LAB3: DOCUMENTATION

Maria Beili Mena Dorado (u199138)
Mireia Pou Oliveras (u198721)

1. Description of the problem

Aquesta tercera pràctica consisteix a implementar l'estructura d'una organització que conté seus, socis, membres, etc., juntament amb els territoris en què es troba. A més a més, també hem hagut d'implementar les funcionalitats que permeten registrar nous membres a l'organització, així com aquelles relacionades amb la proposta d'actuacions i gestions.

Per a fer-ho, se'ns ha proveït uns fitxers xml (amb el codi de Utility) que contenen les regions i ciutats, els delegats i les seus.

Un cop implementats els diferents mètodes i classes, s'haurien de visualitzar tres coses diferents:

- A la terminal: En primer lloc, el nom de l'organització i una llista de les seves seus ("headquarters"). En segon lloc, una llista de regions, on cada regió conté les seves ciutats (amb les poblacions) i, dintre d'aquestes, les seus que es troben en elles i el delegat principal de les seus.
- A la terminal: Al final de tot, el nom “Mathematical” que correspon al propòsit d’una acció.
- A la carpeta Images: dos fotografies de QRs, una per “Simone” i una per “Samantha”.

Per poder fer la tasca hem hagut d’implementar 12 classes amb els seus diferents mètodes:

1. Organization: Està composta per Headquarters i aquests per una llista amb els caps de cada regió.

2. Headquarter: Cada oficina pertany a l’organització que hem anomenat “UPF” (composició), situada en una ciutat o varies amb els seus diferents membres. A més, aquests tenen assignat el “cap” de l’oficina.

3. Region: Té una llista amb les diferents ciutats per cada regió.

4. City: Pertany a una regió (agregació) amb el nombre de població i una llista de les oficines que hi ha a cada ciutat.

5. Member: Està relacionada amb availability i aquests estan associats a un headquarter (agregació). Que estan dividits en dos tipus: els regulars i el delegat, hereten atributs i membres.

6. Regular: Cada membre regular pot tenir un vehicle que pot ser una bicicleta, un cotxe o un ciclomotor. A més, un d'ells pot ser el cap.

7. Delegate: Cada cap d’un headquarter pot proposar noves accions per l’organització, que seran els mètodes, com inscriure nous membres habituals i delegats mitjançant

imatges de codis QR. Com que Delegate hereda els atributs del pare, en el seu constructor utilitzarem la funció super() per assignar els seus atributs.

8. Availability: Tots els membres de l'organització estan disponibles per realitzar accions un nombre de dies de la setmana a algunes hores.

9. Vehicle: Està relacionat amb els membres regulars. Definim el tipus de transport amb la seva capacitat.
10. Action: Realitza les accions de l'organització que vol executar.

11. InfoAction: és la classe associada als Headquarter i Action per guardar la informació que correspon als participants de cada headquarter per cada acció.

12. Finalment, hem creat la classe TestDelegate que inclou els mètodes principals. Per poder fer-ho la classe llegirà els arxius XML per omplir l'estructura de l'organització i el territori amb la classe Utility. Primer, hem creat les instàncies de Region i City. Després d'una nova de Organization, Headquarter i per últim la instància de Delegate.

Per cada classe hem hagut de definir mètodes:

Mètodes gets, sets i adds.

```
public void addCity(City c){
    cityList.add(c);
}

public Organization getOrganization(){
    return organization;
}

public void setHead(Delegate d){
    head = d;
}
```

Per a totes les classes que tenen llistes, les inicialitzem als constructors. A més, creem per a la majoria de les classes els mètodes get(), set () i add() corresponents, ja sigui per afegir atributs a la llista o retornar-los.

També hi ha mètodes get()/ set() per a agafar i posar atributs d'una classe, encara que no siguin llistes.

