Estructures de Dades

Joel Aumedes i Joel Farré - Laboratorio 3

Tarea 1

La *interface* Queue<E> especifica les funcions que ha d'implementar *BankersQueue<E>*. Per a poder fer-ho hem necessitat dues *List<E>*, anomenades **front** i **back**.

Hem creat dos constructors, un que crea una cua buida, i un altre que crea una cua a partir de dues llistes **front** i **back** proporcionades al cridar-lo.

La funció **size()** retorna la suma de les mides de les dues llistes, i **isEmpty()** comprova si size() és 0, i retorna *True* si ho és.

La funció **add(E)** afegeix un element a la llista *back*. Per a llegir un element de la cua, la funció **element()** retorna l'element superior de la llista *front*. Si *front* està buida, es crida a **transfer()** que transfereix els elements de la llista *back* a la llista *front*. Element() llança una excepció si les dues llistes estàn buides.

La funció **remove()** actua igual que *element()* al adquirir el primer element de la cua, però enlloc de retornar-lo, l'elimina.

També hem implementat un **equals(BankersQueue)** que compara dues cues utilitzant **unify()**, una funció que retorna una sola llista amb els elements de la cua.

Als tests, comprovem el funcionament correcte de la cua en diferents situacions.

Tarea 2

Per a la Tarea 2, hem implementat les operacions de la *interface Iterable<E>*.

Primer hem implementat *iterator()*, que retorna una instància d'*Iterator<E>* de la classe interna de BankersQueue **QueueItr**. Aquesta classe implementa les operacions d'*Iterator<E>*.

Per a fer-ho, hem utilitzat tres paràmetres. Un enter que controla les modifcacions per a evitar que la cua no es modifiqui mentre la recorrem, i dos *ListIterator<E>*,un per a *front* i un per a *back*. Els dos constructors de *QueueItr* permeten crear un iterador que comenci a recórrer la cua des del principi, o des de l'índex que especifica l'usuari.

Per a recórrer la cua *front*, l'iterador la recórre del final fins al principi, i quan hi arriba, aleshores es recórre la cua back, aquest cop de principi a final. Si algun dels iteradors té algun element per llegir, la funció **hasNext()** retorna *True*.

El contador de modificacions s'inicialitza al crear l'iterador, i conta les modificacions totals de la cua. Contem com a modificació les operacions *add(E)* i *remove()*. Si al utilitzar **next()** a l'iterador, la cua s'ha modificat, i per tant els contadors de modificacions són diferents, es llança una excepció de que s'ha modificat la cua. Si no, retorna el proper element a partir dels iteradors.

Remove() llança una excepció.

Pels tests, hem comprovat que l'iterador funciona correctament amb certes situacions, i també hem comprovat que si s'utilitza malament, llança una excepció.

```
    @Test
    void ConcurrentModificationException() {
        try {
            BankersQueue<Integer> queue = new BankersQueue<>();
            Iterator<Integer> it = queue.iterator();
            for (int i = 0; i < 10; i++) {
                queue.add(i + 1);
            }
            System.out.println(it.next());
            fail("Didn't throw exception");
        } catch (ConcurrentModificationException c) {
            assertTrue(true);
        }
    }
```

En aquest test, es llança l'excepció ja que s'intenta imprimir un element per pantalla després de modificar la cua.

Tarea 3

A la Tarea 3, hem creat una classe **BankClient** amb 3 paràmetres; temps d'arribada, temps de sortida i temps de servei. Com a mètodes té un constructor que inicia cada paràmetre i un **equals(BankClient)**, que compara els *BankClient* segons el seu temps d'arribada al banc.

BankSimulator realitza la simulació del banc amb cada una de les opcions de caixers, i imprimeix els resultats. *cashiers* és un *array* que guardarà en cada moment el *BankClient* que està atenent. *doneClients* és una cua que guarda els clients que ja han estat atesos, *time* és el temps de la simulació, i *free* és un client amb arribada -1, que ens serveic per indicar que un caixer està lliure. La resta de paràmetres ens serveixen per resumir el codi i fer-lo més entenedor.

La simulació va adelant el temps en 15 unitats de temps, i a cada volta comprova si queden clients per atendre, quan temps porten els clients sent atesos, si hi ha algun caixer lliure... i actua segons els resultats.

Un cop acabada una simulació, en guarda el resultat i executa la següents incrementant el nombre de caixers. Un cop acabades totes, es mostren els resultats.

```
1 caixers: 2692
2 caixers: 1200
3 caixers: 708
4 caixers: 465
5 caixers: 322
6 caixers: 230
7 caixers: 166
8 caixers: 120
9 caixers: 120
10 caixers: 120
```

Com podem veure, a partir de 8 caixers el temps a dins del banc és 120, que és el temps que s'està un client sent atès. Això és perquè al haver-hi 8 caixers o més sempre hi ha algun caixer lliure quan arriba un nou client.