

PROGRAMACIÓ AVANÇADA

PRÀCTICA 3. TARDOR 2015

Objectiu 1: Especificar i implementar una **estructura de dades no lineal**.

Objectiu 2: Els arbres ACB. Generecitat usant la notació **<E>**.

Objectiu 3: Treball amb fitxers de text.

Objectiu 4: Ús d'una instància d'arbre ACB per fer tractaments de fitxers de text.

Durada: Una sessió

Lliurament: Llistat imprès dels fonts i penjar el projecte al Moodle

Data Lliurament: El dia previ a la sessió de la pràctica 4

Enunciat

Aquesta pràctica té dues parts, per un cantó cal implementar una estructura de dades i per l'altra usar-la en la resolució d'una problemàtica.

Part 1

En aquesta primera part de la pràctica implementeu les classes i interfícies necessàries per poder treballar amb objectes d'Arbres de Cerca Binària (ACB). Concretament cal implementar:

- La classe *AbEnll<E>*
- La classe *AcbRecorrible<E>*
- La interfície *Ab<E>* (Arbre binari)
- La interfície *Acb<E>* (Arbre de cerca binari)

La interfície *Ab<E>* ha de contenir les següents operacions:

```
public E arrel () throws ArboreException;  
    // cal llençar una excepció si es demana l'arrel d'un arbre buit  
public Ab<E> fillEsquerre()throws ArboreException;  
    // Exception si l'arbre this és buit. Si no té fill esquerre retorna un arbre buit.  
public Ab<E> fillDret()throws ArboreException;  
    // Excepcion si l'arbre this és buit. Si no té fill dret retorna una arbre buit.  
public boolean abBuit();  
public void buidar();
```

Ab<E>

Els arbres *Acb* emmagatzemen objectes comparables i així ha de quedar reflectit en la definició de la interfície *Acb*, per això cal **indicar que estenen la interfície Comparable <E extends Comparable>**. A més de totes les operacions descrites anteriorment ha d'incloure les següents operacions:

Acb<E extends Comparable>

```
public void inserir (E e) throws ArbreException;
    // llença una excepció si l'element que s'insereix està repetit
public boolean esborrar (E e);
    // retorna true si ha trobat l'element i l'ha esborrat
    // ull!!!! retorna false en cas contrari
public boolean membre (E e);
    // retorna true si l'arbre conté un element com el donat com a paràmetre
```

La interfície Acb ha de ser **extensió** de la interfície Ab per la qual cosa qualsevol classe que implementi la segona també haurà d'implementar la primera.

S'ha d'implementar la classe **AbEnll<E>** i la classe **AcbRecorrible<E extends Comparable>** usant la tècnica enllaçada, és a dir fent ús de punters.

Un requeriment necessari és que els arbres Acb que es **desenvolupin en aquesta pràctica siguin recorribles en els dos sentits: ascendentment i descendentment**. En funció del valor de l'atribut **ordre** es determina si el recorregut s'efectua en un sentit u altre.

Pel que fa la recorribilitat, serà implementada amb l'ajut de tres mètodes públics: **iniInordre**, **finalInordre** i **segInordre**.

AcbRecorrible<E extends Comparable>**Atributs**

```
public static final int ORDRE_ASCENDENT= 1789;
public static final int ORDRE_DESCENDENT= -7895;
```

```
private Queue<E> cua; // per ser recorrible
private int ordre;
```

```
// protected NodeA arrel; ve d'herència!!
// La classe NodeA estarà declarada dins de la classe AbEnll i
//serà amb visibilitat protected
```

Opcionalment si voleu podeu usar la vostra Cua<E> de la Pràctica 2

Mètodes

//Redefinició de mètodes heretats de AbEnll si escau

// tots els mètodes especificats per la interfície Acb.

