15-440
Distributed Systems
Recitation 6

Slides By: Hend Gedawy &

Laila Elbeheiry

Announcements

PS3 Released

Due: Oct. 16th

Project 1

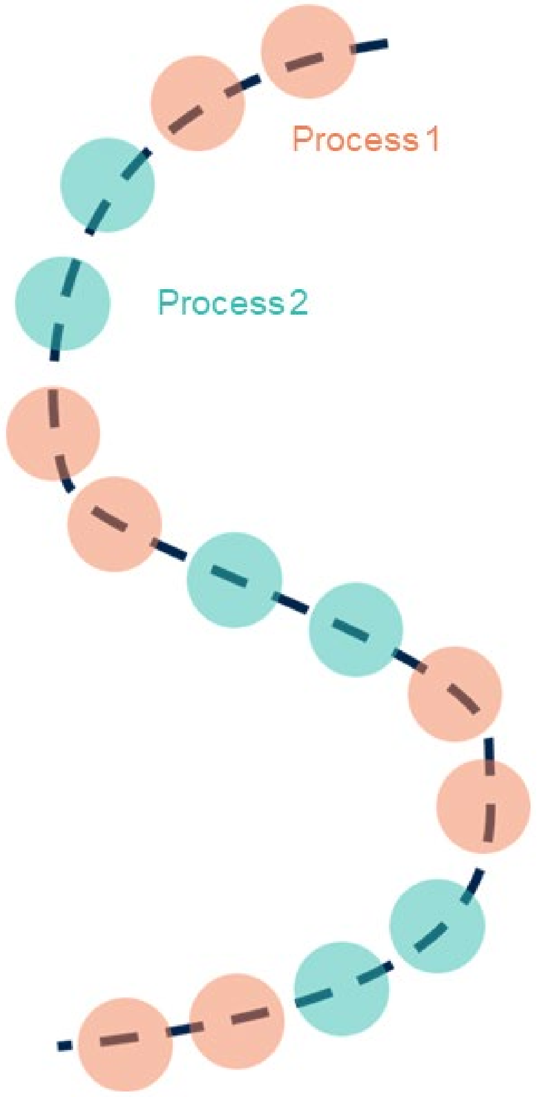
Due: Oct. 1st (Sunday)

Outline

- Concurrent Programming Introduction
  - Defining Concurrency?
  - Concurrency versus parallelism
  - Why Concurrency?
  - Concurrency in Java
- Ensuring Safety in Concurrent Programs
  - Thread Synchronization & challenges
  - Bank Use Case Example: Multiple Threads using abstract shared memory
- More on Concurrency

Outline

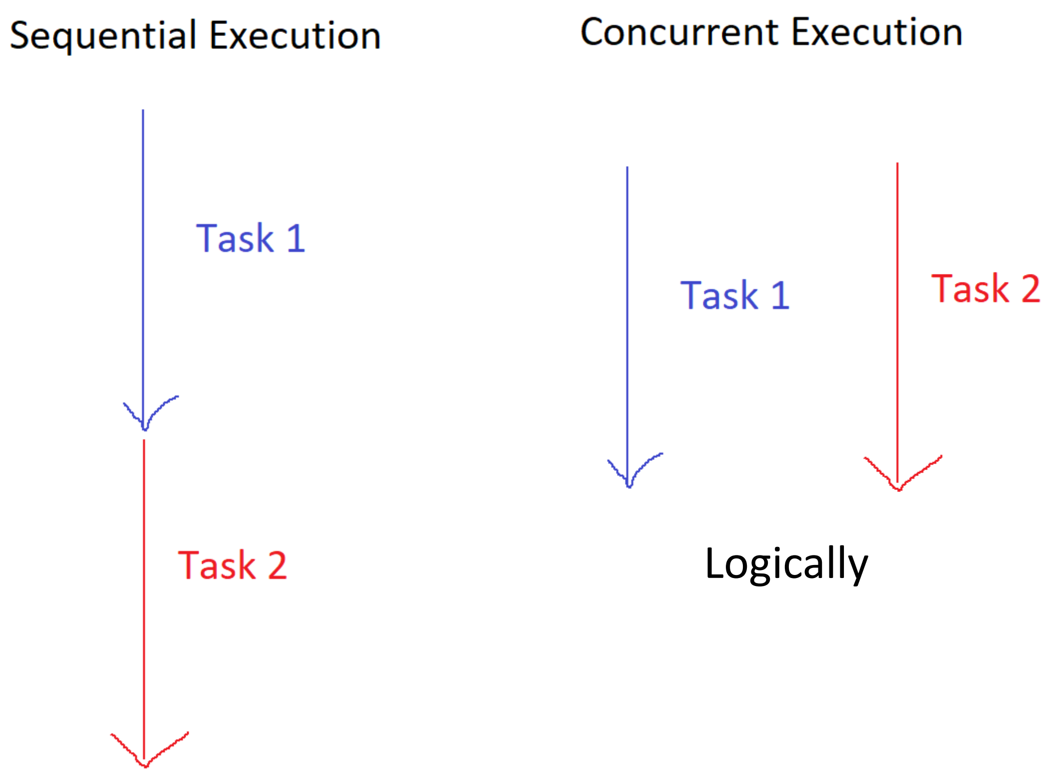
- **Concurrent Programming Introduction**
  - **Defining Concurrency?**
  - **Concurrency versus parallelism**
  - **Why Concurrency?**
  - **Concurrency in Java**
- Ensuring Safety in Concurrent Programs
  - Thread Synchronization & challenges
  - Bank Use Case Example: Multiple Threads using abstract
- More on Concurrency



From Sequential To Concurrent

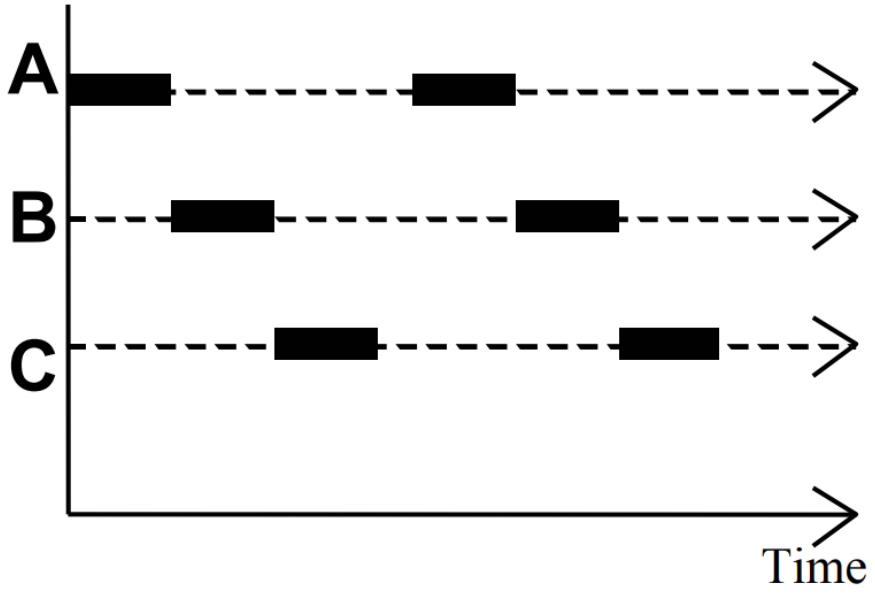
- Sequential Programs
  - Single thread of control
  - Executes one instruction at a time

