:: Memòria ::

Ampliació de Programació Orientada a Objectes

PROGRAMA: Grau de separació entre URL's

ÍNDEX DE CONTINGUT

Nota al professor:

S'ha reutilitzat la documentació de la pràctica de l'any passat (Tecnologia Multimèdia) per a ampliar aquest document amb totes los novetats introduïdes per a la present assignatura de Ampliació de Programació Orientada a Objectes.

Els apartats 1 y 2 són els de la documentació original. Els següents apartats s'han ampliat a partir d'apartats de la documentació original (com l'apartat 3) o s'han fet íntegrament nous.

1 Enunciat original de la pràctica

A partir d'unes classes donades heu de crear una aplicació que permeti:

1. Donada una URL d'origen, una URL destí i un nivell de profunditat màxim definit per l'usuari, trobar el grau de separació mínim entre les dues url's, a partir dels enllaços de la URL d'origen.

2. Donada una URL d'origen, una paraula i un nivell de profunditat màxim definit per l'usuari, trobar el grau de separació mínima des de la URL origen fins una URL on aparegui la paraula, a partir dels enllaços de la URL d'origen.

3. Donada una URL d'origen, una URL destí, una paraula i un nivell de profunditat màxim definit per l'usuari, trobar el grau de separació mínim entre les dues url's, a partir dels enllaços de la URL d'origen, sense que aparegui la paraula en cap de les URL intermitges.

Les classes elaborades es troben dins un fitxer jar que s'haurà d'incloure al projecte Netbeans. S'han d'usar els següents mètodes:

- getLinks(). Torna un vector amb tots els links d'una URL.

- getText(). Torna el text d'una pàgina a partir de la seva URL.

2 Requeriments obligatoris

2.1 Algoritmes de cerca

- Donada una URL d'origen, una URL destí i un nivell de profunditat màxim definit per l'usuari, trobar el grau de separació mínim entre les dues url's, a partir dels enllaços de la URL d'origen.

```
static public void cercarSeparacioMinimaOpcio1(String URLorigen, String URLdesti, int profunditatMaxima) {
   if (profunditatMaxima>=0){
      Interficie.informarUsuari("S'analitzarà la URL "+URLorigen+" al nivell "+profunditatMaxima);
      if (!URLorigen.equals(URLdesti)) {
         //Backtracking
         Vector linksURLorigen = new AgafarUrl(URLorigen).getLinks();
         for (int i=0; i<linksURLorigen.size(); i++) {
            //Per cada link, comprovam que no l'haguem consultat ja, i si no és així
            //procedim a analitzar-lo. Després el guardam per a no tornar a consultar-lo.
            if (!linksComprovats.containsKey((String)linksURLorigen.get(i) + profunditatMaxima)) {
               cercarSeparacioMinimaOpcio1((String)linksURLorigen.get(i), URLdesti, profunditatMaxima-1);
               linksComprovats.put((String)linksURLorigen.get(i) + profunditatMaxima, URLorigen);
            }
         }
      } else {
         //Si arribam aqui és perque hem trobat una possible solució
         if (solucioFinal <= profunditatMaxima) {
            //Si és millor que la solució anterior la guardam
            solucioFinal = profunditatMaxima;
            GestorInterficie.setSolucioFinal(solucioFinal);
            cami.clear();//Borram el cami anterior ja que hem trobat una solució millor
            guardarCami = true;
            nivellSuperiorAGuardar = profunditatMaxima;
         }
      }
      if (guardarCami && (nivellSuperiorAGuardar == profunditatMaxima)) {
         cami.add(URLorigen);
         nivellSuperiorAGuardar++;
         GestorInterficie.setCami(cami);
      }
   }
}
```

