

Partie 2 : Projet Génie Logiciel

- Nom et Prénom de l'élève : MOUWAKEH Fares
- Prof encadrant : Sayar Imen
- Lien vers le dépôt git du projet avec les modifications : <https://github.com/faresmk3/eclipse-collections-GL>
- Lien vers le dépôt original : <https://github.com/eclipse-collections/eclipse-collections>

6 Petites modifications

Supprimer du code mort

Supprimer un nombre magique qui n'avait aucun sens et n'est jamais utilisé dans le code comme on peut le voir dans l'image *no usage* :



et le commit lié à cette opération : <https://github.com/faresmk3/eclipse-collections-GL/commit/4686d32f233d1b36c20feebe7c30f4c82e7d76da>



renommer une méthode

renommer la méthode fastCeil qui se trouve dans la classe `UnifiedSet.java` à **computeCeilingValue** pour donner plus de sens à ce que la méthode fait.

```
private int fastCeil(float v) 3 usages  fares +1
{
    int possibleResult = (int) v;
    if (v - possibleResult > 0.0F)
    {
        possibleResult++;
    }
    return possibleResult;
}
```

<https://github.com/faresmk3/eclipse-collections-GL/commit/694131d788eb9b34fabd31ae1399c454cf95b5d5>

renommer la méthode fastCeil à computeCeilingValue pour donner plus d...
faresmk3 committed 6 minutes ago

renommer une variable

dans la classe `UnifiedSet.java` l'utilisation du nom **curr** pour désigner un élément lorsque l'on manipule quelque chose ce qui n'est pas forcément indicatif et donc j'ai changé toutes les occurrences de **curr** à **currentElement**.

```
public class UnifiedSet<T> 1 inheritor  Eclipse Collections Team +6
{
    private static Object toSentinelIfNull(Object key) 2 usages  Eclipse Collections Team
    {
        if (key == null)
        {
            return NULL_KEY;
        }
        return key;
    }

    private boolean nonNullTableObjectEquals(Object curr, T key) 20 usages  Eclipse Collections Team
    {
        return curr == key || (curr == NULL_KEY ? key == null : curr.equals(key));
    }

    @Override  Eclipse Collections Team
    @Beta
    public ParallelUnsortedSetIterable<T> asParallel(ExecutorService executorService, int batchSize)
    {
        if (executorService == null)
        {
            throw new NullPointerException();
        }
        if (batchSize < 1)
        {
            throw new IllegalArgumentException();
        }
    }
}
```

<https://github.com/faresmk3/eclipse-collections-GL/commit/fd4b75d9129b69e2fe4c3449f6f2cd7ee9da3b6e>

change toutes les occurrences de curr dans la classe UnifiedSet à curre...
faresmk3 pushed 1 commit to master • 951ae09...fd4b75d • 5 seconds ago

réorganiser une classe; méthodes publiques et enfin méthodes privées

on remarque que dans la majorité de classes, les méthodes publiques et privée sont mélangées entre eux ce qu'est pas forcément bien organisé donc j'ai réalisé ce dernier. le commit lié:

<https://github.com/faresmk3/eclipse-collections-GL/commit/91e44a1d5636b2480acccdceea82f0e379a1bd38>

réduire la complexité cyclomatique ou le nombre de lignes d'une méthode

il y a beaucoup de méthodes où lire la méthode pour la première fois n'est pas forcément très lisible ni compréhensible et donc j'ai décidé de réduire cette complexité pour bien rendre la méthode lisible et moins frustrante à lire; la complexité avant :

org.eclipse.collections.impl.set.mutable.UnifiedSet.chainedAdd(T, int)	43	11	11	16
--	----	----	----	----

et après :

org.eclipse.collections.impl.set.mutable.UnifiedSetManagement.chainedAdd(T, int)	16	9	6	9
--	----	---	---	---

bien que la méthode est maintenant plus longue mais elle est beaucoup plus facile à voir ce qu'elle fait. le commit lié : <https://github.com/faresmk3/eclipse-collections-GL/commit/a67be00e7ee834b558bcf878fe19ee904bca7ece>

décomposer une méthode qui à la fois retourne des informations et modifie l'état d'un objet

la méthode `put` de la classe `UnifiedSet` fait plusieurs choses en même temps, elle modifie un objet et retourne des informations donc elle a deux responsabilités

```
@Override
public T put(T key)
{
    int index = this.unifiedSetManagement.index(key);
    Object cur = this.table[index];

    if (cur == null)
    {
        this.table[index] = UnifiedSet.toSentinelIfNull(key);
        if (++this.occupied > this.maxSize)
        {
            this.rehash();
        }
        return key;
    }

    if (cur instanceof ChainedBucket || !this.nonNullTableObjectEquals(cur, key))
    {
        return this.chainedPut(key, index);
    }

    return this.nonSentinel(cur);
}
```