Vehicle/ Availability

Per aquestes classes hem declarat els constructors i atributs i ja està. Per la classe Availability passem les llistes per paràmetre i no és necessari inicialitzar-les (com als altres constructors) ni fer sets/gets/adds.

Mètodes toString()

```
public String toString(){
    return name;
}
```

Els hem implementat a les classes Member, City i Delegate per a poder utilitzar la funció Utility.getObject(). En el cas de Delegate, hem retornat: super.getname(), ja que el nom està guardat a la classe pare.

Main - Test Delegate

A l'apartat de "Test Delegate" hi trobem el main. En aquest, llegim els diferents fitxers i fem les comprovacions necessàries per a assegurar-nos que el codi funciona.

Llegir els fitxers proporcionats és el primer pas perquè la pràctica funcioni correctament. Sempre que hem hagut de llegir fitxers ho hem fet de la següent forma:

```
LinkedList<String[]> regions = Utility.readXML("region");

        for(String[] array: regions){
```

És a dir, creant una LinkedList amb el contingut del fitxer i després iterant sobre aquest array.

1. Region

El primer fitxer que hem llegit és el de Region, que conté tota la informació referent a les regions i ciutats. Un cop estem dintre el bucle d'iteració de l'array, hem creat una regió amb el seu nom (array[0]) i hem creat una llista de ciutats buida. Com que no totes les regions tenen el mateix nombre de ciutats, hem hagut de guardar la informació en forma de bucle.

La informació està guardada a la linkedlist del principi com a (regió, nom ciutat 1, nom ciutat 2, …, població ciutat 1, població ciutat 2, …). Per a poder guardar la informació tenint en compte aquesta disposició, hem creat dues llistes diferents: una pels noms i una altra per les poblacions. Per tant, també hem guardat la informació en dos bucles diferents. El primer va d'1 (perquè el 0 era el nom de la regió) fins a middle+1 (meitat de l'array +1) perquè hem de tenir en compte que el 0 ja l'hem posat. El segon va de middle+2 fins al final. D'aquesta forma evitem igualar un string a integer o al revés. Un cop tenim les dues llistes (que són de la mateixa mida), iterem sobre elles per a crear les ciutats. Finalment, afegim la llista de ciutats a la regió i la regió a la llista de regions.

```
LinkedList<String[]> regions = Utility.readXML(type: "region");
for(String[] array: regions){

    Region region = new Region(array[0]);
    LinkedList<City> cities = new LinkedList<City>();
    int length = array.length;
    int middle = (length-1)/2; //-1 perquè array[0] és la regió

    LinkedList<String> city_names = new LinkedList<String>();
    for (int i=1; i<(middle+1); i++){
        String cityname = array[i];
        city_names.add(cityname);
    }

    LinkedList<Integer> city_population = new LinkedList<Integer>();
    for(int i=(middle+1); i<length; i++){
        int population = Integer.parseInt(array[i]);
        city_population.add(population);
    }

    for (int i=0; i<city_names.size(); i++){
        City city = new City(city_names.get(i), city_population.get(i));
        cities.add(city);
    }

    region.setCities(cities);
    regionsList.add(region);

}
```

2. Organization

Realment aquí no hem de llegir cap fitxer, però la segona cosa que hem fet ha sigut crear una instància d'organització amb un nom inventat. Després, hem creat una llista de cities (treta de region) per a poder buscar-les en el següent fitxer que havíem de llegir.

```
organization = new Organization(n: "UPF");
LinkedList<City> citiesList = new LinkedList<City>();
for (int i=0; i<regionsList.size(); i++){
    LinkedList<City> region_cities = regionsList.get(i).getCities();
    for (int j=0; j<region_cities.size(); j++){
        City city = region_cities.get(j);
        citiesList.add(city);
    }
}
```

3. Headquarters

```
LinkedList<String[]> headquarters = Utility.readXML(type: "headquarter");
for (String[] array: headquarters){
    Headquarter headquarter = new Headquarter(array[0], array[1], organization);
    for(int i=2; i<array.length; i++){
        City city = Utility.getObject(array[i], citiesList);
        headquarter.addCity(city);
        city.addHQ(headquarter);
    }

    organization.addHeadquarter(headquarter);
}
```

Un cop llegit el fitxer de headquarters i posat a la linkedList, simplement hem creat un headquarter (afegint-li l'organització prèviament instanciada). Cada headquarter conté les ciutats a les quals està, però com que el nombre de ciutats és diferent per a cada un, hem iterat sobre elles en un bucle.

