C# without nulls or exceptions

•••

@ReidNEvans

Senior Consultant - @ResultStack

http://tinyurl.com/ReidYouTube

RESULT STACK

Our path forward (there will be code)

1. Be declarative
2. Embrace purity
3. A new type
4. Build out our new type with user stories

“Even the newest .NET developers are likely familiar with the NullReferenceException. This is an exception that almost always indicates a bug **because the developer didn’t perform sufficient null checking** before invoking a member on a (null) object.”

microsoft.com

NO PAIN
NO GLORY
©2016FOX

Hot Potato Operator

```
static string Truncate(string value, int length)
{
  return value?.Substring(
    0,
    Math.Min(value.Length, length)
  );
}
```

Be declarative

```
IEnumerable<int> ___________(List<int> list)
{
    var result = new List<int>();

    for (var i = 0; i < list.Length; i++) {
        if (list[i] % 2 == 0) {
            result.Add(list[i] * 2);
        }
    }

    return result;
}
```

```
IEnumerable<int> DoubleEvens(List<int> list)
{
    var result = new List<int>();

    for (var i = 0; i < list.Length; i++) {
        if (list[i] % 2 == 0) {
            result.Add(list[i] * 2);
        }
    }

    return result;
}
```

```
IEnumerable<int> DoubleEvens(List<int> list)
{
    var result = new List<int>();

    for (var i = 0; i < list.Length; i++) {
        if (list[i] % 2 == 0) {
            result.Add(list[i] * 2);
        }
    }

    return result;
}
```

```
IEnumerable<int> DoubleEvens(List<int> list)
{
   var result = new List<int>();

   for (var i = 0; i < list.Length; i++) {
      if (list[i] % 2 == 0) {
         result.Add(list[i] * 2);
      }
   }

   return result;
}
```

```
IEnumerable<int> DoubleEvens(List<int> list)
{
    var result = new List<int>();

    for (var i = 0; i < list.Length; i++) {
        if (list[i] % 2 == 0) {
            result.Add(list[i] * 2);
        }
    }

    return result;
}
```

IT'S DANGEROUS TO GO ALONE! TAKE THIS.

```
IEnumerable<int> DoubleEvens(List<int> ints)
{
   return ints.Where(x => x % 2 == 0)
              .Select(x => x * 2);
}
```

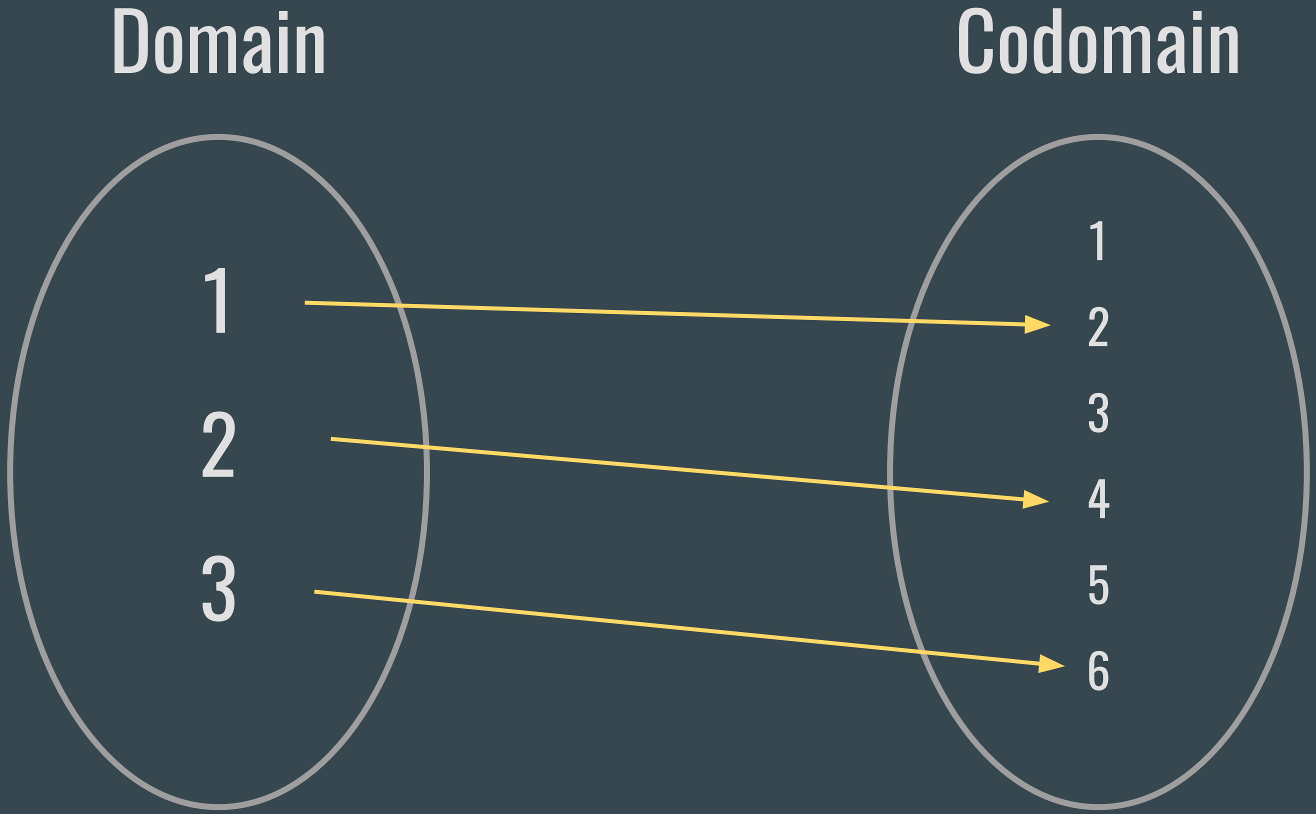
```
bool IsEven(int x) {
    return x % 2 == 0;
}

int Double(int x) {
    return x * 2;
}

IEnumerable<int> DoubleEvens(List<int> ints) {
    return ints.Where(IsEven)
               .Select(Double);
}
```

