

MultiMaps/MultiMaps.Core/gpt/combined_source_files

```
1
2 === File: ../Internal/Bucket.cs ===
3 namespace MultiMaps.Core.Internal;
4
5 internal class Bucket<TKey, TValue>
6 {
7     public TKey Key { get; }
8     public HashSet<TValue> Values { get; }
9
10    public Bucket(TKey key)
11    {
12        Key = key;
13        Values = new HashSet<TValue>();
14    }
15 }
16
17 === File: ../Internal/BucketComparer.cs ===
18 namespace MultiMaps.Core.Internal;
19
20 public class BucketComparer<TKey> : IEqualityComparer<TKey>
21 {
22     private readonly int _bucketCount;
23     private readonly IEqualityComparer<TKey> _innerComparer;
24
25     public BucketComparer(int bucketCount)
26         : this(bucketCount, EqualityComparer<TKey>.Default)
27     {
28     }
29
30     public BucketComparer(
31         int bucketCount,
32         IEqualityComparer<TKey> innerComparer)
33     {
34         if (bucketCount <= 0)
35         {
36             throw new ArgumentOutOfRangeException(
37                 nameof(bucketCount),
38                 "Bucket count must be positive");
39         }
40
41         _bucketCount = bucketCount;
42         _innerComparer = innerComparer
43             ?? throw new ArgumentNullException(nameof(innerComparer));
44     }
45
46     public bool Equals(TKey? x, TKey? y)
47     {
48         return _innerComparer.Equals(x, y);
```

```

49     }
50
51     public int GetHashCode(TKey obj)
52     {
53         if (obj == null)
54             return 0;
55
56         var originalHash = _innerComparer.GetHashCode(obj);
57
58         return Math.Abs(originalHash % _bucketCount);
59     }
60
61     public int GetBucketIndex(TKey key)
62     {
63         return GetHashCode(key);
64     }
65
66     public int BucketCount => _bucketCount;
67 }
68
69 === File: ../Internal/FnvHashStrategy.cs ===
70 namespace MultiMaps.Core.Internal;
71
72 public class FnvHashStrategy<TKey> : IEqualityComparer<TKey>
73 {
74     private const uint FNV_PRIME = 16777619;
75     private const uint FNV_OFFSET_BASIS = 2166136261;
76     private readonly IEqualityComparer<TKey> _innerComparer;
77
78     public FnvHashStrategy()
79         : this(EqualityComparer<TKey>.Default)
80     {
81     }
82
83     public FnvHashStrategy(IEqualityComparer<TKey> innerComparer)
84     {
85         _innerComparer = innerComparer
86             ?? throw new ArgumentNullException(nameof(innerComparer));
87     }
88
89     public bool Equals(TKey? x, TKey? y)
90     {
91         return _innerComparer.Equals(x, y);
92     }
93
94     public int GetHashCode(TKey obj)
95     {
96         if (obj == null)
97             return 0;
98

```

```

99         var originalHash = _innerComparer.GetHashCode(obj);
100         uint hash = FNV_OFFSET_BASIS;
101
102         var bytes = BitConverter.GetBytes(originalHash);
103         foreach (var b in bytes)
104         {
105             hash ^= b;
106             hash *= FNV_PRIME;
107         }
108
109         return (int)hash;
110     }
111 }
112
113 public class MurmurHashStrategy<TKey> : IEqualityComparer<TKey>
114 {
115     private const uint SEED = 0x9747b28c;
116     private const uint M = 0x5bd1e995;
117     private const int R = 24;
118     private readonly IEqualityComparer<TKey> _innerComparer;
119
120     public MurmurHashStrategy()
121         : this(EqualityComparer<TKey>.Default)
122     {
123     }
124
125     public MurmurHashStrategy(IEqualityComparer<TKey> innerComparer)
126     {
127         _innerComparer = innerComparer
128             ?? throw new ArgumentNullException(nameof(innerComparer));
129     }
130
131     public bool Equals(TKey? x, TKey? y)
132     {
133         return _innerComparer.Equals(x, y);
134     }
135
136     public int GetHashCode(TKey obj)
137     {
138         if (obj == null)
139             return 0;
140
141         var originalHash = _innerComparer.GetHashCode(obj);
142         var bytes = BitConverter.GetBytes(originalHash);
143         uint h = SEED ^ (uint)bytes.Length;
144
145         foreach (var b in bytes)
146         {
147             uint k = b;
148             k *= M;