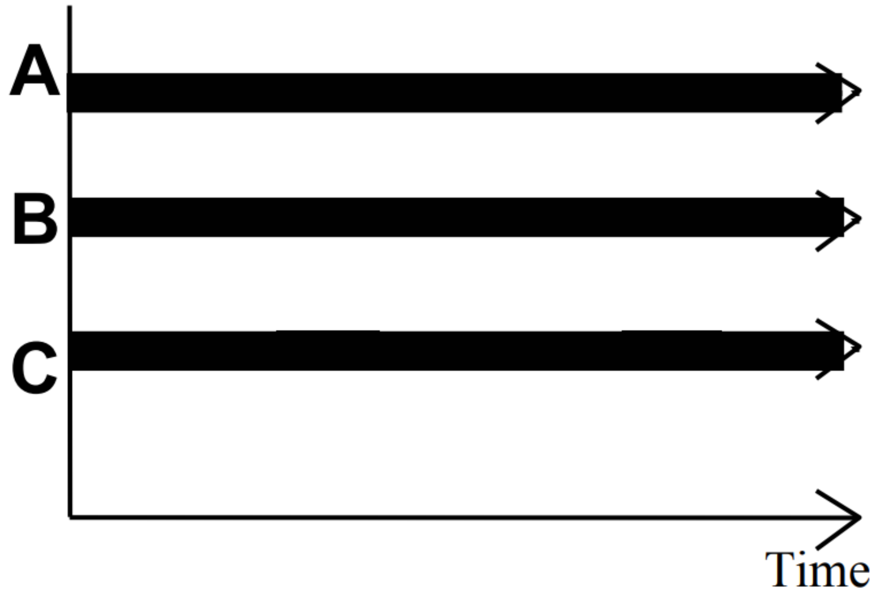
- Concurrent Programs
  - Multiple autonomous sequential threads, executing (logically) in parallel



Concurrency vs. Parallelism

- Concurrency doesn't imply parallelism ← Why?





Concurrency is the basis for writing parallel programs.
Parallel programs have the same correctness issues as concurrent

Implementing/Executing Multiple Threads

- **Multiprogramming** – Threads multiplex their executions on a single processor.
- **Multiprocessing** – Threads multiplex their executions on a multiprocessor or a system
- **Distributed Processing** – Processes multiplex their executions on several different machines

Why Concurrency?

- Natural application structure
- Increased Application throughput & responsiveness
- With multi-cores & multi- processors hardware, you can get parallel execution
- Also, when you are building a large distributed system

Concurrency in Java

- Java has a predefined class `java.lang.Thread`

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

- Java also provides a standard interface

```
public interface Runnable {
    public void run();
}
```

- Any *class* which wishes to *express concurrent execution* must *implement this interface and the* `run` *method*
- *Threads* do *not begin* their execution *until* the `start` method in the `Thread` class is *called*

Concurrency in Javas - Steps

- STEP 1: A class intended *to execute as a thread* must implement the ***Runnable*** interface

```
public class Service implements Runnable
```

  - Implement the method ***run()***

```
public void run() {  //thread's logic goes here }
```

- STEP 2: Instantiate a Thread object *passing an instance of the intended class*

```
Thread t = new Thread(new Service())
```

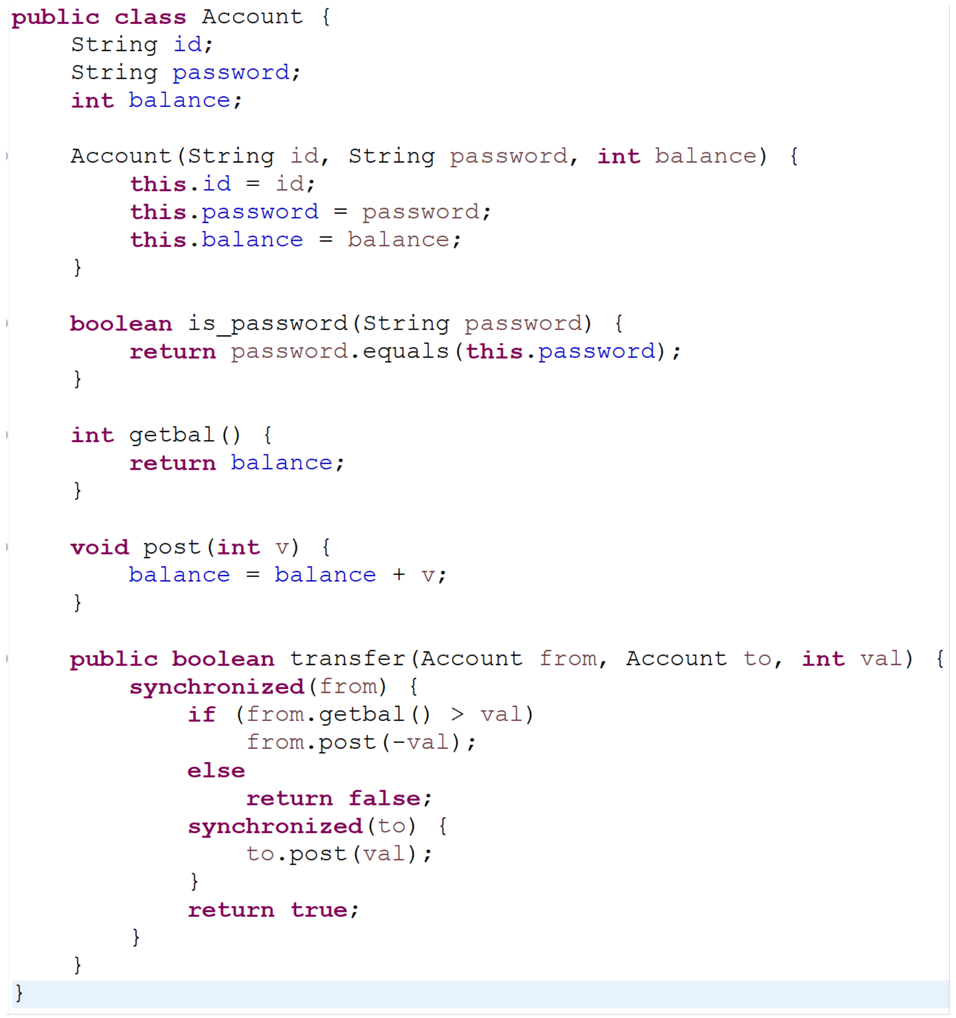
- STEP 3: Invoke ***start()*** on the new thread

```
t.start()              // invokes the run() method implemented in
                           the Service class
```

