Marina Castellano
Mar Moreno

INFORME LAB 3: IMPLEMENTING AN APPLICATION USING LIBRARIES AND INHERITANCE

En aquest laboratori hem implementat la gestió d'organització que vam dissenyar durant el seminari 3. L'objectiu és crear les classes que formen part del projecte, les relacions que hi han entre elles, i implementar els QRs de Delegate i Regular.
Les classes estan organitzades per la classe Headquarter, la qual és la principal, i dintre de cada classe hem implementat els atributs i mètodes necessaris.
La pràctica conté una classe anomenada Utility i un directori xml que conté els fitxers dels quals extreurem la informació que utilitzarem en les altres classes.

CLASSES

Les classes implementades durant aquest laboratori han estat: Organization, Availability, Headquarter, City, Region, Member, Delegate, Regular, Vehicle, Image, QRLib.

- **Organization**: Els atributs de Organization son: name (string), una llista de headQuarters i una llista de action.
  Els mètodes son Organization(constructor), setPlace i getMembershplist. Hem creat la funció getMembership perquè retorni els heads de headquarter, per després imprimirlos en TestDelegate, i una funcio setPlaces a la qual li passarem una llista de headquarters per saber els llocs on s'assenten les seus.
  En el constructor inicialitzem els atributs i els paràmetres.

- **City:** Els atributs de City son: name (string), population (int), i una linked list de headquarter.
  Els métodes son: el constructor City, a la qual li hem passat el nom i la població, i la funció addHeadquarter la qual li passem el headquarter que s'ha creat i l'afegim a la llista headquarterList.

- **Region:** Els atributs de region son: name i una linkedlist de city. Els mètodes son: el constructor region, al qual li hem passat el nom, setCities, i getCityList.

- **Member:** Els atributs d'aquesta classe son: name (string), phone (int), email (string), availability i headquarter. Els mètodes son: Member(constructor), al qual li hem passat el name, phone, email i headquarter; setAvailability i Headquarter.

- **Headquarter:** És la classe principal. Els seus atributs son: name (string), email (string), una linkedlist de member, head, organization i una linkedlist de city.
  Els mètodes implementats son: El constructor Headquarter, al qual li hem passat, el name, email i organization; addMember, per afegir un membre a la llista memberList; getOrganization, que retorna organization; setHead, al qual li passem head; getHead, que retorna head i setCityList, que retorna la llista cityList.

En les classes City, Headquarter, Member, Organization i Region hem implementat la funció toString, que retorna els objectes com un string per després poder afegir-lo a les llistes d'strings.

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

- **Delegate:** La classe està formada pels atributs: una llista dels membres regulars, regularList i un headOf de la classe Headquarter.
  En aquesta classe és on generarem els QRs i els assignarem els textos corresponents amb les dades.
  En el constructor hem utilitzat la paraula clau super en la primera línea per cridar als parametres que estan a la superclase de la subclasse.
  Delegate és una subclasse de Member i podem cridar a als seus paràmetres ja inicialitzats.

```
public Delegate(String n, int p, String e, Headquarter h){
    super(n,p,e,h);
    regularList = new LinkedList<Regular>();
}
```

Per poder fer això, quan creem la classe utilitzem la paraula extends i la superclasse, en aquest cas Member, ja que un Delegate és un tipus de Member.

```
public class Delegate extends Member{
```

Hem creat dos funcions setHeadOf i addDependance.
La primera: setHeadOf, donat un headOf de Headquarter li donen el valor que obtenim al atribut headOf que tenim. I la funció addDependence donat un membre regular afegeix el membre a la llista de regularList que tenim com a atribut.

