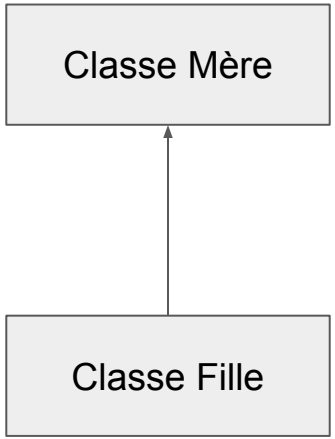
Design pattern decorator

Sommaire :

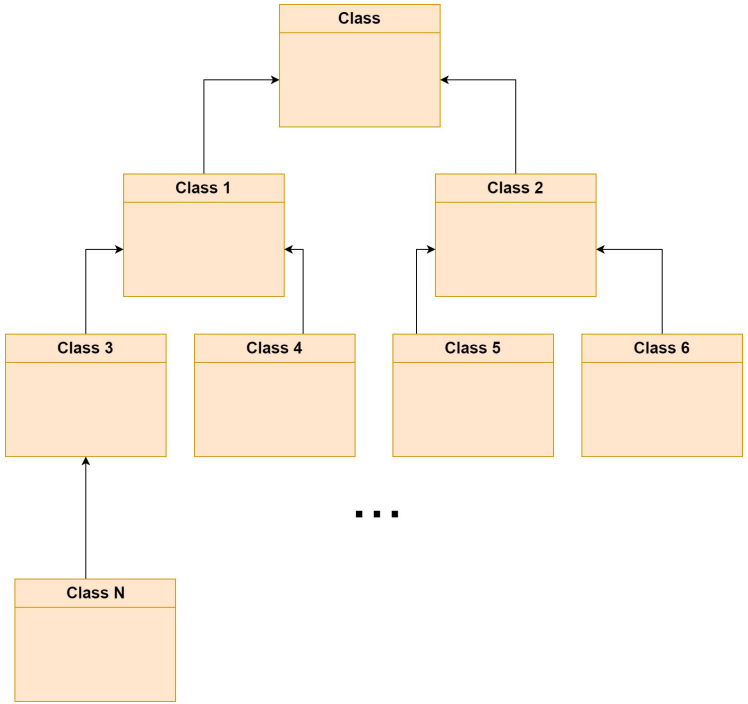
1) Présentation :

Pour ajouter des méthodes à un objet et/ou modifier son comportement sans avoir à modifier la classe mère on utilise : l'héritage.



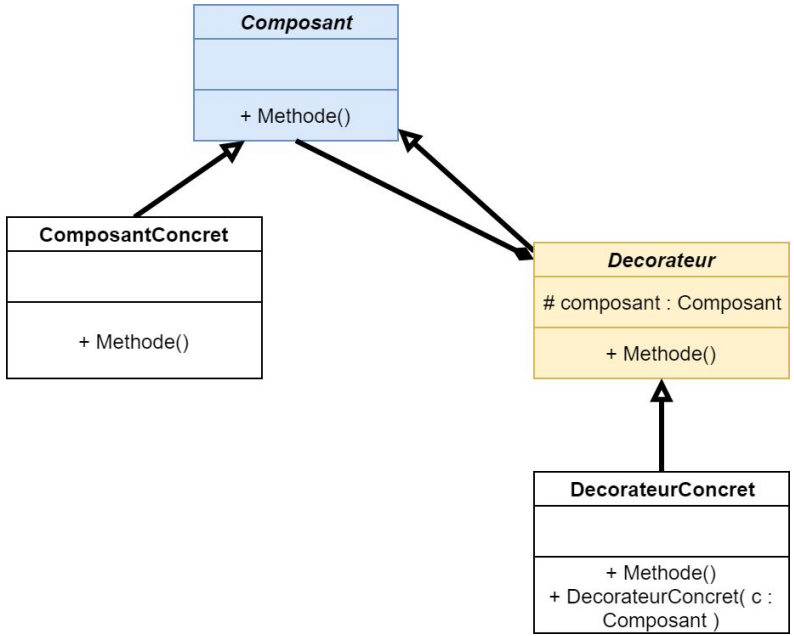
De plus on ne pourra pas ajouter de fonctionnalités de façon dynamique. En effet, une fois le programme compilé on ne peut pas le modifier.

Comment faire pour ajouter des fonctionnalités de façon dynamique à un de nos objets sans utiliser autant de classes ?



Une Solution est le design pattern decorator.