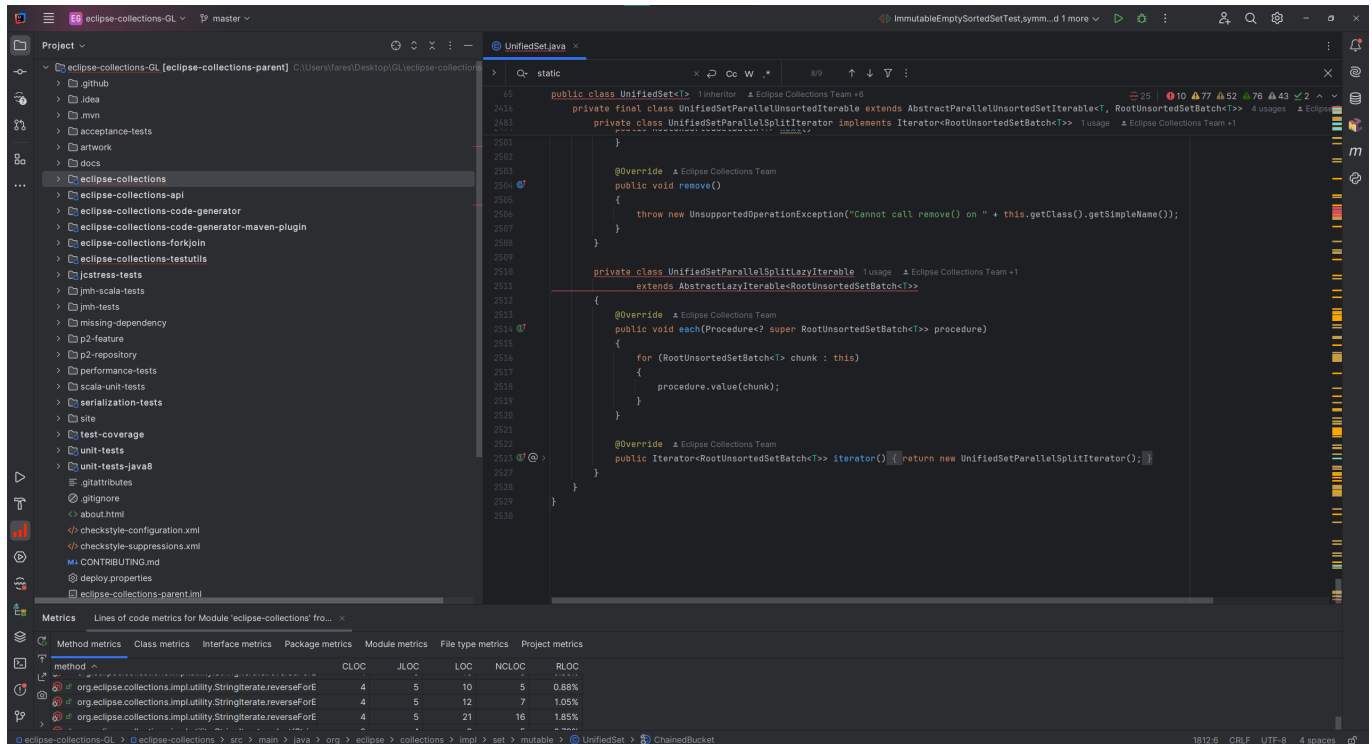
, la première c'est d'insérer la clé si elle n'est pas présente et retourner l'élément stocké dans le set donc séparer cette logique en deux garantit une seule responsabilité et une simplicité donc on a la méthode `insertElement` pour insérer et `getOrInsertElement` pour gérer ces deux idées. le commit lié : <https://github.com/faresmk3/eclipse-collections-GL/commit/4cb7ad4d1c67592854cd00e980402bbc97be6a6f>
<https://github.com/faresmk3/eclipse-collections-GL/commit/44e75c4cb793b55703df3c1d1021cfbdeb9ad89b>