Embrace Purity

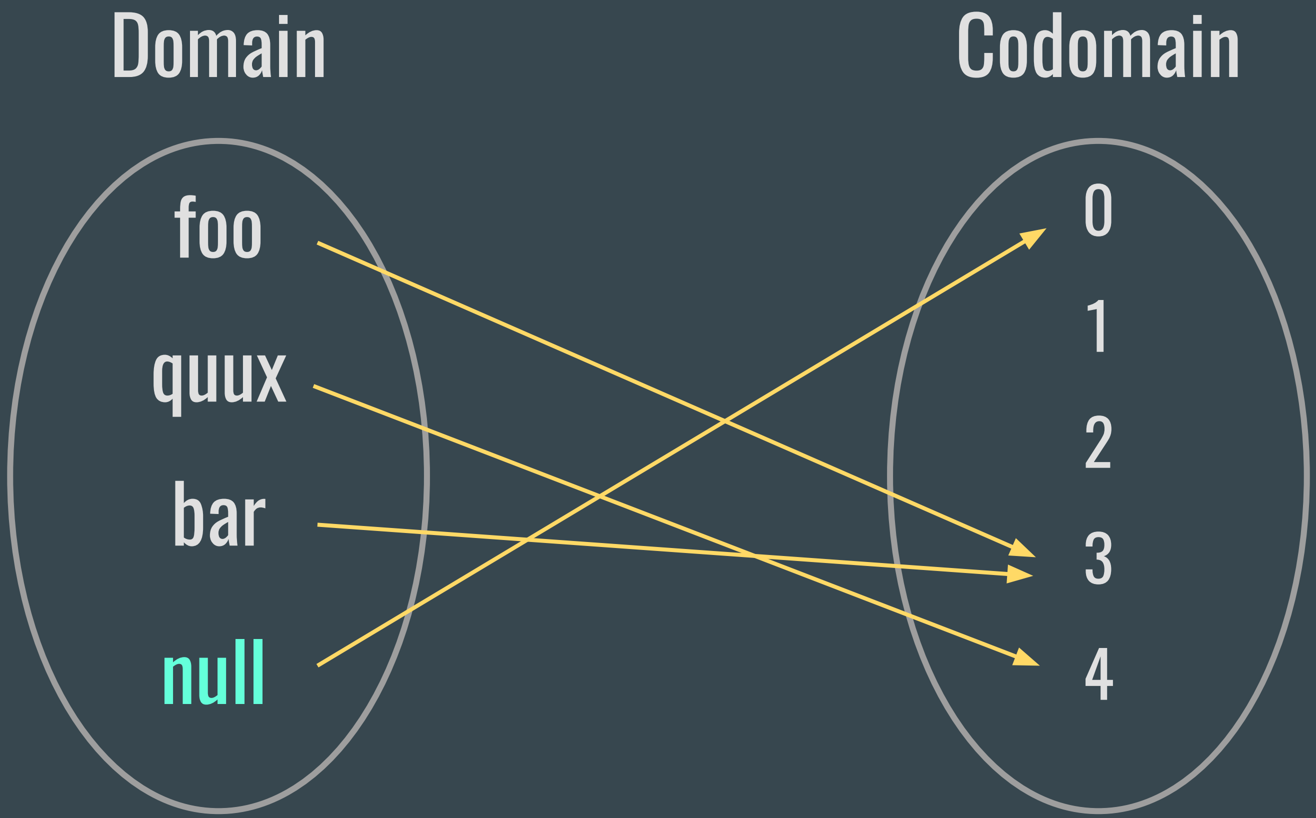
```
int Decrement(int i)

Decrement(1)
//0

Decrement(0)

//ArgumentException: Bob says we can’t decrement below
zero

Assert.Throws<ArgumentException>(() => Decrement(0);)
```

