TecnoCampus Escola Superior Politècnica — Centre adscrit a la Universitat Pompeu Fabra Barcelona

Programació Avançada (UPF)
Programació 3 (UPC)

DATA: 3 de Novembre del 2015
DURACIÓ: 2h

Exercici 1 (8.5 punts)

Es vol construir una aplicació per a que la professora d'aquesta assignatura pugui gestionar els alumnes matriculats, amb les seves corresponents qualificacions en els diferents lliuraments que contempla l'assignatura. Hi hauran lliuraments que es corresponen a exàmens i d'altres a pràctiques i exercicis de classe. Aquest quadrimestre l'assignatura acull als estudiants de l'assignatura de Programació 3 del pla UPC, per tant és necessari saber per cada estudiant si està cursant l'assignatura pla UPF o UPC, encara que, tots tindran les mateixes regles per superar l'assignatura. Després de la fase d'anàlisi de requeriments s'ha optat pel següent disseny de classes, del que heu de **completar** les implementacions dels mètodes que estan pendents.

Comparable.java

```
public interface Comparable{
        public boolean MenorQue(Comparable c);
        public boolean MajorQue(Comparable c);
}
```

Lliurament.java

```
public class Lliurament{
        private int codi;  //identificació
        private boolean examen;
        private double qualificacio;
        public Lliurament(int codi, boolean examen){
                this.codi=codi;
                this.examen=examen; this.qualificacio=0.0;
        }
        public Lliurament(int codi){
                this(codi,true);
        }
        public void setQualificacio(double nota){ this.qualificacio=nota;}
        public double getQualificacio(){ return qualificacio;}
        public int getCodi(){ return codi;}
        public boolean getExamen(){ return examen;}
}
```

Estudiant.java

```
public class Estudiant implements Comparable{
        private class node{
                Lliurament ll; node seg;
                public node(Lliurament prova){this(prova,null);}
                public node(Lliurament prova, node seg){
                        this.ll=prova;this.seg=seg; }
        }
        private String nom;
        private long dni;  //identificador
        private node qualificacions; //seqüència enllaçada lineal sense capçalera
        private boolean pla_UPF;
        public Estudiant(String nom, long dni, String pla){
                this.nom=nom;this.dni=dni;
                if (pla.equalsIgnoreCase("UPF")) pla_UPF=true;
                else pla_UPF=false;
                qualificacions=null;  //creem la seqüència enllaçada sense node capçalera
        }
```

```
        public boolean getPla_UPF(){return pla_UPF;}
        public long getDni(){return dni;}
        public String getNom(){return nom;}
        public void addLliurament(Lliurament ll){ /*Exercici 1*/}
        public void modificaLLiurament(int codi, float qualificacio) throws Exception{ /*Exercici 2*/}
        public double calculaFinal(){/*Exercici 3*/
                //calcula la qualificació final de l'estudiant segons la normativa d'avaluació
        }
        public boolean MajorQue(Comparable c){
                return dni>((Estudiant)c).dni;}
        public boolean MenorQue(Comparable c){
                return dni<((Estudiant)c).dni;}
} // fi classe
```

Acb.java

```
public interface Acb {
        void  Inserir (Comparable e) throws Exception;
        //excepció si l'element a inserir ja està a l'arbre
        void  Esborrar (Comparable e) throws Exception;
        //excepció si l'element a esborrar no està a l'arbre
        boolean  Membre (Comparable e);
        //true si l'element està a l'arbre, false en cas contrari
        Comparable  Arrel () throws Exception;
        Acb FillEsquerre ();
        //no llença excepció, tan si l'arbre this és buit com si no té fill esquerra retorna un arbre buit
        Acb FillDret ();
        //no llença excepció, tan si l'arbre this és buit com si no té fill dret retorna un arbre buit
        boolean ArbreBuit ();
        void Buidar();
} //fi interfície
```

AcbEnll.java

```
public class AcbEnll implements Acb{
        private class NodeA{
                Estudiant inf; NodeA esq,dret;
                public NodeA(Estudiant m, NodeA e, NodeA d){inf=m;esq=e;dret=d;}
        } //fi classe privada
        private NodeA arrel;
        public AcbEnll(){arrel=null;}
        /* implementació de totes les operacions de la interfície*/

} // fi classe
```

Les operacions que caracteritzen l'estructura de dades les suposeu implementades **idènticament** a les treballades a classe.