Per a cada ciutat, amb la funció getObject (de Utility.java) la busquem a la llista creada al punt anterior. Després afegim la ciutat al headquarter i el headquarter a la ciutat. Un cop ha acabat el bucle i ja tenim totes les ciutats, afegim el headquarter a l'organització.

4. Head Delegate

```
headsList = new LinkedList<Delegate>();

LinkedList<String[]> heads = Utility.readXML(type: "head");
for (String[] array: heads){
    int phone = Integer.parseInt(array[1]);
    Headquarter headOf = Utility.getObject(array[3], organization.getHQList());
    Delegate delegate = new Delegate(array[0], phone, array[2], headOf);
    delegate.setHeadOf(headOf);
    headOf.setHead(delegate);

    String[] head_days = array[4].split("\\.");
    List<String> days = Arrays.asList(head_days);

    String[] head_hours = array[5].split("\\.");
    List<String> string_hours = Arrays.asList(head_hours);
    List<Integer> hours = new ArrayList<Integer>();
    for (int i=0; i<head_hours.length; i++){
        hours.add(Integer.parseInt(string_hours.get(i)));
    }

    Availability delegate_availability = new Availability(days, hours);
    delegate.setAvailability(delegate_availability);

    headsList.add(delegate);
}
```

Finalment, ens falta llegir el fitxer de delegats. Un cop ho hem fet (amb el Utility.readXML) iterem sobre la LinkedList on l'hem guardat.

Primerament, hem buscat el headquarter corresponent (array[3]) a la llista de headquarters i l'hem utilitzat per a crear el delegat (juntament amb la informació necessària) i per al headOf.

Un cop hem creat el delegat, necessitem buscar la seva disponibilitat, que es troba a l'array[4] i l'array[5]. Com que les hores i els dies estan separats per punts en una mateixa línia, hem fet servir les funcions "split()" i "asList()" per a poder separar-los i guardar-los en una llista. Per a les hores, a més, hem hagut de fer un bucle for per a passar-les a "integer", ja que eren "strings".

Un cop tenim les llistes, hem creat una instància de disponibilitat amb els dies i les hores i l'hem afegida al delegat. Finalment, afegim el delegat a la llista de delegats del main.

!! Els mètodes print i les comprovacions estan en altres apartats !!

<u>Mètodes QR</u>

Per a fer els QR hem hagut de generar quatre funcions diferents (ubicades a Delegate.java):

- genDelegateQR(): genera una imatge de QR de delegat.
- genRegularQR(): genera una imatge de QR de regular.

- singUpDelegate(): per a afegir el delegat creat a l'organització.
- singUpRegular(): per a afegir el regular creat a l'organització.

1. genDelegateQR() / genRegularQR()

```
public Image genDelegateQR(String path){
    String path_i = "Images/" + path + ".jpg";
    Image i = new Image(path_i, width: 50, height: 50);
    String delegate_name = super.getName();
    String text = "This is a QR for a Delegate Member. You don't have to car
    text = text + delegate_name;
    BitMatrix qr = QRLib.generateQRCodeImage(text, width: 50, height: 50);
    i.setBitMatrix(qr);
    return i;
}
```

Les funcions per generar imatges de delegats i regulars són pràcticament idèntiques. La única cosa que canvia és el text que conté el QR.