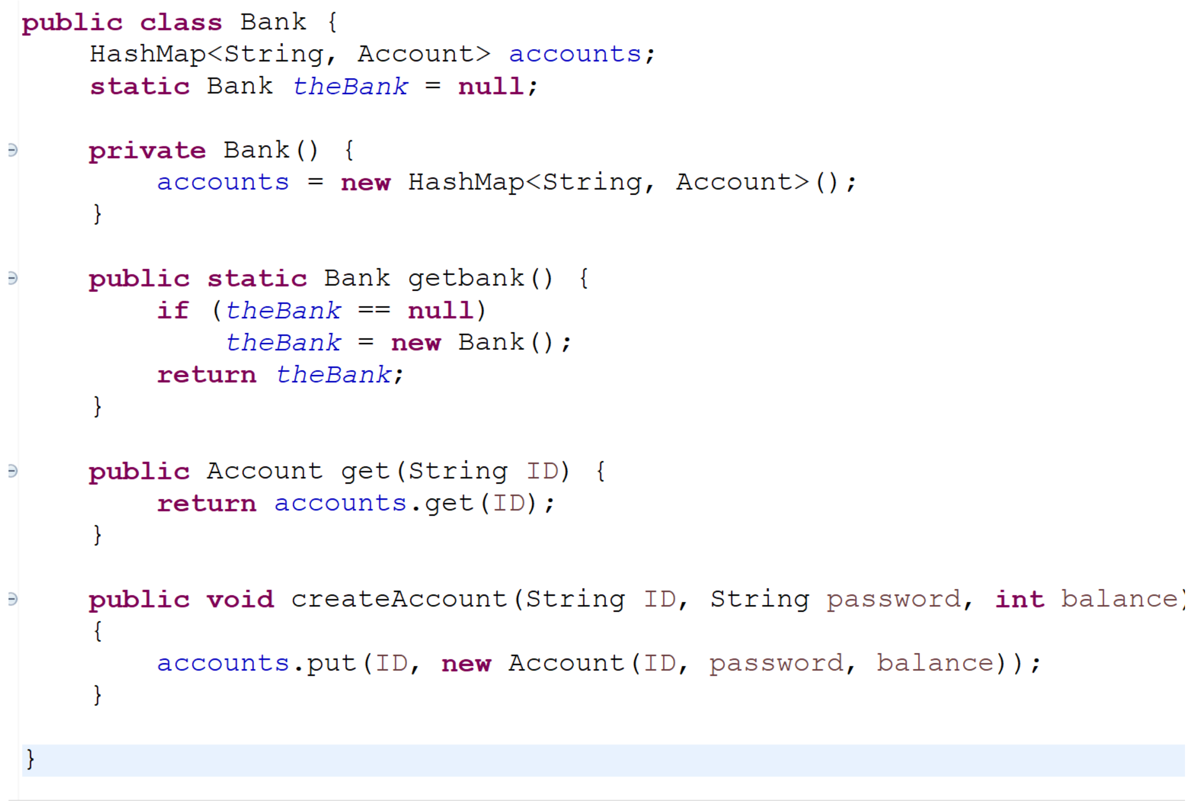
Outline

- Concurrent Programming Introduction
  - Defining Concurrency?
  - Concurrency versus parallelism
  - Why Concurrency?
  - Concurrency in Java
- **Ensuring Safety in Concurrent Programs**
  - **Thread Synchronization & challenges**
  - **Bank Use Case Example: Multiple Threads using abstract shared memory**
- More on Concurrency

Bank Example



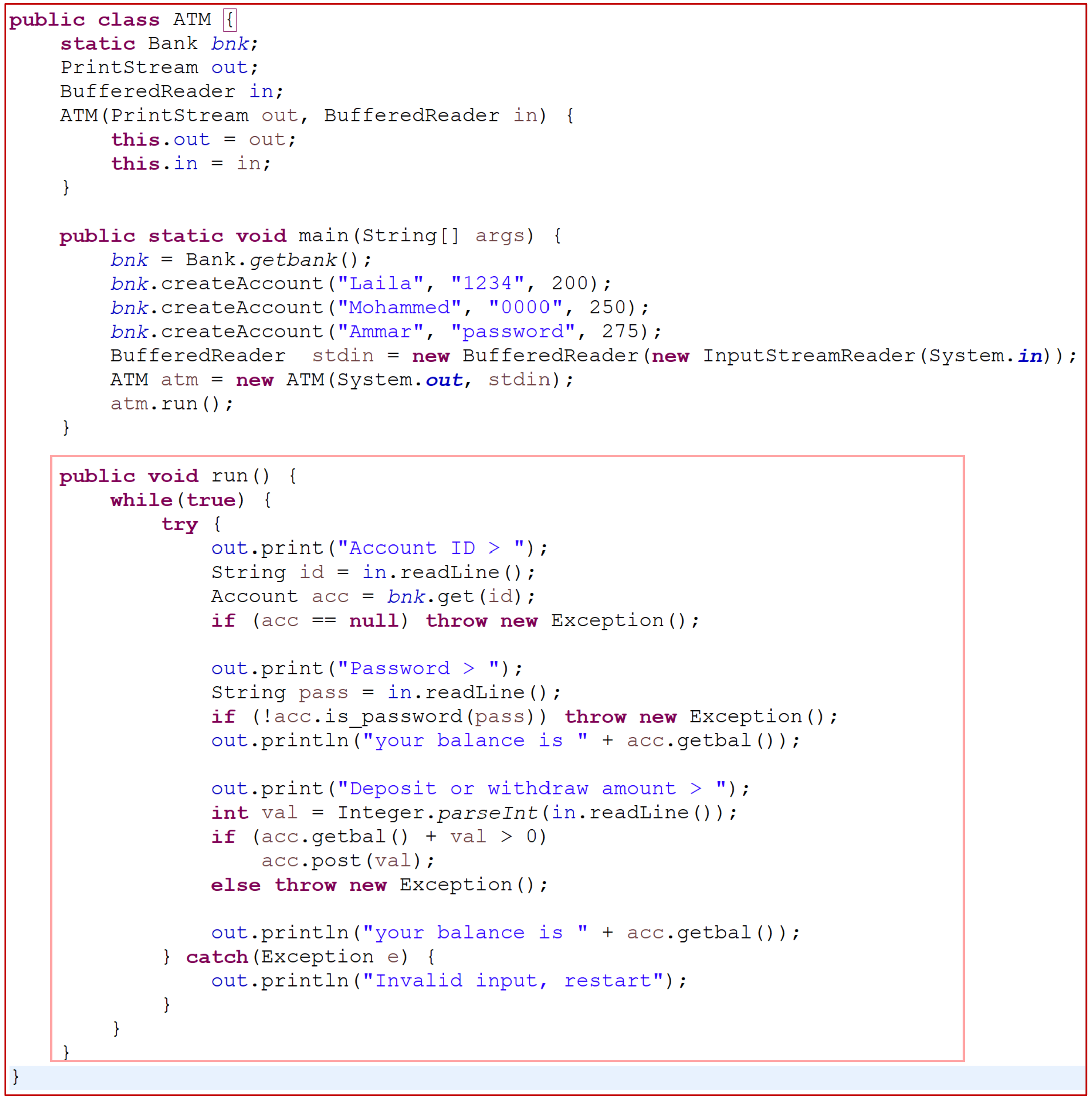
```java
public class Account {
    String id;
    String password;
    int balance;

    Account(String id, String password, int balance) {
        this.id = id;
        this.password = password;
        this.balance = balance;
    }

    boolean is_password(String password) {
        return password.equals(this.password);
    }

    int getbal() {
        return balance;
    }

    void post(int v) {
        balance = balance + v;
    }

    public boolean transfer(Account from, Account to, int val) {
        synchronized(from) {
            if (from.getbal() > val)
                from.post(-val);
            else
                return false;
            synchronized(to) {
                to.post(val);
            }
            return true;
        }
    }
}
```



```java
public class Bank {
    HashMap<String, Account> accounts;
    static Bank theBank = null;

    private Bank() {
        accounts = new HashMap<String, Account>();
    }

    public static Bank getbank() {
        if (theBank == null)
            theBank = new Bank();
        return theBank;
    }

    public Account get(String ID) {
        return accounts.get(ID);
    }

    public void createAccount(String ID, String password, int balance)
    {
        accounts.put(ID, new Account(ID, password, balance));
    }

}
```

Bank Example- With 1 ATM

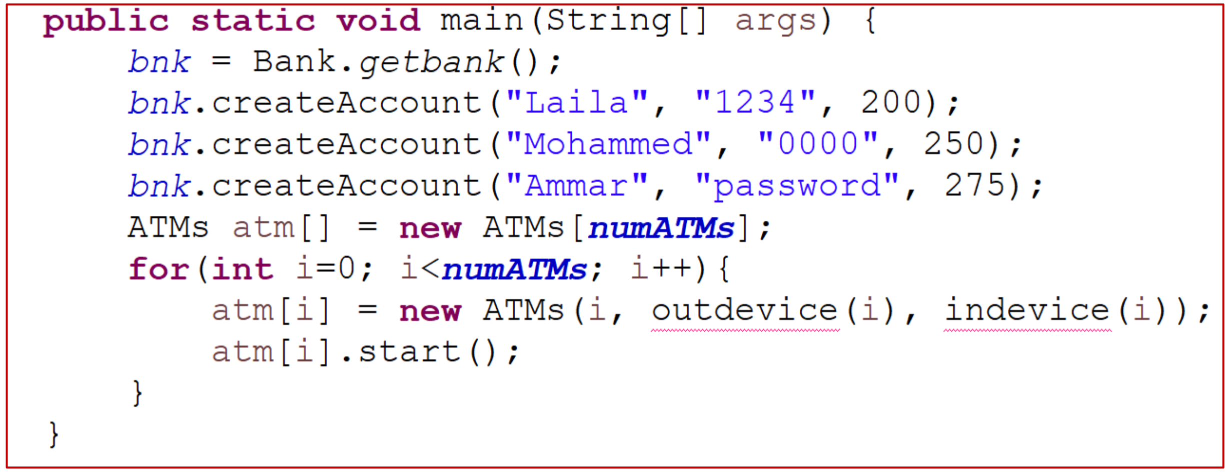
```
Account ID > Hend
Password > 1234
your account balance is 200
Deposit or withdraw amount > -150
your balance is 50
```



