JAC444 - Lecture 5

Threads

Segment 1 - Basics

Objectives

Upon completion of this lecture, you should be able to:

- Examine Concurrent Programming Design
- Create and Use Threads in Java
- Synchronize Threads and Avoid Thread Contention
- Analyze High Level Concurrency Objects

Threads

In this section you will be learning about:

- Process and Threads
- Critical Sections
- Defining and Starting a Thread
- Pausing Thread Execution: Sleep, Interrupts, and Joins

Thread Definition

Thread definition
A thread is a sequence of executing instructions that can run independently.

- Threads organize programs into logically separate paths.
- Thread can perform task independent of other threads.
- Threads can share access to common resources.

Pitfalls:

Race Condition

```
getResource();
modifyResource();
setResource();
```

Example: Bank Account

```
    x = a.getBalance();
x += deposit;
a.setBalance(x);
```

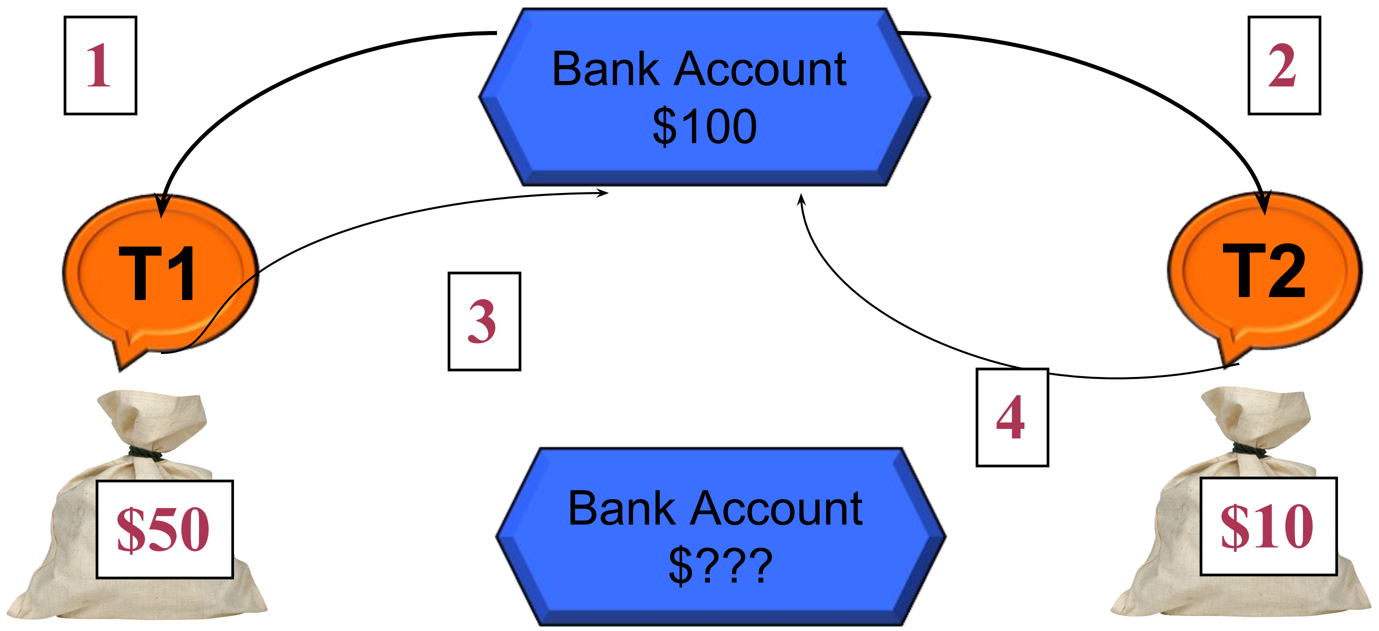
Bank Account – Race Condition

Race Condition

```
getResource();
modifyResource();
setResource();
```

Example: Bank Account

```
I.   x = account.getBalance();
II.  x = x + deposit;
III. account.setBalance(x);
```



Critical Sections

Critical Section definition:

Any part of the code in a program with the property that *only one thread can execute it* at any given time is called *critical section*.

Critical sections are called monitors.

Integrated support for threads is a key facet of Java technology

- Each thread is associated with an instance of the class `Thread`
- Directly control thread creation by building a *thread object*

Defining a Thread

1. **Extend Thread Class:**

```
public class MyThread extends Thread {
    public void run () {
    }
}
```

One must override **`run()`** method.

2. **Create a Runnable Object:**

```
public class MyRunnable implements Runnable {
    public void run() {
    }
}
```

One must implement **`run()`** method.

Thread Constructors

```
Thread()

Thread(Runnable target)

Thread(Runnable target, String name)

Thread(String name)

Thread(ThreadGroup group, Runnable target)

Thread(ThreadGroup group, Runnable target, String name)

Thread(ThreadGroup group, String name)
```

Subclass Thread Class

Create and start a thread by subclassing the `Thread` class:

```
public class MyThread extends Thread {
    int mark;

    MyThread(int m) { mark = m; }

   public void run() {
      // read the database value
      if (mark > value)
         System.out.println("Exam: pass");
   }

    public static void main(String args[]) {
      (new MyThread(75)).start();
   }
}
```

Create a Runnable Object

Build a thread using a Runnable object

```
public class MyRunnable implements Runnable {
  int mark;

  MyRunnable(int m) { mark = m; }

  public void run() {
    // val read from DB
     if (mark > val)
        System.out.println("Exam: pass!");
  }

  public static void main(String args[]) {
     (new Thread(new MyRunnable(75))).start();
  }
}
```

Pausing Execution - `sleep`

`public static void sleep(long millis) throws InterruptedException`
causes the current thread to suspend execution for specified period

Example:

```
for (...) {
    // Pause for 2 seconds
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        // ...
    }
}
```

Pausing Execution - `join`

```
public final void join(long millis) throws InterruptedException
```

The `join` method allows one thread to wait for the completion of another.

```
Thread t = …;
try {
        t.join(1000);
    } catch (InterruptedException e) {
        // ...
    }
}
```

causes the current thread to pause execution until t’s thread terminates

Example: SimpleThread

```
public class SimpleThread extends Thread {
  public SimpleThread(String str) {
    super(str);
  }
  public void run() {
    for (int i = 0; i < 3; i++) {
      System.out.println(i + " " + getName());
      try {
        Thread.sleep((long)(Math.random() * 1000));
      } catch (InterruptedException e) {}
    }
    System.out.println("DONE! " + getName());
  }
  public static void main (String[] args) {
    new SimpleThread("First >>>>>>>>>").start();
    new SimpleThread("Second <<<<<<<<<").start();

    System.out.println("DONE ALL!");
  }
}
```