TP 6 - Implantation d'une table de hachage, classe interne.

Exercice 1 - IntHashSet

1. La classe **Entry** contient les champs **int value** correspondant à la valeur de l'élément et **Entry next** correspondant au prochain élément. La classe **Entry** doit être de visibilité **private** et les champs doivent être **final** c'est pour cela que l'on utilisera ici un **record**.

```
public class IntHashSet {
    private record Entry(int value, Entry next) {
    }
}
```

2. Dans le cas ou la taille de la table est une puissance de 2, on peut utiliser une opération moins coûteuse que le **%** qui est le **&** qui nous permet de directement comparer le **hashCode** avec la taille de la table - 1 qui fait office de masque ici.

```
private int hash(int value) {
    return value & (entries.length - 1);
}
```

3.
```
public void add(int value) {
    var index = hash(value);
    var first = entries[index];

    for (var entry = first; entry != null; entry = entry.next) {
        if (entry.value == value) {
            return;
        }
    }
    entries[index] = new Entry(value, first);
    size ++;
}

public int size() {
    return size;
}
```

4. La méthode **forEach** doit prendre en paramètre un **IntConsumer**.

```
public void forEach(IntConsumer consumer) {
    Objects.requireNonNull(consumer);
    Arrays.stream(entries).forEach(first -> {
        for (var entry = first; entry != null; entry = entry.next) {
            consumer.accept(entry.value);
        }
    });
}
```

5.
```java
public boolean contains(int value) {
    var index = hash(value);
    var first = entries[index];

    for (var entry = first; entry != null; entry = entry.next) {
        if (entry.value == value) {
            return true;
        }
    }
    return false;
}
```

Exercice 2 - DynamicHashSet

1. En java on ne peut pas déclarer un tableau avec un type paramétré. Pour résoudre ce problème il faut **caster**. On a un warning car le compilateur ne sait pas de quel type notre tableau est.
2. **contains** prend en paramètre un **Object** car celle-ci possède une méthode **equals**.
3.
```java
public class DynamicHashSet<E> {
private record Entry<T>(T value, Entry<T> next) {
}

private Entry<E>[] entries;
private int size;

public DynamicHashSet() {
    this.entries = (Entry<E>[]) new Entry<?>[8];
}

private int hash(int value) {
    return value & (entries.length - 1);
}

public void add(E value) {
    var index = hash(value.hashCode());
    var first = entries[index];

    for (var entry = first; entry != null; entry = entry.next) {
        if (entry.value.equals(value)) {
            return;
        }
    }
    if (entries.length / 2 < size) {
        var newEntries = (Entry<E>[]) new Entry<?>[entries.length * 2];
        for (var entry: entries) {
            for (var entryElem = entry; entryElem != null; entryElem =
entryElem.next) {
                newEntries[hash(entry.value.hashCode())] = new Entry<E>
(entryElem.value, first);
```

```
            }
        }
        entries = newEntries;
    }
    entries[index] = new Entry<E>(value, first);
    size++;
}

public int size() {
    return size;
}

public void forEach(Consumer<E> consumer) {
    Objects.requireNonNull(consumer);
    Arrays.stream(entries).forEach(first -> {
        for (var entry = first; entry != null; entry = entry.next) {
            consumer.accept(entry.value);
        }
    });
}

public boolean contains(Object value) {
    var index = hash(value.hashCode());
    var first = entries[index];

    for (var entry = first; entry != null; entry = entry.next) {
        if (entry.value.equals(value)) {
            return true;
        }
    }
    return false;
}
}
```

Steve Chen