Un décorateur permet d'attacher dynamiquement de nouvelles responsabilités à un objet, permettant ainsi d'offrir une alternative assez souple à l'héritage pour composer de nouvelles fonctionnalités

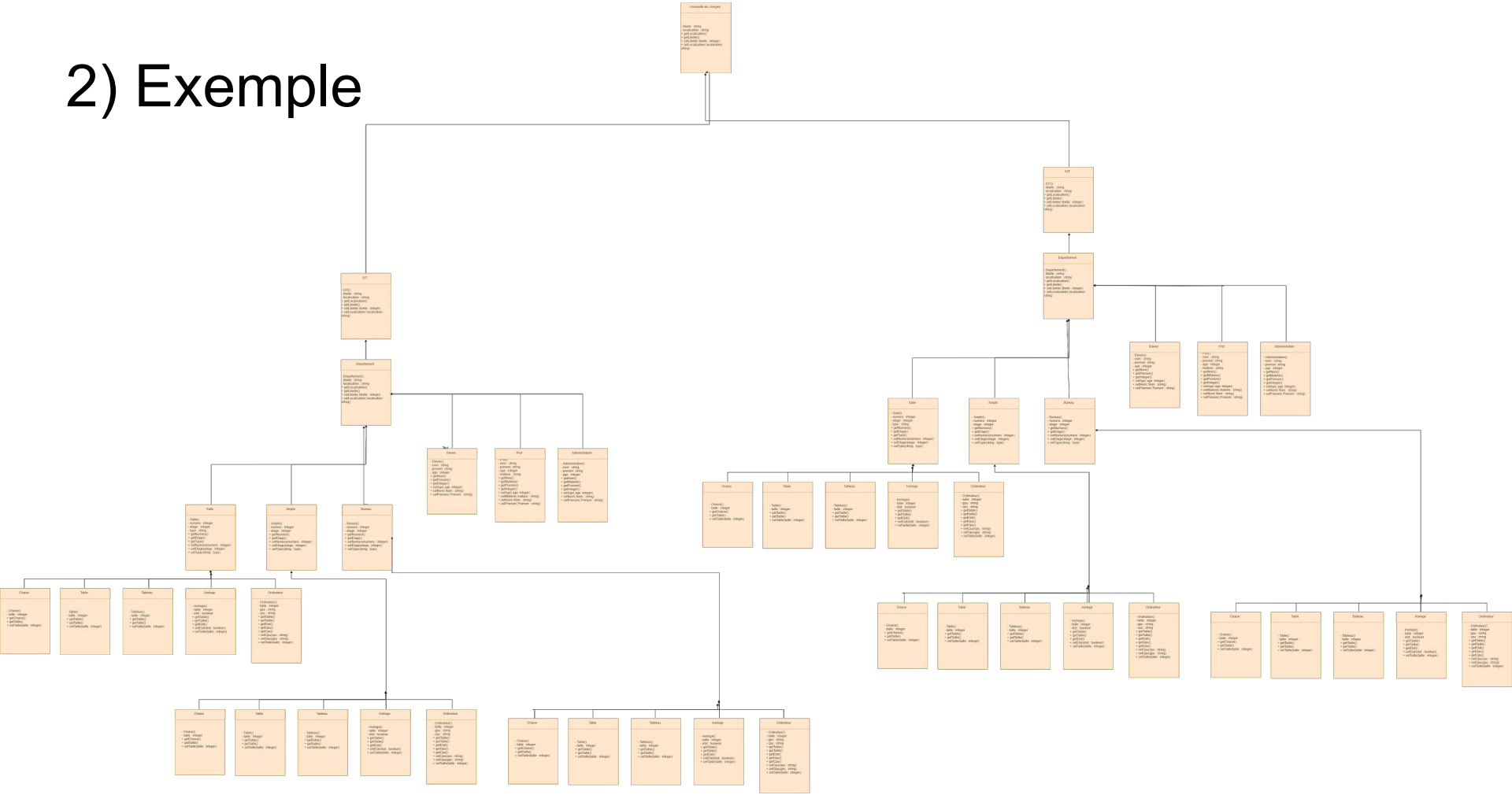


Ce que nous pouvons faire :

