Introduction

Multiple threads may require access to the same object, which if not implemented correctly, may produce undesirable results. The behavior that results in this outcome is usually the result of thread synchronization not being implemented. Thread synchronization is the concept of allowing an object to be accessed exclusively by one thread at a time. In Java, this is commonly accomplished by using “Java’s built-in monitors” (Deitel and Deitel, 2011, location 27821). An object’s monitor and monitor lock have the effect of permitting access to one thread at a time. Requiring threads to possess the lock before gaining access does this. When the operation is complete, then the lock is released, and other threads may have access.

Example of thread synchronization

An obvious example in my everyday life that requires thread synchronization is my bank account. I often will schedule bills to be paid by directly debiting from my checking account, and I also have my employer directly deposit my salary. Added to this mix of transactions are my automatic mortgage and car payments once a month, deposits I make from my rental properties, withdrawing cash through the ATM, and interest accrued. If my banking institution were not implementing thread synchronization, then there would be a high likelihood of my account balance being inaccurate.

Thread synchronization in this instance can be accomplished in Java by creating a bank account class containing synchronized methods for withdrawing and depositing funds:

public class BankAccount {

float accountBalance;

public synchronized void addFunds(float value) {

/\* try-catch statement, using the estimated time needed by other operations

\*/

try {

Thread.sleep(numberOfMilliseconds);

} catch (InterruptException e) {

}

// add value to accountBalance

accountBalance += value;

}

public synchronized void withdrawFunds(float value) {

/\* try-catch statement, using the estimated time needed by other operations

\*/

try {

Thread.sleep(numberOfMilliseconds);

} catch (InterruptException e) {

}

// subtract value from accountBalance

accountBalance -= value;

}

}

By adding the reserved word “synchronized” to the method declarations above, one ensures that these methods can be called by one thread at a time. The try-catch statements would include Thread.sleep statements which take as argument the number of milliseconds we would want it to wait (the time estimated for other threads to finish accessing the synchronized method).

Conclusion

Thread synchronization plays an important role when multiple processes run in parallel, particularly when they must access the same objects. Java provides tools that are more than capable of handling tasks that require this feature.

References:

Deitel, P. & Deitel H. (2011) Java How to Program 9th Edition p. 572. Pearson HE, Inc.. [Kindle Edition].