Per generar les imatges creem les funcions genDelegateQR i genRegularQR, les quals retornaran les imatges amb els QR que els hi correspon a cadascuna. A les funcions els hi passem els QR que hauran de contenir.
El primer que hem fet és crear una nova imatge de la classe Imatge, on li passarem el path i les dimensions que haurà de tindre la imatge. En el path li hem posat una carpeta nova que hem crear, QR perquè es guardin alla les imatges dels QR que es creein, i el nom que els hi posarem als QRs, per els delegats delegateQR.png i a els regulars regularQR.png. En el nostre cas li hem posat que 300 tant per alçada com en amplada. Després hem creat una variable String que sera el text que haurà de contenir el QR que ens demanen en cada cas. Per delegate ha de contenir: `"This is QR for a Delegate Member. You don't have to care about rising sea levels, if you live on a mega yatch."`, i li hem afegit també el this.getName() perquè a l'hora de que surti també aparegui el nom del delegat que ho ha creat (en la imatge no apareix perquè no cabía en la imatge). Per els membres regulars hem afegit el text: `"This is a QR for a Regular Member. Climate change doesn't matters, if you stay indoors."` Després hem creat el QR amb el text que li passem, amb les mateixes dimensions de la imatge amb la funció de la classe donada QRLib generateQRCodeImage, el qual sera un BitMatrix per després cridar a la funció setBitMatrix de Image amb la imatge creada.

```java
public Image genDelegateQR(QRLib qrLib){
    Image image = new Image(path: "src/QR/delegateQR.png", width: 300, height: 300);
    String text = "This is QR for a Delegate Member. You don't have to care about rising sea levels, if you live on a mega yatch." + " "+
    BitMatrix bm = QRLib.generateQRCodeImage(text, width: 300, height: 300);
    image.setBitMatrix(bm);

    return image;
}

public Image genRegularQR(QRLib qrLib){
    Image image = new Image(path: "src/QR/regularQR.png", width: 300, height: 300);
    String text = "This is a QR for a Regular Member. Climate change doesn't matters, if you stay indoors.";
    BitMatrix bm = QRLib.generateQRCodeImage(text, width: 300, height: 300);
    image.setBitMatrix(bm);

    return image;

}
```

Implementem els mètodes singUpDelegate i singUpRegular per crear nous membres en l'Organització. Li passem els paràmetres Delegate i Regular respectivament, i el QR i la imatge. Primer recuperem el text que hi ha en en el QR utilitzant el mètode decodeQRCodeImage de la classe QRLib per poder comprobar que el text que hi ha coincideix amb el que ha de contenir. En cas de que coincideixi afegirem el membre i retornarà true, en cas contrari retornarà false.

```
public boolean singUpDelegate(Delegate d, QRLib q, Image i){
    String textqr = QRLib.decodeQRCodeImage(i.getBitmap());
    String text = "This is QR for a Delegate Member. You don't have to care about rising sea levels, if you live on a mega yatch." + this.g
    if(textqr.equals(text)){
        headOf.addMember(d);
        //Image im = new Image("src/delegate.png", 300, 300);
        return true;
    }
    return false;
}

public boolean singUpRegular(Regular r, QRLib q, Image i){
    String textqr = QRLib.decodeQRCodeImage(i.getBitmap());
    String text = "This is a QR for a Regular Member. Climate change doesn't matters, if you stay indoors";
    if(textqr.equals(text)){
        headOf.addMember(r);
        return true;
    }

    return false;
}
```

- **Regular:** Els atributs implementats són responsable de tipus Delegate, que seria el delegate que tenen com a responsable els membres regulars, i una LinkedList (vList) que és una llista de vehicles.
  Igual que Delegate aquesta classe és una subclasse de Member, per tant, al crear-la hem d'utilitzar extends Member.

```
public class Regular extends Member{
```

Els mètodes implementats son: el constructor Regular, al qual li hem passat el nom, phone, email, headquarter i delegate. Igual que en Delegate, en el constructor hem utilitzat la paraula clau super en la primera línea per cridar als paràmetres que estan a la superclase de la subclasse.

```
public Regular(String n, int p, String e, Headquarter h, Delegate r){
    super(n,p,e,h);
    this.responsable = r;
    vList = new LinkedList<Vehicle>();

}
```

També hem implementat addVehicle, que afegeix vehicle a la llista de vehicle (funció que no hem utilitzat en aquesta pràctica).