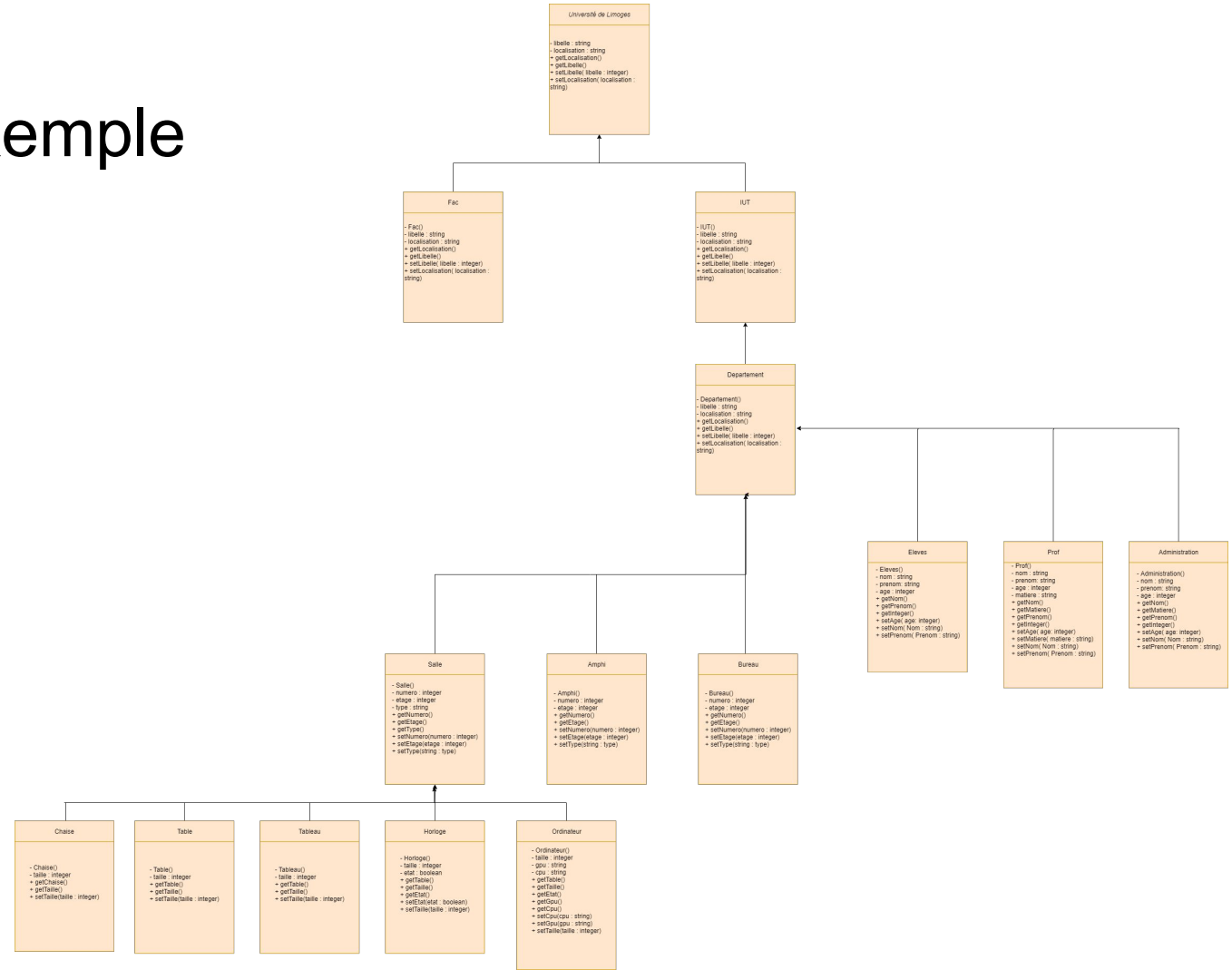
- Créer plusieur composants concret.
- Le décorateur permet d’étendre les fonctionnalités de nos composants.
- Chaque décorateur concret redéfinie les méthodes de décorateur.

Quand le développeur creer un jeu ou une application qui est ou sera sujette à beaucoup de modification et d’upgrade le design pattern permet de simplifier l’app, les tâches et de l’ouvrir vers l’avenir.

