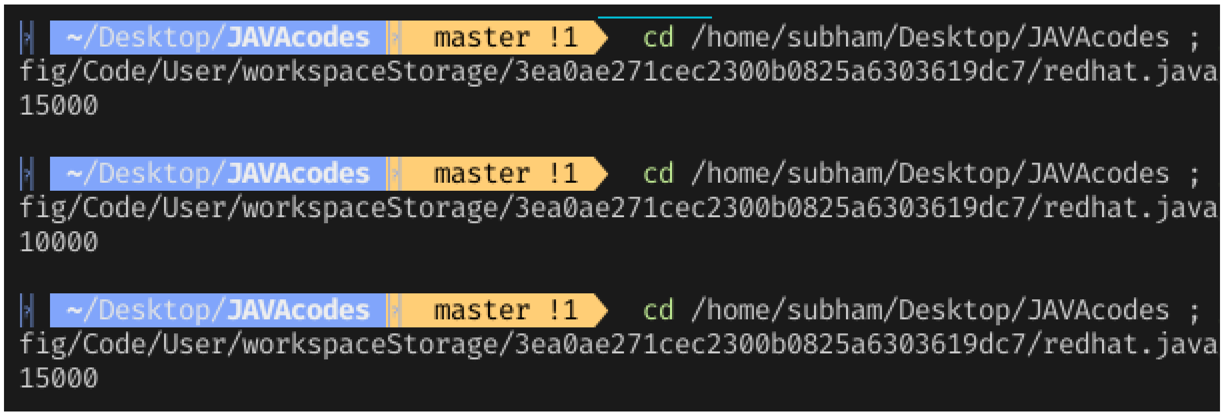
JAVA PROGRAMMING EXERCISE - APRIL 15th

1. Unsynchronized Threads

Code:

```
public class ThreadUnsynchronized {
   public static void main(String[] args) throws Throwable {

       // UNSYNCHRONIZED THREADS
       account s = new account(20000);
       Thread thr1 = new Thread(new Runnable(){
           @Override
           public void run() {
               for(int i=0;i<50;i++) {
                   s.withdraw(100);
               }
           }
       });
       Thread thr2 = new Thread(new Runnable(){
           @Override
           public void run() {
               for(int i=0;i<50;i++) {
                   s.withdraw(100);
               }
           }
       });
       thr1.start();
       thr2.start();
       thr1.join();
       thr2.join();
       System.out.println(s.balance);
   }
}

class account {
   public int balance;

   public account(int deposit) {
       this.balance = deposit;
   }

   public void withdraw(int withdraw_amount) {
       this.balance = this.balance - withdraw_amount;
   }
}
```

Output:



```
 ~/Desktop/JAVAcodes  master !1  cd /home/subham/Desktop/JAVAcodes ;
fig/Code/User/workspaceStorage/3ea0ae271cec2300b0825a6303619dc7/redhat.java
15000

 ~/Desktop/JAVAcodes  master !1  cd /home/subham/Desktop/JAVAcodes ;
fig/Code/User/workspaceStorage/3ea0ae271cec2300b0825a6303619dc7/redhat.java
10000

 ~/Desktop/JAVAcodes  master !1  cd /home/subham/Desktop/JAVAcodes ;
fig/Code/User/workspaceStorage/3ea0ae271cec2300b0825a6303619dc7/redhat.java
15000
```

Since the threads are unsynchronized, the output of the account balance is different every time we run the output, and even the final balance is wrong sometimes.

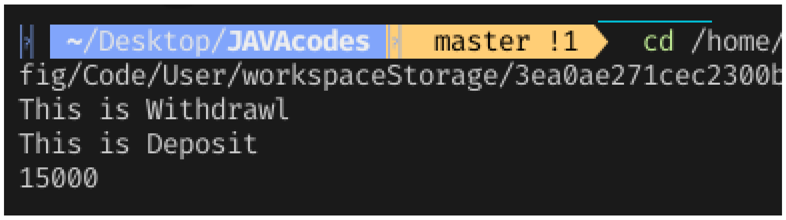
2. Synchronized threads - Using synchronized functions:

Code:

```java
public class ThreadSynchronized {
   public static void main(String[] args) throws Throwable {

       // SYNCHRONIZED THREADS
       bankAccount s = new bankAccount(20000);
       Thread thr1 = new Thread(new Runnable(){
           @Override
           public void run() {
               s.withdraw(10000);
           }
       });
       Thread thr2 = new Thread(new Runnable(){
           @Override
           public void run() {
               s.deposit(5000);
           }
       });
       thr1.start();
       thr2.start();
       thr1.join();
       thr2.join();
       System.out.println(s.balance);
   }
}

class bankAccount {
   public int balance;

   public bankAccount(int deposit) {
       this.balance = deposit;
   }

   public synchronized void withdraw(int withdraw_amount) {
       System.out.println("This is Withdrawl");
       this.balance = this.balance - withdraw_amount;
   }
```

```java
    public synchronized void deposit(int deposit_amount) {
        System.out.println("This is Deposit");
        this.balance = this.balance + deposit_amount;
    }
}
```