Hem implementat la funció toString en les classes Regular i Delegate, però al ser subclasses, la implementació és diferent a les demés ja que hem d'agafar el nom que té la superclasse, encara que fa la mateixa utilitat.

```
public String toString() {
        return this.getName();
}
```

- **Availability:** Els atributs son dos llistes: la llista de days, que es de strings, i la llista hours, que es d'enters.
  Hem implementat la funció Availability, que es el constructor, la única funció que hem creat, que es on inicialitzem les llistes i els paràmetres.

- **TestDelegate:** Aquesta és la classe on s'executa el programa, a partir de les funcions implementades en aquesta classe podrem mostrar per pantalla la informació que hem agafat dels diferents fitxers per a comprovar que s'hagin implementat bé. També serà on cridarem a les funcions dels QRs, per a què es creïn.
  Els atributs son tres llistes de: region, delegate i city; i un atribut de classe organization. Al ser atributs, son privats, perquè s'utilitzen només en aquesta classe, pero també son static per a que els valors vagin canviant, ja que quan canviïn en algun mètode i s'utilitzen, també es canviïn en testDelegate, i no tinguin sempre els mateixos valors.

  El primer que fem abans de començar el programa és inicialitzar els atributs.
  La primera cosa que fem abans de començar el programa és inicialitzar els atributs.
  L'atribut d'Organització l'hem inicialitzat amb el nom Organització (no sabíem si s'haurien d'anar canviant els noms i l'hem posat un nom fixe).

  Començem el codi llegint el fitxer regions.xml gràcies a la funció d'Utility readXML, creant una linkedlist de la classe d'un array d'strings que tindrà la informació del fitxer.
  Fent un for de string[ ]array de regions que recorrerà l'array creem un nou element cridant al constructor Region que li passarem la posició 0 del array ja que fa referència a el nom de la regió.
  A continuació, fem un if que si la longitud del seu array és igual a 3 (l'igualem a 3, ja ja qué regions.xml té 3 posicions principalment però pot canviar).

Si es compleix el if, es crea una City, que li passarem el nom de la regió i el de la ciutat, i afegim aquesta City a la llista de City que hem inicialitzat en els atributs.
Si no entra al if, és a dir no compleix que la longitud del array sigui igual 3, va al else, on fem un for, que si i (contador) és més petit que la longitud del array dividit entre 2; és a dir, per exemple si la longitud de l'array és 5, 5/2+1 = 2'5, que ho arrodoneix per la baixa, per tant 2, per tant agafarà la posició 1 i 2 (li sumem 1 perquè així pugui arribar a tots els valors de l'array). Llavors li passarem la posició "i" de l'array que fa referència a el nom de la ciutat i la d'apartir de la meitat de l'array sumant-li i perquè vagi iterant els valors que agafi, el número de la població de la ciutat que la passem a int utilitzant Intenger.parsenInt() per transformar l'string a un número enter, després ho afegim a la llista de city cityList.
Al final fem un add, perquè afegeixi la regió a la llista de regions.

```
LinkedList<String[]> regions = Utility.readXML(type: "region");
for(String[]array : regions){
    Region region = new Region(array[0]);
    if(array.length == 3){
        City city = new City(array[1], Integer.parseInt(array[2]));
        cityList.add(city);
    }else{
        for(int i = 1; i < (array.length/2)+1; i++){
            int medium = array.length/2;
            City city = new City(array[i], Integer.parseInt(array[medium+i]));
            cityList.add(city);
        }
    }
    regionList.add(region);

}
```