El cas base d'aquest algoritme representa que s'ha arribat a la profunditat màxima establerta per l'usuari. Però aquest cas es troba implícit, ja que si es fan crides recursives només mentre la profunditat sigui menor a l'establerta s'assegura la finitud de les crides recursives. Per tant, aquest és el motiu de la primera condició de l'algoritme. Un cop s'ha assegurat la condició booleana anterior, es defineix el cas que general (no trivial) de no trobar solució. Si la URL de la cridada no coincideix amb la que ha especificat l'usuari per fer la cerca, s'agafen tots els enllaços continguts en aquesta. Aleshores es crida recursivament per tots aquests enllaços a un nivell de profunditat posterior. Per tant es decrementa la profunditat màxima (fins que es redueix a 0, aleshores es compleix la booleana inicial, i queden així definits tots els casos possibles excepte el de trobar solució). Un cop comprovat el link s'afegeix al conjunt de links comprovats per evitar futures comprovacions redundants que augmentarien l'arbre de solucions inútilment. Finalment, el cas de trobar solució queda definit pels casos restants a tots els casos anteriors. És a dir, si la URL de la cridada recursiva coincideix amb la URL especificada per l'usuari com a destí. Llavors, si la solució és millor que l'anterior s'agafa la nova i es guarda el camí de la millor solució.

- Donada una URL d'origen, una paraula i un nivell de profunditat màxim definit per l'usuari, trobar el grau de separació mínima des de la URL origen fins una URL on aparegui la paraula, a partir dels enllaços de la URL d'origen.

```
static public void cercarSeparacioMinimaOpcio2(String URLorigen, String paraula, int profunditatMaxima) {
   if (profunditatMaxima>=0){
      Interficie.informarUsuari("S'analitzarà la URL "+URLorigen+" al nivell "+profunditatMaxima);
      String textURL = new AgafarUrl(URLorigen).getText();
      if (!textURL.contains(paraula)) {
         //Backtracking
         Vector linksURLorigen = new AgafarUrl(URLorigen).getLinks();
         for (int i=0; i<linksURLorigen.size(); i++) {
            //Per cada link, comprovam que no l'haguem consultat ja, i si no és així
            //procedim a analitzar-lo. Després el guardam per a no tornar a consultar-lo.
            if (!linksComprovats.containsKey((String)linksURLorigen.get(i) + profunditatMaxima)) {
               cercarSeparacioMinimaOpcio2((String)linksURLorigen.get(i), paraula, profunditatMaxima-1);
               linksComprovats.put((String)linksURLorigen.get(i) + profunditatMaxima, URLorigen);
            }
         }
      } else {
         //Si arribam aqui és perque hem trobat una possible solució
         if (solucioFinal <= profunditatMaxima) {
            //Si és millor que la solució anterior la guardam
            solucioFinal = profunditatMaxima;
            GestorInterficie.setSolucioFinal(solucioFinal);
            cami.clear();//Borram el cami anterior ja que hem trobat una solució millor
            guardarCami = true;
            nivellSuperiorAGuardar = profunditatMaxima;
         }
      }
      if (guardarCami && (nivellSuperiorAGuardar == profunditatMaxima)) {
         cami.add(URLorigen);
         nivellSuperiorAGuardar++;
         GestorInterficie.setCami(cami);
      }
   }
}
```

Tant aquest algoritme com el següent són variants de l'algoritme anterior. L'únic que canvia són els paràmetres i el cas base de la cerca. És fan totes les comprovacions anteriors a excepció de la condició que implica haver trobat solució. Que en aquest cas consisteix en trobar el substring especificat per l'usuari contingut en l'enllaç de la cridada recursiva,

