1 Threading (abstractly)

- Allows simultaneous execution of pieces of code within one process, each of which is a thread
- Provides behavior similar to that of multiple processes executing simultaneously, but since threads are within one process, they share state (variables, etc...)
- Issues can arise with synchronization of shared memory

2 Threads (Java)

- So far, we have dealt with only one thread – the main thread (the one started when the program is run)
- Ordinary classes can be made to execute as threads
- The interface Runnable represents a class that can represent a thread – it has one method, public void run()
- The class Thread represents a class that implements Runnable but is a generic thread (it doesn’t have a specified behavior)
- Thread contains a method public void start() that creates a new thread of execution and executes the run() method within that new thread
- A call to the method join() on a thread variable will cause the current thread to wait until the other thread has finished executing before continuing

Example

class UI extends Thread {
    public void run() {
        // give the user progress updates
    }
}

class Indexer {
    public static void main(String[] args) {
        Thread ui = new UI();
        ui.start();
        // index volumes of information
    }
}

3 Synchronizing code

- Java provides the keyword synchronized that allows locking of critical regions of code
• Methods may have the `synchronized` keyword, indicating that only one thread may be executing any synchronized methods within that object instance at a time

• Each object may only be locked by one thread, so upon execution of a synchronized piece of code, the thread tries to `acquire` the lock (if the object is locked by another thread, this thread waits), and after the thread finishes executing a synchronized piece of code from that object, it `releases` the lock

Example

```java
class T1 {
    static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    static Ball b = new Ball();

    public static void main(String[] args) throws Exception {
        Thread t1 = new SynchHolder("A");
        Thread t2 = new SynchHolder("B");
        t1.start();
        t2.start();
        t1.join();
        t2.join();
    }

    static class SynchHolder extends Thread {
        String name;

        SynchHolder(String name) {
            this.name = name;
        }

        public void run() {
            while (true) {
                b.hold(name);
            }
        }
    }

    static class Ball {
        public synchronized void hold(String s) {
            try {
                System.out.println(s + " is holding the ball");
                br.readLine();
            } catch (Exception e) {} 
        }
    }
}
```

4 Other synchronization

• There are other methods of synchronization, which we will discuss later

• Synchronization in java is far more convenient than in many other languages
To a great extent, handling of synchronization problems causes much of the complexity in operating systems, which have to deal with management of shared resources accessed simultaneously.