A pure function always supplies a result value for all possible input values

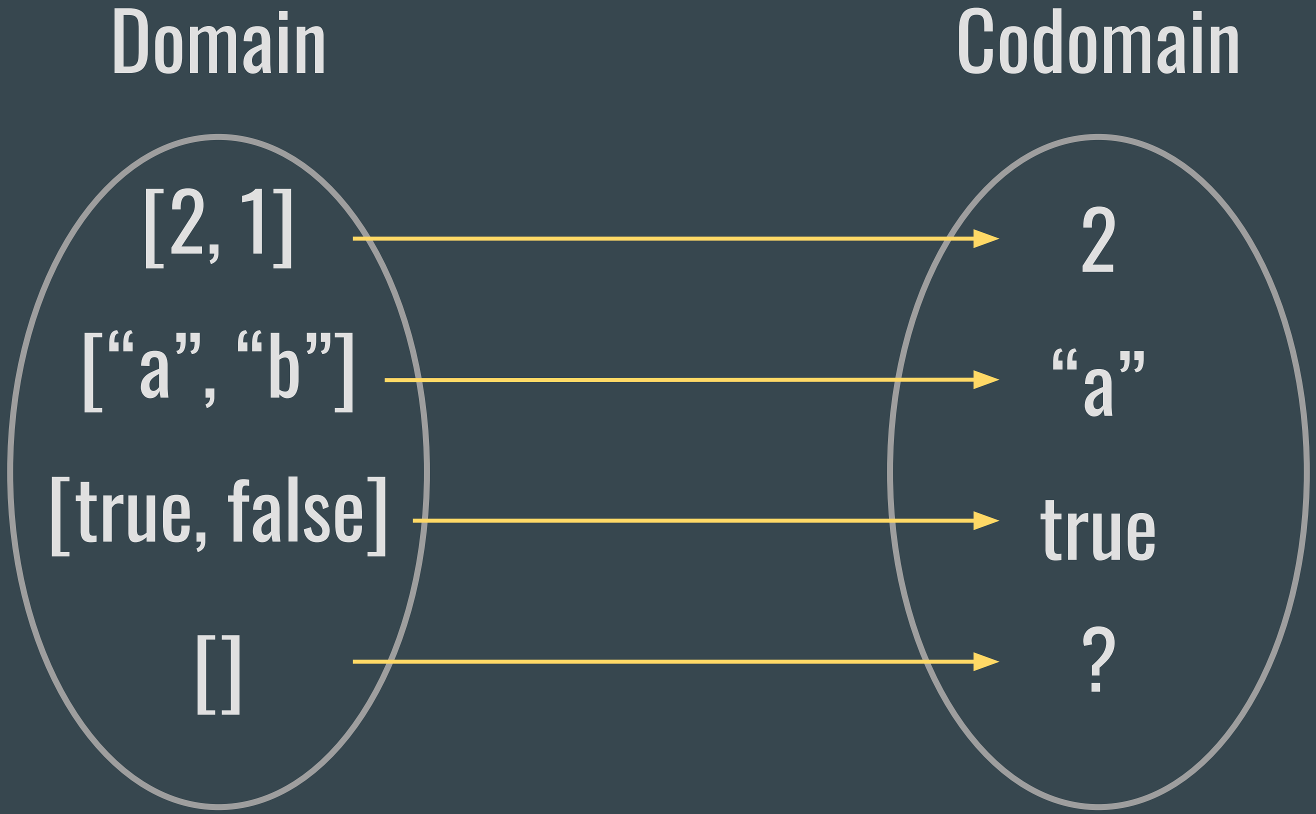
$$f(x) = 2x$$



```
public int Length(string s) { return s.Length; }
```



```
public static T FirstOrDefault(this IEnumerable<T> x)
```



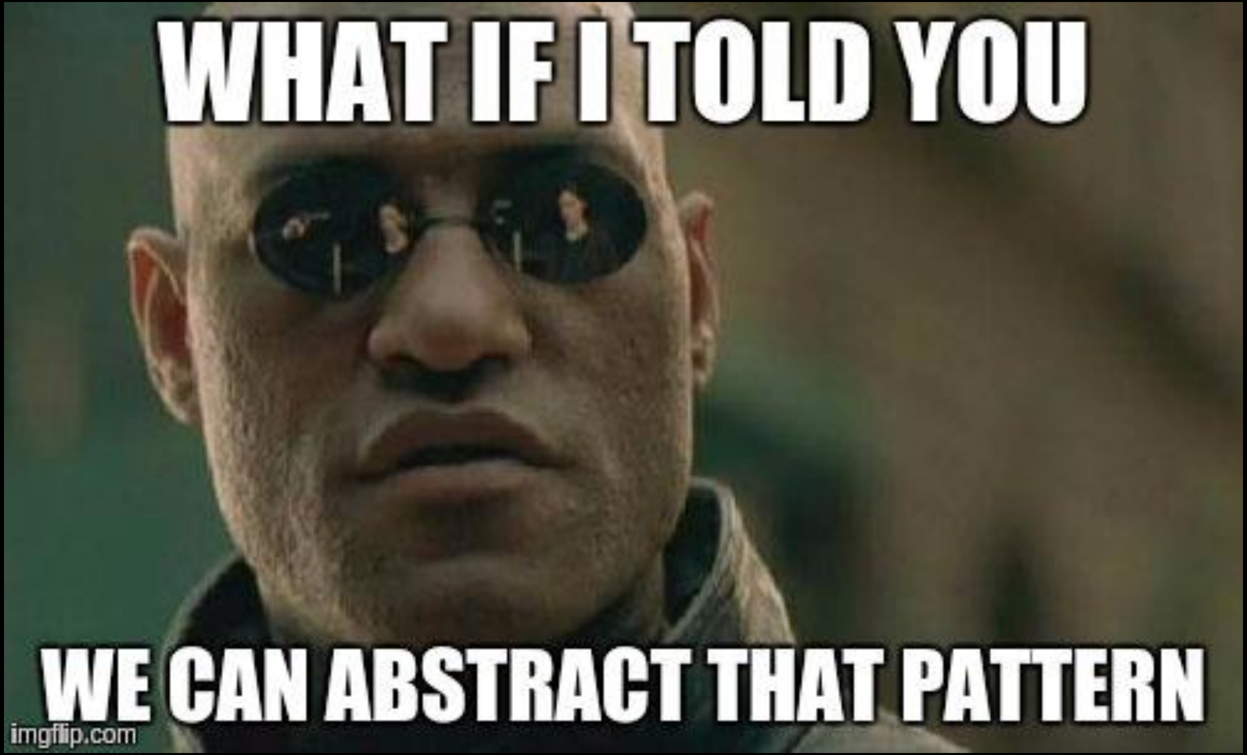
“I call it my billion-dollar mistake. It was the invention of the null reference in 1965.”