```

```

149         k ^= k >> R;
150         k *= M;
151
152         h *= M;
153         h ^= k;
154     }
155
156     h ^= h >> 13;
157     h *= M;
158     h ^= h >> 15;
159
160     return (int)h;
161 }
162 }
163
164 === File: ../Internal/HashStrategyFactory.cs ===
165 namespace MultiMaps.Core;
166
167 public enum HashingAlgorithm
168 {
169     Default,
170     FowlerNollVo,
171     Murmur
172 }
173
174 public static class HashingStrategies
175 {
176     public static IEqualityComparer<TKey> Create<TKey>(
177         HashingAlgorithm algorithm)
178     {
179         return algorithm switch
180         {
181             HashingAlgorithm.FowlerNollVo
182                 => new Internal.FnvHashStrategy<TKey>(),
183
184             HashingAlgorithm.Murmur
185                 => new Internal.MurmurHashStrategy<TKey>(),
186
187             _ => EqualityComparer<TKey>.Default
188         };
189     }
190
191     public static IEqualityComparer<TKey> CreateBucketed<TKey>(
192         int bucketCount,
193         HashingAlgorithm algorithm = HashingAlgorithm.Default)
194     {
195         var baseComparer = Create<TKey>(algorithm);
196
197         return new Internal.BucketComparer<TKey>(
198             bucketCount,

```

```

199         baseComparer);
200     }
201 }
202
203 === File: ../Internal/MapEnumerator.cs ===
204 using System.Collections;
205
206 namespace MultiMaps.Core.Internal;
207
208 internal class MapEnumerator<TKey, TValue>
209     : IEnumerator<KeyValuePair<TKey, ISet<TValue>>>
210 {
211     private readonly List<Bucket<TKey, TValue>> _buckets;
212     private int _currentIndex;
213     private KeyValuePair<TKey, ISet<TValue>> _current;
214
215     public MapEnumerator(List<Bucket<TKey, TValue>> buckets)
216     {
217         _buckets = buckets;
218         _currentIndex = -1;
219         _current = default;
220     }
221
222     public KeyValuePair<TKey, ISet<TValue>> Current => _current;
223
224     object IEnumerator.Current => Current;
225
226     public void Dispose()
227     {
228         // No unmanaged resources to dispose
229     }
230
231     public bool MoveNext()
232     {
233         if (_currentIndex < _buckets.Count - 1)
234         {
235             _currentIndex++;
236
237             var bucket = _buckets[_currentIndex];
238             _current = new KeyValuePair<TKey, ISet<TValue>>(
239                 bucket.Key,
240                 bucket.Values);
241
242             return true;
243         }
244
245         return false;
246     }
247
248     public void Reset()