Per a fer-les, en primer lloc, generem el "path" de la imatge (a on es guardara) i creem una instància d'imatge amb aquest path i unes mesures inventades. Tot seguit, creem el text que s'ha d'afegir (diferent si és delegat i si és regular) i afegim el nom del creador al text. Finalment, creem un QR amb el text i les mateixes mesures que la imatge, l'afegim a la imatge i retornem la imatge.

2. singUpRegular() / singUpDelegate()

```
public boolean singUpDelegate(Delegate d, Image i, Headquarter h){
    BinaryBitmap b = i.getBitmap();
    String delegate_name = super.getName();
    String text = QRLib.decodeQRCodeImage(b);
    String delegate_text = "This is a QR for a Delegate Member. You don
    delegate_text = delegate_text + delegate_name;

    if(! headOf.equals(h)){
        System.out.println("El delegat no pertany a la seu");
        return false;
    }

    if (! delegate_text.equals(text)){
        System.out.println("El QR no és de delegat");
        return false;
    }

    i.save();
    headOf.addMember(d);
    return true;
}
```

Les funcions de singUpDelegate() i singUpRegular() també són idèntiques, exceptuant el text.

En primer lloc, hem de comprovar si el text és el que es correspon amb delegate i regular. Per això, tornem a generar el text de la mateixa forma que a la funció anterior i, després, descodifiquem el text del QR.

Per a comprovar si es correspon mirem si són iguals i, si no ho són, retornem fals.

De la mateixa manera, volem comprovar si el creador pertany a la seu on volem afegir el membre (sinó no el podríem crear). Per això, passem la seu per paràmetre i comprovem que sigui igual al headOf del creador. Si no ho és, també retornem fals.

Finalment, un cop s'han comprovat totes les condicions, guardem la imatge fent imtage.save(), afegim el membre a la seu i retornem true.

<u>Mètodes print()</u>

```
//Primer imprimim la organització amb els headquarters per mirar que s'ha agafat bé la informació
organization.printOrganization();

//Després imprimim la informació per regions.
for (int i=0; i<regionsList.size(); i++){
    regionsList.get(i).printRegion();
}
```

La finalitat dels mètodes print() que hem implementat és poder comprovar que els fitxers xml s'han llegit correctament i que la informació s'ha guardat on li pertoca. Per a poder imprimir a la terminal (des de "**TestDelegate**")hem creat un seguit de mètodes que es van cridant entre ells:

1. printOrganization()

```
public void printOrganization(){
    System.out.println("Organization:" + name);
    System.out.println("    Headquarters:");
    for (int i=0; i<placesList.size(); i++){
        System.out.println("        " + placesList.get(i));
    }
    System.out.print("\n");
}
```

La primera cosa que s'imprimeix a la terminal és la organització amb els seus headquarters. Per a aconseguir això només hem hagut d'implementar un mètode a la classe Organització "printOrganization()" i cridar-lo a Test Delegate. Aquest mètode, l'única cosa que fa és imprimir el nom de l'organització i la seva llista de seus (amb el bucle for).

2. printRegion()

La segona cosa que s'imprimeix a la terminal són les regions, amb les seves ciutats, i les seus i delegats de cada una d'aquestes ciutats. Per a poder aconseguir això, al contrari que amb organització, hem hagut de crear una varietat de mètodes:

```
public void printRegion(){
    System.out.println("\nRegion:" + name);
    for (int i=0; i<citiesList.size(); i++){
        citiesList.get(i).printCity();
    }
}
```

En primer lloc, printRegion(), que es el que cridem des de Test Delegate (i ubicada a Region.java). Aquesta funció printeja el nom de la Regió i després, per a cada ciutat (bucle for) crida a un mètode per imprimir-la:

```
public void printCity(){
    System.out.print("      City name:" + name + "\n");
    System.out.print("      Population:" + population + "\n");
    System.out.print("      Headquarters:");
    for (int i=0; i<hqsList.size(); i++){
        hqsList.get(i).printHeadquarter();
    }
    System.out.print("\n");
}
```