8

Grandes modifications

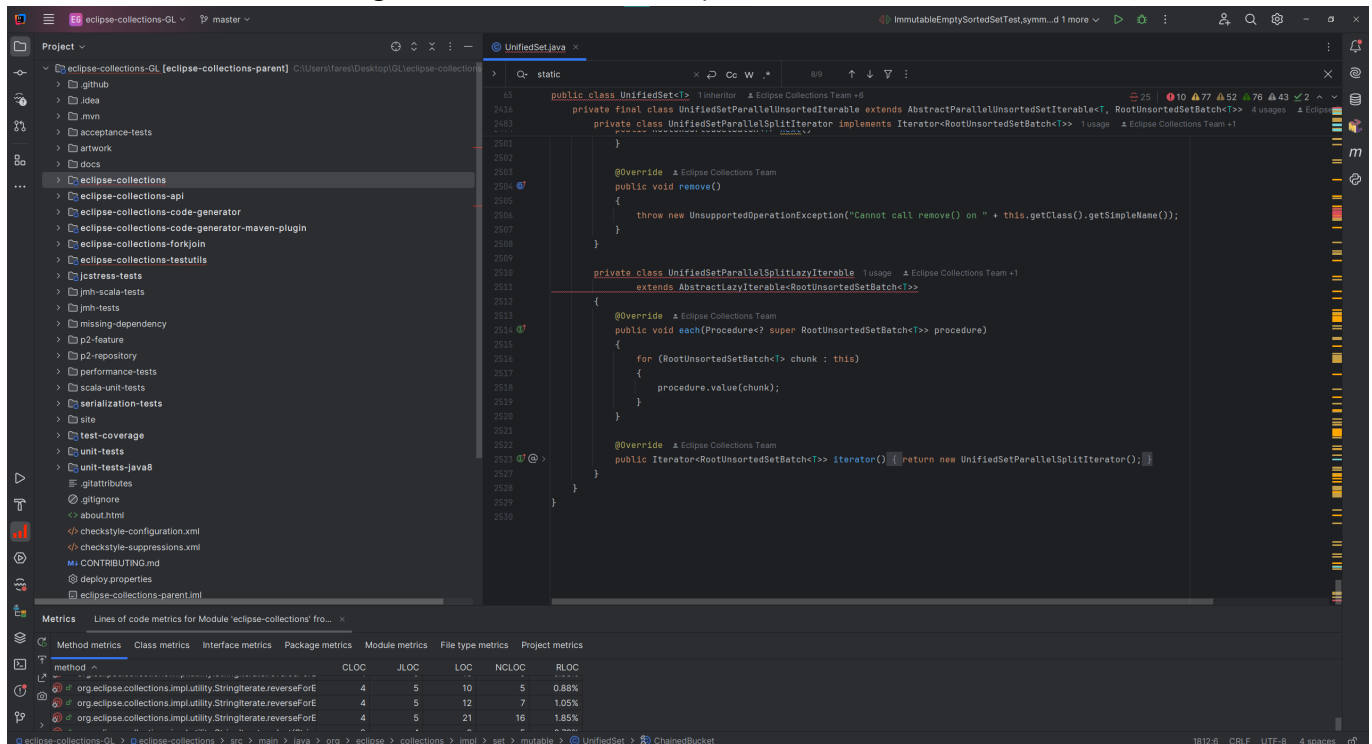
Décomposition de God Classes

comme mentionné auparavant dans l'analyse de la première partie du projet, on a remarqué qu'il y avait beaucoup de God Classes dans l'ensemble de projet, et comme la majorité de ces classes contiennent une logique compacte et spécifique d'une logique mais on peut toute façons de décomposer ces classes au maximum lorsque c'est possible; comme dans le cas de la classe `UnifiedSet.java` :

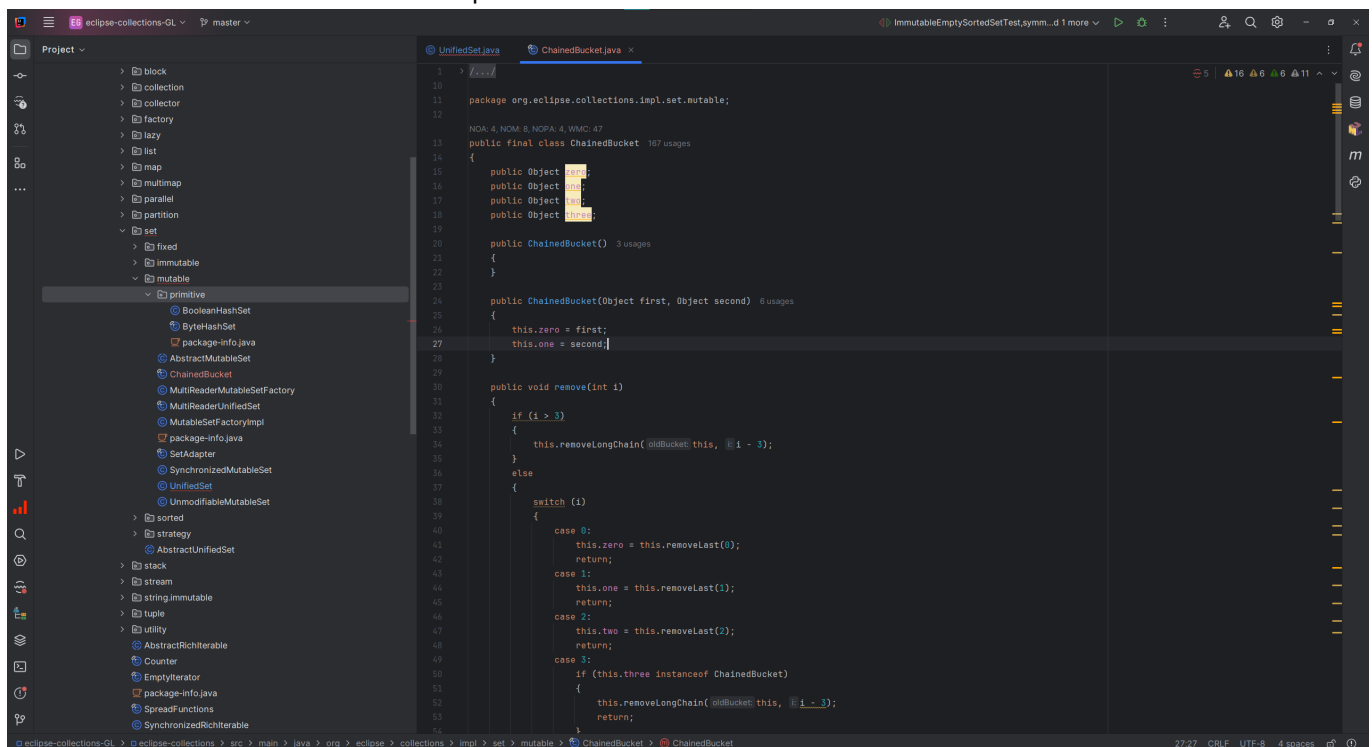


Supprimer des classes static

la classe **Chained Bucket** s'agit d'une classe final static privée dans la classe **UnifiedSet.java** :




qui contient la logique générale de : ajouter, supprimer, getter etc... on peut séparer cette classe de la classe mère en la mettant dans une classe à part.



