

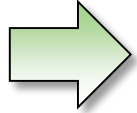
Threading Pattern Library

Donald Belcham
www.igloocoder.com
@dbelcham



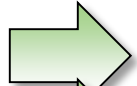
pluralsight 
hardcore dev and IT training

Thread Unsafe



Ability to set thread access policy on code constructs

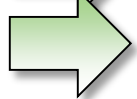
- Instance
- Static
- Thread Affine



Throws a runtime error

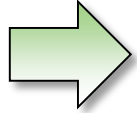


All fields must be private or protected



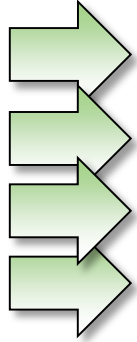
static policy:

- Static methods can access fields
- Static methods can invoke private methods



***[ThreadUnsafe]* applied at the class level**

Actor



Message queue based access to class instances

Queue processing happens in a single thread

Calls to instance are handled asynchronously and in order

Classes will inherit the Actor class

```
class Player
{
    readonly string name;
    readonly Random random = new Random();
    int totalBallsReceived;
    readonly double skills;

    public Player(string name, double skills )
    {
        this.name = name;
        this.skills = skills;
    }

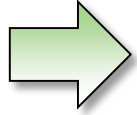
    public Player Ping(Player peer)
    {
        if (random.NextDouble() <= this.skills)
        {
            this.totalBallsReceived++;
            return peer.Ping(this);
        }
        else
        {
            return peer;
        }
    }
}
```

```
class Player : Actor
{
    readonly string name;
    readonly Random random = new Random();
    int totalBallsReceived;
    readonly double skills;

    public Player(string name, double skills )
    {
        this.name = name;
        this.skills = skills;
    }

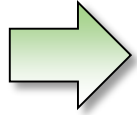
    public async Task<Player> Ping(Player peer)
    {
        if (random.NextDouble() <= this.skills)
        {
            this.totalBallsReceived++;
            return await peer.Ping(this);
        }
        else
        {
            return peer;
        }
    }
}
```

Reader/Writer Synchronized Object Model



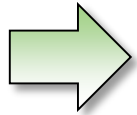
Principle of preventing changes while reading from an instance
[ReaderWriterSynchronized]

- Block writes while reading *[ReaderLock]*
- Block reads while writing *[WriterLock]*



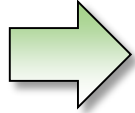
Eventing

- Blocks writes during event handling *[ObserverLock]*



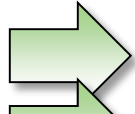
Can be manually applied using *IReaderWriterSynchronized*

Dispatching Method to Background and the UI

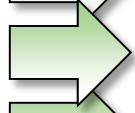


Use BackgroundWorkerThread

- RunWorkerAsync
- DoWork
- RunWorkerCompleted



Works in place of BackgroundWorker and asynchronous processing

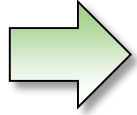


***[Background]* to execute in a background thread**



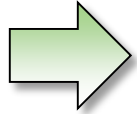
***[Dispatched]* to resynchronize with UI thread**

Waiving Verification



Some code will have

- Manual locks
- No need for thread specific code

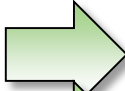

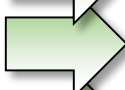
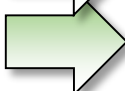


[ExplicitlySynchronized]

Runtime Deadlock Detection

- ➡ Prevents code hanging in deadlocks
- ➡ Triggered after an object waits 200ms
- ➡ Throws a *DeadlockException* in all threads involved in the cycle
- ➡ Applied to each assembly in the solution [*DeadlockDetectionPolicy*]

Summary

-  Multiple options to deal with thread isolation
-  You still need to know which to choose
-  The ability to exclude items from threading patterns
-  Deadlock detection