Tony Hoare

```
public static T FirstOrDefault(this IEnumerable<T> x)
```

```
var result = list.FirstOrDefault();
if (result != null) {
   //Do something
} else {
   //Do something else
}
```

Spidey Sense x2


WHAT IF I TOLD YOU
WE CAN ABSTRACT THAT PATTERN
imgflip.com

We can encode logic into our type system so that the compiler helps us write correct code.

Me + Compiler = ♥

```csharp
var result = list.FirstOrDefault();
if (result != null) {
    //Do something
} else {
    //Do something else
}
```

```
public interface IMaybe {  }

public class Some : IMaybe { }

public class None : IMaybe { }
public interface IMaybe<T> {  }


public class Some<T> : IMaybe<T>
{
    readonly T _obj;
    public Some(T obj)
    {
        _obj = obj;
    }
}

public class None<T> : IMaybe<T> { }
public static IMaybe<T> SafeFirst(this IEnumerable<T> x)
{
    var result = list.FirstOrDefault();
    return result == null
        ? new None<T>()
        : new Some(result);
}
```

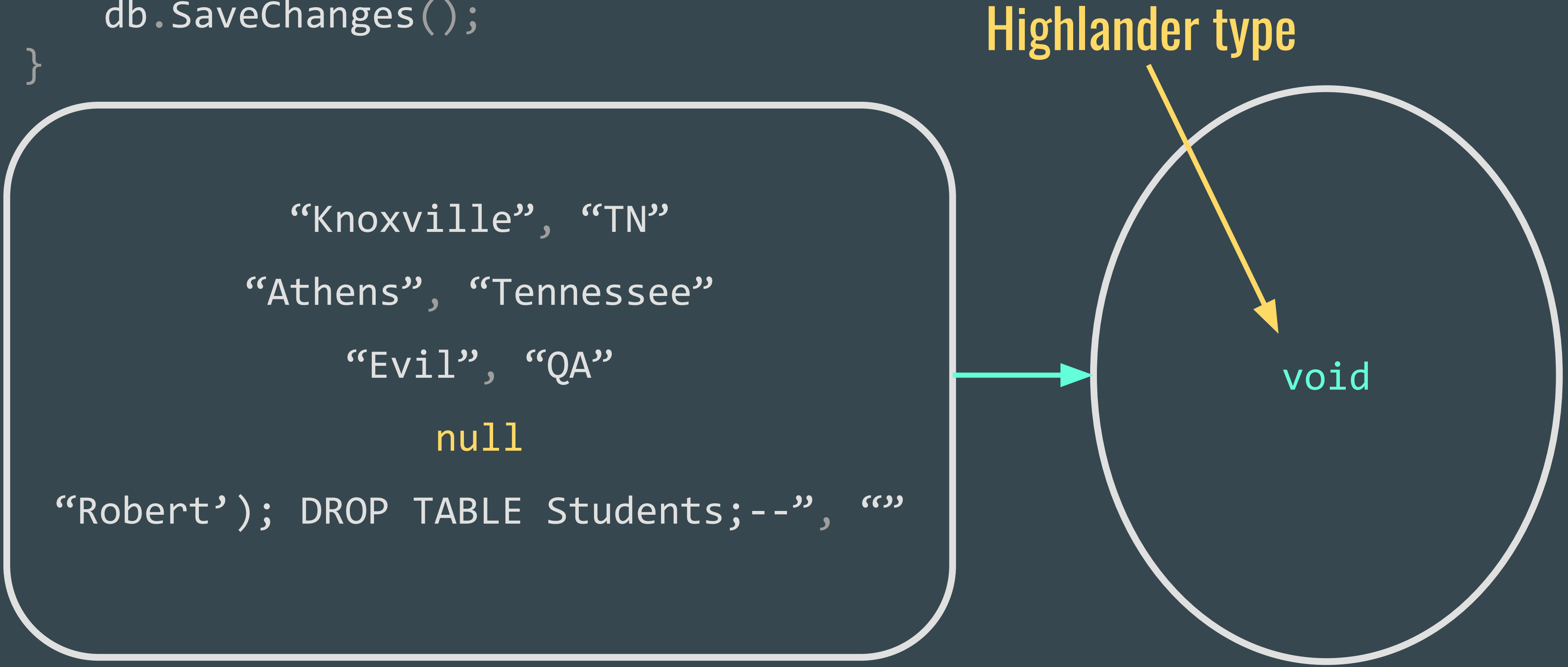
```
new List<string>().SafeFirst(); //None<string>

new List<string> { “foo” }.SafeFirst(); //Some<string> “foo”
```