```

```

249     {
250         _currentIndex = -1;
251         _current = default;
252     }
253 }
254
255 === File: ../MultiMaps.Core.csproj ===
256 <Project Sdk="Microsoft.NET.Sdk">
257
258     <PropertyGroup>
259         <TargetFramework>net9.0</TargetFramework>
260         <ImplicitUsings>enable</ImplicitUsings>
261         <Nullable>enable</Nullable>
262     </PropertyGroup>
263
264 </Project>
265
266
267 === File: ../OneToManyMap.cs ===
268 using System.Collections;
269 using MultiMaps.Core.Internal;
270
271 namespace MultiMaps.Core;
272
273 public class OneToManyMap<TKey, TValue>
274     : IEnumerable<KeyValuePair<TKey, TValue>>
275 {
276     private readonly List<Bucket<TKey, TValue>> _buckets;
277     private readonly IEqualityComparer<TKey> _comparer;
278
279     public OneToManyMap()
280         : this(EqualityComparer<TKey>.Default)
281     {
282     }
283
284     public OneToManyMap(IEqualityComparer<TKey> comparer)
285     {
286         _buckets = new List<Bucket<TKey, TValue>>();
287         _comparer = comparer
288             ?? throw new ArgumentNullException(nameof(comparer));
289     }
290
291     public ISet<TValue> this[TKey key]
292     {
293         get
294         {
295             var bucket = FindBucket(key)
296                 ?? throw new KeyNotFoundException(
297                     $"The key '{key}' was not found.");
298         }
299     }

```

```

299         return bucket.Values;
300     }
301 }
302
303 public int Count => _buckets.Count;
304
305 public ICollection<TKey> Keys =>
306     _buckets.Select(b => b.Key).ToList();
307
308 public ICollection<ISet<TValue>> Values =>
309     _buckets.Select(b => (ISet<TValue>)b.Values).ToList();
310
311 public void Add(TKey key, TValue value)
312 {
313     var bucket = FindBucket(key);
314     if (bucket == null)
315     {
316         bucket = new Bucket<TKey, TValue>(key);
317         _buckets.Add(bucket);
318     }
319     bucket.Values.Add(value);
320 }
321
322 public void AddRange(TKey key, IEnumerable<TValue> values)
323 {
324     if (values == null)
325         throw new ArgumentNullException(nameof(values));
326
327     var bucket = FindBucket(key);
328     if (bucket == null)
329     {
330         bucket = new Bucket<TKey, TValue>(key);
331         _buckets.Add(bucket);
332     }
333
334     foreach (var value in values)
335         bucket.Values.Add(value);
336 }
337
338 public bool Remove(TKey key)
339 {
340     for (int i = 0; i < _buckets.Count; i++)
341     {
342         if (_comparer.Equals(_buckets[i].Key, key))
343         {
344             _buckets.RemoveAt(i);
345             return true;
346         }
347     }
348     return false;

```

```

349     }
350
351     public bool RemoveValue(TKey key, TValue value)
352     {
353         var bucket = FindBucket(key);
354         return bucket != null && bucket.Values.Remove(value);
355     }
356
357     public bool ContainsKey(TKey key) => FindBucket(key) != null;
358
359     public bool ContainsValue(TKey key, TValue value)
360     {
361         var bucket = FindBucket(key);
362         return bucket != null && bucket.Values.Contains(value);
363     }
364
365     public bool TryGetValues(TKey key, out ISet<TValue> values)
366     {
367         var bucket = FindBucket(key);
368         if (bucket != null)
369         {
370             values = bucket.Values;
371             return true;
372         }
373         values = new HashSet<TValue>();
374         return false;
375     }
376
377     public void Clear() => _buckets.Clear();
378
379     private Bucket<TKey, TValue>? FindBucket(TKey key)
380     {
381         if (key == null)
382             throw new ArgumentNullException(nameof(key));
383
384         return _buckets.FirstOrDefault(b =>
385             _comparer.Equals(b.Key, key));
386     }
387
388     public IEnumerator<KeyValuePair<TKey, ISet<TValue>>> GetEnumerator()
389     {
390         return new MapEnumerator<TKey, TValue>(_buckets);
391     }
392
393     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
394
395     public ISet<TValue> GetOrCreate(TKey key)
396     {
397         var bucket = FindBucket(key);
398         if (bucket == null)

```



```
399     {
400         bucket = new Bucket<TKey, TValue>(key);
401         _buckets.Add(bucket);
402     }
403     return bucket.Values;
404 }
405 }
406
```