<u>Output:</u>



```
 ~/Desktop/JAVAcodes   master !1    cd /home/
fig/Code/User/workspaceStorage/3ea0ae271cec2300b
This is Withdrawl
This is Deposit
15000
```

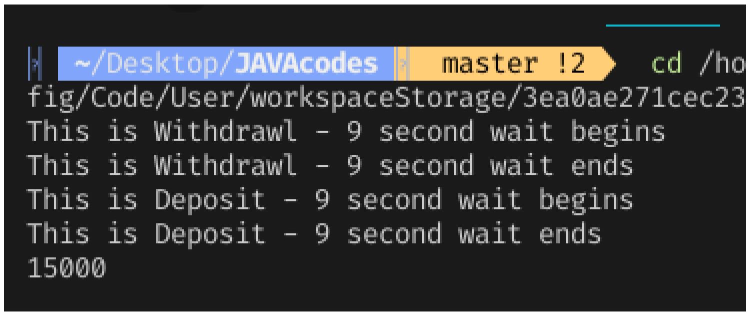
The code on the threads is now synchronized. The withdrawal runs first then the deposit and finally we get the correct final account balance.

3. Synchronized Threads - Using sleep() to simulate time taken to run the withdraw and deposit function:

Code:

```java
public class ThreadSynchronizedWithSleep {
   public static void main(String[] args) throws Throwable {

       // SYNCHRONIZED THREADS
       bankAccount1 s = new bankAccount1(20000);
       Thread thr1 = new Thread(new Runnable() {
           @Override
           public void run() {
               try {
                   s.withdraw(10000);
               } catch (Throwable e) {
                   e.printStackTrace();
               }
           }
       });
       Thread thr2 = new Thread(new Runnable() {
           @Override
           public void run() {
               try {
                   s.deposit(5000);
               } catch (Throwable e) {
                   e.printStackTrace();
               }
           }
       });
       thr1.start();
       thr2.start();
       thr1.join();
       thr2.join();
       System.out.println(s.balance);
   }
}

class bankAccount1 {
   public int balance;
   public bankAccount1(int deposit) {
        this.balance = deposit;
   }

  public synchronized void withdraw(int withdraw_amount) throws Throwable
{
        System.out.println("This is Withdrawl - 9 second wait begins");
        Thread.currentThread().sleep(9000);
        System.out.println("This is Withdrawl - 9 second wait ends");
        this.balance = this.balance - withdraw_amount;
   }

   public synchronized void deposit(int deposit_amount) throws Throwable {
        System.out.println("This is Deposit - 9 second wait begins");
        Thread.currentThread().sleep(9000);
        System.out.println("This is Deposit - 9 second wait ends");
        this.balance = this.balance + deposit_amount;
   }
}
```

<u>Output:</u>



```
 ~/Desktop/JAVAcodes   master !2   cd /ho
fig/Code/User/workspaceStorage/3ea0ae271cec23
This is Withdrawl - 9 second wait begins
This is Withdrawl - 9 second wait ends
This is Deposit - 9 second wait begins
This is Deposit - 9 second wait ends
15000
```

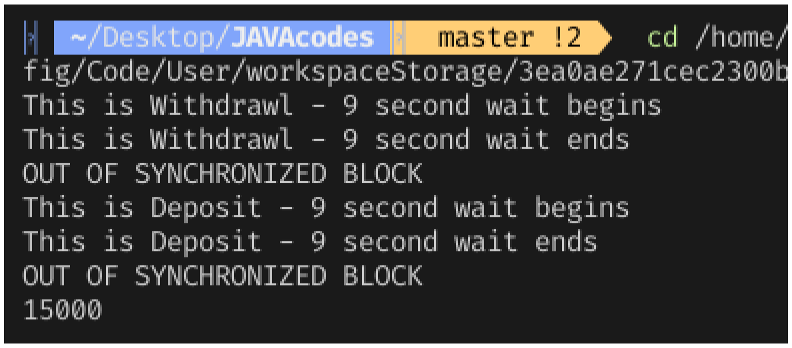
4. Synchronized Threads - Using sleep() together with synchronized blocks:

Code:

```
public class ThreadSynchronizedWithSleepAndSynchronizedBlocks {
   public static void main(String[] args) throws Throwable {

       // SYNCHRONIZED THREADS
       bankAccount1 s = new bankAccount1(20000);
       Thread thr1 = new Thread(new Runnable() {
           @Override
           public void run() {
               try {
                   s.withdraw(10000);
               } catch (Throwable e) {
                   e.printStackTrace();
               }
           }
       });
       Thread thr2 = new Thread(new Runnable() {
           @Override
           public void run() {
               try {
                   s.deposit(5000);
               } catch (Throwable e) {
                   e.printStackTrace();
               }
           }
       });
       thr1.start();
       thr2.start();
       thr1.join();
       thr2.join();
       System.out.println(s.balance);
   }
}

class bankAccount1 {
   public int balance;
```

```java
    public bankAccount1(int deposit) {
        this.balance = deposit;
    }

    public void withdraw(int withdraw_amount) throws Throwable {
        synchronized (this) {
            System.out.println("This is Withdrawl - 9 second wait begins");
            Thread.currentThread().sleep(9000);
            System.out.println("This is Withdrawl - 9 second wait ends");
            this.balance = this.balance - withdraw_amount;
        }
        System.out.println("OUT OF SYNCHRONIZED BLOCK");
    }

    public void deposit(int deposit_amount) throws Throwable {
        synchronized (this) {
            System.out.println("This is Deposit - 9 second wait begins");
            Thread.currentThread().sleep(9000);
            System.out.println("This is Deposit - 9 second wait ends");
            this.balance = this.balance + deposit_amount;

        }
        System.out.println("OUT OF SYNCHRONIZED BLOCK");
    }
}
```

<u>Output:</u>



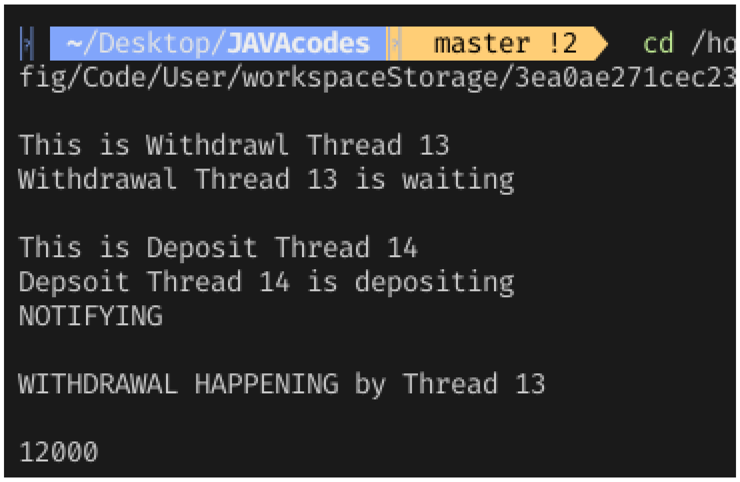
```
 ~/Desktop/JAVAcodes   master !2   cd /home/
fig/Code/User/workspaceStorage/3ea0ae271cec2300b
This is Withdrawl - 9 second wait begins
This is Withdrawl - 9 second wait ends
OUT OF SYNCHRONIZED BLOCK
This is Deposit - 9 second wait begins
This is Deposit - 9 second wait ends
OUT OF SYNCHRONIZED BLOCK
15000
```

5. Synchronized threads - Using wait() to make threads wait for a particular action, and notify() to notify one of the threads waiting:

Code:

```
public class ThreadWaitNotify {
   public static void main(String[] args) throws Throwable {
       Account subham = new Account(2000);
       Thread thr1=new Thread(new Runnable(){
           @Override
           public void run() {
               try {
                   subham.withdraw(30000);
               } catch (Throwable e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
           }
       });
       Thread thr2=new Thread(new Runnable(){
           @Override
           public void run() {
               try {
                   subham.deposit(40000);
               } catch (Throwable e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
           }
       });
       thr1.start();
       thr2.start();
       thr1.join();
       thr2.join();
       System.out.println(subham.balance);
   }
}

class Account {
   int balance;
   public Account(int balance) {
       this.balance = balance;
   }

   public synchronized void withdraw(int withdraw_amount) throws Throwable
{
       System.out.println();
       System.out.println("This is Withdrawl Thread " +
Thread.currentThread().getId());
       while (withdraw_amount > balance) {
           System.out.println("Withdrawal Thread " +
Thread.currentThread().getId() + " is waiting");
           wait();
       }
       System.out.println("WITHDRAWAL HAPPENING by Thread "+
Thread.currentThread().getId());
       System.out.println();
       this.balance = this.balance - withdraw_amount;
   }

   public synchronized void deposit(int deposit_amount) throws Throwable {
       System.out.println();
       System.out.println("This is Deposit Thread
"+Thread.currentThread().getId());
       System.out.println("Depsoit Thread " +
Thread.currentThread().getId() + " is depositing");
       System.out.println("NOTIFYING");
       System.out.println();
       this.balance = this.balance + deposit_amount;
       notify();
   }
}
```