et la commit liée à cette modification est celle là : <https://github.com/faresmk3/eclipse-collections-GL/commit/4351b4527f6c5a8304b8c153f86807cdc1fae469>

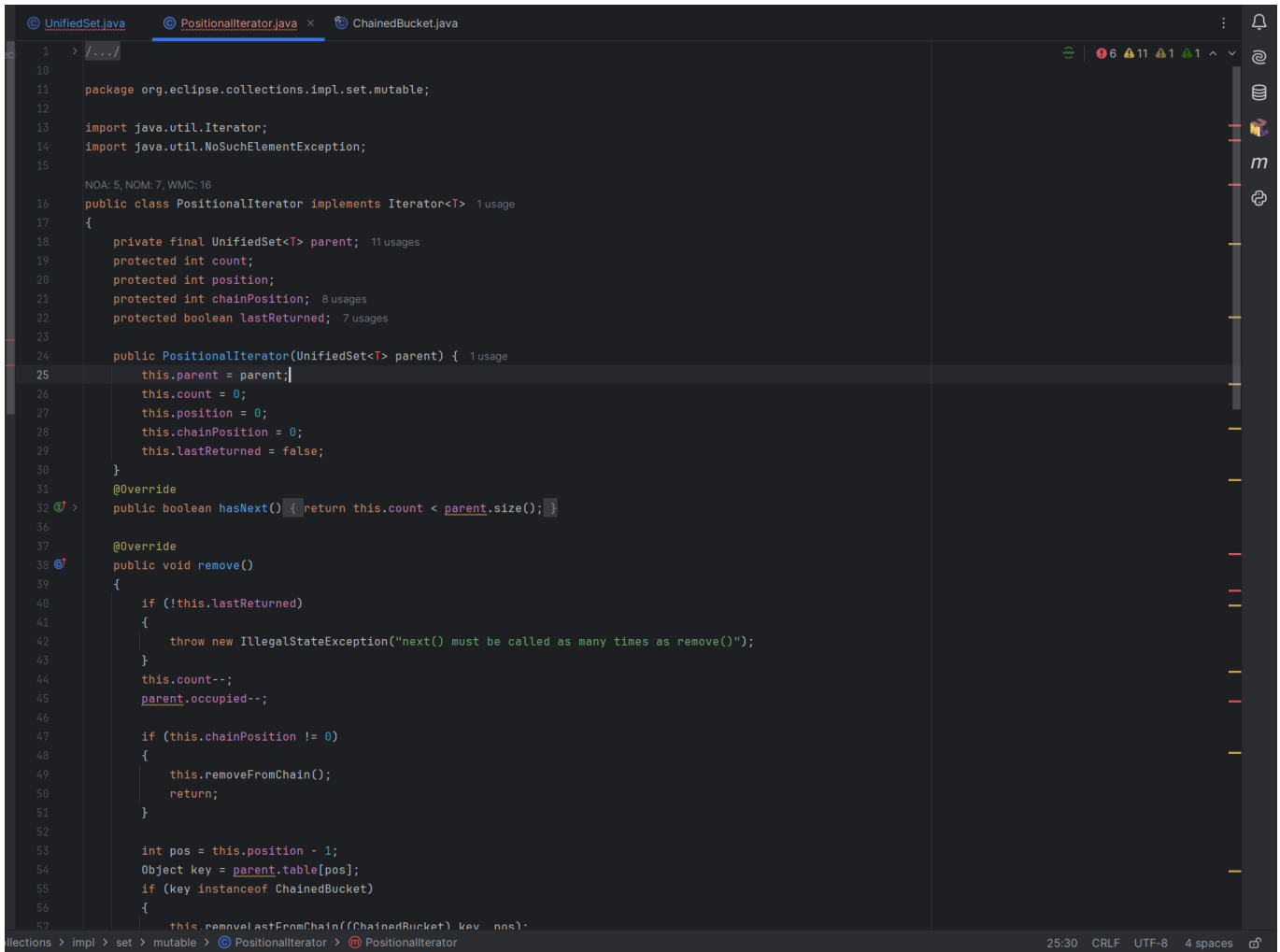
séparer la classe static privée dans la classe UnifiedSet à une class...