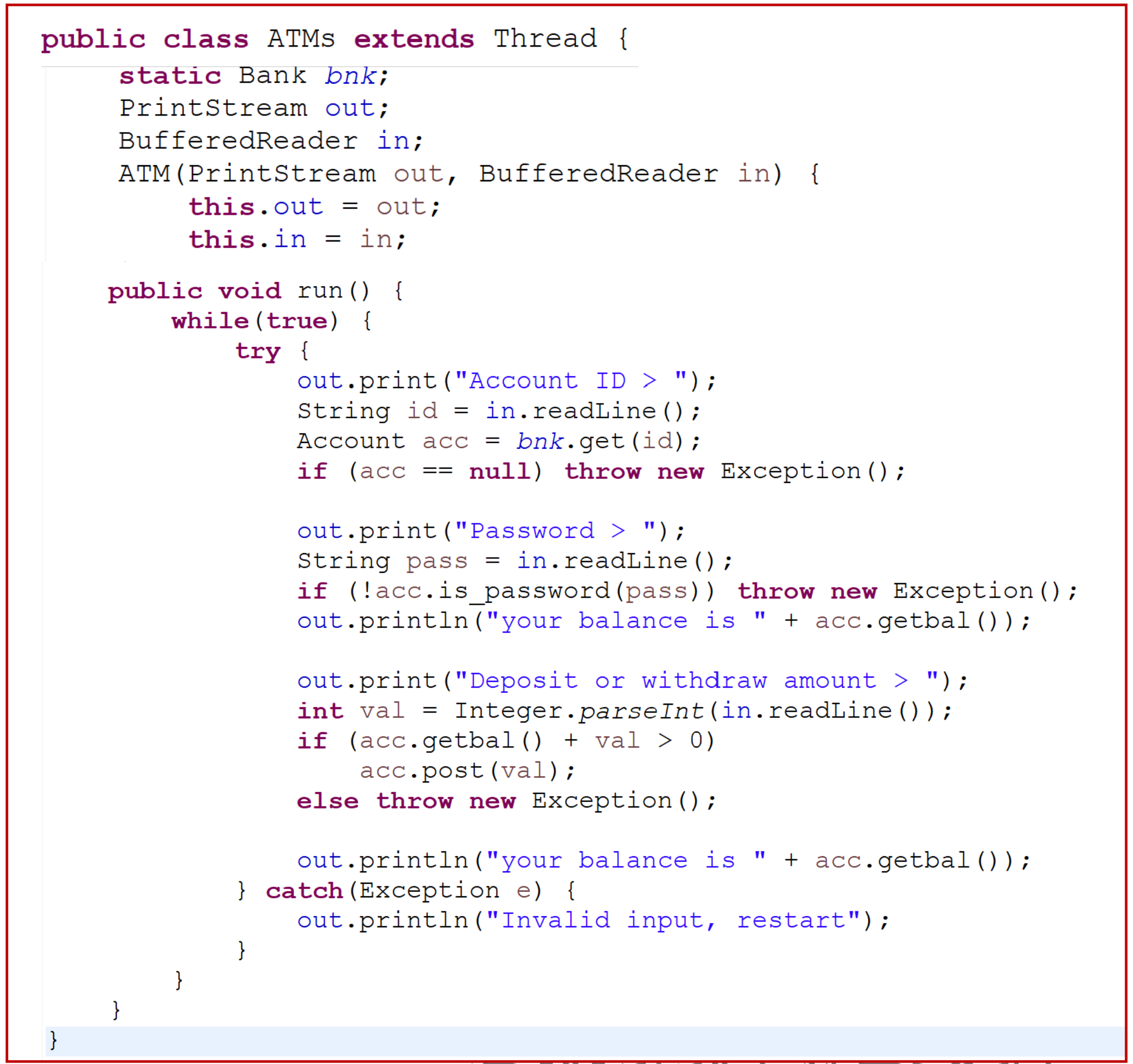
```
public class ATM {
    static Bank bnk;
    PrintStream out;
    BufferedReader in;
    ATM(PrintStream out, BufferedReader in) {
        this.out = out;
        this.in = in;
    }

    public static void main(String[] args) {
        bnk = Bank.getbank();
        bnk.createAccount("Laila", "1234", 200);
        bnk.createAccount("Mohammed", "0000", 250);
        bnk.createAccount("Ammar", "password", 275);
        BufferedReader  stdin = new BufferedReader(new InputStreamReader(System.in));
        ATM atm = new ATM(System.out, stdin);
        atm.run();
    }

    public void run() {
        while(true) {
            try {
                out.print("Account ID > ");
                String id = in.readLine();
                Account acc = bnk.get(id);
                if (acc == null) throw new Exception();

                out.print("Password > ");
                String pass = in.readLine();
                if (!acc.is_password(pass)) throw new Exception();
                out.println("your balance is " + acc.getbal());

                out.print("Deposit or withdraw amount > ");
                int val = Integer.parseInt(in.readLine());
                if (acc.getbal() + val > 0)
                    acc.post(val);
                else throw new Exception();

                out.println("your balance is " + acc.getbal());
            } catch(Exception e) {
                out.println("Invalid input, restart");
            }
        }
    }
}
```

Bank Example- Multiple ATMs

Create Multiple ATM Threads



```java
public static void main(String[] args) {
    bnk = Bank.getbank();
    bnk.createAccount("Laila", "1234", 200);
    bnk.createAccount("Mohammed", "0000", 250);
    bnk.createAccount("Ammar", "password", 275);
    ATMs atm[] = new ATMs[numATMs];
    for(int i=0; i<numATMs; i++){
        atm[i] = new ATMs(i, outdevice(i), indevice(i));
        atm[i].start();
    }
}
```



```java
public class ATMs extends Thread {
     static Bank bnk;
     PrintStream out;
     BufferedReader in;
     ATM(PrintStream out, BufferedReader in) {
          this.out = out;
          this.in = in;

    public void run() {
        while(true) {
            try {
                out.print("Account ID > ");
                String id = in.readLine();
                Account acc = bnk.get(id);
                if (acc == null) throw new Exception();

                out.print("Password > ");
                String pass = in.readLine();
                if (!acc.is_password(pass)) throw new Exception();
                out.println("your balance is " + acc.getbal());

                out.print("Deposit or withdraw amount > ");
                int val = Integer.parseInt(in.readLine());
                if (acc.getbal() + val > 0)
                    acc.post(val);
                else throw new Exception();

                out.println("your balance is " + acc.getbal());
            } catch(Exception e) {
                out.println("Invalid input, restart");
            }
        }
    }
}
```

Activity Trace 1 of ATMs

```
out.print("Deposit or withdraw amount > ");
int val = Integer.parseInt(in.readLine());
if (acc.getbal() + val > 0)
    acc.post(val);
else throw new Exception();

out.println("your balance is " + acc.getbal());
```

Time

Thread 1	Thread 2
	Account ID > Sana
Account ID > Hend	
Password > 1234	
	Password > 0000
	your account balance is 250
your account balance is 200	
	Deposit or withdraw amount > -50
	your balance is 200
Deposit or withdraw amount > -150	
your balance is 50	

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

```
out.print("Deposit or withdraw amount > ");
int val = Integer.parseInt(in.readLine());
if (acc.getbal() + val > 0)
    acc.post(val);
else throw new Exception();

out.println("your balance is " + acc.getbal());
```

Activity Trace 2 of ATMs

Time

Thread 1	Thread 2
Account ID >	Account ID >
Hend	
Password >	Hend
	Password >
1234	
Your account balance is 200	
Deposit or withdraw amount >	1234
	Your account balance is 200
	Deposit or withdraw amount >
-150	-150
your balance is 50	
	your balance is 50

200 – 150 – 150 = 50!!!