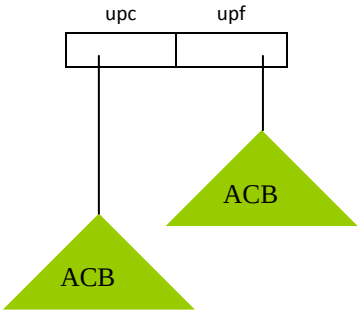
ProgramacioAvançada.java

```
public class ProgramacioAvançada{
        private Acb matriculats[];
        public ProgramacioAvançada(){
                matriculats=new AcbEnll[2];
                matriculats[0]=new AcbEnll();  //UPC
                matriculats[1]=new AcbEnll();  //UPF
        }
        public void AfegirEstudiant(Estudiant e) throws Exception { /*Exercici 4*/ }
        public void EsborrarEstudiant(Estudiant e) throws Exception{ /*Exercici 5*/ }
        public int[] matriculatsDeCadaPla(){ /*Exercici 6*/ }
        public String toString(){ /*Exercici 7*/ }
        public double obtenirNotafinal(long dni) throws Exception{/*Exercici 8*/ }
} // fi classe
```

Aclariments:

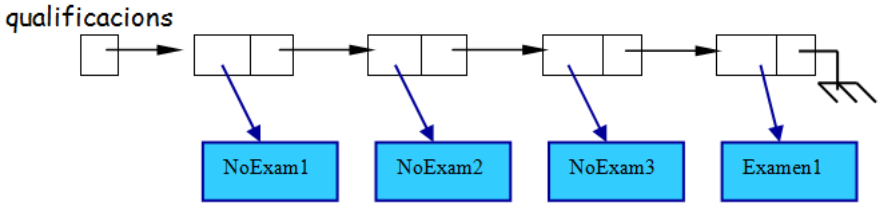
- **Tots els estudiants matriculats en l'assignatura estan emmagatzemats en un arbre Acb, de fet l'assignatura té dos arbres, un que té els estudiants que cursen l'assignatura pla UPC i altre que conté els de UPF. Cadascun d'aquests arbres estan referenciats per una posició d'una taula indexada, la d'índex 0 té la referència al contenidor de matriculats UPC i la d'índex 1 als de pla UPF.**
- **Cada objecte Estudiant emmagatzema:**
  - **Les dades personals de l'estudiant, nom, dni i pla d'estudis (UPC o UPF), i**
  - **Un magatzem amb els lliuraments que ha fet. Aquest és una seqüència enllaçada lineal sense capçalera de nodes. Cada node referencia a un objecte de la classe Lliurament. Els lliuraments que són exàmens han d'estar seguits i al final de la seqüència enllaçada i la resta, els que no ho són, a l' inici de la seqüència.**

Un objecte ProgramacioAvançada té el magatzem: matriculats



Els ACB estan implementats de la mateixa manera que la treballada a classe.
Cada node de l'ACB emmagatzema un objecte de la classe Estudiant. Per pròpia definició de l'estructura ACB no poden haver mots repetits.

Un objecte Estudiant té el magatzem: qualificacions



Tots són objectes de la classe Lliurament

Implementeu els mètodes pendents:

1.- **(1 punt)** El mètode **public void** addLliurament(Lliurament ll) que ha d'afegir un lliurament al magatzem de lliuraments d'un estudiant. Podeu suposar i per tant no cal que feu la comprovació, que el lliurament a afegir NO està al magatzem. **És imprescindible que els lliuraments que són exàmens estiguin ubicats al final de la seqüència, tots seguits, i la resta abans que aquests.**

2.- (1 punt) El mètode **public void** modificaLLiurament(**int** codi, **float** qualificacio) **throws** Exception que ha de modificar la qualificació d'un dels lliuraments d'un estudiant, el que té el codi indicat en el primer paràmetre. La nova qualificació ve donada en el segon paràmetre. Si el lliurament a modificar no està al contenidor s'ha de llençar una exception. Es suposa que la nova qualificació és correcte.