 faresmk3 pushed 1 commit to master • 6256e7c...4351b45 • 1 minute ago

de la même façon, la classe `PositionalIterator` qui est une classe protégée dans la classe mère qui contient une logique d'itérateur tout en gérant la chaîne et la séquence et en la gardant cohérente.

```
65 public class UnifiedSet<T> 1 inheritor Eclipse Collections Team +6
1714 protected class PositionalIterator implements Iterator<T> 1 usage Eclipse Collections Team +1
1715 {
1716     protected int count;
1717     protected int position;
1718     protected int chainPosition; 7 usages
1719     protected boolean lastReturned; 6 usages
1720
1721     @Override Eclipse Collections Team
1722     public boolean hasNext() { return this.count < UnifiedSet.this.size(); }
1723
1724     @Override Eclipse Collections Team
1725     public void remove()
1726     {
1727         if (!this.lastReturned)
1728         {
1729             throw new IllegalStateException("next() must be called as many times as remove()");
1730         }
1731         this.count--;
1732         UnifiedSet.this.occupied--;
1733
1734         if (this.chainPosition != 0)
1735         {
1736             this.removeFromChain();
1737             return;
1738         }
1739
1740         int pos = this.position - 1;
1741         Object key = UnifiedSet.this.table[pos];
1742         if (key instanceof ChainedBucket)
1743         {
1744             this.removeLastFromChain((ChainedBucket) key, pos);
1745             return;
1746         }
1747         UnifiedSet.this.table[pos] = null;
1748         this.position = pos;
1749         this.lastReturned = false;
1750     }
1751
1752     protected void removeFromChain() 1 usage Eclipse Collections Team
1753     {
1754         ChainedBucket chain = (ChainedBucket) UnifiedSet.this.table[this.position];
1755         chain.remove(--this.chainPosition);
1756         this.lastReturned = false;
1757     }
1758 }
```

on peut facilement la mettre dans une classe à part. comme cette classe était positionnée dans une classe mère, j'ai dû créer une référence vers cette classe dans la classe mise à part et prendre cette référence en paramètre, comme on crée une instance de cette classe dans la classe mère il suffit de fournir *this* dans le constructeur.



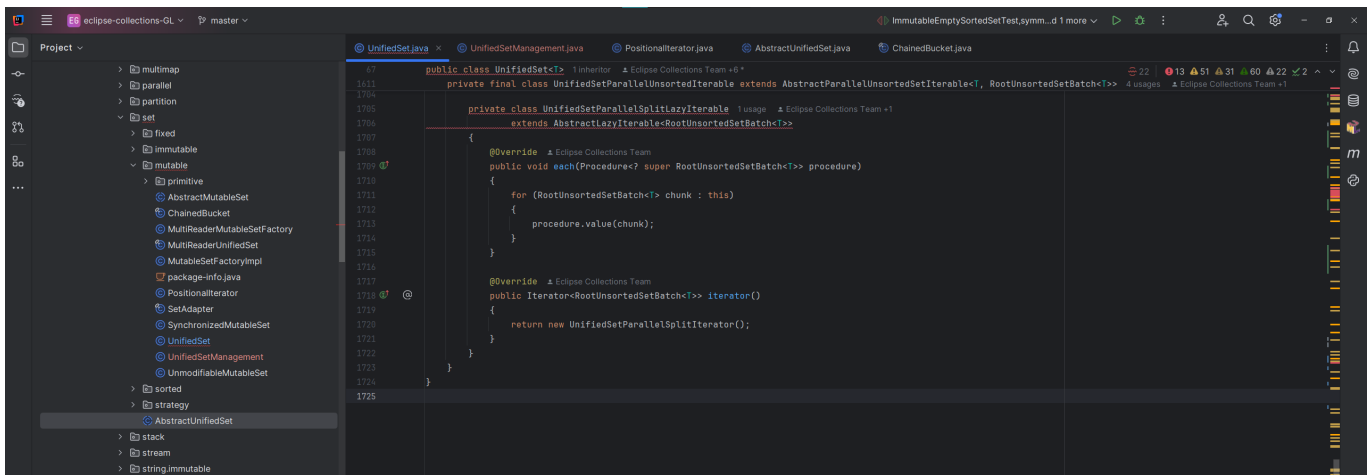
```
1  > [...]
10
11 package org.eclipse.collections.impl.set.mutable;
12
13 import java.util.Iterator;
14 import java.util.NoSuchElementException;
15
16 NOA: 5, NOM: 7, WMC: 16
17 public class PositionalIterator implements Iterator<T> { 1 usage
18 {
19     private final UnifiedSet<T> parent; 11 usages
20     protected int count;
21     protected int position;
22     protected int chainPosition; 8 usages
23     protected boolean lastReturned; 7 usages
24
25     public PositionalIterator(UnifiedSet<T> parent) { 1 usage
26     {
27         this.parent = parent;
28         this.count = 0;
29         this.position = 0;
30         this.chainPosition = 0;
31         this.lastReturned = false;
32     }
33
34     @Override
35     public boolean hasNext() { return this.count < parent.size(); }
36
37     @Override
38     public void remove()
39     {
40         if (!this.lastReturned)
41         {
42             throw new IllegalStateException("next() must be called as many times as remove()");
43         }
44         this.count--;
45         parent.occupied--;
46
47         if (this.chainPosition != 0)
48         {
49             this.removeFromChain();
50             return;
51         }
52
53         int pos = this.position - 1;
54         Object key = parent.table[pos];
55         if (key instanceof ChainedBucket)
56         {
57             this.removeFromChain((ChainedBucket) key, pos);
58         }
59     }
60 }
```

et la commit liée à cette modification est celle là: <https://github.com/faresmk3/eclipse-collections-GL/commit/e4ca9938c41393aab49c317d854bf430b79c2aa6>



```
séparer la classe PositionalIterator de la class UnifiedSet à une cla...
faresmk3 pushed 1 commit to master • 4351b45...e4ca993 • 30 seconds ago
```