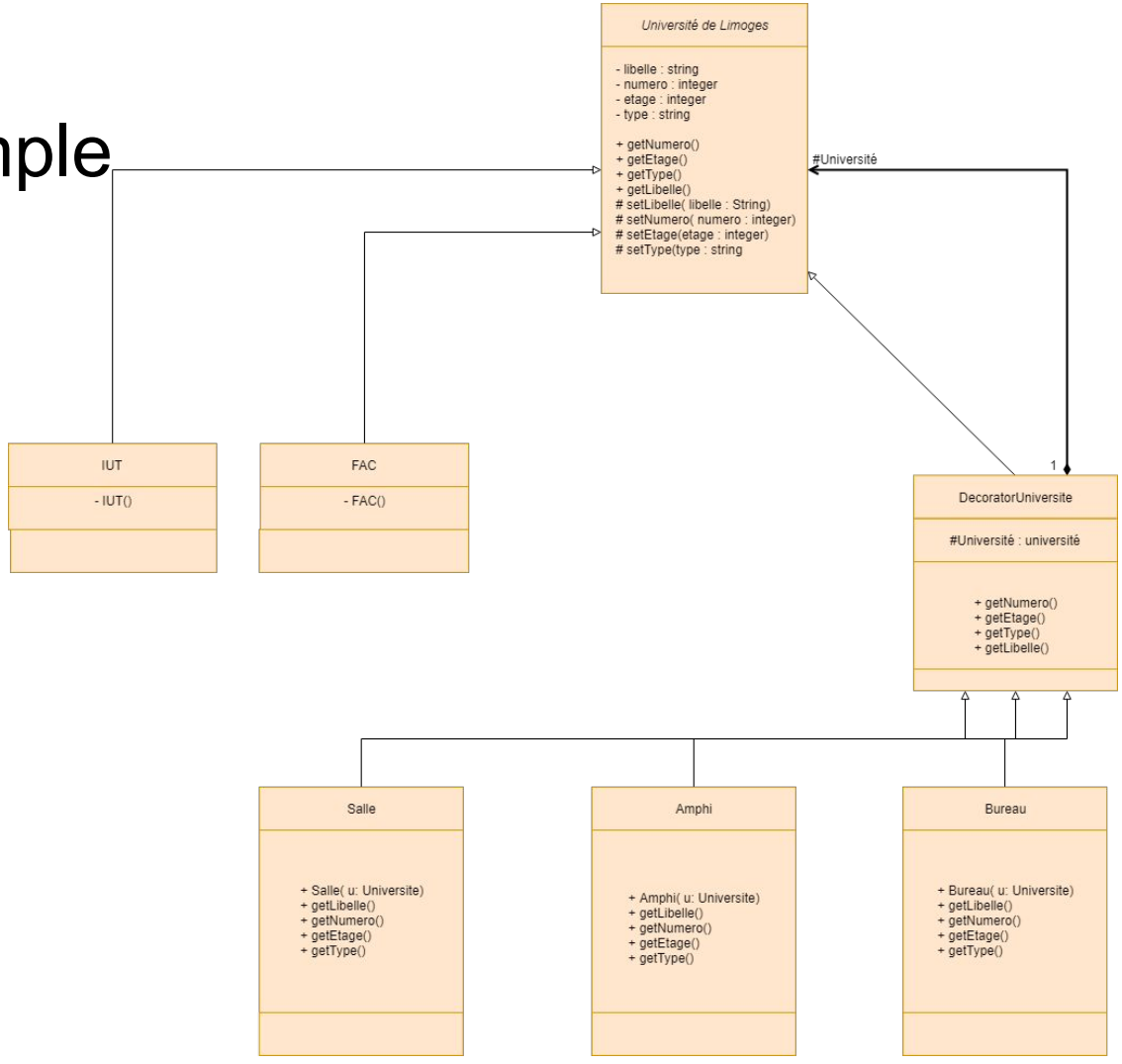
2) Exemple



2) Exemple


Université de Limoges
- libelle : string
- localisation : string
+ getLocalisation()
+ getLibelle()
+ setLibelle(libelle : integer)
+ setLocalisation(localisation : string)
Fac
- Fac()
- libelle : string
- localisation : string
+ getLocalisation()
+ getLibelle()
+ setLibelle(libelle : integer)
+ setLocalisation(localisation : string)
IUT
- IUT()
- libelle : string
- localisation : string
+ getLocalisation()
+ getLibelle()
+ setLibelle(libelle : integer)
+ setLocalisation(localisation : string)
Departement
- Departement()
- libelle : string
- localisation : string
+ getLocalisation()
+ getLibelle()
+ setLibelle(libelle : integer)
+ setLocalisation(localisation : string)
Eleves
- Eleves()
- nom : string
- prenom: string
- age : integer
+ getNom()
+ getPrenom()
+ getInteger()
+ setAge(age: integer)
+ setNom(Nom : string)
+ setPrenom(Prenom : string)
Prof
- Prof()
- nom : string
- prenom: string
- age : integer
- matiere : string
+ getNom()
+ getMatiere()
+ getPrenom()
+ getInteger()
+ setAge(age: integer)
+ setMatiere(matiere : string)
+ setNom(Nom : string)
+ setPrenom(Prenom : string)
Administration
- Administration()
- nom : string
- prenom: string
- age : integer
+ getNom()
+ getMatiere()
+ getPrenom()
+ getInteger()
+ setAge(age: integer)
+ setNom(Nom : string)
+ setPrenom(Prenom : string)
Salle
- Salle()
- numero : integer
- etage : integer
- type : string
+ getNumero()
+ getEtage()
+ getType()
+ setNumero(numero : integer)
+ setEtage(etage : integer)
+ setType(string : type)
Amphi
- Amphi()
- numero : integer
- etage : integer
+ getNumero()
+ getEtage()
+ setNumero(numero : integer)
+ setEtage(etage : integer)
+ setType(string : type)
Bureau
- Bureau()
- numero : integer
- etage : integer
+ getNumero()
+ getEtage()
+ setNumero(numero : integer)
+ setEtage(etage : integer)
+ setType(string : type)
Chaise
- Chaise()
- taille : integer
+ getChaise()
+ getTaille()
+ setTaille(taille : integer)
Table
- Table()
- taille : integer
+ getTable()
+ getTaille()
+ setTaille(taille : integer)
Tableau
- Tableau()
- taille : integer
+ getTable()
+ getTaille()
+ setTaille(taille : integer)
Horloge
- Horloge()
- taille : integer
- etat : boolean
+ getTable()
+ getTaille()
+ getEtat()
+ setEtat(etat : boolean)
+ setTaille(taille : integer)
Ordinateur
- Ordinateur()
- taille : integer
- gpu : string
- cpu : string
+ getTable()
+ getTaille()
+ getEtat()
+ getGpu()
+ getCpu()
+ setCpu(cpu : string)
+ setGpu(gpu : string)
+ setTaille(taille : integer)