- Donada una URL d'origen, una URL destí, una paraula i un nivell de profunditat màxim definit per l'usuari, trobar el grau de separació mínim entre les dues url's, a partir dels enllaços de la URL d'origen, sense que aparegui la paraula en cap de les URL intermitges.

```
    static public void cercarSeparacioMinimaOpcio3(String URLorigen, String URLdesti, String paraula, int
profunditatMaxima) {
        if (profunditatMaxima>=0){
            Interficie.informarUsuari("S'analitzarà la URL "+URLorigen+" al nivell "+profunditatMaxima);
            String textURL = new AgafarUrl(URLorigen).getText();
            if (!textURL.contains(paraula)) {
                //Si la URL origen no conté la paraula prohibida procedim al seu anàlisi
                if (!(URLorigen.equals(URLdesti))) {
                    //Backtracking
                    Vector linksURLorigen = new AgafarUrl(URLorigen).getLinks();
                    for (int i=0; i<linksURLorigen.size(); i++) {
                        //Per cada link, comprovam que no l'haguem consultat ja, i si no és així
                        //procedim a analitzar-lo. Després el guardam per a no tornar a consultar-lo.
                        if (!linksComprovats.containsKey((String)linksURLorigen.get(i) + profunditatMaxima)) {
                            cercarSeparacioMinimaOpcio3((String)linksURLorigen.get(i), URLdesti, paraula,
profunditatMaxima-1);
                            linksComprovats.put((String)linksURLorigen.get(i) + profunditatMaxima, URLorigen);
                        }
                    }
                } else {
                    //Si arribam aqui és perque hem trobat una possible solució
                    if (solucioFinal <= profunditatMaxima) {
                        //Si és millor que la solució anterior la guardam
                        solucioFinal = profunditatMaxima;
                        GestorInterficie.setSolucioFinal(solucioFinal);
                        cami.clear();//Borram el cami anterior ja que hem trobat una solució millor
                        guardarCami = true;
                        nivellSuperiorAGuardar = profunditatMaxima;
                    }
                }
                if (guardarCami && (nivellSuperiorAGuardar == profunditatMaxima)) {
                    cami.add(URLorigen);
                    nivellSuperiorAGuardar++;
                    GestorInterficie.setCami(cami);
                }
            }
        }
        //El cas trivial es troba implícit
    }
```

Tal com s'ha dit abans, aquest algoritme és una variant del primer, també canvien els arguments i el cas base, que tot i comprovar el mateix que el primer, comprova a més que les URLs intermitges no contenguin la paraula prohibida per l'usuari.

2.2 Complexitat asimptòtica

L'ordre de complexitat de tots els tres algoritmes és el mateix. Es tracta d'un backtracking, el qual no acostuma a tenir costos asimptòtics baixos degut a ser una solució que necessita comprovar totes les solucions possibles per retornar la millor. Es tracta d'una complexitat O(n^m). La variable n és el nombre d'enllaços a analitzar i m representa el nombre de nivells màxim al qual es comprovarà el resultat de la cerca (la profunditat màxima).

3 Estructura de la pràctica

Per a la resolució de la pràctica s'ha tengut en compte el paradigma de la programació orientada a objectes que s'ha introduït al llarg del curs. Es tracta d'un paradigma que utilitza objectes i les seves interaccions per a dissenyar aplicacions i programes informàtics. Està basat en diferents tècniques com l'herència, l'abstracció, el polimorfisme i l'encapsulament de la informació.

En el cas concret d'aquesta pràctica s'ha estructurat l'aplicació en 5 classes:

- **Main.** Classe des d'on s'inicia el programa.
- **Interfície.** Representa el conjunt de components gràfics com menús, botons i finestres que serviran per a guiar qualsevol usuari de l'aplicació i plantejar-li un entorn de funcionament intuïtiu.
- **GestorInterficie.** Representa el conjunt d'accions que s'han d'executar en funció de l'estat de la interfície. Per tant, ha de gestionar les accions que fa un usuari a través dels components de la interfície i alhora les accions que han de dur a terme les altres classes dins el conjunt de l'aplicació. A més, és la classe encarregada de transferir les dades que introduirà l'usuari per mitjà de la interfície a la classe CalculadoraDistancies, que necessita aquesta informació per processar les tasques que li encomani el mateix usuari. És com un mediador entre la classe Interficie i CalculadoraDistancies, permetent una comunicació correcta entre ambdues.
- **CalculadoraDistancies.** Aquesta classe és l'encarregada de treballar amb diferents URLs i altres paràmetres convinguts a la pràctica de manera que realitza cerques i n'extreu resultats. Conté informació sobre el problema que es vol resoldre i els seus mètodes representen la resolució dels problemes plantejats a l'enunciat. Els resultats obtinguts els comparteix amb la classe GestorInterficie, qui s'encarregarà de cridar a la interfície perquè mostri aquests resultats als usuaris.
- **Memento.** Classe especifica per a emmagatzemar l'estat de l'aplicació de manera que després es pugui recuperar.