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Activity Trace 2 of ATMs – Zoomed In

```
out.print("Deposit or withdraw amount > ");
int val = Integer.parseInt(in.readLine());
if (acc.getbal() + val > 0)
    acc.post(val);
else throw new Exception();

out.println("your balance is " + acc.getbal());
```

Time

Balance	Thread 1	Thread 2
	Deposit or withdraw amount >	Deposit or withdraw amount >
	-150	
		-150
	val= -150	
		val=-150
200	acc.getbal()+val > 0 //yes	
200		acc.getbal()+val > 0 //yes
50	acc.post(-150)	acc.post(-150)
50	Your balance is 50	
		Your balance is 50

200 – 150 – 150 = 50!!!

Thread 2 calls post() while the Thread 1 is not finished yet, so the execution interleave

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

```
void post(int v) {
    balance = balance + v;
}
```

How Could this Happen? – Post()

Time

Thread 1

Post(int v) // v=-150

- Balance = 200 — Read value
- Balance -150 — You subtracted but didn't write the result yet

Thread 2

Post (int v) //v=-150

- Balance = 200 — Read value
- Balance - 150 — You subtracted but didn't write the result yet
- Balance =50 — You write balance value 50

Thread 1

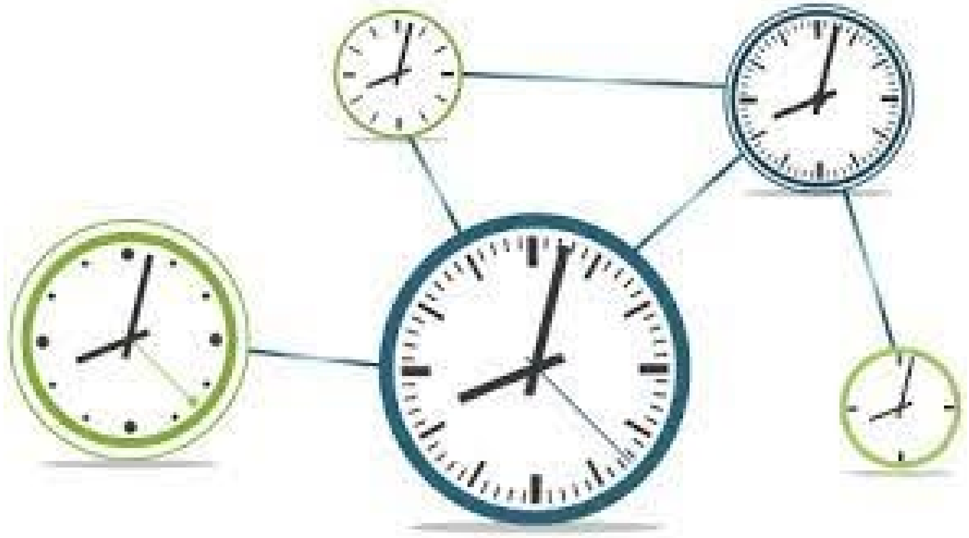
- Balance = 50 — You write balance value 50

Source of the problem

- Threads can be arbitrarily interleaved
- Some interleavings are NOT correct

How to Resolve it

- Java provides *synchronization* mechanism to restrict the interleavings



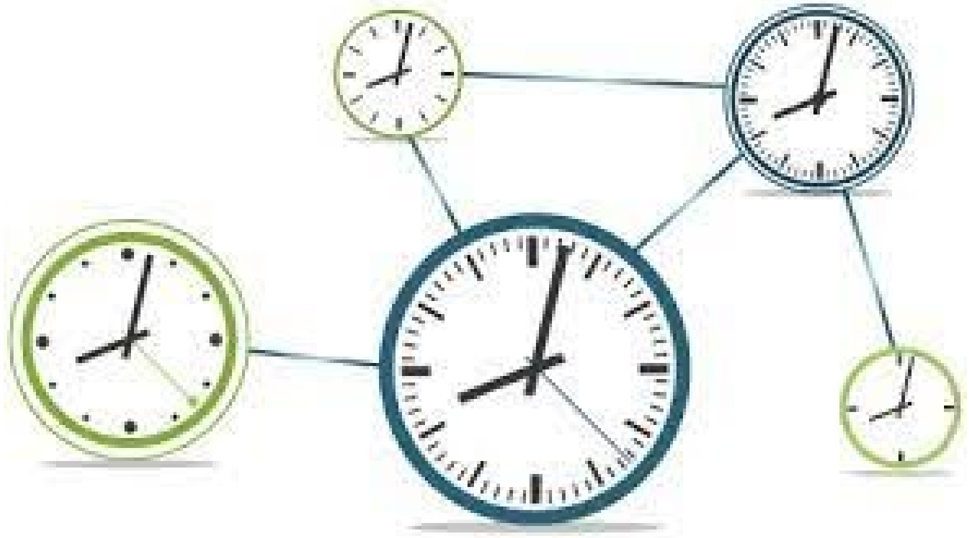
Synchronization: Restricting Intervealings

Synchronization serves two purposes:

- **Ensure safe threads access** for shared updates/resources – Avoid race conditions.
- **Coordinate actions** of threads – Parallel computation – Event notification

Multiple Threads access to a shared resource is Safe only if:

- All accesses have no effect on resource, – e.g., reading a variable
- All accesses are atomic
- Only one access at a time: mutual exclusion



Synchronization: Restricting Intervealings
Mutual Exclusion

- Prevent more than one thread from accessing critical section at a given time
- Once a thread is in the critical section, no other thread can enter that critical section until the first thread has left the critical section.
- *No interleavings* of threads within the critical section
- Serializes access to section

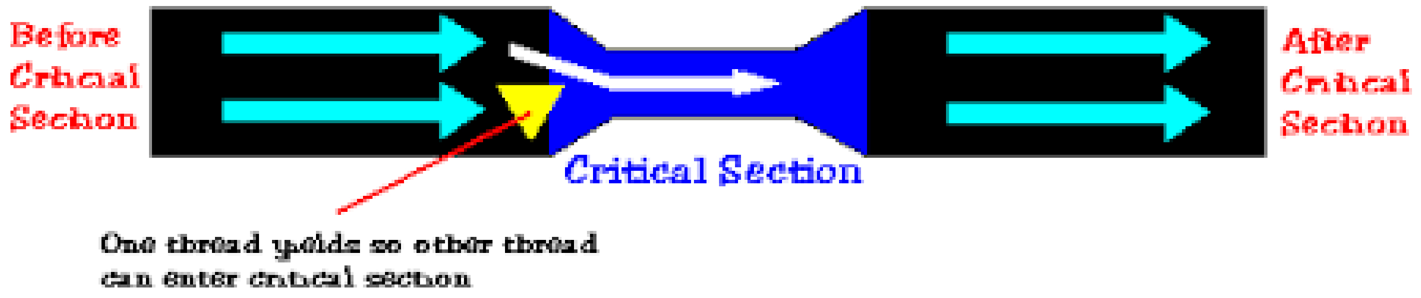


Photo-Credit: http://www.delphicorner.f9.co.uk/articles/op4.htm

How to Synchronize? – Mutual Exclusion In Java

ATM Thread Logic

```
while(true) {
    try {
        out.print("Account ID > ");
        String id = in.readLine();
        Account acc = bnk.get(id);
        if (acc == null) throw new Exception();

        out.print("Password > ");
        String pass = in.readLine();
        if (!acc.is_password(pass)) throw new Exception();
        out.println("your balance is " + acc.getbal());

        out.print("Deposit or withdraw amount > ");
        int val = Integer.parseInt(in.readLine());
        if (acc.getbal() + val > 0)
            acc.post(val);
        else throw new Exception();

        out.println("your balance is " + acc.getbal());
    } catch(Exception e) {
        out.println("Invalid input, restart");
    }
}
```

- Identify critical sections in code
- Add Synchronized keyword on critical sections
  - one thread can be executing it at any one time

Is this Good Enough??