Després llegim el fitxer "headquarter.xml". Per a aquest fitxer hem fet algo semblant, la diferència és que en comptes de fer un if i un else, hem fet un for dintre del for string[ ] array : headquarter, que crea Headquarter en la posició 0, és a dir el seu nom, la posició 2, què és el correu i la organització. El segon for té la condició i < array.length / 2+1, que ja hem explicat com funciona anteriorment. Aquesta condició l'hem implementat per agafar els noms de les ciutats que apareixen en el fitxer per això comencem per la posició 2 ja que és on comencen a aparèixer. Dins del for de la llista citylist que hem creat i per cada regió li assignem la llista de les regions. Després cridem a la funció getObject que està en la classe Utility, aquesta funció a través d'un string i la llista corresponent ens retorna l'objecte city passant-li la posició de l'array i i la llista de les cities per després afegir city a la llista de cities i el headquarter creat anteriorment s'afegirà a la city.

Al final fem un headquarter.setCityList(citylist) i afegim el headquarter a la llista de headquarters (hqlist) que hem creat a l'inici de la implementació d'aquesta part del codi.

```
LinkedList<Headquarter> hlist = new LinkedList<Headquarter>();

LinkedList<String[]> headquarters = Utility.readXML(type: "headquarter");
for(String[]array : headquarters){
    Headquarter headquarter = new Headquarter(array[0], array[1], organization);
    LinkedList<City> citylist = new LinkedList<City>();
        for(int i=2; i<(array.length); i++){
            citylist = regionList.get(i).getCityList();
            City city = Utility.getObject(array[i], cityList);
            citylist.add(city);
            city.addHeartquarter(headquarter);
        }

    headquarter.setCityList(citylist);
    hlist.add(headquarter);
    }
```

I per últim llegim el fitxer "heads.xml" el qual conté el name, el seu número de telèfon, l'email de cada delegat, de quina seu és el head i els dies i les hores que estan disponibles.

Per començar creem i inicialitzem dos LinkedList de cada tipus de membre, una dels delegats i una del regulars per després omplir-les. Com els fitxers anteriors fem un procediment igual per llegir-lo. El primer que fem és agafar l'objecte headquarter, agafant la posició 3 (la seu de la qual el delegat és head), perquè ara quan creem un delegate li puguem passar, ja que li passem la posició 0 que és el nom, la 1 que l'hem de convertir en enter que és el número de teléfon, la 2 què és la direcció d'email i el headquarter anterior. Afegim el delegat creat a la llista de heads (headList) i assignem a el delegat com a head en el Headquarter. Ara fem el mateix que hem fet amb el delegate quan l'hem creat, peró li passem un paràmetre més, el delegat que hem creat. Amb les dues posicions restants de la informació que té el fitxer les afegirem a la classe Availability perquè són els dies i les hores que estan disponibles els delegats. Per fer això hem creat dos arrays, una per dies i l'altre per les hores. Per els dies hem agafat la posició 4 ja que és on estan els dies i els hem separat amb un split per obtenir cadascun per separat i afegir-los a l'array i hem fet el mateix però amb la posició 5 per les hores. El procediment per els dies i les hores és el mateix, primer creem una linkedlist, recorrem els arrays que hem fet anteriorment i els afegim a les llistes, en el cas de les hores convertim d'string a

enter, i així obtenim una llista dels dies i una de les hores que estan disponibles els membres.

```
LinkedList<Delegate> delegateList = new LinkedList<Delegate>();
LinkedList<Regular> regularList = new LinkedList<Regular>();

LinkedList<String[]> heads = Utility.readXML(type: "head");
for(String[]array : heads){
        Headquarter headquarter = Utility.getObject(array[3], hlist);
        Delegate delegate = new Delegate(array[0], Integer.parseInt(array[1]), array[2], headquarter);
        headList.add(delegate);
        headquarter.setHead(delegate);

        Regular regular = new Regular(array[0], Integer.parseInt(array[1]), array[2], headquarter, delega

        String[] array_days = array[4].split(regex: "\\.");
        String[] array_hours = array[5].split(regex: "\\.");

        LinkedList<String> days = new LinkedList<String>();
        int a = array_days.length;
        int b = array_hours.length;
        for(int i = 0; i < a; i++){
            days.add(array_days[i]);
        }
        LinkedList<Integer> hours = new LinkedList<Integer>();
        for(int j = 0; j < b; j++){
            hours.add(Integer.parseInt(array_hours[j]));

        }
```