4 Contribucions addicionals de la pràctica original

- **Nomenclatura i convenis de java.**

Convenis de nomenclatura		
Identificador	**Convenció**	**Exemple del programa**
Paquets	El prefix d'un nom de paquet (únic) s'escriu sempre en minúscules de l'ASCII. Els components posteriors del nom del paquet varien en funció dels convenis de nomenclatura interns.	separaciourls;

Classes	Per als noms de classes posar la primera lletra en majúscules i les altres en minúscules. Cada paraula nova s'inicia amb majúscula.	GestorInterficie;
Mètodes	Els mètodes han de començar amb un verb i aquest en minúscula. Si té més d'una paraula, s'escriuen juntes i sempre amb la primera lletra en majúscula.	afegirComponents();
Variables	El nom d'una variable ha de ser breu però significatiu (significat mnemotècnic). No hauria de començar ni amb signe de subratllat ni $, tot i que estan permesos. S'alternen majúscules i minúscules començant per una minúscula.	profunditatMaxima();
Constants	Estan declarades com “static final”. Totes les seves lletres van en majúscula i la separació entre paraules es fa amb “_”.	NEW_LINE;

5 Contribucions afegides per a APOO

No sols s'han implementat patrons de disseny sinó que també s'ha millorat el programa en general tal i com s'explica a continuació:

- **Javadoc.**

Tal i com es va encomanar per a la pràctica de APOO, hem comentat tot el programa amb javadoc i hem generat aquest a partir del NetBeans per a poder veure'l des del browser. En concret s'han comentat les classes, constructors i mètodes.

El javadoc no es limita sols a explicar que fa cada mètode, classe, etc. sinó que també inclou tags per a enriquir-lo com @version, @author, @param, @return, @see, @deprecated i @throws.

El javadoc es troba dins el projecte Netbeans i es pot obrir des del programa.

**També s'han usat comentaris aclaratius d'una sola línia que es poden veure directament al codi. Aquests comentaris són per facilitar la comprensió del funcionament de l'aplicació.*

- **Taules de Hashing**

S'han afegit estructures de Hashing per tal de millorar el rendiment i l'eficiència de l'aplicació. Per a evitar entrar a bucles dins l'arbre de cerca que augmentin el cost computacional cada URL visitada es guarda per tal de no tornar-hi. Si empram una estructura ordenada d'accés seqüencial, cada cop que volguéssim fer la comprovació de si ja s'ha visitat una URL consumiríem O(n), essent n la quantitat d'elements de la llista.

Per a combatre aquest augment del cost computacional (explicat a l'apartat 2.2), a la nova versió del programa per a APOO s'han modificat els 3 algoritmes de cerca per a que funcionin amb les taules de hashing, que es troben properes a un cost asimptòtic de O(1), depenent del nombre de col·lisions i de l'aleatorietat de la funció de hash. Així doncs, aquest tipus d'estructures acceleren els processos de cerca segons una clau emprada per generar la posició pseudoaleatoria on residirà la informació a guardar. S'ha assignat una clau a les URLs comprovades segons la tupla d'informació: URL + nivell de profunditat comprovat. Aquest conjunt de dos valors assegura que la clau serà unívoca.

