Introduction to Java for C++ Programmers

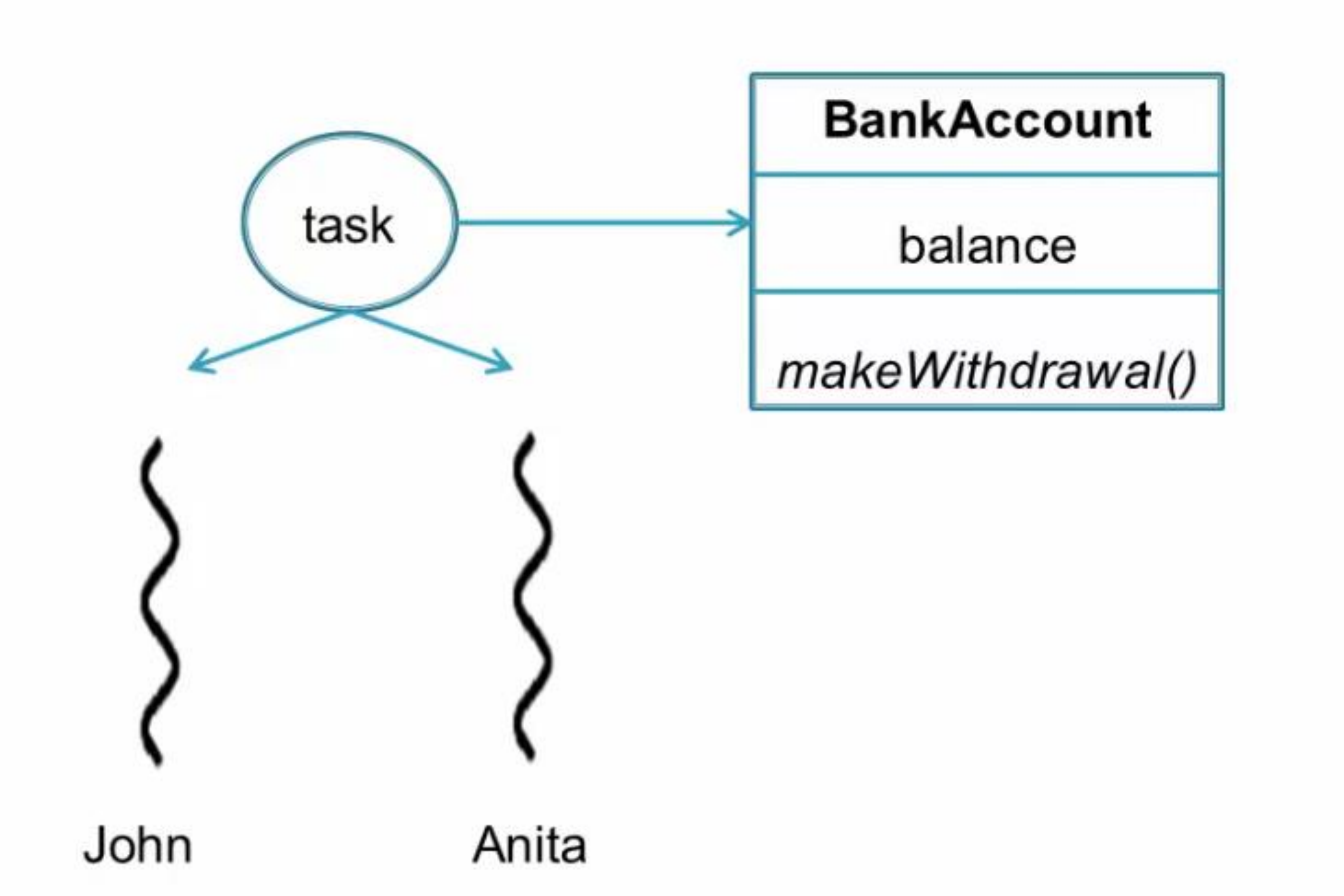
Thread Synchronization

By: Mahboob Ali

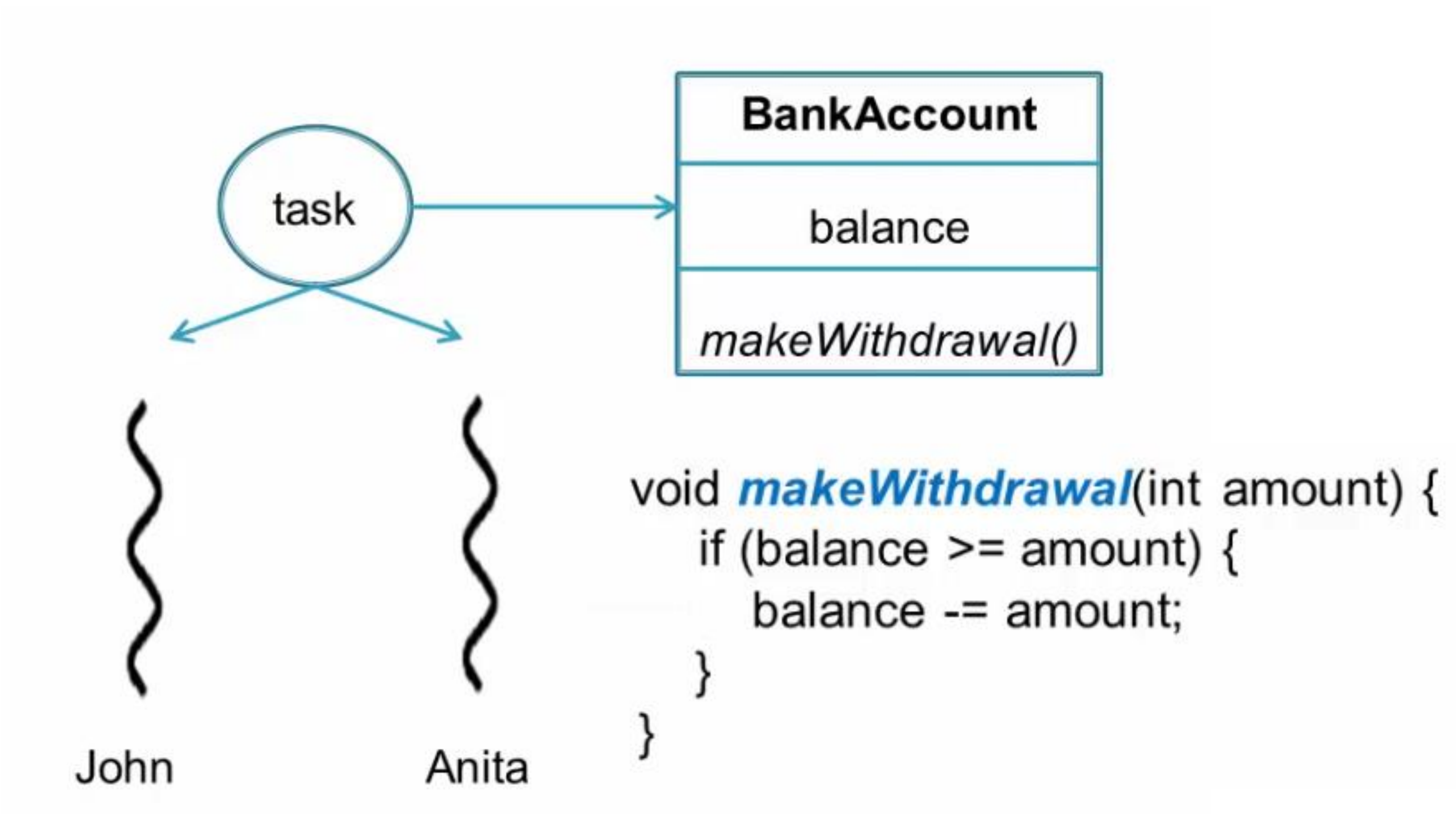
Thread Synchronization

- 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.

Concurrency Hazard: Race Condition



Concurrency Hazard: Race Condition



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

John and Anita wants to withdraw $75

John

Enters *makeWithdrawal()*

checks *balance* >= *amount*

Moved to RUNNABLE

Moved to Running

Overdraws assume the

balance is 100

Anita

Enters *makeWithdrawal()*

checks *balance* >= *amount*

balance -= amount

Balance -> 25

BankAccount object was not ***thread safe***

mutable state → *shared* → *not properly managed*

Race Condition ~ *check-then-act*

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

atomic unit

Synchronization Concept

- Synchronization is built around the concept known as the <u>*intrinsic lock*</u>
- 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

Why Synchronization?

- The synchronization is mainly used to
  - Prevent thread interference.
  - Prevent consistency problem.

Types of Synchronization

- There are two types of synchronization
  - Process Synchronization.
  - Thread Synchronization.

Thread Synchronization

- There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   - Synchronized method.
   - Synchronized block.
   - static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

- Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

- by synchronized method
- by synchronized block
- by static synchronization

Synchronized Method

- When a thread invokes a synchronized method, it automatically acquires the intrinsic lock for that method's object

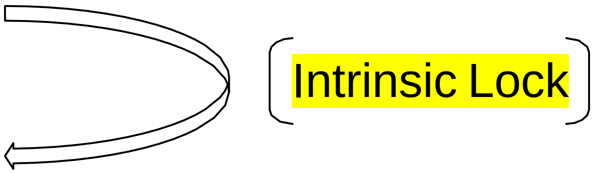
- In a synchronized method, the thread releases the acquired lock when the method returns

```
class X implements Runnable {
...
 synchronized void method(...) {
   ...
  return;
 }

  public static void main(...) {

       Thread t = new Thread(new X());

       t.start();
 }
}
```



Intrinsic Lock

```
Class Table{
  void printTable(int n){//method not synchronized
   for(int i = 1; i <= 5; i++){
     System.out.println( n * i);
     try{  Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
   }
  }
}
class MyThread1 implements Runnable{
   Table t;
   MyThread1(Table t){
        this.t=t; }
    @Override
    public void run(){
         t.printTable(5);
       }
  }
```

```
class MyThread2 implements Runnable{
     Table t;
     MyThread2(Table t){
        this.t=t;
    }
    @Override
    public void run(){
        t.printTable(100);
    }
}
class TestSynchronization1{
   public static void main(String args[]){
   Table obj = new Table();
   Thread t1 = new Thread(new MyThread1(obj));
   Thread t2 = new Thread(new MyThread2(obj));
   t1.start();
   t2.start();
  }
}
```

