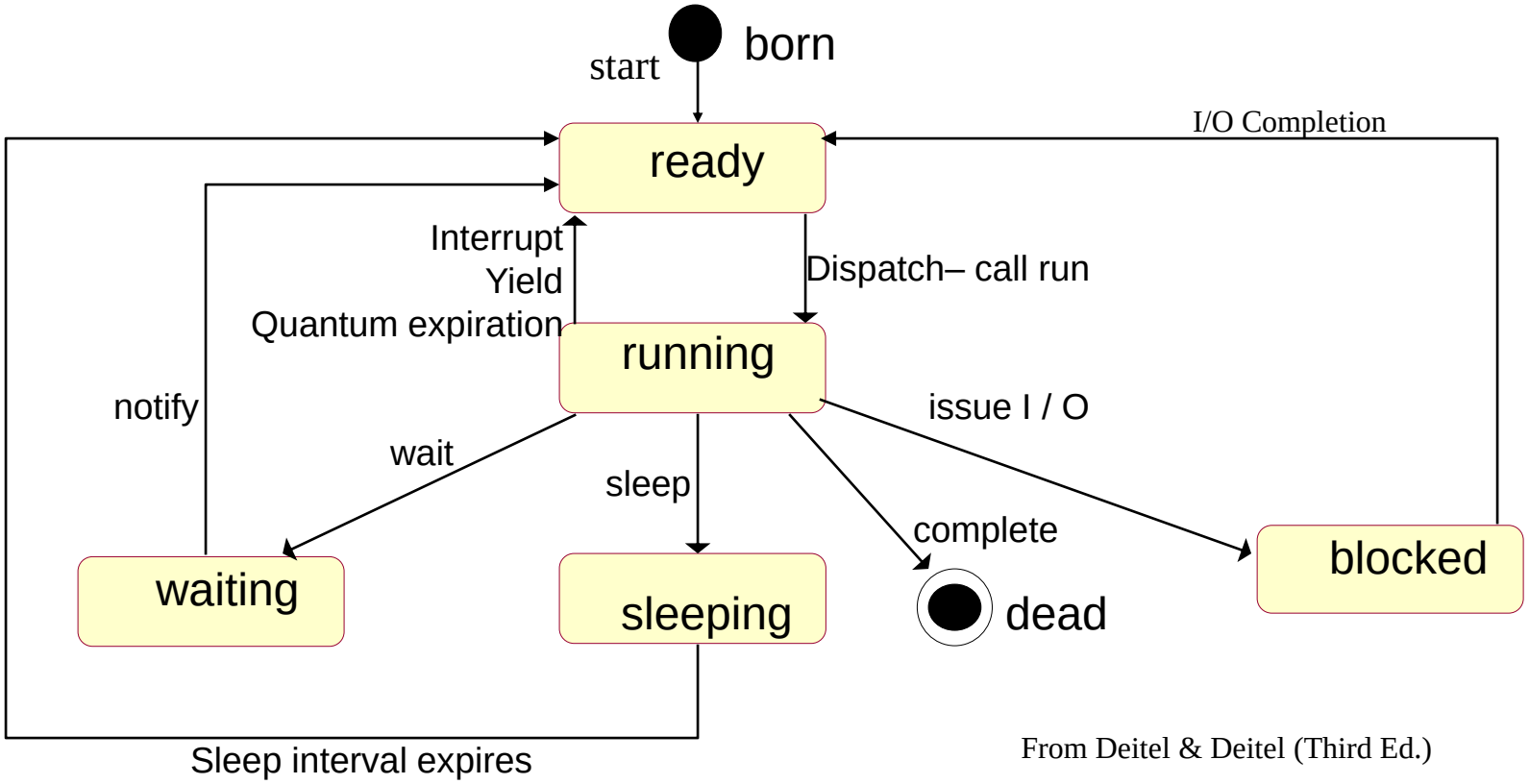
What are Thread States

UML

State Transition Diagram

Thread State

- At anytime a thread is said to be in one of the several thread states.



From Deitel & Deitel (Third Ed.)

Java How to program, Prentice Hall

Class Thread Methods

- start()
  - A program lunches a thread's execution by calling the thread's start method.
- run()
  - Called by the start method
- sleep(milliseconds)
  - Static method that indicates how long the thread should sleep.
    - it may not wait exactly as long as you specify.
    - responds to an interrupt by exiting with an InterruptedException.