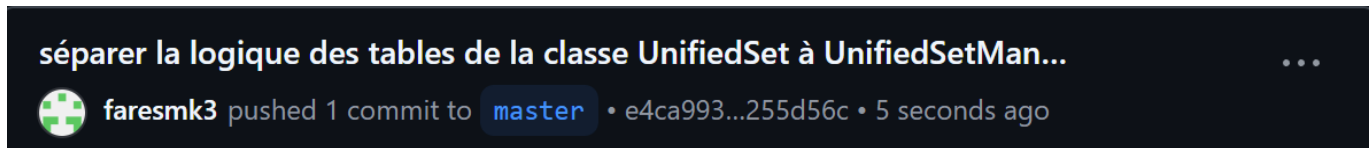
En plus, on peut voir que la majorité de la logique concernant : comment indexer dans la table et l'allocation, add, remove, contains ainsi que les fonction de hashage avec les fonctions qui gèrent la capacité etc... peuvent être placés dans une classe à part. voici ce que l'on obtient: à la base la classe `UnifiedSet.java` était quasiment 2600 lignes de code et on arrive avec cette séparation d'occupation à atteindre 1700 lignes de code.



```
1700 public class UnifiedSet<T> { 1 inheritance
1701     private final class UnifiedSetParallelUnsortedIterable extends AbstractParallelUnsortedSetIterable<T, RootUnsortedSetBatch<T>>
1702     {
1703         extends AbstractLazyIterable<RootUnsortedSetBatch<T>>
1704     {
1705         @Override
1706         public void each(Procedure<? super RootUnsortedSetBatch<T>> procedure)
1707         {
1708             for (RootUnsortedSetBatch<T> chunk : this)
1709             {
1710                 procedure.value(chunk);
1711             }
1712         }
1713     }
1714
1715     @Override
1716     public Iterator<RootUnsortedSetBatch<T>> iterator()
1717     {
1718         return new UnifiedSetParallelSplitIterator();
1719     }
1720 }
```

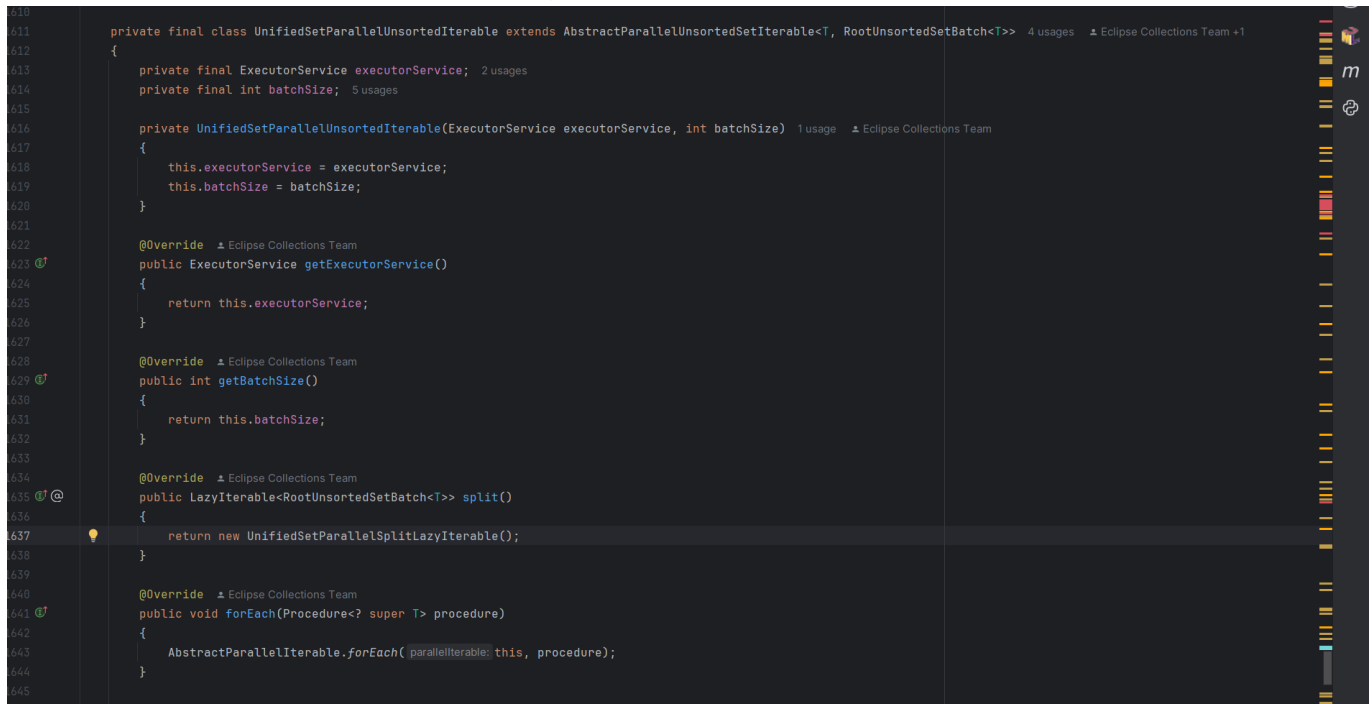
voici le commit liée à cette modification: <https://github.com/faresmk3/eclipse-collections-GL/commit/e4ca9938c41393aab49c317d854bf430b79c2aa6>