// i a més:

```
public AcbRecorrible(int ordre) {
    if (ordre!=AcbRecorrible.ORDRE_ASCENDENT &&
        ordre!=AcbRecorrible.ORDRE_DESCENDENT)
        throw new IllegalArgumentException("ordre: "+ ordre);
    this.ordre=ordre;
    this.cua=null
}

public void setOrdre (int ordre) {
    if (ordre!=AcbRecorrible.ORDRE_ASCENDENT &&
        ordre!=AcbRecorrible.ORDRE_DESCENDENT)
        throw new IllegalArgumentException("ordre: "+ ordre);
    this.ordre=ordre;
    this.cua=null
}

public void inlInordre ()
// prepara l'arbre per a ser recorregut en inordre. Després d'invocar
// aquest mètode, la invocació del mètode segInordre retornarà el primer
// element en lInordre de l'arbre

public boolean finalInordre ()
/* retorna true si ja s'ha arribat al final del recorregut en inordre
de l'arbre. Això és si:
    - l'arbre és buit
    - la darrera vegada que es va invocar segInordre aquest mètode
    ja va retornar el darrer element en inordre de l'arbre.
Tot això és el mateix que dir que retorna true quan no té sentit
invocar el mètode segInordre */

public E segInordre () throws ArboreException
/*retorna el següent element en lInordre, si n'hi ha.
    llença una excepció si:
    - abans d'invocar-lo no s'ha invocat el mètode inlInordre
    - la darrera vegada que es va invocar ja va retornar
    el darrer element en inordre (finalInordre retornaria true)
    - s'invoca quan entre la invocació de inlInordre i la del mètode
    s'ha produït una modificació de l'arbre, això és, s'ha fet ús del
    mètode inserir, esborrar, buidar o setOrdre */
```

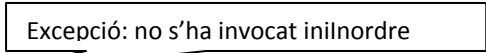
En la implementació d'aquesta classe heu d'usar el **mètode compareTo**, és a dir usar la interfície Comparable de l'API de Java. I **evitar l'ús del mètode equals** per la comparació.

Perquè us feu una idea de com s'efectuaria un recorregut en inordre d'un arbre d'aquesta mena, podeu observar el següent fragment de codi:

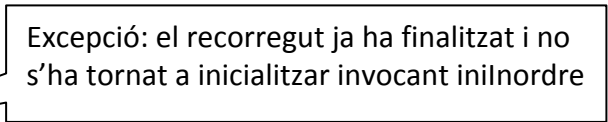
```
AcbRecorrible<E> arbre;  
arbre = new AcbRecorrible(AcbRecorrible.ORDRE_ASCENDENT);  
Comparable c;  
arbre.inserir(...);  
...  
arbre.iniInordre();  
while (!arbre.finalInordre()) {  
    c = arbre.segInordre();  
    // fer el que sigui amb c  
}
```

Perquè us feu una idea de quan el mètode **segInordre** llença una excepció, observeu els següents fragments de codi:

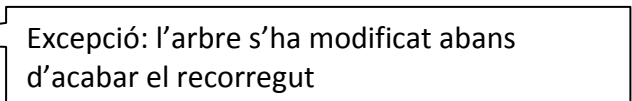
```
AcbRecorrible<E> arbre;  
arbre = new AcbRecorrible( AcbRecorrible.ORDRE_DESCENDENT);  
Comparable c;  
arbre.inserir(...);  
...  
c = arbre.segInordre();
```



```
AcbRecorrible<E> arbre;  
arbre = new AcbRecorrible( AcbRecorrible.ORDRE_DESCENDENT);  
Comparable c;  
arbre.inserir(...);  
...  
arbre.iniInordre();  
while (!arbre.finalInordre()) {  
    c = arbre.segInordre();  
    // fer el que sigui amb c  
}  
c = arbre.segInordre();
```



```
AcbRecorrible<E> arbre;  
arbre = new AcbRecorrible( AcbRecorrible.ORDRE_DESCENDENT);  
Comparable c;  
arbre.inserir(...);  
...  
arbre.iniInordre();  
// suposem que l'arbre té algun element  
c = arbre.segInordre();  
arbre.inserir(...)  
c = arbre.segInordre();
```



Part 2

PROCESSAMENT D'ARXIU DE TEXT

Es vol escriure un programa que faci el següent:

Donat un arxiu de text (el programa en demanarà el nom) obtenir un llistat (pantalla) **ordenat alfabèticament** de totes les paraules que conté (sense repeticions) i que tenen una mida de 10 o més lletres. Usar una instància de la classe AcbRecorrible per donar solució al demanat.

