RACE CONDITION

- A race condition is a situation in which two or more threads or processes are reading or writing some shared data, and the final result depends on the timing of how the threads are scheduled. Race conditions can lead to unpredictable results and subtle program bugs. A thread can prevent this from happening by locking an object. When an object is locked by one thread and another thread tries to call a synchronized method on the same object, the second thread will block until the object is unlocked.

SUJATABATRA

Synchronizing Code

- Threads can share resources.
- Sometimes it is desirable that only one thread at a time has access to a shared resource.
- Achieved by using the keyword: `synchronized`, Only *methods* can be synchronized, not variables.
- Protects access to code, not to data
  - Make data members private
  - Synchronize accessor methods
- A lock can be associated with a shared resource. Threads gain access to a shared resource by first acquiring the lock associated with the resource.
- one thread can hold the lock and thereby have access to the shared resource.

Synchronizing Code

- A class can have both synchronized and no synchronized methods.
- If a class has both synchronized and nonsynchronized methods, *multiple threads can still access the nonsynchronized methods* of the class!
- *A thread can acquire more than one lock.*
- A thread can enter a synchronized method, thus acquiring a lock, and then immediately invoke a synchronized method on a different object, thus acquiring *that* lock as well.
- As the stack unwinds, locks are released again.

Thread Synchronization

```
class TwoStrings {

synchronized static void print(String str1, String str2) {

System.out.print(str1);

try {
Thread.sleep(500);
} catch (InterruptedException ie)
{
ie.printStackTrace();
}
System.out.println(str2);
}
}
```

Thread Synchronization

```
class PrintStringsThread implements Runnable {

String str1, str2;

PrintStringsThread(String str1, String str2) {

    this.str1 = str1;
    this.str2 = str2;

 Thread thread = new Thread(this);
 thread.start();
}

public void run() {
 TwoStrings.print(str1, str2);

 }
}
```

SUJATA BATRA

Thread Synchronization

```
public class Test {

public static void main(String args[])
{
new PrintStringsThread("Hello ", "there.");
 new PrintStringsThread("How are ", "you?");
 new PrintStringsThread("Thank you ","very much!");
 new PrintStringsThread("Bye ","Bye!");

}
}
```

Output- May be in Different Order

Hello there

How are you

Thank you very much

Bye,Bye

Volatile

- Volatile keyword is used to modify the value of a variable by different threads.
- Used to make classes thread safe.
- The volatile keyword does not cache the value of the variable and always read the variable from the main memory.
- Use the volatile keyword with variables. Using volatile keyword with classes and methods is illegal.