Thread Methods Cont'd

- interrupt()
  - Despite its name, this method does NOT interrupt a running thread. It changes the interrupted status of a thread
  - It is an indication to a thread that it should stop what it is doing, and do something else.
    - It is implemented using an internal flag. Invoking `Thread.interrupt` sets this flag.
  - When a thread checks for an interrupt by `Thread.interrupted`, interrupt status gets clear.
  - Also, any method that throws an `InterruptedException`, clears the interrupt status.

Thread Methods Cont'd

- `static boolean interrupted()`
  - returns true if thread is interrupted
  - checks the current thread.
  - clears the status flag of the thread.
  - Never call it on an instance.
- `boolean isInterrupted()`
  - returns true if thread is interrupted
  - is an instance method and can check any Thread object.
  - doesn't clear the status.

Thread Methods

- join()
  - Asks current thread to wait for another thread to complete its job.
  - For example: If thread, t, is currently executing within the `thread main, t.join()` causes the current thread (main) to pause execution until t's thread terminates.
  - it may not wait exactly as long as you specify.
  - Like sleep, responds to an interrupt by exiting with an `InterruptedException.`
- join(long milliseconds)
  - Asks current thread to wait for another thread for an specific period or when the other threads job is completed (whichever comes first)

Other Methods

- Thread object methods are used on instantiated thread objects to control a thread.
- These methods include:
  - `getName()`,
  - `getPriority()`,
  - `boolean isAlive()`,
  - `setName(string)`,
  - `setPriority(int)`,
  - `static Thread currentThread ()`

Lets Look at Some Code

Exercise 1

- Develop another version of `SimpleThread` class with a data field of type long that allows each thread to have its own `sleepTime`, and a `run` method that uses a for loop to iterate 5 times, displaying the thread's name, and its `sleepTime`.
- Develop a second class called Demo that creates three instances of `SimpleTread` in its main method.

Program's Sample Run Output

```
Name: A; sleep: 1000
Name: B; sleep: 1
Name: C; sleep: 1
Three Threads A, B, and C Started:
B    Started.
C    Started.
C    Started.
B    Started.
B    Started.
C    Started.
B    Started.
C    Started.
B    Started.
C    Started.
A    Started.
A    Started.
A    Started.
A    Started.
A    Started.
```

Exercise 1 - Solution

```
import java.lang.*;
class MyThread extends Thread{
    private long sleepTime;

   MyThread (String s, long time){
        super(s);
        sleepTime = time;
        System.err.println("Name: "
        +getName() + "; sleep: "+sleepTime);
    }
//The rest of the code for class MyThread is on
//the next slide
```

Exercise 1 – Solution (Cont'd)

```
public void run() {
    for (int i = 0; i < 5; i++) {
        try {
            sleep(sleepTime);
        } catch (InterruptedException e) {
            System.err.println(e.toString());
        }

    System.err.println(getName() + " Started.");
}
}//End of class MyThread
```

Exercise 1 – Solution (Cont'd)

```
public class MyThreadDemo {
   public static void main(String[] args) {
      MyThread t1 = new MyThread("A", 1000);
      MyThread t2 = new MyThread("B", 1);
      MyThread t3 = new MyThread("C", 1);
      t1.start();
      t2.start();
      t3.start();
      System.out.println("Three Threads A, B, and c
      Started:");
  } // end of main function
} // and of class MyThreadDemo
```

Exercise 2

- Develop a new version of `SimpleThread` implementing `Runnable interface`.
- This class should have data field of type long that allows each thread to have its own `sleepTime`, and a `run` method that uses a for loop to iterate 5 times, displaying the thread's name, and its `sleepTime`.
- Develop a second class called `Demo` that creates three instances of `SimpleTread` in its `main method`.