2) Exemple


Université de Limoges
- libelle : string
- numero : integer
- etage : integer
- type : string
+ getNumero()
+ getEtage()
+ getType()
+ getLibelle()
# setLibelle(libelle : String)
# setNumero(numero : integer)
# setEtage(etage : integer)
# setType(type : string
#Université
1
IUT
- IUT()
FAC
- FAC()
DecoratorUniversite
#Université : université
+ getNumero()
+ getEtage()
+ getType()
+ getLibelle()
Salle
+ Salle(u: Universite)
+ getLibelle()
+ getNumero()
+ getEtage()
+ getType()
Amphi
+ Amphi(u: Universite)
+ getLibelle()
+ getNumero()
+ getEtage()
+ getType()
Bureau
+ Bureau(u: Universite)
+ getLibelle()
+ getNumero()
+ getEtage()
+ getType()

2) Exemple

```
public abstract class Universite {
	String libelle;
	String localisation;
	int numero;
	int etage;
	String type;

	public String getLibelle(){ return libelle; }
	public int getNumero() { return numero; }
	public int getEtage() { return etage; }
	public int getType() { return type; }

protected void setLibelle(String libelle) {this.libelle = libelle;}
	protected void setNumero(int numero) {this.numero = numero;}
	protected void setEtage(int etage) {this.etage = etage;}
	protected void setType(int type) {this.type = type;}

	public String toString() { return "Université de Limoges : " +
getLibelle() + ", Etage: " + getEtage() + ", Type : " + getType() + ",
Type : "+ getType() + ", Numero : " + getNumero(); }

}
```

2) Exemple

```
class Fac extends Universite{
        public Fac(){
                setLibelle("FLSH");
                setEtage(3);
setType("Litterature");
setNumero(4);
    }
}
class Iut extends Universite{
        public Iut(){
                setLibelle("INFO");
                setEtage(2);
setType("Informatique");
setNumero(1);
    }
}
```

2) Exemple

```
abstract class DecoratorUniversite extends Universite{


    protected Universite universite;
    public abstract String getLibelle();
    public abstract String getType();
    public abstract int getNumero();
    public abstract int getEtage();
}
```

2) Exemple

```
class Salle extends DecorateurUniversite{
      public Salle(Universite u){ universite = u:}
      public String getLibelle(){ return universite.getLibelle() +
"Salle";}
      public int getNumero(){ return universite.getNumero() + "2S";}
      public int getEtage(){ return universite.getEtage() + "2S";}
      public String getType(){ return universite.getType() +
"Numerique";}
}
class Amphi extends DecorateurUniversite{
      public Salle(Universite u){ universite = u:}
      public String getLibelle(){ return universite.getLibelle() + "Amphi
";}
      public int getNumero(){ return universite.getNumero() + "3A";}
      public int getEtage(){ return universite.getEtage() + 0A;}
      public String getType(){ return universite.getType() +
"Classique";}
}
class Bureau extends DecorateurUniversite{
      public Salle(Universite u){ universite = u:}
      public String getLibelle(){ return universite.getLibelle() +
"Bureau ";}
      public int getNumero(){ return universite.getNumero() + "2B";}
      public int getEtage(){ return universite.getEtage() + "3B";}
      public String getType(){ return universite.getType() +
"Classique";}
}
```

