Cribillers, Arnau; Lázaro, Paul

Estructura de dades
Pràctica 3 - Estudi previ

Tokenizer

L'objectiu principal de la primera part de la pràctica és implementar un programa que pugui identificar, igual que fan les IDEs, si l'obertura i tancament de parèntesi o claus és correcte.

La manera en que es proposa controlar aquets problema és amb la utilització d'una pila i la implementació d'un algoritme que ens permet identificar els símbols.

En primer lloc, haurem d'organitzar el funcionament de la pila. La idea serà intentar imitar el funcionament real de la mateixa. Per tant crearem els mètodes que pertoquen, Push() , Pop(), getSize(), isEmpty(). D'aquest mode serem capaços de gestionar els símbols.

D'altra banda ens falta trobar una manera en la qual puguem organitzar la identificació dels símbols. En aquest sentit, el que farem és crear una classe abstracte que ens permetrà instanciar els diferents símbols que necessitem.

Per últim, crearem un mètode que s'encarregà de gestionar la pila i els nostres objectes de tipus símbol, i ens permeti retornar al usuari si la estructura de parèntesi i claus és correcte, o en cas contrari, no ho és.

Simulador de cues

L'objectiu d'aquest apartat és crear una simulació per a un cine, determinant quants caixers necessiten perquè els clients hagin de fer el minim de cua possible quan arriben.

Per a la realització d'aquest apartat, s'han realitzat les següents classes:

Implementació de la classe ListCua<T>

La classe ListCua<T> implementa una cua amb una llista doblement enllaçada i genèrica, en què cada node té referències tant a l'element següent com a l'anterior. A continuació, veurem una explicació detallada dels seus components principals, així com els passos de les seves funcions.

Aquesta classe implementa l'interficie **CuaInterficie**, la qual defineix que ListCua ha de tenir la funció treure(), insertar() i isEmpty().

1. Estructura Bàsica de la Classe

```
public class ListCua<T> implements CuaInterface<T>{
    9 usages
    private Node<T> front;  //Cap, inici de la cua
    7 usages
    private Node<T> back;   //Cua, final de la cua
```

Els atributs front i back representen els extrems de la cua: el primer (front) i l'últim (back) element. La classe també defineix una classe interna Node que conté les referències per l'element i les connexions amb els nodes adjacent (next i previous).

2. Mètodes

2.1 Constructor ListCua

```
public boolean isEmpty() {
    return front == null;
}
```

Inicialitza una cua buida, on front i back es defineixen com a null.

2.2 Mètode isEmpty

```
public boolean isEmpty() {
    return front == null;
}
```

Determina si la cua és buida, retorna true si front és null, indicant que la cua no té elements.

2.3 Mètode inserir (Enqueue)

Aquest mètode afegeix un nou element al final de la cua. S'explica pas a pas com es realitza el procés:

```
public void inserir(T element) { //Enqueue

    Node<T> aux;

    if(front == null && back == null){ //Si és el primer node que afeg
        front = back = new Node<T>(element, next: null, previous: null);
    } else {
        aux = new Node<T>(element, next: null, back); //El nou node no
        back.next = aux;     //Actualitzem i apuntem el next del penult
        back = aux; //Actualitzem el final de la cua
    }
}
```

Es comprova si la cua està buida (front i back són null). Si és així, es crea un nou node que es defineix com tant front com back.

Si la cua ja té elements, es crea un nou node amb element com a valor, sense next (ja que és l'últim), i amb previous apuntant a l'antic back.

Es configura next de l'antic back per apuntar al node que acabem de crear.

El back s'actualitza perquè apunti al nou node.

2.4 Mètode treure (Dequeue)

Aquest mètode elimina el primer element de la cua i l'allibera.

```
public T treure() { //Dequeue
    T aux = null;

    if(isEmpty()){
        System.out.println("No hi ha elements, no es pot eliminar l'últim element de la llista");
    } else {
        aux = front.element;
        front = front.next;

        if (front == null) {
            back = null;
        }

    }

    return aux;
};
```

Passos:

1. Es comprova si la cua està buida. Si és així, es mostra un missatge d'error.
2. Si no està buida:
    - Es guarda l'element de front a aux.
    - Es reassigna front perquè apunti al seu next.
    - Si front esdevé null (cua buida), back també s'estableix a null.
3. Retorna aux, l'element extret del front.