Output:

```
 5
100
10
200
15
300
20
400
25
500
```

Inconsistent

```
Class Table{
  synchronized void printTable(int n){
   //method synchronized
   for(int i = 1; i <= 5; i++){
     System.out.println( n * i);
     try{  Thread.sleep(400);
      }catch(Exception e){System.out.println(e);}
   }
  }
}
class MyThread1 implements Runnable{
   Table t;
   MyThread1(Table t){
        this.t=t; }
    @Override
    public void run(){
         t.printTable(5);
       }
  }
```

```
class MyThread2 implements Runnable{
     Table t;
     MyThread2(Table t){
        this.t=t;
    }
    @Override
    public void run(){
        t.printTable(100);
    }
}
class TestSynchronization1{
   public static void main(String args[]){
   Table obj = new Table();
   Thread t1 = new Thread(new MyThread1(obj));
   Thread t2 = new Thread(new MyThread2(obj));
   t1.start();
   t2.start();
  }
}
```

Output:

```
 5
10
15
20
25
100
200
300
400
500
```

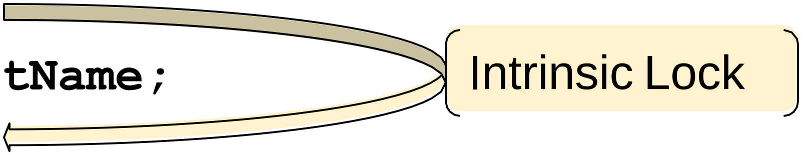
Consistent

Synchronized Block

- 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
- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

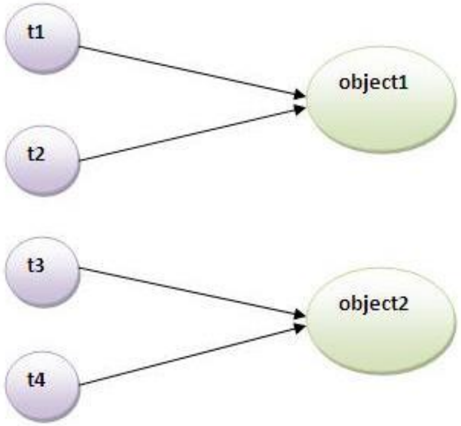
```
public void addName(String studentName) {
    synchronized(this) {
        lastName = studentName;
        nameCount++;
    }
    studentList.add(studentName);
}
```



Intrinsic Lock

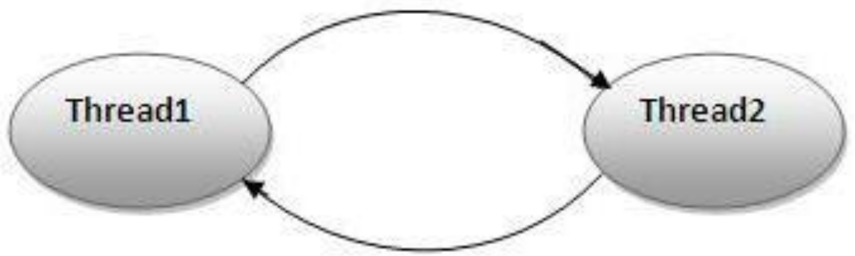
Static Synchronization

- If you make any static method as synchronized, the lock will be on the class not on object.
- Suppose there are two objects of a shared class(e.g. Table) named object1 and object2.
- In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock.
- But there can be interference between t1 and t3
- or t2 and t4 because t1 acquires another lock and t3 acquires another lock.
- I want no interference between t1 and t3 or t2 and t4.
- Static synchronization solves this problem.



Deadlock Example

- The threads t1and t2 are blocked forever, waiting for each other - this problem is defined as being a *deadlock*



```
public class TestDeadlockExample1 {
  public static void main(String[] args) {
    final String resource1 = “Some Name";
    final String resource2 = “Other Name";

    // t1 tries to lock resource1 then resource2
    Thread t1 = new Thread() {
      public void run() {
       synchronized (resource1) {
         System.out.println("Thread 1: locked resource 1");


         try { Thread.sleep(100);} catch (Exception e) {}

       synchronized (resource2) {
         System.out.println("Thread 1: locked resource 2");

           }
         }
      }
    };
// t2 tries to lock resource2 then resource1

   Thread t2 = new Thread() {
      public void run() {
        synchronized (resource2) {
        System.out.println("Thread 2: locked resource 2");

          try { Thread.sleep(100);} catch (Exception e) {}

          synchronized (resource1) {
           System.out.println("Thread 2:locked resource 1");

          }
        }
      }
    };
    t1.start();
    t2.start();
  }
}
```

Output: Thread 1: locked resource 1
Thread 2: locked resource 2