Seguidament, creem un availability i els hi passem les llistes anterior amb els dies i hores. A el delegate li assignem el availability i el headquarter, i al headquarter el delegate que hem creat anteriorment. Per últim afegim el delegate i el regular creat a les seves respectives llistes.

```
Availability availability = new Availability(days, hours);
delegate.setAvailability(availability);
delegate.setHeadOf(headquarter);
headquarter.setHead(delegate);
delegateList.add(delegate);
regularList.add(regular);
```

A continuació, hem imprès els head de cada headquarter, a partir de la funció que hem creat en organization, que es diu getMemberhqlist, que retorna el head delegate de cada headquarter.

```
System.out.println(x: " Miembros de Organization:");
for(int k = 0; k<hlist.size(); k++){
    System.out.println("El head delegate "+ organization.getMembershqlist(hlist.get(k)) +" del headquarte
}
```

Per últim per poder comprobar que els QRs funcionen correctament farem què els QRs es creïn. Per començar creem un nou qr de la classe QRLib i una imatge que al principi sera null. Després obtindrem un objecte delegate al qual li passarem un head de la organització, convertit en string, amb el mètode que hem explicat anteriorment i la llista de delegats. Després creem un delegat i un regular amb valors aleatoris i a les imatges les generem amb el mètode genRegularQR i genDelegateQR passant el qr. Seguidament, amb el delegate cridem a el mètode singup per cadascun, el delegat i el regular, passant els paràmetres corresponents. Per acabar fem un save de les imatges perquè es mostrin.

```
QRLib qr = new QRLib();
Image i = null;
Delegate delegate  = Utility.getObject(String.valueOf(organization.getMembershqlist(hlist.get(index: 5))), delegateList);
Delegate dele = new Delegate(n: "Mar Moreno", p: 123444, e: "mar@gmail.com", hlist.get(index: 0));
Regular regular = new Regular(n: "Mar Moreno", p: 564748, e: "mar@gmail", hlist.get(index: 1), hlist.get(index: 1).getHead());
i = delegate.genRegularQR(qr);
Image i2 = delegate.genDelegateQR(qr);
delegate.singUpDelegate(dele, qr, i2);
delegate.singUpRegular(regular, qr, i);
i.save();
i2.save();
```

També hem implementat linkedlists de: Member, City, regular, Headquarter, Delegate i Region, en els constructors de les classes, depenent de amb quines classes estan associades.

```
memberList = new LinkedList<Member>();
cityList = new LinkedList<City>();
regularList = new LinkedList<Enrollment>();
regionList = new LinkedList<Region>();
headList = new LinkedList<Delegate>();
headquarteList = new LinkedList<Headquarter>();
```

- **Image:** Aquesta classe ja venia implementada en el codi proporcionat per dur a terme aquesta pràctica. La seva funció és crear la imatge en el que anirà el QR corresponent.
- **QRLib:** Aquesta classe igual que la anterior, Image, ja venia i la seva funció és generar el QR corresponent.

- **Vehicle i InfoAction:** En aquestes dos classes no hem implementat gairebé res ja que no l'has hem utilitzat en cap moment per realitzar la pràctica.

CONCLUSIÓ

Una vegada implementat el codi hem observat que el resultat creiem que ha estat l'esperat, ja que s'ha imprès bé els heads de headquarter, i els QRs amb la informació corresponent. Des del mòbil hem pogut llegir els QRs, i en el cas de Delegate, ens ha aparegut en el nom i el missatge correcte, al igual que amb Regular.