Build out our new type with user stories

As a User I want to be able to add new locations

```csharp
class Location
{
    public string City { get; set; }
    public string State { get; set; }
}
```

```
public void InsertIntoDb(Location x)
{
    db.Locations.AddObject(x);
    db.SaveChanges();
}
```



As a DBA I’m sick of junk data in my database

```
class Location
{
    public string City { get; set; }
    public string State { get; set; }
}
```

Why not use the constructor?

```
class Location
{
    private Location() { }
    public string City { get; private set; }
    public string State { get; private set; }

    public static IMaybe<Location> Create(string city, string state)
    {
        return (string.IsNullOrWhiteSpace(city) ||
                string.IsNullOrWhiteSpace(state))
            ? new None<Location>()
            : new Some(new Location {
                City = city,
                State = state
            });
    }
}
```

Can still be null or empty

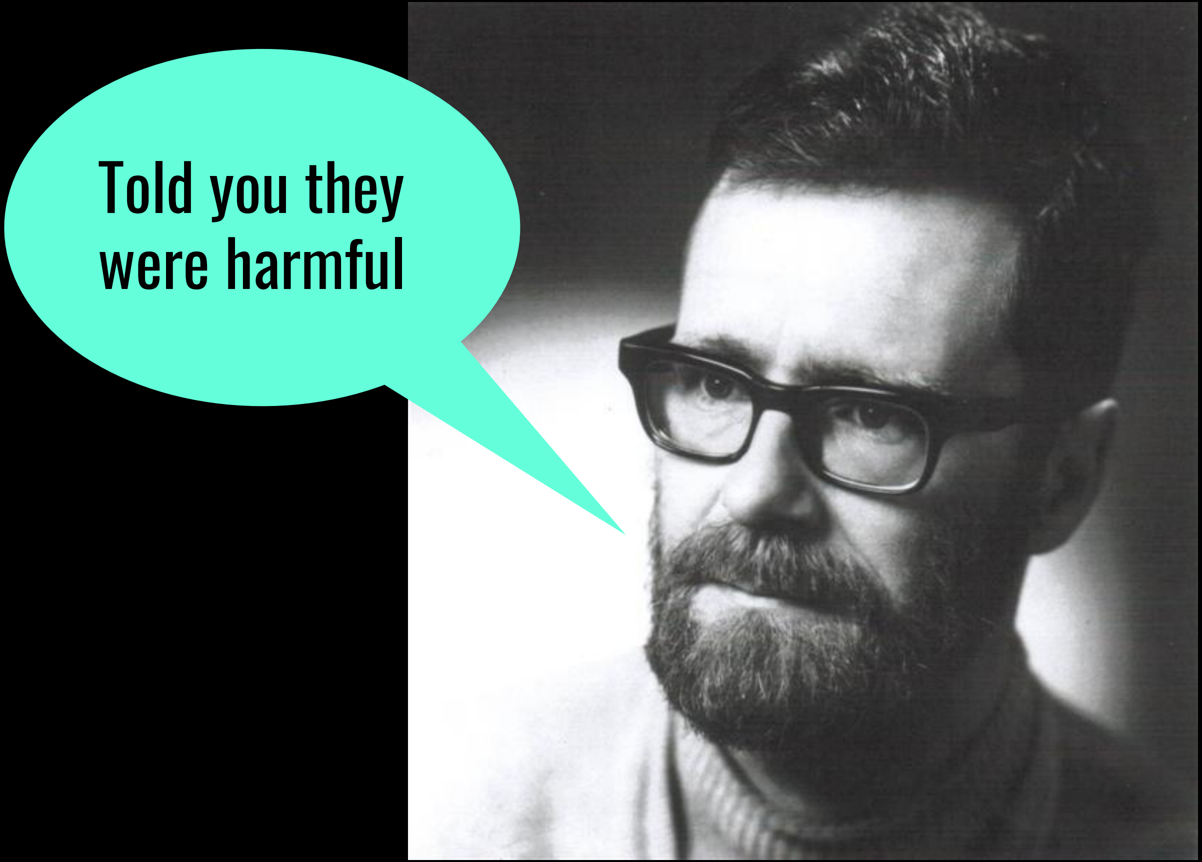
```
class Location
{
    public Location(string city, string state) {
        City = city;
        State = state;
    }
    public string City { get; private set; }
    public string State { get; private set; }
}
```

```
class Location
{
    public Location(string city, string state) {
        if (String.IsNullOrWhiteSpace(city) ||
            String.IsNullOrWhiteSpace(state))
            throw new Exception(“”);

        City = city;
        State = state;
    }
    public string City { get; private set; }
    public string State { get; private set; }
}
```