Post() method in the Account class

```
void post(int v) {
    balance = balance + v;
}
```

```
synchronized void post(int v) {
    balance = balance + v;
}
```

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Activity Trace 2 of ATMs: Is it Fixed Now?

```
out.print("Deposit or withdraw amount > ");
int val = Integer.parseInt(in.readLine());
if (acc.getbal() + val > 0)
    acc.post(val);
else throw new Exception();

out.println("your balance is " + acc.getbal());
```

Balance	Thread 1	Thread 2
	Deposit or withdraw amount >	Deposit or withdraw amount >
		-150
	-150	
	val= -150	
		val=-150
200	acc.getbal()+val > 0 // yes	
200		acc.getbal()+val > 0 //yes
	acc.post(-150)	
50		acc.post(-150)
-100	your balance is -100	
		your balance is -100

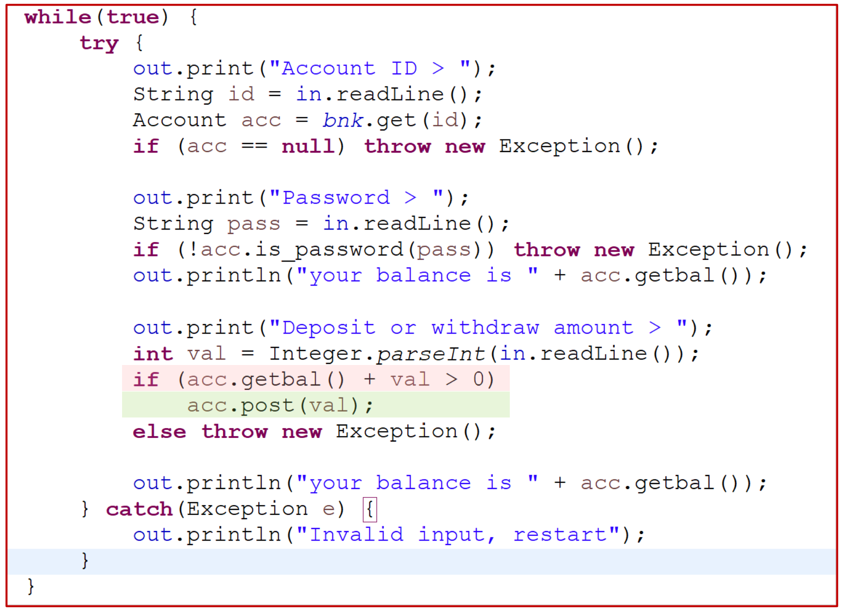
Time

Negative Bank Balance !

Thread 2 calls post only when Thread 1 returns from post

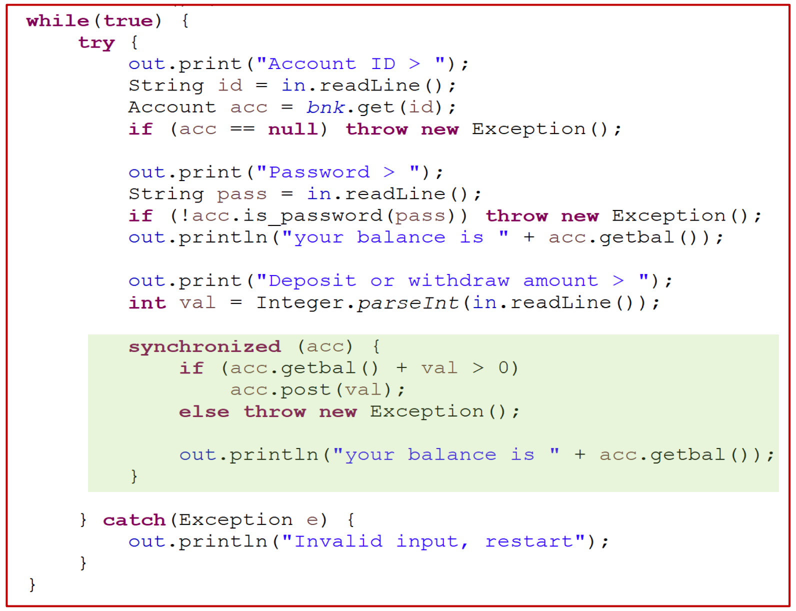
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

How to Synchronize? – Block Synchronization



```
while(true) {
    try {
        out.print("Account ID > ");
        String id = in.readLine();
        Account acc = bnk.get(id);
        if (acc == null) throw new Exception();

        out.print("Password > ");
        String pass = in.readLine();
        if (!acc.is_password(pass)) throw new Exception();
        out.println("your balance is " + acc.getbal());

        out.print("Deposit or withdraw amount > ");
        int val = Integer.parseInt(in.readLine());
        if (acc.getbal() + val > 0)
            acc.post(val);
        else throw new Exception();

        out.println("your balance is " + acc.getbal());
    } catch(Exception e) {
        out.println("Invalid input, restart");
    }
}
```

Let's Lock the account starting from when a transaction request is made until response it sent to user



```
while(true) {
    try {
        out.print("Account ID > ");
        String id = in.readLine();
        Account acc = bnk.get(id);
        if (acc == null) throw new Exception();

        out.print("Password > ");
        String pass = in.readLine();
        if (!acc.is_password(pass)) throw new Exception();
        out.println("your balance is " + acc.getbal());

        out.print("Deposit or withdraw amount > ");
        int val = Integer.parseInt(in.readLine());

        synchronized (acc) {
            if (acc.getbal() + val > 0)
                acc.post(val);
            else throw new Exception();

            out.println("your balance is " + acc.getbal());
        }

    } catch(Exception e) {
        out.println("Invalid input, restart");
    }
}
```

Synchronized Methods execute the body of the method as an atomic unit.

May need to synchronize not only the method but a lot more in there;

- Synchronize an entire code region where an object is manipulated and execute this code as an atomic unit
- For this, you have to do **Block Synchronization**
- Synchronized keyword takes as a parameter an object that the system needs to obtain lock for, before it continues

Activity Trace 2 of ATMs: Is it Fixed Now?

```
synchronized (acc) {
    if (acc.getbal() + val > 0)
        acc.post(val);
    else throw new Exception();

    out.println("your balance is " + acc.getbal());
}
```

Time	Balance	Thread 1	Thread 2
			your account balance is 200
		your account balance is 200	
			Deposit or withdraw amount >
		Deposit or withdraw amount >	
			-150
		-150	
		val= -150	
			val=-150
		Synchronized(acc)	
			Synchronized(acc)
	200	acc.getbal()+val > 0 //yes	
	50	acc.post(-150)	
	50	your balance is 50	
			acc.getbal()+val > 0 //No
			Throw Exception()

Thread 2 starts the code inside the synchronized block only when Thread 1 is done

Balance in Thread 2 Already Showed 200 but Couldn't Withdraw!!

