Introducing ConcurrentDictionary

Simon Robinson

@TechieSimon | TechieSimon.com

Module 2 Overview

- *Usual* generic dictionary methods *not suitable* for multithreaded coding
- `ConcurrentDictionary` *has new methods instead*
- `GetOrAdd()` and `AddOrUpdate()`

Course Overview

Concurrent dictionary

Producer-consumer

Best practices

3. Concurrent Dictionary Demo

5. Producer-Consumer and BlockingCollection Demo

2. Introducing Concurrent Dictionary

4. Queues, Stacks and Bags

6. Some best practices

1. Introducing the Concurrent Collections

Dictionary vs. ConcurrentDictionary

`Dictionary<TKey, TValue>`

`ConcurrentDictionary<TKey, TValue>`

Same functionality

Many methods in common

⚠ But using `ConcurrentDictionary` requires a different mindset

Key ConcurrentDictionary Methods

`AddOrUpdate()`

`GetOrAdd()`

This Module:

1 Start with simple code based on `Dictionary<TKey, TValue>`

2 Convert to use `ConcurrentDictionary<TKey, TValue>`

This will teach the mindset too!

This Module:

1 Start with simple code based on `Dictionary<TKey, TValue>`

2 Convert to use `ConcurrentDictionary<TKey, TValue>`

To just see the key methods –
skip ahead to
The AddOrUpdate() Method

CODE DEMO

This slide must not appear in the recorded course

Thread Scheduling

Two threads must make sure an element is in the dictionary,….

```
myDict.Add("pluralsight", 4);
```

Which thread gets here first?

- It's impossible to tell

CODE DEMO

This slide must not appear in the recorded course

Thread-Friendly ConcurrentDictionary Methods

Also available on generic dictionary

```
TryGetValue()
TryAdd()
TryRemove()
TryUpdate()
```

Won't throw exceptions if they fail

```
GetOrAdd()
AddOrUpdate()
```

Will always succeed

No state-dependent exceptions – so great for multithreading

The TryGetValue() Method

From `IDictionary<Tkey, TValue>`

Using indexer:

```
int psStock = stock["pluralsight"];
```

Using TryGetValue():

```
int psStock;
bool success = stock.TryGetValue("pluralsight", out psStock);
```

CODE DEMO

This slide must not appear in the recorded course

The TryUpdate() Method

```
bool TryUpdate(TKey key,
               TValue newValue,
               TValue comparisonValue)
```

Expected to match existing value for the key

Update proceeds only if:

1. `key` is in dictionary
2. `dict[key] == comparisonValue`

CODE DEMO

This slide must not appear in the recorded course

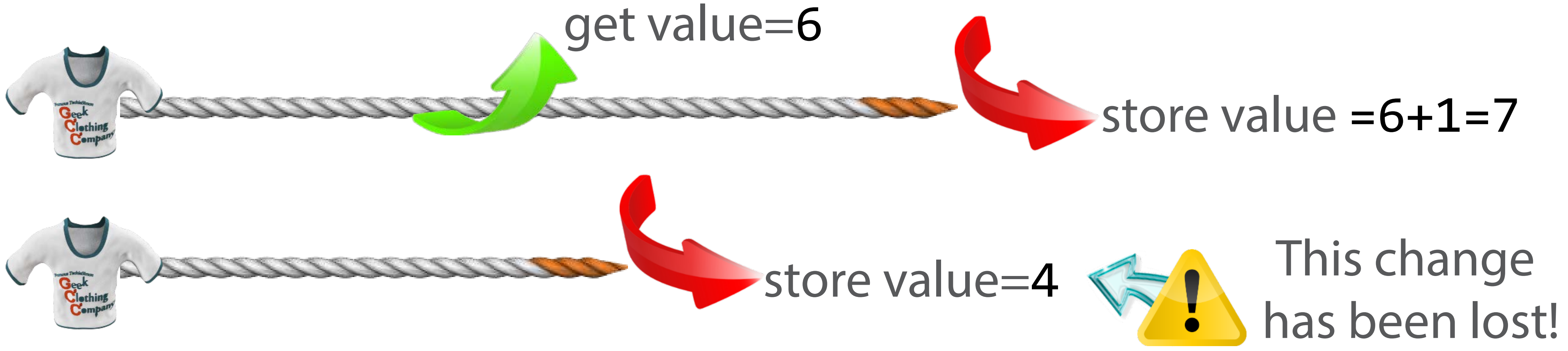
How Do You Add 1 to a Value?

Single-threaded solution:

```
int temp = stock["pluralsight"];

stock["pluralsight"] = temp + 1;
```

This is not atomic!



Can You Use TryUpdate Instead?

```
int temp = stock["pluralsight"];
bool success = stock.TryUpdate("pluralsight", temp + 1, temp);
if (!success)
{
    // what do you do?
    temp = stock["pluralsight"];
    // etc. – this still won't work
```

We need another solution...

...`ConcurrentDictionary<TKey, TValue>.AddOrUpdate()`

CODE DEMO

This slide must not appear in the recorded course

CODE DEMO

Grey area must not appear in the recorded course

Multiple Threads

If another thread might update the element…

Use return values from `AddOrUpdate()` etc.

✓ It's more efficient

✓ It avoids race conditions

Using an Update Result

```
int psStock = stock.AddOrUpdate(
    "pluralsight", 1, (key, oldValue) => oldVal

Console.WriteLine("New value is " + psStock);
```

Displays new value after the update

Do each operation in **ONE** concurrent collection method call

```
int psStock = stock.AddOrUpdate(
    "pluralsight", 1, (key, oldValue) => oldValue + 1);

Console.WriteLine("New value is " + stock["pluralsight"]
```

Displays what's in the dictionary (at a later time)

Possible race condition here

Using an Update Result

This is the key
to using concurrent collections
correctly

Do each operation in **ONE** concurrent collection method call

Concurrent collections
do not protect you from race conditions
between method calls

Values Can Go Out-of-Date

```
int psStock = stock.AddOrUpdate(
    "pluralsight", 1, (key, oldValue) => oldValue + 1);
```

Guarantees to return a value...

…but this value could
become out of date
as soon as you have it!

The GetOrAdd() Method

```
TryGetValue()
TryAdd()
TryRemove()
TryUpdate()
```

Won't throw exceptions if they fail

```
AddOrUpdate()

GetOrAdd()
```

For modifying values

For looking up values

Will always succeed

The GetOrAdd() Method

```
TValue GetOrAdd(TKey key, TValue value)
```

Key to look up

Value to add for key
if key was missing

1. Tries to get value for the key from the dictionary
2. If key wasn't in the dictionary, adds it

CODE DEMO

This slide must not appear in the recorded course

Looking up a Value

```
int psStock = stock["pluralsight"];
```

```
int psStock;
bool success = stock.TryGetValue("pluralsight", out psStock);
```

```
int psStock = stock.GetOrAdd("pluralsight", 0);
```

Module 2 Summary

- Most Dictionary methods are not good for multithreading
- For thread-safe coding:
  - `ConcurrentDictionary.TryXXX()` methods will fail gracefully
  - `GetOrAdd()` and `AddOrUpdate()` won't fail
- One concurrent dictionary method call for each operation