Exercise 2 – First Solution

```
class MyThread implements Runnable {
        private long sleepTime;
        private String name;

        MyThread(String s, long time) {
                sleepTime = time;
                name = s;
                System.err.println("Name: " + s + "; sleep: " + sleepTime);
        }

        public void run() {
                for (int i = 0; i < 5; i++) {
                        try {
                                Thread.sleep(sleepTime);
                                } catch (InterruptedException e) {
                                        System.err.println(e.toString());
                                }
                                System.err.println(name + "    Started.");
                }
        }
}
```

Exercise 2 – First Solution (Cont'd)

```
public class MyThreadDemo{
   public static void main(String [] args){
       MyThread myThread1 = new MyThread("A", 1000);
       MyThread myThread2 = new MyThread("B", 1);
       MyThread myThread3 = new MyThread("C", 1);
       Thread t1 = new Thread (myThread1);
       Thread t2 = new Thread (myThread2);
       Thread t3 = new Thread (myThread3);
       t1.start();
       t2.start();
       t3.start();
       System.out.println("Three Threads A, B, and C Started:");
   }
}
```

Thread Issues

- Threads have their own call stack, which simplifies many applications, but it can effect the program's performance.
  - There is a limit on how many threads you can create without degrading the performance
- Other issues:
  - Data corruption: Threads that are unaware of sharing the same data.
  - Deadlocks: The program does not react anymore due to problems in the concurrent access of data deadlocks.
  - Safety failure: The program creates incorrect data.

How to Avoid Issues

- Java provides two keywords for this purpose:
  - `synchronized` – A keyword to avoid lock an object when it's used by another thread.
  - `volatile` – A keyword to indicate that a variable's value will be modified by different threads.
    - Access to the variable **acts as though it is enclosed in a synchronized block.**

Thread Synchronization

- Java uses **monitors** to perform **synchronization**.
- **Every object** with synchronized method or synchronized statement is a monitor.
- Invocation of a synchronized method will lock the object.
- When execution of the method is terminated the lock will be released.

Thread Synchronization

- To make a class useable in a multithreaded environment **the appropriate methods must be synchronized.**
- For in class `Account`, method `withdraw` is an appropriate method to become **synchronized**.

Example

```
class Account {
        private double amount;

        Account(double initDeposit) {
            amount = initDeposit;
        }

        synchronized public void withdraw(double amnt) {
            if (amount >= amnt) {
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                amount -= amnt;
            }
        }
// class definition continues on the next slide
```

Example (continued)

```
    public double balance() {
        return amount;
    }

    public void deposit(double amnt) {
        amount += amnt;
    }
} // end of class Account
```

Example (continued)

```
class MyThread extends Thread {
      private String s;
      private Account account;

      public MyThread(String ss, Account acc) {
            super(ss);
            account = acc;
      }
}
```

Example (continued)

```
    public void run() {
        for (int i = 0; i < 5; i++) {

            if (account.balance() >= 100)
                account.withdraw(100);

            System.err.println(getName() + " Started." + " balance: " +
            account.balance());

        } // end of for loop
    } // end of function run
} // end of class MyThread
```

Example (continued)

```
public class Demo {
   public static void main(String [] args){
      Account acc = new Account (200.00);
      MyThread t1 = new MyThread("A",  acc);
      MyThread t2 = new MyThread("B", acc);
      MyThread t3 = new MyThread("C", acc);
      t1.start();
      t2.start();
      t3.start();
      System.out.println("Three Threads A, C,
      and C Started:");
   }
}
```

Program Output when function Withdraw is Defined Synchronized

Problems @ Javadoc Declaration Console

```
<terminated> Demo [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe
Three Threads A, C, and C Started:
C    Started. balance: 100.0
A    Started. balance: 0.0
A    Started. balance: 0.0
A    Started. balance: 0.0
B    Started. balance: 0.0
B    Started. balance: 0.0
B    Started. balance: 0.0
B    Started. balance: 0.0
C    Started. balance: 0.0
B    Started. balance: 0.0
A    Started. balance: 0.0
C    Started. balance: 0.0
A    Started. balance: 0.0
C    Started. balance: 0.0
C    Started. balance: 0.0
```

Program Output When Function Withdraw is NOT Defined Synchronized

Problems @ Javadoc Declaration Console

```
<terminated> Demo [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (Feb 15, 2017,
Three Threads A, C, and C Started:
C    Started. balance: -100.0
C    Started. balance: -100.0
C    Started. balance: -100.0
C    Started. balance: -100.0
C    Started. balance: -100.0
B    Started. balance: -100.0
A    Started. balance: -100.0
A    Started. balance: -100.0
A    Started. balance: -100.0
A    Started. balance: -100.0
A    Started. balance: -100.0
B    Started. balance: -100.0
B    Started. balance: -100.0
B    Started. balance: -100.0
B    Started. balance: -100.0
```