2.5 Mètode getFront()

Aquest mètode retorna l'element del primer node de la cua sense eliminar-lo.

```
public T getFront() {
    T aux = null;

    if(!isEmpty()){
        return this.front.element;
    }

    return null;
}
```

Retorna l'element del front o null si la cua és buida.

3. Classe Interna Node

```
private static class Node<T>{
    3 usages
    T element;
    3 usages
    Node<T> next;
    1 usage
    Node<T> previous;
    2 usages  Paul
    Node(T e, Node<T> next, Node<T> previous){
        element = e;
        this.next = next;
        this.previous = previous;
    }
};
```

La classe Node defineix l'estructura de cada node dins de la cua.

Implementació de la classe Client

Utilitzem Client per introduir aquest objecte a la classe cua, client també ens permet saber quan ha arribat aquest objecte a la cua.

```
public class Client {

    2 usages
    private int arribada;
    no usages
    private int sortida;

    1 usage   Paul
    public Client(int arribada) { this.arribada = arribada; }

    1 usage   Paul
    public int getArribada() {
        return arribada;
    }
}
```

Implementació del Main / simuladorDeCues()

Aquest codi simula una cua de clients que esperen ser atesos per diversos caixers. Cada cert interval de temps, arriba un nou client a la cua. Cada caixer atén el client del front de la cua quan està disponible i acumula el temps d'espera per calcular una mitjana després de la simulació.

Línia 121: Es crea una cua genèrica ListCua<Client> cuaClients, que conté els clients que esperen ser atesos.

Línia 122: Es declara un array caixers, que representa el temps en què cada caixer estarà disponible per atendre el següent client. Això serveix per controlar el moment en què cada caixer finalitza la seva atenció actual.

Línies 125-130: Es tracta d'un bucle for que inicialitza les variables necessàries per la simulació i que cada cop que s'executa, afegeix un caixer a la simulació

Línia 132: S'inicia un bucle while que continuarà executant-se mentre hi hagi clients per atendre (numClients > 0).

Línia 134: A cada iteració, es comprova si el temps actual (temps) és múltiple de TEMPS_ARRIBADA, que indica cada quants cicles arriba un nou client.

- Si es compleix la condició, es crea un nou Client amb el temps d'arribada actual (temps) i s'afegeix a cuaClients (funció inserir, línia 135).

Línia 137-146: Bucle for que recorre cada caixer (caixers[n]):

Línia 139: Es comprova si el caixer n està disponible (caixers[n] <= temps) i si hi ha clients a la cua (!cuaClients.isEmpty()).

Línia 140: Si el caixer està disponible, es fa el següent:

  - S'actualitza caixers[n] sumant TEMPS_CLIENT, que indica el temps que aquest caixer necessitarà per atendre el client.
  - Línia 141: Es calcula i acumula el temps d'espera per al client en tempsMitja. Aquest valor es calcula com la suma del temps d'atenció (TEMPS_CLIENT) i la diferència entre el temps actual (temps) i el temps d'arribada del client (cuaClients.getFront().getArribada()).
  - Línia 142: Es treu el client de la cua (cuaClients.treure()) ja que ja està sent atès pel caixer.
  - Línia 143: Es decreix el nombre de clients per atendre (numClients--).

Línia 146: Es incrementa el temps, simulant el pas del temps en cada iteració del bucle while.

```
125         for(int i = 0; i < NUM_CAIXERS; i++){
126             cuaClients = new ListCua<>();
127             caixers = new int[i + 1];
128             temps = 0;
129             tempsMitja = 0;
130             numClients = NUM_CLIENTS;
131 
132             while(numClients > 0){ //Hem decidit fer-ho així perquè és una "simulació"
133 
134                 if(temps % TEMPS_ARRIBADA == 0){ //Comptador circular
135                     cuaClients.inserir(new Client(temps));
136                 }
137 
138                 for(int n = 0; n < (i+1) && !cuaClients.isEmpty(); n++){
                        if(caixers[n] <= temps){
140                         caixers[n] += TEMPS_CLIENT;
141                         tempsMitja += TEMPS_CLIENT + (temps - cuaClients.getFront().getArribada());
142                         cuaClients.treure();
143                         numClients--;
144                     }
145                 }
146 
147                 temps++;
148             }
```