GL/commit/4351b4527f6c5a8304b8c153f86807cdc1fae469

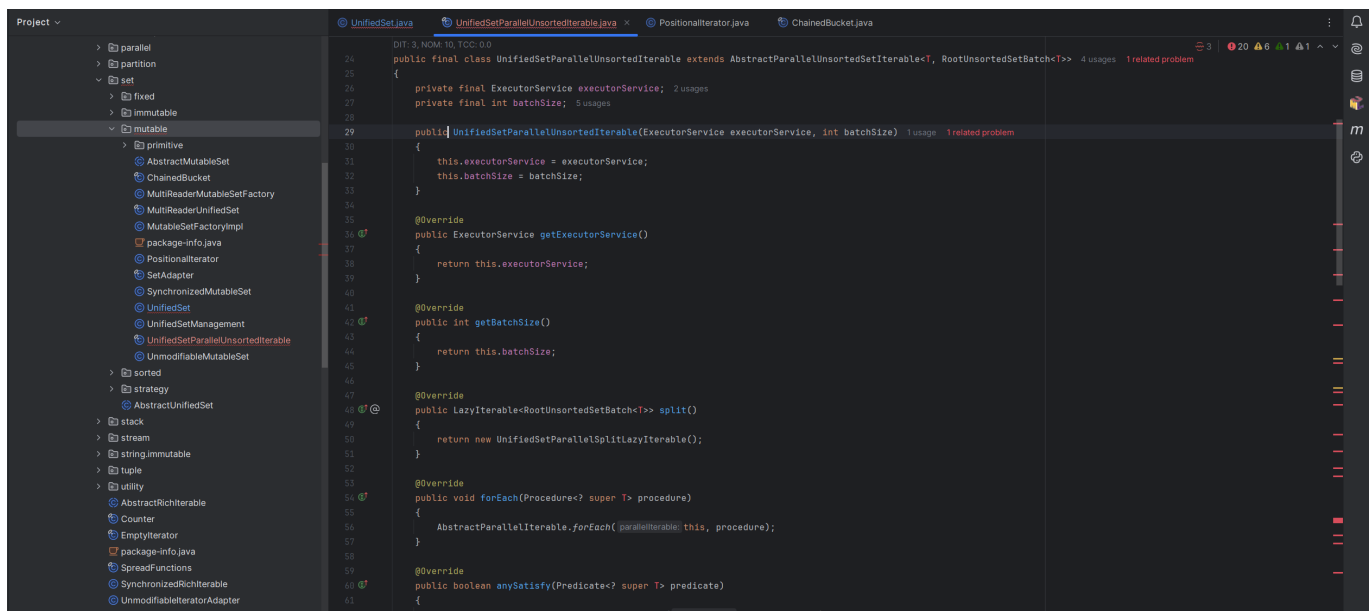


On peut même remarquer qu'il y a encore de classes privée et finales dans cette classe comme :

UnifiedSetParallelUnsortedIterable.java:



on peut facilement mettre sa logique à part car elle n'affectera directement les autres méthodes ou création d'instances dans la classe mère



voici le commit lié à cette modification: [https://github.com/faresmk3/eclipse-collections-](https://github.com/faresmk3/eclipse-collections-GL/commit/6932809d079ca92a871b7b53b32fb5f83a760fe1)

GL/commit/6932809d079ca92a871b7b53b32fb5f83a760fe1



ce qu'on peut observer aussi c'est le fait qu'il y a encore une classe privée dans la classe même qui est