- **Altres contribucions: excepcions & finestres emergents & obrir arxius**

L'aplicació també fa front a possibles excepcions d'usuari durant el funcionament de l'aplicació. Alguns exemples són l'avís de que s'ha intentat iniciar una cerca amb un dels camps necessaris buit o també que una URL no pot contenir espais en blanc, de manera que s'informa a l'usuari. També s'han controlat altres tipus d'excepcions que poguessin ocórrer en temps d'execució: al obrir un arxiu, al mostrar codi HTML al panell dret, etc. Algunes d'aquestes excepcions es controlen per mitjà d'una classe anidada ExcepcioValorNul que hereda directament de Exception.

La interfície no consta d'una sola finestra, sinó que depenent de la situació que es doni durant l'execució del programa o la informació que vulgui consultar l'usuari, apareixeran diferents finestres emergents. Aquestes dotaran d'informació addicional com és el cas de la finestra que informa sobre la versió del programa i els seus autors o els missatges emergents.

A més, s'han especificat mètodes per a obrir arxius diversos amb les aplicacions predeterminades de l'usuari. El següent codi representa l'algoritme que apareix a la pràctica. És capaç d'obrir un arxiu i en cas de que es produeixi un error, informar a l'usuari de que no ha estat possible obrir-lo. Des del menú principal és possible obrir el pdf amb el manual d'usuari o bé el HTML del javadoc.

```
private void obrirArxiu() {
    try {
        File path = new File("dist/javadoc/index.html");
        Desktop.getDesktop().open(path);
    } catch (IOException ex) {
        Interficie.informarUsuari("No s'ha pogut obrir el fitxer.");
    }
}
```

6 Manual d'usuari

Des del menú superior 'Opcions' es pot consultar el 'Manual d'usuari', que inclou les instruccions necessàries per a poder usar el programa correctament.

També es pot consultar directament del document que es troba dins el projecte de Netbeans.

7 Patrons implementats

Després d'analitzar tots els patrons vists a classe, a més d'alguns altres per Internet, s'han implementat un total de tres que trobam que s'adapten millor al programa que hem reutilitzat. A més, cada un d'ells és d'un tipus diferent: de sistema, de comportament i de construcció.

El model-vista-controlador s'ha triat ja que d'aplicació gairebé obligatòria quan el teu programa té una GUI, ja que soluciona molts de problemes i ajuda a separar d'acord amb el seus criteris la implementació en 3 blocs diferents.

El patró memento s'ha triat per ser útil a l'hora de fer backups i rollbacks de manera que l'usuari gaudeixi de la possibilitat de recuperar un estat anterior del programa.

Finalment, el patró singleton s'ha aplicat també a la classe Memento, ja que tan sols es vol poder guardar un estat alhora i per tant ens basta amb una sola instancia de la classe. A més, volem poder accedir-hi globalment des de diferents llocs del programa.

Aquests 3 patrons es poden veure explicats a continuació:

7.1 Model-vista-controlador (patró de sistema)

Va ser el primer patró en implementar a la pràctica i a partir d'ell es van desenvolupar tota la resta de novetats introduïdes per a la versió de l'aplicació de Ampliació de Programació Orientada a Objectes. Amb la implementació d'aquest patró en primera instancia hem aconseguit que les classes estiguin estructurades segons el patró de manera que el desenvolupament de la resta de patrons i funcionalitats extres afegides ha resultat més senzill gràcies a aquesta estructura imposada.

El model-vista-controlador està format per un total de 3 parts:

- **Model (classe CalculadoraDistancies, classe Memento).** És el problema que tractam de resoldre. És a dir, es tracta del conjunt d'algoritmes que tracten les dades del nostre programa d'una manera específica (model de negoci). En el cas de la nostra aplicació en concret, el model representa la forma en que estan implementats els tres algoritmes de cerca.

- **Vista (classe Interficie).** És la presentació visual del programa, la interfície gràfica. Representa les dades del nostre model. Per a la realització de la interfície del programa hem usat la llibreria swing de Java.

