Locks
CSCI 201L

Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

HTTP://WWW-SCF.USC.EDU/~CSCI201
Outline

- Locks
  - Lock Conditions
The `synchronized` keyword puts a restriction on a method that only one thread can be inside at a time.

- This is accomplished using a monitor.
- No other thread will be able to enter that method if another thread is currently executing inside of it, regardless of whether it is in the CPU currently or not.
Explicitly Acquiring Locks

- A synchronized method or block of code implicitly acquires a lock on the instance before executing.
- Java gives us the ability to explicitly acquire locks using the `java.util.concurrent.locks.Lock` interface and `java.util.concurrent.locks.ReentrantLock` class.
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class AddAPenny implements Runnable {
  private static PiggyBank piggy = new PiggyBank();

  public void run() {
    piggy.deposit(1);
  }

  public static void main(String[] args) {
    ExecutorService executor = Executors.newCachedThreadPool();
    for (int i=0; i < 100; i++) {
      executor.execute(new AddAPenny());
    }
    executor.shutdown();
    // wait until all tasks are finished
    while(!executor.isTerminated()) { }
    System.out.println("Balance = " + piggy.getBalance());
  }
}

class PiggyBank {
  private int balance = 0;
  public int getBalance() {
    return balance;
  }

  public synchronized void deposit(int amount) {
    int newBalance = balance + amount;
    Thread.yield();
    balance = newBalance;
  }
}
Explicitly Acquiring Locks Example

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class AddAPenny implements Runnable {
    private static PiggyBank piggy = new PiggyBank();

    public void run() {
        piggy.deposit(1);
    }

    public static void main(String [] args) {
        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i=0; i < 100; i++) {
            executor.execute(new AddAPenny());
        }
        executor.shutdown();
        // wait until all tasks are finished
        while(!executor.isTerminated()) {} 
        System.out.println("Balance = " + piggy.getBalance());
    }
}

class PiggyBank {
    private int balance = 0;
    private Lock lock = new ReentrantLock();
    public int getBalance() {
        return balance;
    }
    public void deposit(int amount) {
        lock.lock();
        try {
            int newBalance = balance + amount;
            Thread.yield();
            balance = newBalance;
        } finally {
            lock.unlock();
        }
    }
}
```
Thread Conditions

- Threads are able to communicate with each other based on specific conditions
- Threads can perform the following actions using the `java.util.concurrent.Condition` interface
  - `await()`
    - Wait for the condition to be signaled
  - `signal()`
    - Wake up one thread waiting on the condition
  - `signalAll()`
    - Wake up all of the threads waiting on the condition
- A **Condition** is created from a **Lock** object by calling the `newCondition()` method
Thread Conditions Example

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class AddAndRemoveAPenny implements Runnable {
    private static PiggyBank piggy = new PiggyBank();
    private boolean isWithdrawal;

    public void run() {
        if (isWithdrawal) {
            piggy.deposit((int)(Math.random() * 9 + 1));
        }
        else {
            piggy.withdraw((int)(Math.random() * 9 + 1));
        }
    }

    public static void main(String[] args) {
        ExecutorService executor = Executors.newCachedThreadPool();
        for (int i=0; i < 100; i++) {
            AddAndRemoveAPenny penny = new AddAndRemoveAPenny();
            if (i < 50) { // exactly 50 threads will withdraw
                penny.isWithdrawal = false;
            }
            else { // and exactly 50 threads will deposit
                penny.isWithdrawal = true;
            }
            executor.execute(penny);
        }
        executor.shutdown();
        // wait until all tasks are finished
        while(!executor.isTerminated()) {

        }
        System.out.println("Balance = " + piggy.getBalance());
    }
}

class PiggyBank {
    private int balance = 0;
    private Lock lock = new ReentrantLock();
    private Condition depositMade = lock.newCondition();
    public int getBalance() {
        return balance;
    }

    public void withdraw(int amount) {
        lock.lock();
        try {
            while (balance < amount) {
                System.out.print("Waiting for deposit...");
                System.out.print(" to withdraw $" + amount);
                System.out.println(" from balance of $" + balance);
                depositMade.await();
            }
            balance -= amount;
            System.out.print("\$" + amount + " withdrawn,");
            System.out.println(\" leaving balance of \$" + balance);
        } catch (InterruptedException ie) {
            System.out.println("IE: " + ie.getMessage());
        } finally {
            lock.unlock();
        }
    }

    public void deposit(int amount) {
        lock.lock(); // acquires lock
        try {
            balance += amount;
            System.out.print("\$" + amount + " deposited,");
            System.out.println(\" making balance of \$" + balance);
            depositMade.signalAll();
        } finally {
            lock.unlock(); // release the lock
        }
    }
}
```

Locks – Lock Conditions
Thread Conditions Issues

- Once a thread invokes `await()` on a condition, the thread will move to the waiting state until it is signaled
  - If `signal()` or `signalAll()` is never called on the condition, the thread will wait forever
  - In the previous example, some executions of the program never terminate if a withdrawal was not able to be made based on the balance after all the deposits were made

- You must first obtain the lock on the object before you are able to invoke a method on its `Condition`
  - When a thread calls `await()`, it will release its lock
    - It will not be able to start executing again, even after it is signaled, if it is not able to obtain the lock again