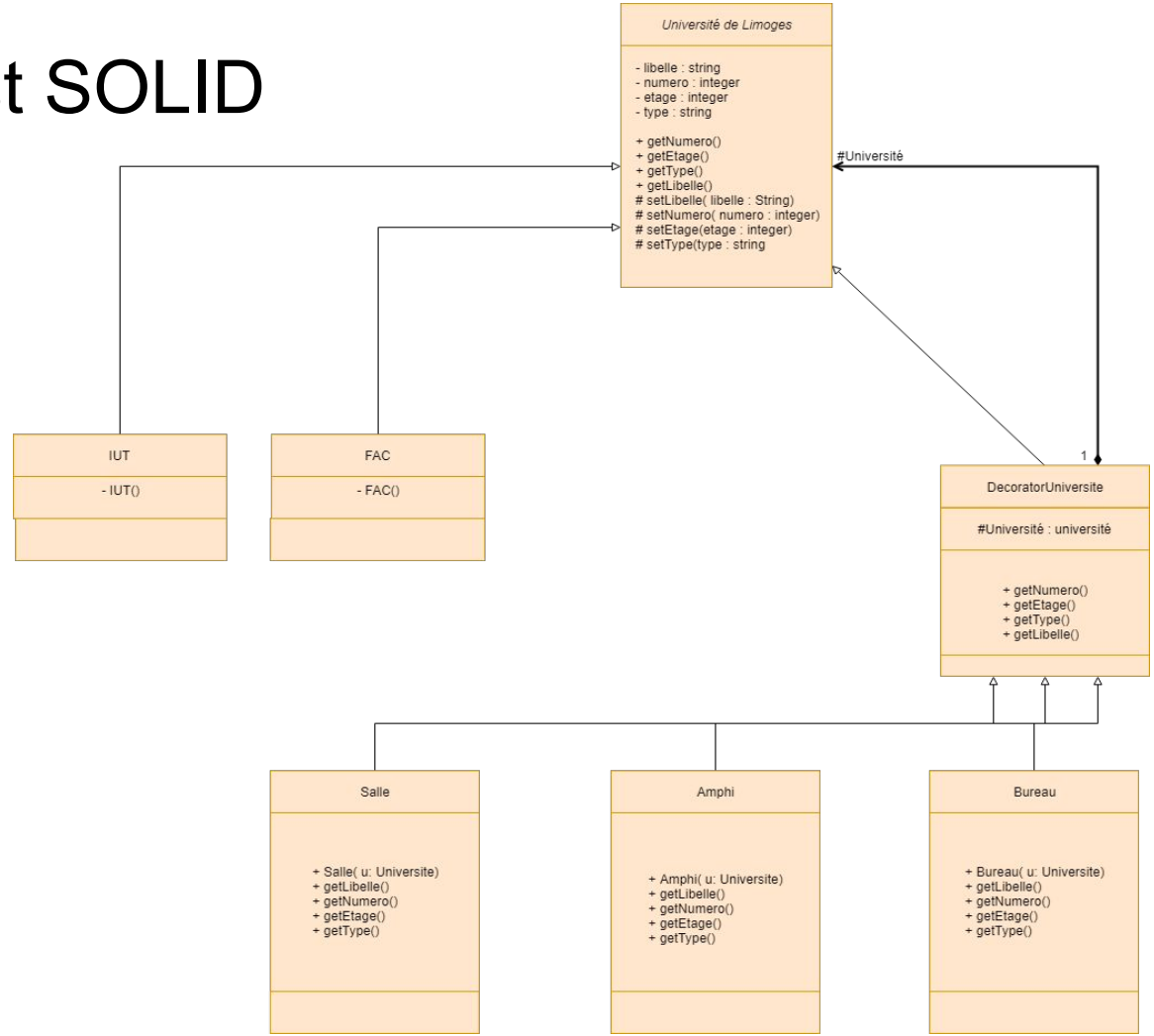
2) Exemple

```
public class Main {

public static void main(String[] args){
      Universite u1 = new Fac();
      System.out.println(u1);
      u1 = new Salle(u1);
      System.out.println(u1);

}
```

2) Exemple

```
SORTIE :
Premier print : Universite : Université de Limoges : FLSH, Etages : 3
,Type : Litterature , Numero : 4

Second print : Salle + Université de Limoges : FLSH, Etages : 3 2S,Type
: Litterature , Numero : 4 2S
```

3) En quoi c’est SOLID



4.1) SRP (Single Responsibility Principle)