Output:



```
  ~/Desktop/JAVAcodes    master !2    cd /ho
fig/Code/User/workspaceStorage/3ea0ae271cec23

This is Withdrawl Thread 13
Withdrawal Thread 13 is waiting

This is Deposit Thread 14
Depsoit Thread 14 is depositing
NOTIFYING

WITHDRAWAL HAPPENING by Thread 13

12000
```

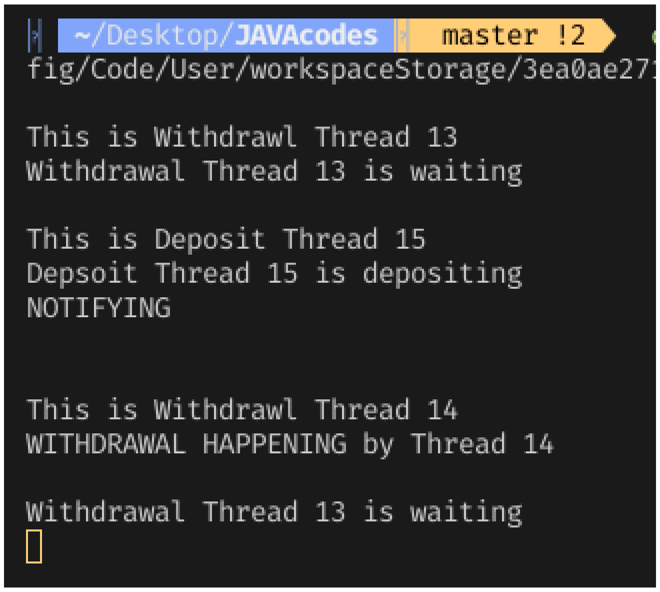
6. Synchronized threads - Using wait() to make threads wait for a particular action, and notifyAll() to notify all of the threads waiting:

Code:

```
public class ThreadWaitNotifyAll {
   public static void main(String[] args) throws Throwable {
       BankAccount subham = new BankAccount(2000);
       Thread thr1=new Thread(new Runnable(){
           @Override
           public void run() {
               try {
                   subham.withdraw(30000);
               } catch (Throwable e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
           }
       });
       Thread thr2=new Thread(new Runnable(){
           @Override
           public void run() {
               try {
                   subham.withdraw(40000);
               } catch (Throwable e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
           }
       });
       Thread thr3=new Thread(new Runnable(){
           @Override
           public void run() {
               try {
                   subham.deposit(40000);
               } catch (Throwable e) {
                   // TODO Auto-generated catch block
                   e.printStackTrace();
               }
           }
       });
       thr1.start();
       thr2.start();
       thr3.start();
       thr1.join();
       thr2.join();
       thr3.join();
       System.out.println(subham.balance);
   }
}

class BankAccount {
   int balance;

   public BankAccount(int balance) {
       this.balance = balance;
   }

   public synchronized void withdraw(int withdraw_amount) throws Throwable
{
       System.out.println();
       System.out.println("This is Withdrawl Thread " +
Thread.currentThread().getId());
       while (withdraw_amount > balance) {
           System.out.println("Withdrawal Thread " +
Thread.currentThread().getId() + " is waiting");
           wait();
       }
       System.out.println("WITHDRAWAL HAPPENING by Thread "+
Thread.currentThread().getId());
       System.out.println();
       this.balance = this.balance - withdraw_amount;
   }

   public synchronized void deposit(int deposit_amount) throws Throwable {
       System.out.println();
       System.out.println("This is Deposit Thread
"+Thread.currentThread().getId());
```

```
        System.out.println("Depsoit Thread " +
Thread.currentThread().getId() + " is depositing");
        System.out.println("NOTIFYING");
        System.out.println();
        this.balance = this.balance + deposit_amount;
        notifyAll();
    }
}
```

Output:



```
 ~/Desktop/JAVAcodes   master !2
fig/Code/User/workspaceStorage/3ea0ae27

This is Withdrawl Thread 13
Withdrawal Thread 13 is waiting

This is Deposit Thread 15
Depsoit Thread 15 is depositing
NOTIFYING


This is Withdrawl Thread 14
WITHDRAWAL HAPPENING by Thread 14

Withdrawal Thread 13 is waiting
```

In this case the Withdrawal thread 13 is waiting since there is not enough balance to withdraw and it will keep waiting until the user deposits money and notifies all the waiting withdrawing threads and there is enough money to withdraw the given amount.