Using Associative Collections

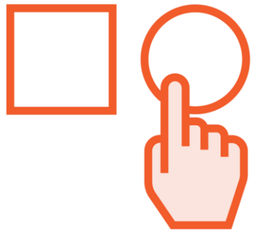
Zoran Horvat
CEO at Coding Helmet

@zoranh75 https://codinghelmet.com

Working with Associative Collections

There must be the key-to-value mapping

- One key can only map to one value
- Otherwise, there would be a collision between objects



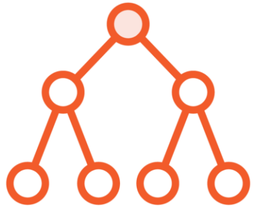
Associative collections offer fast retrieval by the key

- Retrieving by position is usually slow
- That is directly opposite to lists

Working with Associative Collections

The cost of search, insert and delete may vary

- From $O(1)$ time in hash table to $O(\log n)$ time in balanced tree
- List appends in $O(1)$ time but searches and deletes in $O(n)$ time



Associative collections rely on hash tables and balanced trees

- `HashSet` and `Dictionary` use the hash table
- `SortedDictionary` uses the balanced tree

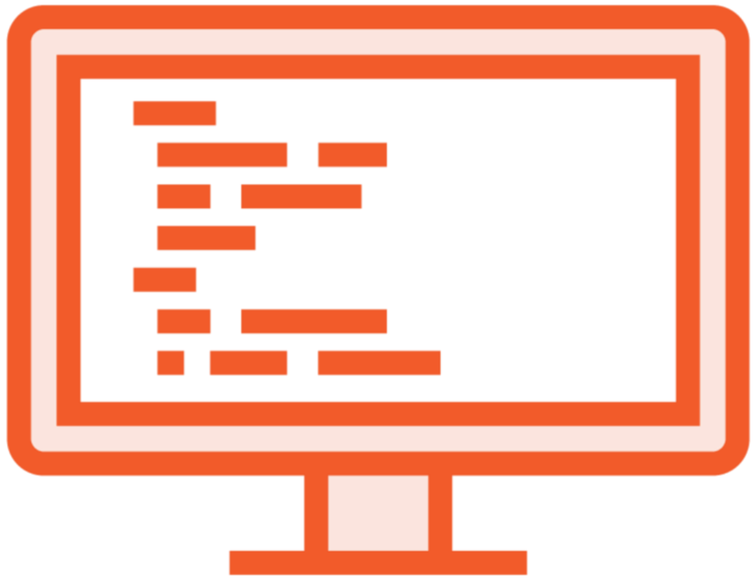
Demos in This Module

Aggregating `Money` objects by their currency

- `Dictionary`, `SortedDictionary`
- `ImmutableSortedDictionary`

Caching equal objects

- `HashSet`



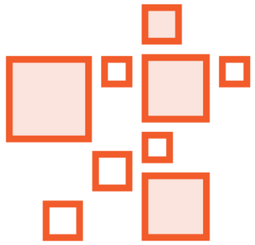
Seen so far:

- `Dictionary<TKey, TValue>`
- `SortedDictionary<TKey, TValue>`
- `ImmutableSortedDictionary<TKey, TValue>`

Next demo:

- Using the `HashSet<T>`

Addressing Repeated `Currency` Objects

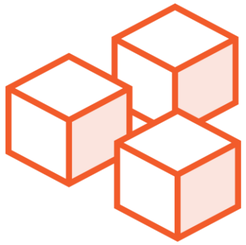


There is no upper bound to the number of objects

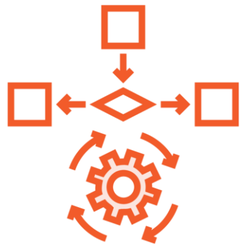
We are working with millions of objects anyway

ISO 4217 defines less than 300 currencies worldwide

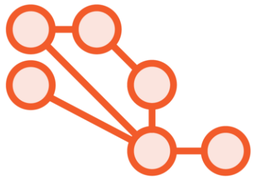
Addressing Repeated `Currency` Objects



A few distinct currency codes will repeat endlessly in memory



Why don’t we share references to the same strings?



`Currency` **is immutable and safe for sharing**

Frequently shared objects are more likely to remain present in the fast CPU cache

Holding numerous copies
of the same data will cause
performance degradation

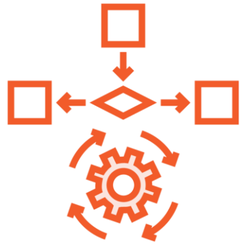
Demo

Measure CPU and memory performance

Operate on millions of `Currency` objects

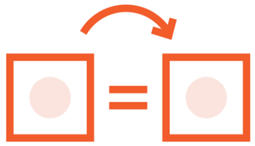
Apply a `HashSet` to improve performance

The Flyweight Design Pattern



Flyweight applies to shareable objects (e.g. immutable objects)

Applies when lack of object sharing hurts performance



Serve unique instances from a collection instead of creating again

String Interning in .NET

```
string a =
  new StringBuilder().Append("Some").Append("thing").ToString();    // "Something"

string b = string.Intern(a);

string temp =
  new StringBuilder().Append("So").Append("me").Append("thing").ToString();
string c = string.Intern(temp);     // "Something"
```

String Interning in .NET

```
string a =
  new StringBuilder().Append("Some").Append("thing").ToString();    // "Something"

string b = string.Intern(a);

string temp =
  new StringBuilder().Append("So").Append("me").Append("thing").ToString();
string c = string.Intern(temp);    // "Something"
```

Two distinct string instances

String Interning in .NET

```
string a =
  new StringBuilder().Append("Some").Append("thing").ToString();   // "Something"

string b = string.Intern(a);
```

Two distinct string instances

```
string temp =
  new StringBuilder().Append("So").Append("me").Append("thing").ToString();
string c = string.Intern(temp);     // "Something"
```

Interned into a single object

String Interning in .NET

```
string a =
  new StringBuilder().Append("Some").Append("thing").ToString();   // "Something"

string b = string.Intern(a);
```

Two distinct string instances

```
string temp =
  new StringBuilder().Append("So").Append("me").Append("thing").ToString();
string c = string.Intern(temp);     // "Something"
```

Interned into a single object

```
object.ReferenceEquals(a, b);       // true
object.ReferenceEquals(b, c);       // true
```

String Interning in .NET

```
string a =
  new StringBuilder().Append("Some").Append("thing").ToString();    // "Something"

string b = string.Intern(a);

string temp =
  new StringBuilder().Append("So").Append("me").Append("thing").ToString();
string c = string.Intern(temp);     // "Something"
```

Drawbacks of interning:

- It is global and static
- Strings remain in the global string pool

Summary

Applied the `Dictionary`

- Aggregated objects under their key
- Utilized constant time for operations (insert, find by key, remove)

Applied the `SortedDictionary`

- Sorts values by the key
- Operations run in logarithmic time (insert, find by key, remove)

Applied the `ImmutableSortedDictionary`

- Made containing class deeply immutable
- Class becomes safe for multithreading

Summary

Analyzing memory patterns

- Lack of object sharing hurts CPU efficiency and takes memory
- Applied a `HashSet` to reduce memory footprint and improve speed

Up Next: Engineering Solutions Using Associative Collections