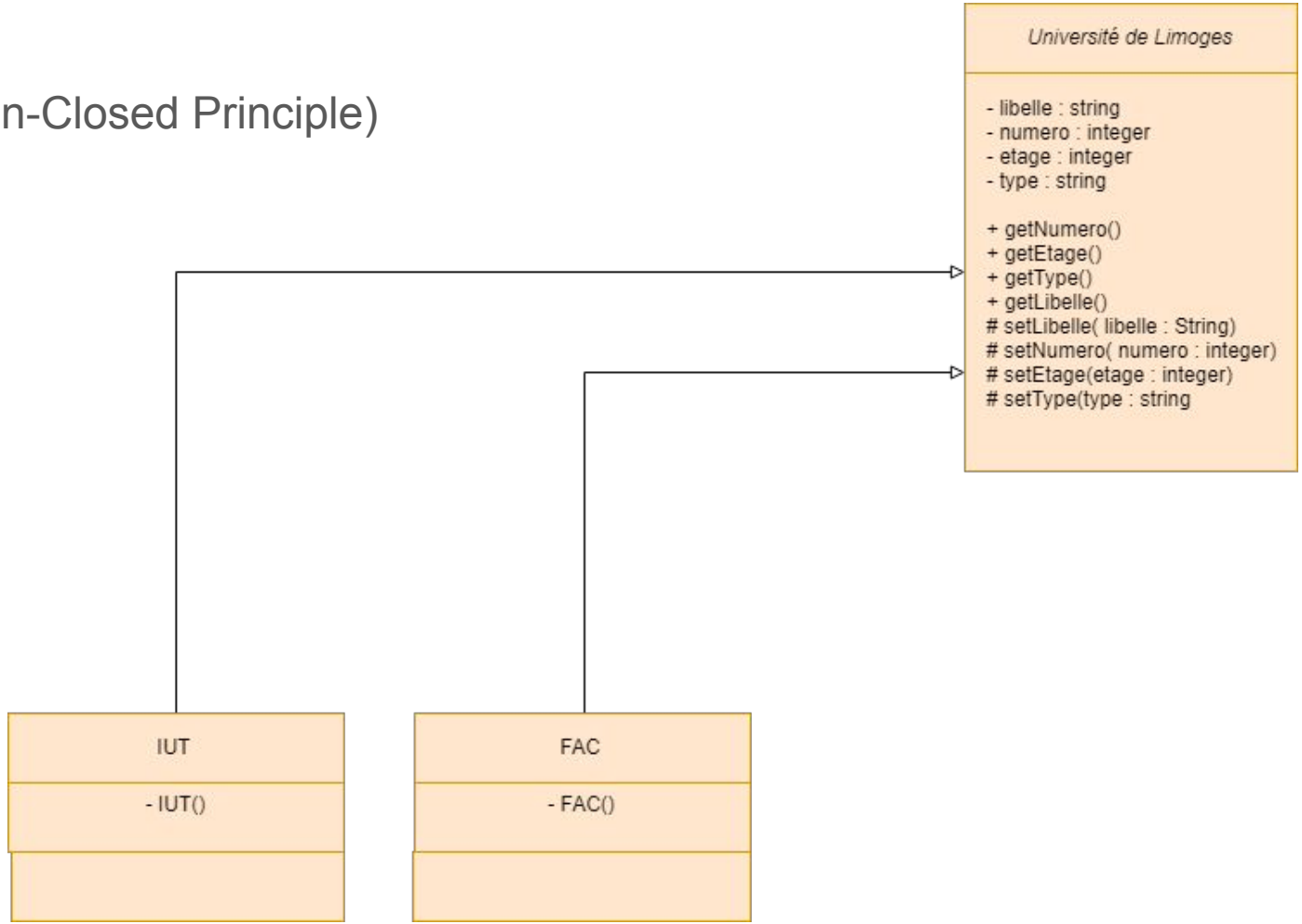
La salle donne son libelle elle même

La salle donne son numéro elle même

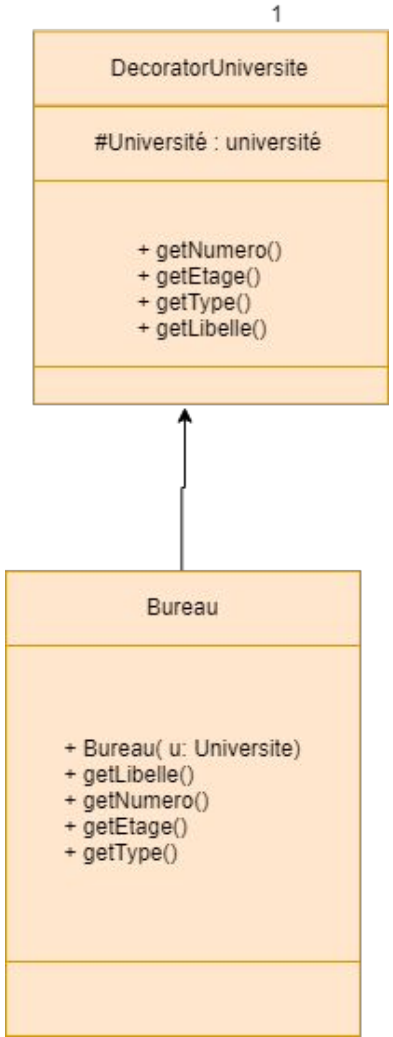
La salle donne son étage elle même

La salle donne son type elle même

4.2) OCP (Open-Closed Principle)



4.3) LSP (Liskov Substitution Principle)



4.4) ISP (Interface Segregation Principle)

4) Conclusion

QCM

Le design pattern decorator utilise le principe :

- de composition
- d’héritage
- d’abstraction

http://www.strawpoll.me/16827808

Le design pattern decorator sert à :