3.- (1 punt) El mètode **public double** calculaFinal() calcula i retorna la qualificació final de l'estudiant. La normativa d'avaluació a aplicar és 70% els exàmens i 30% la resta de lliuraments. Tots els exàmens tenen el mateix pes, a l'igual que la resta de lliuraments.

No es coneix ni el número d'exàmens ni els lliuraments no exàmens que es faran al llarg del trimestre. Però si sabem que per cada lliurament sol·licitat en l'assignatura hi ha un node a la seqüència enllaçada,

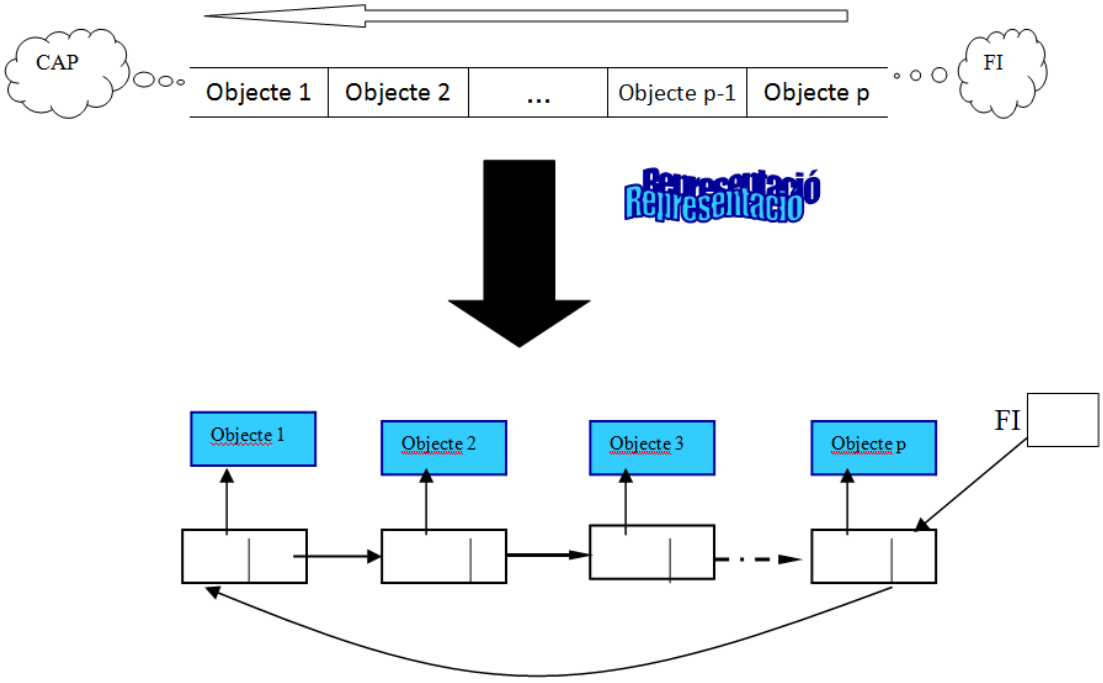
un suposat no lliurament d'un estudiant té un node amb qualificació de 0 i per tant s'ha de contemplar en els càlculs de la nota final.
Exemple: si es fan 2 exàmens i 5 lliuraments no exàmens la normativa és:
35%examen1+35%examen2+6%lliu1+6%lliu2+6%lliu3+6%lliu4+6%lliu5

