JAVA AVANCEE

Implantation d'une table de hachage, classe interne.

TP4

Saravanane MANICOME

IINFO 2020 - 2021

Saravanane MANICOME

Exercice 1

Question 1

```
protected static record Entry(int value, Entry next) {}
```

Puisque nous voulons que les champs de la classe Entry soient spécifiées à l'instanciation et qu'il ne soient pas modifiables par la suite, il est préférable de définir Entry comme un record.

Question 2

```
private static int getIndex(int hashcode) { return hashcode & (bucketsNumber - 1); }
```

Pour implémenter la fonction de hachage nous pouvons tout simplement utiliser la méthode hashcode héritée de la classe Object puis appliquer le modulo à 2 dessus.
Néanmoins il n'est pas préférable de le coder ainsi car l'opération modulo est lente. Puisque le nombre de sceaux est toujours exposant à deux nous pouvons exploiter cette propriété pour appliquer sur le hashcode le masque du nombre de sceaux - 1.
De cette manière on s'assure que le résultat obtenu est toujours dans l'interval des index.

Question 3

```
public void add(int value) {
    if(!contains(value)) {
        int index = getIndex(value);
        Entry entry = entries[index];
        entries[index] = new Entry(value, entry);
    }
}

public int size() {
    int size = 0;
    for (int i = 0; i < bucketsNumber; i++)
        for (Entry j = entries[i]; j != null ; j = j.next())
            size++;

    return size;
}
```

Question 4

Il suffit tout simplement de définir Entry comme étant interne à la classe IntHashSet

Saravanane MANICOME

Question 5

```
public void forEach(IntConsumer entryConsumer) {
    Objects.requireNonNull(entryConsumer);
    for (int i = 0; i < bucketsNumber; i++)
        for (Entry entry = entries[i]; entry != null ; entry = entry.next())
            entryConsumer.accept(entry.value());
}
```

L’interface fonctionnelle passée en paramètre est Function<Integer.class, void.class> qui peut être remplacer par IntConsumer.

Question 6

```
public boolean contains(int value) {
    Entry entry = entries[getIndex(value)];
    for (Entry i = entry; i != null ; i = i.next()) {
        if(i.value() == value)
            return true;
    }

    return false;
}
```

Exercice 2

Question 1

```
@SuppressWarnings("SafeIgnore")
private Entry<T>[] entries = (Entry<T>[]) new Entry[bucketsNumber];
```

Pour palier au problème de typage du tableau nous pouvons le caster. Le warning généré est dû au fait que le cast peut ne pas fonctionner et faire planter le programme.
De notre point de vue nous pouvons l’ignorer car nous sommes certains que les éléments contenu dans le tableau seront de type T.
Pour ignorer ce warning ont peut utiliser l’annotation SuppressWarnings.

Saravanane MANICOME

Question 2

```
public boolean contains(Object value) {
    Objects.requireNonNull(value);
    Entry<T> entry = entries[getIndex(value.hashCode())];
    for (Entry<T> i = entry; i != null ; i = i.next()) {
        if(i.value().equals(value))
            return true;
    }

    return false;
}
```

On utilise un Object plutôt que E car nous ne souhaitons qu'utiliser les méthodes equals et hashcode qui sont présentent dans Object et toutes les classes qui l'héritent.

Question 3

```
public void add(T value) {
    Objects.requireNonNull(value);
    if(!contains(value)) {
        int index = getIndex(value.hashCode());
        Entry<T> entry = entries[index];
        entries[index] = new Entry<T>(value, entry);
        increaseSize();
    }
}

@SuppressWarnings("SafeIgnore")
private void increaseSize() {
    size++;
    if(size < (bucketsNumber / 2))
        return;

    Object[] valueList = new Object[size];
    AtomicInteger index = new AtomicInteger( initialValue: 0);
    forEach((T value) -> {
        valueList[index.getAndIncrement()] = value;
    });

    bucketsNumber = bucketsNumber * 2;
    entries = (Entry<T>[]) new Entry[bucketsNumber];

    size = 0;
    for(Object value : valueList) {
        add((T) value);
    }
}
```

Saravanane MANICOME

Question 4

Telle qu'elle est, la classe peut être hérité et peut donc donner lieu à une réécriture de la méthode forEach. Pour éviter cela nous spécifions le modificateur final pour la classe, empêchant donc d'hérité de celle-ci.

Conclusion

Ce TP contenant les classes internes et les types paramétrés est plus compliqué que les précédents. La difficulté est plus prononcée sur l'usage des types car il ne faut pas seulement que le code fonctionne mais il faut qu'il soit le plus efficace possible.
Par exemple l'utilisation de Object plutôt que E ne m'a pas paru évidente, bien le fait que je comprend l'intention. De même avec l'usage d'une méthode public à l'intérieur d'une classe.

Finalement il reste de la place à l'amélioration mais je suis conscient qu'une compréhension intégrale et immédiate des notions abordées n'est pas toujours possible.