**JAC444 - Lecture 5**

Threads

Segment 2 - Synchronization

# Threads

**In this section you will be learning about:**

* Synchronization
* Synchronized Methods
* Deadlock
* Starvation and Livelock

# Bank Account – Race Condition

Race Condition Example: Bank Account

**getResource(); I. x = account.getBalance(); modifyResource(); II. x = x + deposit; setResource(); III. account.setBalance(x);**



Bank Account

$100



**T1**



**$50**



**T2**



**$10**

**1**

**2**

**3**

**4**



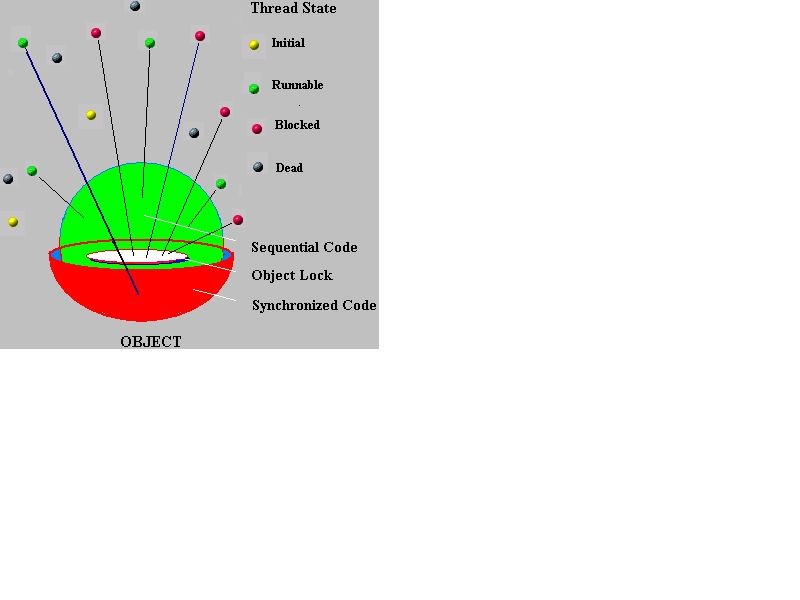
Bank Account

$???

# Synchronization Concepts

* 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

# Thread State and Intrinsic Lock



1

. Initial

2

. Runnable

3

. Blocked

4

. Dead

Object Intrinsic Lock

Unsynchronized Code

Synchronized Code

# Synchronized Methods

* When a thread invokes a synchronized method, it automatically acquires the intrinsic lock for that method's object
* In a synchronized method, the thread releases the acquired lock when the method returns

**class X extends Thread {**

**...**

**synchronized void method(...) {**

**... return;**

Intrinsic Lock

**}**

**public static void main(...) { Thread t = new X(); t.method();**

**}**

**}**

# Synchronized Statements

* Synchronized statements must specify the object that provides the intrinsic lock
* In a synchronized statements , the thread releases the acquired lock when the last statement is executed

**public void addName(String studentName) { synchronized(this) { lastName = studentName; nameCount++;**

Intrinsic Lock

**}**

**studentList.add(studentName);**

**}**

# Example Synchronized Method

public class SynThread implements Runnable { private String holdA = "This is "; private int[] holdB = {1,2,3,4,5,6,7,8,9,10};

//without synchronized keyword public void run() {

for(int w = 0; w < 10; w++) {

System.out.println(holdA + holdB[w] + ".");

}

}

public static void main(String args[]) {

SynThread z = new SynThread(); new Thread(z).start(); new Thread(z).start();

}

}

Run this code twice: 1. as is, and 2. add **synchronized** keyword to **run** method.

Can you see the difference?

# Liveness

*Liveness* is the property of a concurrent application to execute in a timely manner.

Liveness Problems:

1. *Deadlock*

Deadlock occurs when multiple threads need the same locks but obtain them in different order

1. *Starvation*

Starvation occurs when a thread is unable to gain regular access to shared resources and is unable

# Deadlock Example

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

**public class Deadlock { public static void main(String[] args) {**

**final Object r1 = "r1"; final Object r2 = "r2";**

**Thread t1 = new Thread(() -> {synchronized(r1){ synchronized(r2){} } });**

**Thread t2 = new Thread(() -> {synchronized(r2){ synchronized(r1){}**

**} });**

**t1.start(); t2.start();**

**}**

**}**