Carnegie Mellon University Qatar

How to Synchronize? – Even Bigger Synchronization Blocks

```java
while(true) {
    try {
        out.print("Account ID > ");
        String id = in.readLine();
        Account acc = bnk.get(id);
        if (acc == null) throw new Exception();

        out.print("Password > ");
        String pass = in.readLine();
        if (!acc.is_password(pass)) throw new Exception();
        out.println("your balance is " + acc.getbal());

        out.print("Deposit or withdraw amount > ");
        int val = Integer.parseInt(in.readLine());

        synchronized (acc) {
            if (acc.getbal() + val > 0)
                acc.post(val);
            else throw new Exception();

            out.println("your balance is " + acc.getbal());
        }

    } catch(Exception e) {
        out.println("Invalid input, restart");
    }
}
```

Let's Lock the account starting from when a transaction request is made response it sent to user

```java
while(true) {
    try {
        out.print("Account ID > ");
        String id = in.readLine();
        Account acc = bnk.get(id);
        if (acc == null) throw new Exception();

        out.print("Password > ");
        String pass = in.readLine();
        if (!acc.is_password(pass)) throw new Exception();

        synchronized (acc) {
        out.println("your balance is " + acc.getbal());

        out.print("Deposit or withdraw amount > ");
        int val = Integer.parseInt(in.readLine());

            if (acc.getbal() + val > 0)
                acc.post(val);
            else throw new Exception();

            out.println("your balance is " + acc.getbal());
        }

    } catch(Exception e) {
        out.println("Invalid input, restart");
    }
}
```

Activity Trace 2 of ATMs: Is it Fixed Now?

```
synchronized (acc) {
out.println("your balance is " + acc.getbal());

out.print("Deposit or withdraw amount > ");
int val = Integer.parseInt(in.readLine());

    if (acc.getbal() + val > 0)
      acc.post(val);
    else throw new Exception();

    out.println("your balance is " + acc.getbal());
}
```

Time

Thread 1

Account ID > Hend

Password > 1234

synchronized(acc)

out.println("your balance is " + acc.getbal());

your balance is 200

Deposit or withdraw amount >

Thread 2

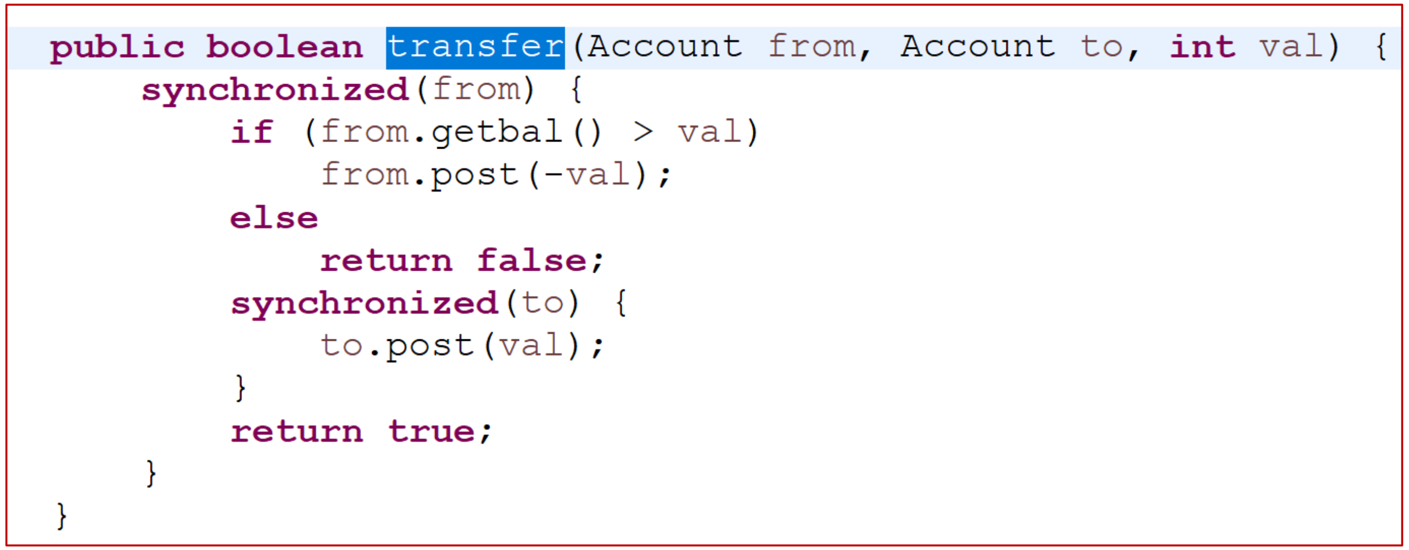
Account ID > Hend

Password > 1234

synchronized(acc)

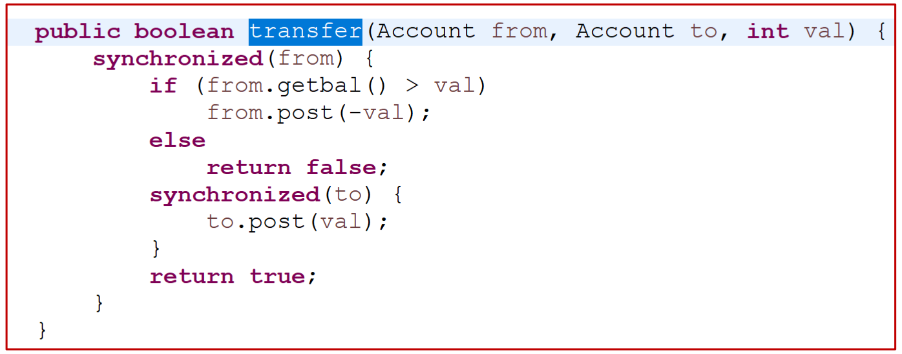
NO RESPONSE!!!

Concurrency Issues - Account Transfer Example



```
public boolean transfer(Account from, Account to, int val) {
    synchronized(from) {
        if (from.getbal() > val)
            from.post(-val);
        else
            return false;
        synchronized(to) {
            to.post(val);
        }
        return true;
    }
}
```

Account Transfer- Execution Trace



```
public boolean transfer(Account from, Account to, int val) {
    synchronized(from) {
        if (from.getbal() > val)
            from.post(-val);
        else
            return false;
        synchronized(to) {
            to.post(val);
        }
        return true;
    }
}
```

Time

Sana -> Abdalla

```
synchronized(from) {
if (from.getbal() > val)
from.post(-val);

synchronized(to)
```

Abdalla -> Sana

```
synchronized(from) {
if (from.getbal() > val)

from.post(-val);

synchronized(to)
```

DEADLOCKED!!!!

How to fix?

Sana wants to transfer 10 riyals to Abdalla
Abdalla wants to transfer 20 riyals to Sana
Will our code always work?

Avoiding deadlocks

- Cycle in locking graph = deadlock
- Standard solution: canonical order for locks
  - Acquire in increasing order
  - Release in decreasing order
- Ensures deadlock-freedom, but not always easy to do

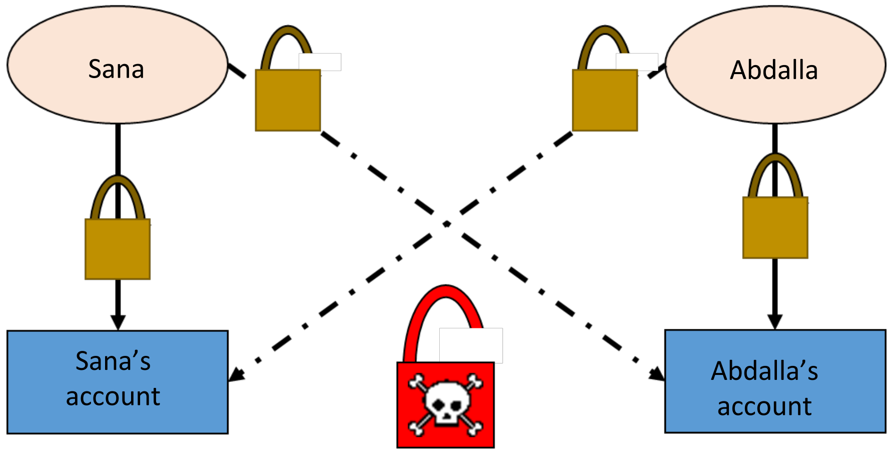
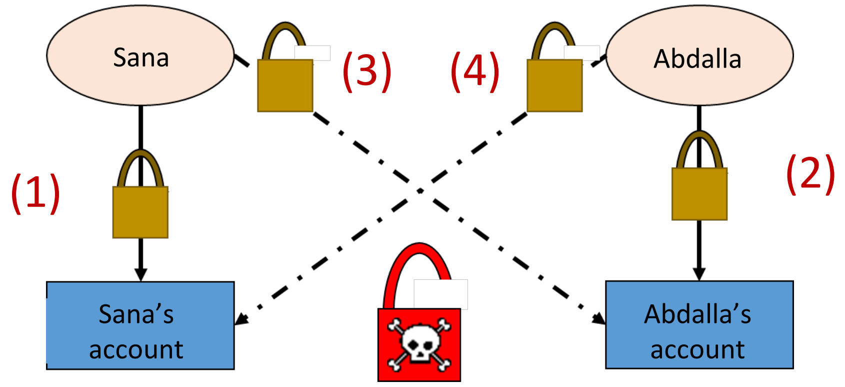