`UnifiedUnsortedSetBatch.java` :

```

1557 private final class UnifiedUnsortedSetBatch extends AbstractBatch<T> implements RootUnsortedSetBatch<T> { 1 usage Eclipse Collections Team +1
1558 {
1559     private final int chunkStartIndex; 5 usages
1560     private final int chunkEndIndex; 5 usages
1561
1562     private UnifiedUnsortedSetBatch(int chunkStartIndex, int chunkEndIndex) 1 usage Eclipse Collections Team
1563     {
1564         this.chunkStartIndex = chunkStartIndex;
1565         this.chunkEndIndex = chunkEndIndex;
1566     }
1567
1568     @Override Eclipse Collections Team
1569     public void forEach(Procedure<? super T> procedure)
1570     {
1571         UnifiedSet.this.each(procedure, this.chunkStartIndex, this.chunkEndIndex);
1572     }
1573
1574     @Override Eclipse Collections Team
1575     public boolean anySatisfy(Predicate<? super T> predicate)
1576     {
1577         return UnifiedSet.this.shortCircuit(predicate, expected: true, onShortCircuit: true, atEnd: false, this.chunkStartIndex, this.chunkEndIndex);
1578     }
1579
1580     @Override Eclipse Collections Team
1581     public boolean allSatisfy(Predicate<? super T> predicate)
1582     {
1583         return UnifiedSet.this.shortCircuit(predicate, expected: false, onShortCircuit: false, atEnd: true, this.chunkStartIndex, this.chunkEndIndex);
1584     }
1585
1586     @Override Eclipse Collections Team
1587     public T detect(Predicate<? super T> predicate)
1588     {
1589         return UnifiedSet.this.detect(predicate, this.chunkStartIndex, this.chunkEndIndex);
1590     }
1591
1592     @Override Eclipse Collections Team +1
1593     public UnsortedSetBatch<T> select(Predicate<? super T> predicate)
1594     {
1595         return new SelectUnsortedSetBatch<>{ unsortedSetBatch: this, predicate };
1596     }

```

et donc après avoir séparé les deux fichiers on a :

```

1557 private final class UnifiedUnsortedSetBatch extends AbstractBatch<T> implements RootUnsortedSetBatch<T> { no usages new *
1558 {
1559     private final int chunkStartIndex; 5 usages
1560     private final int chunkEndIndex; 5 usages
1561
1562     private UnifiedUnsortedSetBatch(int chunkStartIndex, int chunkEndIndex) no usages new *
1563     {
1564         this.chunkStartIndex = chunkStartIndex;
1565         this.chunkEndIndex = chunkEndIndex;
1566     }
1567
1568     @Override new *
1569     public void forEach(Procedure<? super T> procedure)
1570     {
1571         UnifiedSet.this.each(procedure, this.chunkStartIndex, this.chunkEndIndex);
1572     }
1573
1574     @Override new *
1575     public boolean anySatisfy(Predicate<? super T> predicate)
1576     {
1577         return UnifiedSet.this.shortCircuit(predicate, expected: true, onShortCircuit: true, atEnd: false, this.chunkStartIndex, this.chunkEndIndex);
1578     }
1579
1580     @Override new *
1581     public boolean allSatisfy(Predicate<? super T> predicate)
1582     {
1583         return UnifiedSet.this.shortCircuit(predicate, expected: false, onShortCircuit: false, atEnd: true, this.chunkStartIndex, this.chunkEndIndex);
1584     }
1585
1586     @Override new *
1587     public T detect(Predicate<? super T> predicate)
1588     {
1589         return UnifiedSet.this.detect(predicate, this.chunkStartIndex, this.chunkEndIndex);
1590     }
1591
1592     @Override new *

```

voici le commit lié à cette opération: <https://github.com/faresmk3/eclipse-collections-GL/commit/5930f857ab5bd4a2a0d3280db0dcc3f8ccf74baa>

séparer la classe `UnifiedUnsortedSetBatch` de la classe `UnifiedSet`

...



faresmk3 pushed 1 commit to master • 90c5dc3...5930f85 • 8 seconds ago

après avoir fait ces modifications on peut constater que l'on a fait le suivant:

- décomposer une god classe `UnifiedSet`

- supprimer des classes static en les séparant de la class mère.
- diminuer les lignes de code de la classe *UnifiedSet* de 2700 lignes de code à quasiment 1500 lignes de code.

cela reste beaucoup mais dans ce cas là on a séparé l'occupation des sous-classes qui était soit privée finales ou soit privée static à une autre classe qui est totalement séparée de la classe mère et ne change pas la fonctionnalité générale que l'on a eu avant cette séparation.

utiliser un design pattern (MVC, Strategy, Composite, Decorator)

on aurait pu pour ce projet d'implémenter des design patterns, comme par exemple **Iterator Pattern** car on sait que pour ce projet on utilise et on implémente beaucoup d'itérateurs et donc ce genre de design patter facilitera le travail et ne nous nécessite plus de les avoir imbriqué dans le classes qui les concernent. autre idée est d'utiliser le **Null object pattern** comme dans la plupart de classes dans ce projet on a besoin de quelque chose pour comparer à null et on a déjà **NULL_KEY** comme manière de faire, on peut créer un objet qui représente ce genre de choses et a deux méthodes pour la comparaisons pour rendre les choses plus faciles.

Finalemment...

j'ai tenté plusieurs choses et je n'ai pas réussi à faire marcher les tests donc j'en n'ai pas implémenté, autres modifications que je n'ai pas fait est dû au fait que c'est déjà bien implémenté et donc ce n'est pas nécessaire de faire quelque chose comme **supprimer de la duplication de codes entre méthodes** car comme il s'agit d'une API et structures de données il y aura forcément du code dupliqué car les méthodes se ressemblent mais pas forcément d'une manière à pouvoir les factoriser et faire de telle sorte d'avoir une super classe et toute ces idées/détails ont été mentionnées dans la première partie de ce projet.