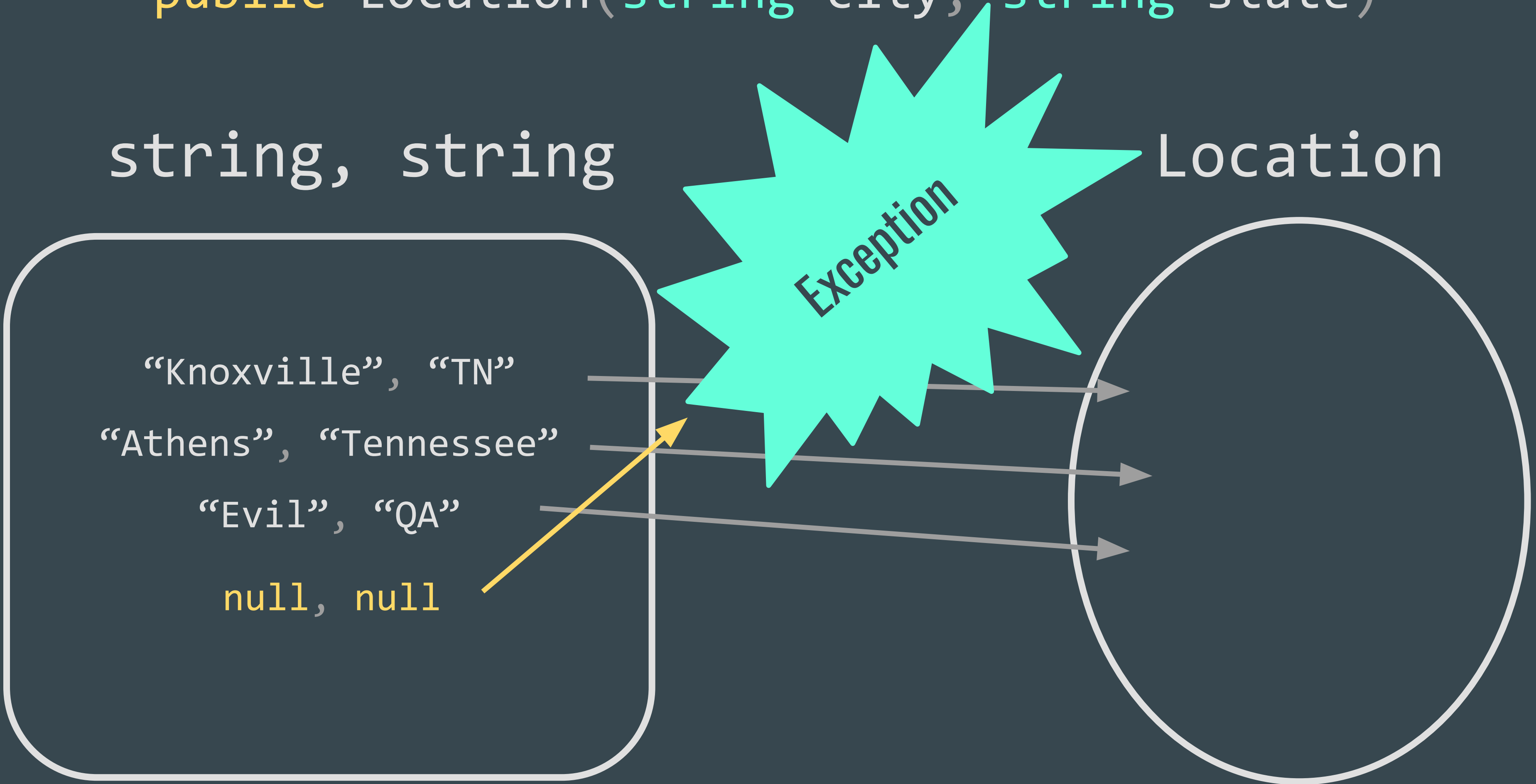
“An exception represents an immediate, nonlocal transfer of control - **It is a kind of cascading goto**”

The Pragmatic Programmer (127)