- Implémenter de nouvelles méthodes de manière dynamique
- à décorer son code
- Ajouter dynamiquement des responsabilités
- Modifier le comportement de méthode déjà implémentée

http://www.strawpoll.me/16827813

Peut on décorer un objet avec plusieurs décorateurs:

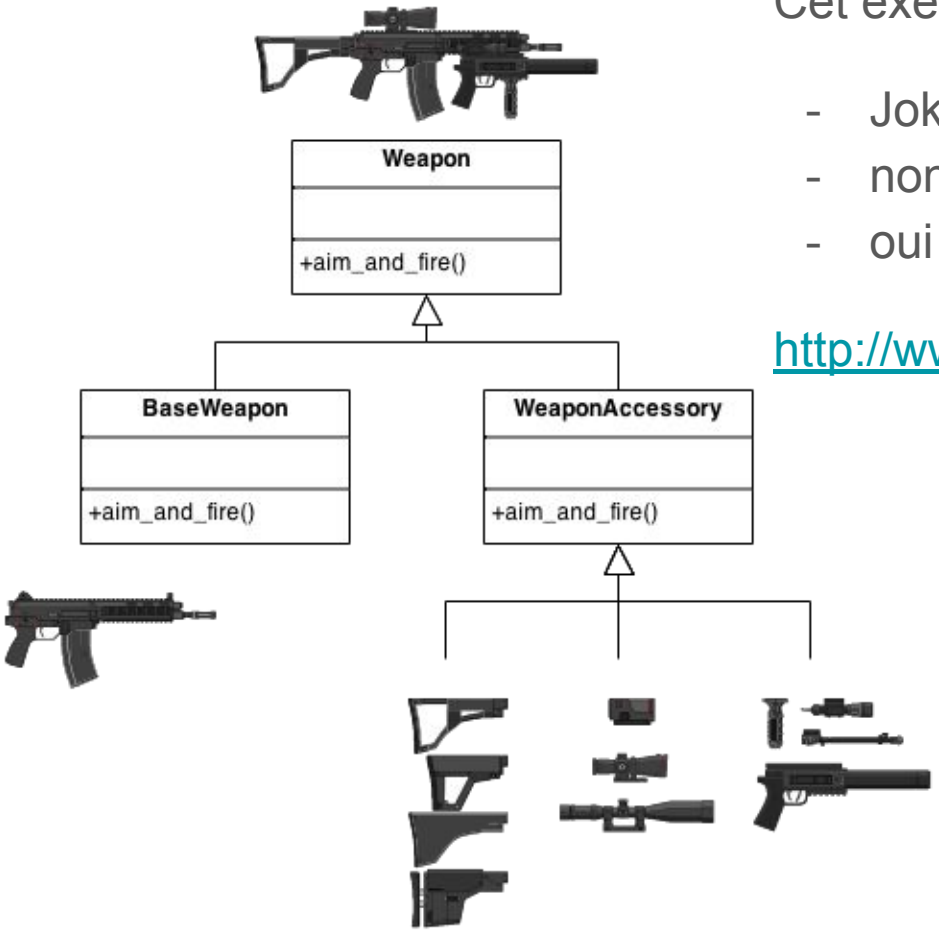
- oui
- non
- joker

http://www.strawpoll.me/16827815

Peut-on utiliser des décorateurs dans n’importe quel ordre :

- oui
- joker
- non

http://www.strawpoll.me/16827819



Cet exemple, utilise-t'il le design pattern decorator:

- Joker
- non
- oui

http://www.strawpoll.me/16827821