Ens hem trobat amb alguns problemes durant la pràctica, que hem estat capaç de resoldre, ja que ens ha semblat més complicada i ens ha costat més temps.
Principalement vam tenir un problema en les classes Image i QRLib, i vam veure que no estaven carregades en el classpath les llibreries i les vam afegir.

Altres problemes van ser a l'hora de fer que se'ns creessin els QR, i al generar les funcioncs genDelegateQR i genRegularQR, que al princpi no s'haviem com implementar les funcions, però al final ho vam solucionar:

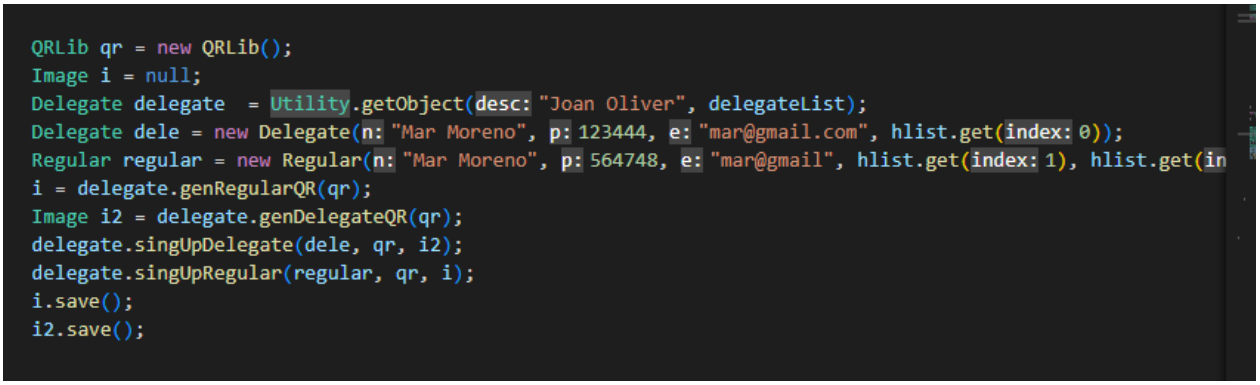
```
public Image genDelegateQR(QRLib qrLib){
    Image image = new Image(path: "code/src/QR/delegateQR.png", width: 300, height: 300);
    String text = "This is QR for a Delegate Member. You don't have to care about rising sea levels, if you l
    BitMatrix bm = QRLib.generateQRCodeImage(text, width: 300, height: 300);
    image.setBitMatrix(bm);

    return image;
}

public Image genRegularQR(QRLib qrLib){
    Image image = new Image(path: "code/src/QR/regularQR.png", width: 300, height: 300);
    String text = "This is a QR for a Regular Member. Climate change doesn't matters, if you stay indoors.";
    BitMatrix bm = QRLib.generateQRCodeImage(text, width: 300, height: 300);
    image.setBitMatrix(bm);

    return image;
```

També vam tenir un problema en el testDelegate, que no imprimia bé els QRs i no ens els guardaba bé, al cap de diferents intents, ens va sortir:



```
QRLib qr = new QRLib();
Image i = null;
Delegate delegate  = Utility.getObject(desc: "Joan Oliver", delegateList);
Delegate dele = new Delegate(n: "Mar Moreno", p: 123444, e: "mar@gmail.com", hlist.get(index: 0));
Regular regular = new Regular(n: "Mar Moreno", p: 564748, e: "mar@gmail", hlist.get(index: 1), hlist.get(in
i = delegate.genRegularQR(qr);
Image i2 = delegate.genDelegateQR(qr);
delegate.singUpDelegate(dele, qr, i2);
delegate.singUpRegular(regular, qr, i);
i.save();
i2.save();
```

Creiem que aquest laboratori ens ha ajudat molt ha entendre com es relacionen les diferents classes.