- Andrew Hunt
- Dave Thomas


Told you they were harmful

```
public Location(string city, string state)
```



```csharp
class Location
{
    private Location() { }
    public string City { get; private set; }
    public string State { get; private set; }

    public static IMaybe<Location> Create(string city, string state)
    {
        return (String.IsNullOrWhiteSpace(city) ||
                String.IsNullOrWhiteSpace(state))
            ? new None<Location>()
            : new Some(new Location {
                City = city,
                State = state
            });
    }
}
```

```
IMaybe<Location> location = Location.Create("Cincinnati", "OH");

public void InsertIntoDb(Location x)
{
    db.Locations.AddObject(x);
    db.SaveChanges();
}
```

```
IMaybe<Location> != Location
```

```
public interface IMaybe<T> {
    IMaybe<U> Select<U>(Func<T, U> mapper);
}

public class Some<T> : IMaybe<T> {
    readonly T _obj;
    public Some(T obj)
    {
        _obj = obj;
    }
    public IMaybe<U> Select<U>(Func<T, U> mapper) {
        U value = mapper(_obj);
        return new Some(value);
    }
}
```

```
public interface IMaybe<T> {
    IBool<U> Select<U>(Func<T, U> mapper);
}

public class Some<T> : IMaybe<T>
{
    readonly T _obj;
    public Some(T obj)
    {
        _obj = obj;
    }
    public IMaybe<U> Select<U>(Func<T, U> mapper) {
        U value = mapper(_obj);
        return new Some(value);
    }
}
```

```
public class None<T> : IMaybe<T>
{
    public IMaybe<U> Select<U>(Func<T, U> mapper) {

    }
}
```

```
public class None<T> : IMaybe<T>
{
    public IMaybe<U> Select<U>(Func<T, U> mapper) {
        return new None<U>();
    }
}
```

```
public void InsertIntoDb(Location x)
{
    db.Locations.AddObject(x);
    db.SaveChanges();
}

Location.Create("Knoxville", "TN") // Some Location
        .Select(InsertIntoDb);

Location.Create(null, "") // None
        .Select(InsertIntoDb);
```

No databases were harmed during this call

4GIFS.com

```
public class Some<T> : IMaybe<T>
{
    public IMaybe<U> Select<U>(Func<T, U> mapper) {
        U value = mapper(_obj);
        return new Some(value);
    }
}

public class None<T> : IMaybe<T>
{
    public IMaybe<U> Select<U>(Func<T, U> mapper) {
        return new None<U>();
    }
}
```

Mapper function never called

As a Business Owner I only want Tennessee locations

```
public bool StateIsTennessee(Location x)
{
    return x.State == “TN”;
}
```

```
public interface IMaybe<T> {
    IMaybe<T> Where(Func<T, bool> predicate);
}

public class Some<T> : IMaybe<T> {
    readonly T _obj;
    public Some(T obj) {
        _obj = obj;
    }

    public IMaybe<T> Where(Func<T, bool> predicate) {
        return predicate(_obj)
            ? this
            : new None<T>();
    }
}
```

```
public interface IMaybe<T> {
    IMaybe<T> Where(Func<T, bool> predicate);
}

public class Some<T> : IMaybe<T> {
    readonly T _obj;
    public Some(T obj) {
        _obj = obj;
    }

    public IMaybe<T> Where(Func<T, bool> predicate) {
        return predicate(_obj)
            ? this
            : new None<T>();
    }
}
```

```csharp
public class None<T> : IMaybe<T>
{
    ...
    public IMaybe<T> Where(Func<T, bool> predicate) {

    }
}
```

```
public class None<T> : IMaybe<T>
{
    ...
    public IMaybe<T> Where(Func<T, bool> predicate) {
        return this;
    }
}
```

```
public bool StateIsTennessee(Location x)
{
    return x.State == “TN”;
}

Location.Create(“Knoxville”, “TN”) // Some Location
    .Where(StateIsTennessee)       // Some Location
    .Select(InsertIntoDb);


Location.Create(“Cincinnati”, “OH”) // Some Location
    .Where(StateIsTennessee)          // None
    .Select(InsertIntoDb);
```

Still no databases harmed during this call

4GIFs
.com

```
void CreateLocation(string city, string state)
{
    if (String.IsNullOrWhiteSpace(city) ||
         String.IsNullOrWhitespace(state)) {
         throw new Exception(“city and state are required”);
     }
     if (state == “TN”)
     {
        db.Locations.AddObject(x);
        db.SaveChanges();
    }
}
```

```
public bool StateIsTennessee(Location x)
{
    return x.State == “TN”;
}

Location.Create(“Knoxville”, “TN”) // Some Location
    .Where(StateIsTennessee)       // Some Location
    .Select(InsertIntoDb);


Location.Create(“Cincinnati”, “OH”) // Some Location
    .Where(StateIsTennessee)          // None
    .Select(InsertIntoDb);
```

As Ops Support we want logging

```
void LogLocation(Location x) {
    Logger.info("about to insert " + x.City + ", " + x.State);
}

void LogNoLocation() {
    Logger.info("Location was not valid");
}
```

```
public interface IMaybe<T> {
    IMaybe<T> Do(Action<T> someCase, Action noneCase);
}

public class Some<T> : IMaybe<T>
{
    readonly T _obj;
    public Some(T obj)
    {
        _obj = obj;
    }
    public IMaybe<T> Do(Action<T> someCase, Action noneCase) {
        someCase(_obj);
        return this;
    }
}
```