4.- (0.5 punts) El mètode **public void** AfegirEstudiant(Estudiant e) **throws** Exception que ha de donar d'alta un nou matriculat en l'assignatura. S'ha d'afegir al corresponent magatzem. La situació anormal que cal controlar mitjançant el llançament d'una excepció és quan l'estudiant a afegir ja està al magatzem.
5.- (0.5 punts) El mètode **public void** EsborrarEstudiant(Estudiant e) **throws** Exception, que ha de fer l'operació contraria, donar-lo de baixa. Si no hi és cal llençar l'excepció.
6.- (1 punt) El mètode **public int[]** matriculatsDeCadaPla() que ha de retornar dins d'una taula el número d'estudiants matriculats en l'assignatura, separadament els que ho són del pla UPC dels que ho són del pla UPF.
7.- **(1.5 punts)** Redefinir el mètode **public** String toString(), ha de mostrar a pantalla, per cada grup d'estudiants, segons siguin UPC o UPF, **un llistat ordenat ascendentment per DNI** de matriculats. Per cada estudiant es visualitza únicament, nom, dni i qualificació final.
8.- (2 punts) El mètode **public double** obtenirNotafinal(**long** dni) **throws** Exception ha de localitzar l'estudiant amb dni indicat al paràmetre i retornar-ne la seva qualificació final. Si l'estudiant no hi és, voldrà dir que no està matriculat i per tant es llença una excepció.
És imprescindible per fer aquesta implementació que considereu el fet de que els elements dins d'un ACB estan ordenats, és dir **feu una cerca intel·ligent**.

Per implementar el mètodes de la classe ProgramacioAvançada **podeu afegir mètodes** a la classe AcbEnll i la classe NodeA, evidentment cal implementar-los i deixar molt clar en quina classe els ubiqueu.
No podeu afegir cap atribut en cap de les classes.

Exercici 2 (1.5 punts)

Continuem treballant amb la cua que heu implementat a la pràctica 2.



Especialitzem la nostra Cua, ara es vol que sigui **travessable**. Direm que un magatzem és travessable quan, independentment de la seva naturalesa –pila, cua, llista,...- permet ser interrogat sobre el seu

contingut i permet suprimir-ne elements de manera directa, sense haver de passar per les operacions pròpies del magatzem.

```
public interface Cua<E>{
        //operacions que caracteritzen una cua
}
public CuaEnll<E> implements Cua<E>{
        private class Node<E>{
                E inf;
                Node<E> seg;
        }
        protected Node<E> fi; //referència al node fi de la cua
        // implementació de les operacions
}
//especialització de la CuaEnll
public class CuaTravessable<E> extends CuaEnll<E> {
      //redefinició de mètodes
      //implementació de les operacions pròpies de la "travessibilitat"
      public  void eliminar (E element) throws Exception{
        //implementació
      }
}
```

Implementa el mètode **eliminar** que ha de treure de la cua travessable l'element especificat via paràmetre. Si aquest no està a la cua es llença una excepció.

Solució

Exercici 1

```
  public void addLliurament(Lliurament ll) {
      // Si examen darrera sino davant
      // no cal mirar si repe -- suposem no hi és
      // cas buida
      if (qualificacions == null)
          qualificacions = new node(ll);
      else {
          if (ll.getExamen()) {
              // inserim al darrera, fem recorregut
              node aux = qualificacions;
              while (aux.seg != null)
                  aux = aux.seg;
              aux.seg = new node(ll);
          } else { // inserim davant
              qualificacions = new node(ll, qualificacions);
          }
      }

  }
public void modificaLLiurament(int codi, float qualificacio)
        throws Exception {
    // cerca
    node aux = qualificacions;
    boolean trobat = false;
    while (!trobat && aux != null) {
        if (aux.ll.getCodi() == codi)
            trobat = true;
        else
            aux = aux.seg;
    }
    if (trobat) {
        aux.ll.setQualificacio(qualificacio);
    } else
        throw new Exception("Lliurament no existent");
}
```

```
public double calculaFinal() {
    // Exàmens repartir 70% i reste 30% no examens
    // Valoració eficiència només 1 recorregut
    float exam = 0.0F;
    float noExam = 0.0F;
    int quantsE = 0, quantsNoE = 0;
    // Recorregut
    node aux = qualificacions;
    while (aux!= null) {
        if (aux.ll.getExamen()) {
            quantsE++;
            exam += aux.ll.getQualificacio();
        } else {
            quantsNoE++;
            noExam += aux.ll.getQualificacio();
        }
        aux = aux.seg;
    }
    return (0.70 * (exam / quantsE) + 0.30 * (noExam / quantsNoE));
}

public void AfegirEstudiant(Estudiant e) throws Exception {
    if (e.getPla_UPF()) ((AcbEnll)matriculats[1]).Inserir(e);
    else ((AcbEnll)matriculats[0]).Inserir(e);
    //el mètode inserir ja controla la repetició.
}

public void EsborrarEstudiant(Estudiant e) throws Exception {
    if (e.getPla_UPF()) ((AcbEnll)matriculats[1]).Esborrar(e);
    else ((AcbEnll)matriculats[0]).Esborrar(e);
    //el mètode esborrar ja controla la no existència
}

public int[] matriculatsDeCadaPla(){
    //recorregut de cada arbre
    int q[]={0,0};
    q[0]=((AcbEnll)matriculats[0]).quants();
    q[1]=((AcbEnll)matriculats[1]).quants();
    return q;
}
```