- **Controlador (classe GestorInterficie).** No té a veure ni amb les finestres visuals ni amb amb les regles del nostre model, sinó que s'encarrega de prendre decisions quan rep esdeveniments. D'aquesta manera, les interaccions vista-model passen a través seu: invoca peticions al model i a la vista. El controlador ha de fer que les accions de la vista provoquin canvis en el model i que aquests canvis siguin visualitzats a la vista. En aquest sentit, és com una interface: adapta la vista al model i viceversa.

7.2 Memento (patró de comportament)

Representa la imatge de l'estat d'un objecte (objecte que es troba declarat com a privat). Quan es necessita emmagatzemar l'estat d'un objecte, el patró permet transmetre l'estat al Memento (classe Memento del nostre projecte, independent de la classe a la que pertany l'objecte). Posteriorment, -i des de l'objecte original-, podem recuperar l'estat del propi objecte en el punt anterior en que va ser guardat. La nostra classe Memento guarda les variables de GestorInterficie necessàries per a recuperar l'estat del programa, però tan sols les variables necessàries, no totes (eficiència).

7.3 Singleton (patró de construcció)

Permet tenir una única instancia d'una classe (en el nostre cas la classe a la que se li ha aplicat el patró singleton és Memento) dins el programa, però a la vegada totes les classes del programa hi poden tenir accés. Això és possible gràcies a que el constructor és marcat com a private i existeix un mètode static per recuperar la instancia de Memento quan se la necessita. La creació pot fer-se al carregar la classe o quan se la necessita. Nosaltres hem optat pel darrer cas, ja que així guanyam eficiència: computacionalment i en quant a memòria, ja que pot ser que l'usuari mai faci un backup de l'estat del programa (inicialització sota demanda).

Un inconvenient del patró és que en un entorn multithreated pot haver-hi problemes de concurrència. Per exemple, dos threats cridant alhora al constructor des de dins la classe. Tot i que actualment en el programa mai hi ha més d'un threat executant aquesta porció de codi, ens hem cobert l'esquena per si en futures revisions del programa sí és així. La solució passa per usar la paraula reservada synchronized. En cas de que dos threats executassin alhora la instrucció se'ls hi tornaria dues referencies al mateix objecte.

8 Conclusions de la pràctica original

La pràctica final de l'assignatura ens ha servit per a consolidar molts dels conceptes vists al llarg del curs com puguin ser: les interfícies gràfiques d'usuari i la programació dirigida per esdeveniments; les implicacions de la POO (amb la corresponent estructura de classes); la recursivitat, en concret el backtracking; i també l'anàlisi de cost computacional dels algoritmes de cerca.

Precisament amb l'anàlisi de cost computacional hem comprovat que la tècnica de backtracking és molt ineficient ja que consisteix en recórrer tots els possibles camins i estudiar tots els possibles casos. Conseqüentment l'eficiència hauria de ser una de les metes que s'hauria de plantejar tot programador. No sols per la necessitat d'obtenir una resposta ràpida sinó per l'ús eficient dels recursos que pugui oferir un computador.

9 Conclusions de la pràctica de APOO

Aquesta pràctica ha ajudat a introduir-nos dins el món dels patrons de disseny. Hem vist que hi ha diversos nivells d'abstracció a l'hora de reutilitzar codi per a estalviar temps. Amb els patrons de disseny se'ns ha mostrat un nou nivell d'abstracció superior, que va més enllà del simple copy-paste de fragments de codi, la reutilització d'algoritmes o l'ús de llibreries, per citar alguns exemples. Un patró de disseny és un model reusable i efectiu que es pot aplicar recurrentment al llarg de diversos problemes que comparteixin un mateix context.

Així doncs, se'ns ha introduït el tema dels patrons de disseny i ara tenim una visió general d'aquests patrons explicats a classe, de manera que quan programem un projecte amb un determinat context podrem decidir usar-los si existeix un patró propici que ens solucioni problemes del projecte.