PrintCity() (City.java) printeja el nom i la població de la ciutat. Un cop ho ha fet, per a cada "headquarter" que conté, crida a una funció per a imprimir-lo.

```
public void printHeadquarter(){
    System.out.print("  " + name + "\n");
    System.out.print("          Head:");
    head.printDelegate();
}
```

PrintHeadquarter() (Headquarter.java) printeja el nom de cada seu. A més a més, crida a una funció per a imprimir el nom del cap de la seu.

```
public void printDelegate(){
    System.out.print(super.getName() + "\n");
    super.printAvailability();
}
```

PrintDelegate() (Delegate.java) printeja el nom del Delegat principal del Headquarter. Utilitza super perquè membre és la seva classe pare. Després crida a un altre mètode per a imprimir la seva disponibilitat. Com que la disponibilitat també es troba a la classe pare, la funció va presidida per super.

```
public void printAvailability(){
    available.printDays();
    available.printHours();
}
```

PrintAvailability() (Member.java) que simplement crida a dues funcions ubicades a la classe "availability.java" per obtenir els dies i les hores que el delegat està disponible.

```
public void printDays(){
    System.out.print("            Days available:" + "\n");
    System.out.print("                ");
    for (int i=0; i<daysList.size(); i++){
        System.out.print(daysList.get(i) + ", ");
    }
    System.out.print("\n");
}

public void printHours(){
    System.out.print("            Hours available:" + "\n");
    System.out.print("                ");
    for (int i=0; i<hoursList.size(); i++){
```

Les dues funcions cridades anteriorment que simplement imprimeixen cada dia disponible a la llista de dies i cada hora disponible a la llista de hores.

Un cop s'ha cridat les dues funcions printOrganization() i printRegion() des de Test Delegate, a la terminal surt això:

```
Region:Tarragonès
      City name:Tarragona
      Population:132299
      Headquarters:  Sud
          Head:Pau Muñoz
              Days available:
                  dimarts, dijous, divendres,
              Hours available:
                  18, 19, 20, 21,
```

```
Organization:UPF
    Headquarters:
        Santsenc
        Gracienc
        Barceloní
        Nord
        Sud
        Costaner
        Central
        Llobregat
        Ponent
        VallOriental
        VallOccidental
```

Mètodes Action(), InfoAction() - Part Opcional

Per a action() i infoaction() hem implementat els mètodes, el constructor i atributs corresponents a les classes. A més, hem implementat alguns mètodes i atributs a Headquarter i Delegate.

- Headquarter: hem dissenyat els mètodes signUpAction() creant un info d'InfoAction per després afegir-la a les accions dels participants de l'oficina. També getAction() on mirem que les accions coincideixen.
- Delegate: hem implementat proposeAction() i signUpAction(), utilitzant la d'Headquarter per poder fer-la funcionar, ja que l'head és l'únic que pot realitzar i iniciar accions.

Comprovació - Main

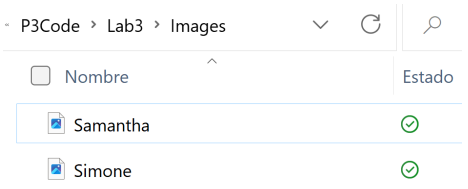
A part de la part dels prints() per a comprovar que l'estructura estigui ben implementada, al main hi podem trobar una altra part per a comprovar les funcions de QR i les funcions de action(), infoaction().