Afegit classe AcbEnll

```
public int quants(){
 if (arrel==null) return 0;
 else return arrel.quants();
}
```

Afegit classe NodeA

```
public int quants(){
    //recorregut en preordre (es irrellevant quin). Comptar nodes
    int cont=1;
    if (esq!=null) cont+=esq.quants();
    if (dret!=null) cont+=dret.quants();
    return cont;
}
public String toString() {
   // llista alumnes UPF ordenats per dni amb la qualificació final
   //lista alumnes UPC ordenats per dni amb la qualificació final
   return ((AcbEnll)matriculats[1]).toString() + ((AcbEnll)matriculats[0]).toString();
}
```

Afegit classe AcbEnll

```
public String toString(){
    if (arrel==null) return "";
    return (arrel.toString());
}
```

Afegit classe NodeA

```
public String toString(){
    String r="";
    if (esq!=null) r+=esq.toString();
    r += inf.getDni() + inf.getNom()+"    "+inf.calculaFinal()+"\n";
    if (dret!=null) r+=dret.toString();
    return r;
}
```

```
public double obtenirNotafinal(long dni) throws Exception{
    //localitzar a l'arbre. Si no hi és exception
    //cal tenir present que l'arbre esta ordenat!!!!!
    double nota;
    try{
        nota= ((AcbEnll)matriculats[0]).obtenirNotaFinal(dni);
    }
    catch(Exception e){
        try{
            nota = ((AcbEnll)matriculats[1]).obtenirNotaFinal(dni);
        }
        catch(Exception h){
            throw new Exception("Alumne no matriculat");
        }
    }
    return nota;
}
```

Afegit classe AcbEnll

```
public double obtenirNotaFinal(long dni) throws Exception{
    //buscar element de l'arbre. LLançar excepcion si no hi es
    if (arrel==null) throw new Exception("NO hi és");
    return (arrel.obtenirNotaFinal(dni));
    //no fer mètode inordre ni res ....
}
```

Afegit classe NodeA

```
public double obtenirNotaFinal(long dni)throws Exception {
    //Cerca cal tenir en compte que l'arbre està ordenat
    if (inf.getDni()==dni){
        //calcula la nota final
        return(inf.calculaFinal());
    }
    else{
        if (inf.getDni()<dni){  //ha d'estar a la dreta
            if (dret==null)throw new Exception("No hi és");
            return (dret.obtenirNotaFinal(dni));
        }
        else{ //ha d'estar a l'esquerra
            if (esq==null)throw new Exception("No hi és");
            return (esq.obtenirNotaFinal(dni));
        }
    }
}
```

Exercici 2

```
public  void eliminar (E element) throws Exception{
        //treure un element d'un seqüència enllaçada lineal circular
        // la referència al darrer element es diu fi, és un atribut al que hi tenim accés
        if (fi==null) throw new Exception("La cua és buida. Element inexistent");
        //Cerca de l'element
        boolean trobat=false;
        Node<E> aux=fi;
        while (!trobat && aux.seg!=fi){
                if (aux.seg.inf.equals(E)) trobat=true;
                else aux=aux.seg;
        }
        if (trobat){
           aux.seg=aux.seg.seg;
        }
        else{
                //manca tractar el referenciat per fi, en la iteració no és tracta
                if (fi.inf.equals(E)){
                        if (fi==fi.seg) //esborrem el darrer element
                                fi=null;
                        else{
                                aux.seg=aux.seg.seg; fi=aux;
                        }
                }
                else throw new Exception("Element inexistent");
        }
}
```