Després, donat un altre arxiu de text (el programa també en demanarà el nom) que contindrà algunes paraules que cal excloure (censurar) del llistat, modificar la instància usada abans (mètode esborrar...) de tal manera que les paraules excloses ja no hi siguin. Tornar a generar el llistat **però ara en ordre lexicogràfic descendent**.

Si useu l'arxiu Doc1.txt que se us subministra llavors el primer llistat hauria de ser aquest:

LLISTA DE PARAULES LLARGUES

----- -- -----

ACQUISITION
APPLICABILITY
APPROACHES
CIRCUMSCRIBED
CONSIDERABLE
DISSERTATION
HANDWRITING
HORIZONTAL
INFORMATION
PARTICULAR
RECOGNITION
SCIENTIFIC
TEXT-DEPENDENT
TIME-DEPENDENT
TRAJECTORIES
TRANSITIONING
VERIFICATION

Si per a les paraules que cal excloure useu l'arxiu Excl.txt subministrat. El segon llistat seria com aquest:

LLISTA DE PARAULES LLARGUES ('CENSURAT')

----- -- -----

TRAJECTORIES
SCIENTIFIC
RECOGNITION
PARTICULAR
HANDWRITING
DISSERTATION
CONSIDERABLE
CIRCUMSCRIBED
APPROACHES
ACQUISITION

Per a resoldre aquest problema us pot ser útil usar instàncies de la classe StringTokenizer (paquet java.util) per a separar les diferents paraules que hi ha en cada línia dels arxius. Atès que a més de paraules hi ha símbols separadors (espais en blanc...) i caràcters de puntuació (punts, comes, ...) podeu fer que els separadors considerats en el moment de separar paraules (tokens) siguin els que hi ha en aquesta cadena:

```
"():.,; \t\n\r\f"
```

Organització

Els noms dels vostres projectes, en les pràctiques d'aquesta assignatura han de seguir el següent patró: PràcticaXCognomNom del propietari de la pràctica. En cas de fer-la en parella cal que poseu PràcticaXCognom1Nom1&&Cognom2Nom2. En aquesta pràctica X és 3.

Creeu un paquet per cada part de la pràctica, Paquet 1 amb la Part 1 (amb les interfícies i classes) i Paquet 2 amb la Part 2 (amb el programa que fa el tractament de fitxers).

Què se us subministra?

Fitxer amb l'enunciat de la pràctica. Arxius de text per comprovar la correcte funcionalitat i aclariments.

Què s'ha de lliurar i com?

S'ha de lliurar la carpeta que conté el projecte Eclipse amb el vostre desenvolupament de la pràctica. La carpeta s'ha de lliurar amb tot el seu contingut i comprimida amb ZIP o RAR.

També s'ha de lliurar el llistat en paper de tot el codi desenvolupat.

On s'ha de lliurar?

El lliurament del projecte es farà a través de la plataforma Moodle i no s'acceptarà cap altra via. Feu atenció a la data i hora límit.

El lliurament en paper es farà directament a la professora a l'inici de la pràctica 4.

Quan s'ha de lliurar?

El lliurament es podrà fer fins **el dia abans de la sessió de la pràctica 4**. Tingueu present que a partir d'aquesta hora el sistema bloquejarà, de manera automàtica, la possibilitat de lliurament.

Grup 101 → 18 de Novembre a les 23:50h

Grup 102 → 11 de Novembre a les 23:50h