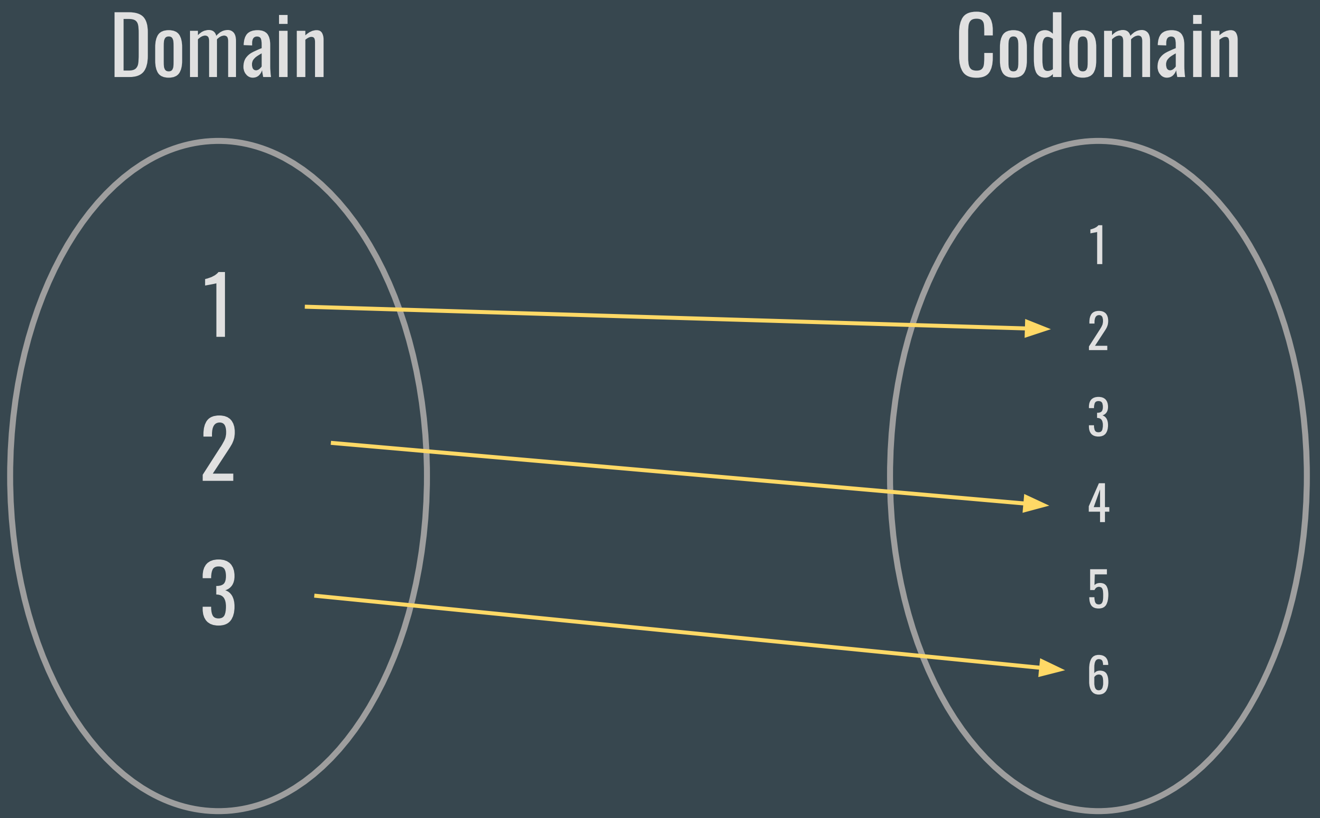
```
public class None<T> : IMaybe<T>
{
    public IMaybe<T> Do(Action<T> someCase, Action noneCase) {
        noneCase();
        return this;
    }
}
```

```
void LogLocation(Location x) {
    Logger.info(“about to insert ” + x.City + “, “ + x.State);
}

void LogNoLocation() {
    Logger.info(“Location was not valid”);
}

Location.Create(“Knoxville”, “TN”)  //Some Location
    .Where(StateIsTennessee)        //Some Location
    .Do(LogLocation, LogNoLocation) //Some Location
    .Select(InsertIntoDb);
    //about to insert Knoxville, TN
```

```
void LogLocation(Location x) {
    Logger.info(“about to insert ” + x.City + “, “ + x.State);
}

void LogNoLocation() {
    Logger.info(“Location was not valid”);
}

Location.Create(“Cincinnati”, “OH”)  //Some Location
    .Where(StateIsTennessee)          //None
    .Do(LogLocation, LogNoLocation) //None
    .Select(InsertIntoDb);
    //Location was not valid
```

Key Takeaways

$$f(x) = 2x$$



We can encode logic into our type system so that the compiler helps us write correct code.

Additional Resources

Code from this presentation: http://tinyurl.com/ReidIMaybe

These Slides: http://tinyurl.com/csharpNoNull

https://github.com/dotnet/csharplang/wiki/Nullable-Reference-Types-Preview

Daily FP videos: http://tinyurl.com/ReidYouTube

Optionally https://github.com/cameronpresley/Optionally

@ReidNEvans