1. QR

```
Delegate creator = Utility.getObject(desc: "Dolors Gil", headsList);
Delegate delegate = new Delegate(n: "Simone", p: 567891234, e: "simone@upf.edu", creator.getHeadquarter());
Image i = creator.genDelegateQR(path: "Simone");
creator.singUpDelegate(delegate, i, creator.getHeadquarter());

Regular regular = new Regular(n: "Samantha", p: 567891234, e: "samantha@upf.edu", creator.getHeadquarter(), creator);
Image regular_i = creator.genRegularQR(path: "Samantha");
creator.singUpRegular(regular, regular_i, creator.getHeadquarter());
```

Per a fer la comprovació dels QR hem dut a terme un petit codi. Hem escollit un delegat qualsevol i l'hem buscat a la llista de delegats de "Test Delegate" amb Utility.getObject(). Després hem creat un delegat i un regular inventats (nom, telèfon i correu inventats, però

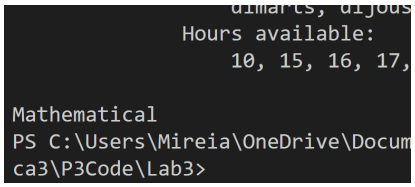


amb la seu del creador). Un cop estan creats hem generat una imatge de QR per a cadascun i l'hem afegit a l'organització. Hem comprovat que el codi funciona perquè ens han aparegut dues imatges a la carpeta "Images".

2. Action(), InfoAction()

```
Date d = new Date();
Action a = new Action(p: "Mathematical", d, d: 60);
a.addHeadquarter(creator.getHeadquarter());
creator.getHeadquarter().singUpAction(a, members: 20, nonmembers: 4, press: false);
creator.proposeAction(a);
System.out.println(organization.getAction(d).getPurpose());
```

Per a comprovar que els mètodes de action() i infoaction() estiguin ben fets també hem fet un petit codi. Primer hem creat una data i, amb aquesta, hem instànciat una acció inventada (de propòsit matemàtiques). Aleshores, hem afegit la seu a l'acció i l'acció a la seu (un altre cop, amb membres inventats). Finalment, l'hem proposat a l'organització. Sabem que ha funcionat perquè hem imprès el propòsit de l'acció ubicada a l'organització i s'ha imprès: "Mathematical".



2. Alternative solution discussed

En aquesta pràctica hem tingut molts dubtes, sobretot a l'apartat de main.

- A l'hora de crear l'organització no sabíem si havíem d'inventar el nom o no, ja que no se'n proporciona cap.
- A l'hora d'agafar la disponibilitat al fitxer de delegats, no sabíem com desfer l'array de dies i hores per poder-los posar en una llista. Vam discutir diverses formes de fer-ho fins que finalment vam fer servir les dues funcions split() i asList().
- A l'hora de crear el delegat no sabíem si s'havia d'utilitzar la mateixa seu al constructor que al headOf.
- A l'hora d'obtenir les ciutats del fitxer de regions vam haver de fer diferents bucles abans de trobar el correcte. El principal problema que hi havia és que la disposició de ciutats a la linkedList no era la mateixa que al fitxer → al fitxer hi havia ciutat 1 població 1, ciutat 2, població 2, ... i a la llista hi havia ciutat 1, ciutat 2..., població 1, població 2...,

Un cop vam resoldre tots els punts explicats, no ens funcionava el main. Ens havíem oblidat de crear les funcions toString() per aconseguir objectes amb el Utility.Object().

D'altra banda, vam tenir qüestions amb les funcions per crear QRs. No enteníem exactament com fer les funcions singUpDelegate()/ singUpRegular().

3. Conclusion

En resum i mirant-la globalment, creiem que la pràctica ha estat ben implementada. Els mètodes i els atributs han estat fets tal com consta a l'enunciat i el test ha tingut resultats positius.

És cert i en som conscients, però, que no hem tingut en compte cap mena d'error a l'hora de l'input. És a dir, si una funció necessita un "Headquarter" i li passem una "Region" la pràctica donarà error i ja està, no hi haurà cap opció per canviar-ho ni al codi s'ha implementat res per evitar-ho.

Tots els problemes que hem tingut estan al punt 2.