Photo credit: https://www.sqlshack.com/what-is-a-sql-server-deadlock/

Avoiding deadlocks through ranking– Account Transfer Example

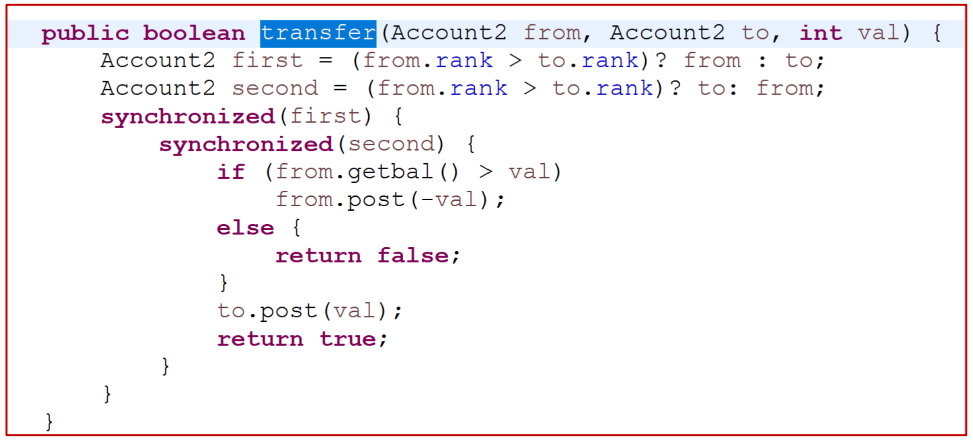


```
public boolean transfer(Account from, Account to, int val) {
    synchronized(from) {
        if (from.getbal() > val)
            from.post(-val);
        else
            return false;
        synchronized(to) {
            to.post(val);
        }
        return true;
    }
}
```

Let's Apply Ranking

```
public boolean transfer(Account2 from, Account2 to, int val) {
    Account2 first = (from.rank > to.rank)? from : to;
    Account2 second = (from.rank > to.rank)? to: from;
    synchronized(first) {
        synchronized(second) {
            if (from.getbal() > val)
                from.post(-val);
            else {
                return false;
            }
            to.post(val);
            return true;
        }
    }
}
```

Account Transfer-Execution Trace – Is it Fixed



```
public boolean transfer(Account2 from, Account2 to, int val) {
    Account2 first = (from.rank > to.rank)? from : to;
    Account2 second = (from.rank > to.rank)? to: from;
    synchronized(first) {
        synchronized(second) {
            if (from.getbal() > val)
                from.post(-val);
            else {
                return false;
            }
            to.post(val);
            return true;
        }
    }
}
```

Time

Sana -> Abdalla

synchronized(SanaAccount)

synchronized(AbdallAccount)

if (SanaAccount.getbal() > val)

SanaAccount.post(-val)

AbdallaAccount.post(val)

Abdalla -> Sana

Synchronized(SanaAccount)

synchronized(AbdallaAccount)
if (AbdallaAccount.getbal() > val)
AbdallaAccount.post(-val)
SanaAccount.post(val)

Suppose Sana's account has higher rank

Sana wants to transfer 10 riyals to Abdalla
Abdalla wants to transfer 20 riyals to Sana

Outline

- Concurrent Programming Introduction
  - Defining Concurrency?
  - Concurrency versus parallelism
  - Why Concurrency?
  - Concurrency in Java
- Ensuring Safety in Concurrent Programs
  - Synchronization
  - Bank Use Case Example: using abstract shared memory
- **More on Concurrency**

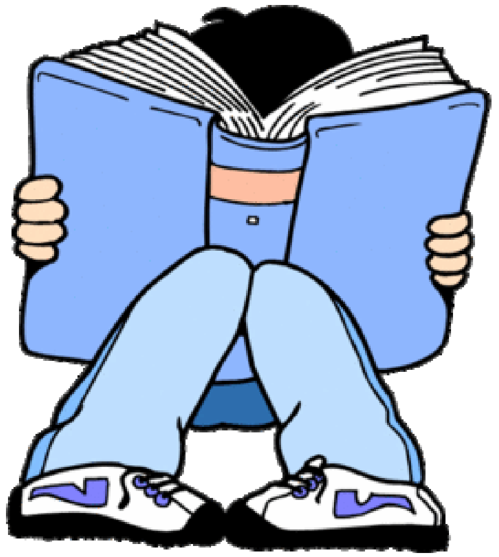
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Potential Concurrency Problems

- **Deadlock**
  - Two or more threads stop and wait for each other
- **Livelock**
  - Two or more threads continue to execute, but make no progress toward the ultimate goal.
- **Starvation**
  - Some thread gets deferred forever.
- **Lack of fairness**
  - Each thread gets a turn to make progress.
- **Race Condition**
  - Some possible interleaving of threads results in an undesired computation result

More on Concurrency in Java

- Semaphores
- Blocking & non-blocking queues
- Concurrent hash maps
- Copy-on-write arrays
- Exchangers
- Barriers
- Futures
- Thread pool support



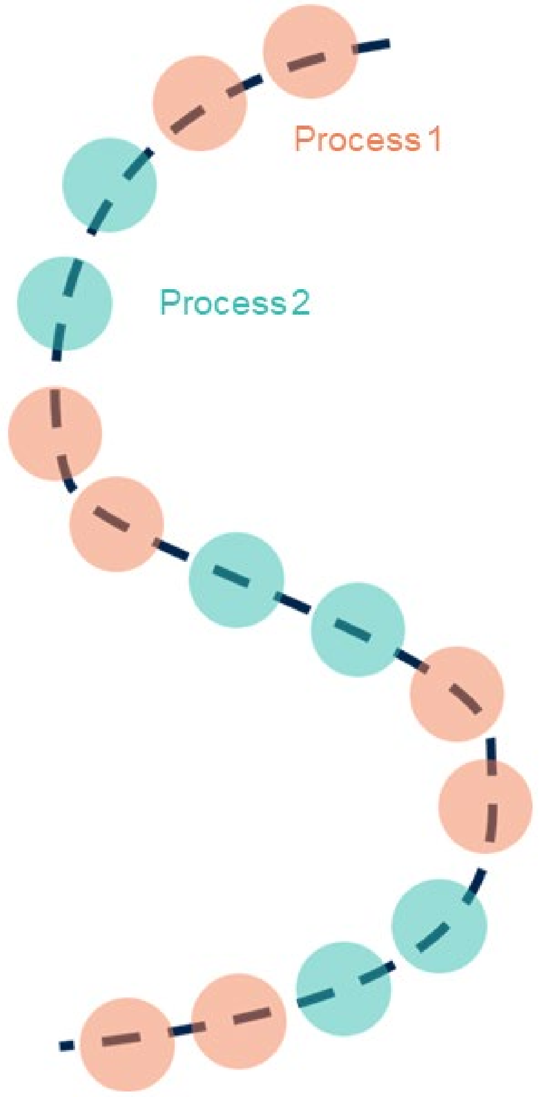
Check the
Java.util.concurrent
Java Package!

Interesting Ongoing Research on Concurrency

- Automatic parallelizers (e.g. Parsynt)
- Verification of concurrent programs (e.g. Duet)
- Concurrent program testing (e.g. Penelope)
- PL approached to deadlock freedom

Recap

- Concurrency and Parallelism are important concepts in Computer Science
- It can be very hard to understand and debug concurrent programs
- Parallelism is critical for high performance
  - From Supercomputers in national labs to Multicores and GPUs on your desktop
- Concurrency is the basis for writing parallel programs
- Next Recitation: Project 2



Credits

- The bank use case code and some slides are taken from 6.189 IAP 2007 MIT concurrent programming lecture