UNIVERSITÉ CÔTE D'AZUR

UTILISATION DES PATRONS DE CONCEPTIONS AVEC ANDROID

@.fR Frédéric RALLO – mars 2021

A QUOI SERVENT LES PATRONS DE CONCEPTION ?

- ✓ **Extensible**
- ✓ **Flexible**
- ✓ **Facile à maintenir**
- ✓ **Réutilisable**
- ✓ **Qualités internes**
- ✓ **Meilleure spécification, construction, documentation**

CLASSIFICATION DES PATRONS DE CONCEPTION

Création

- Comment un objet peut être créé
- Indépendance entre la manière de créer et la manière d'utiliser

Structure

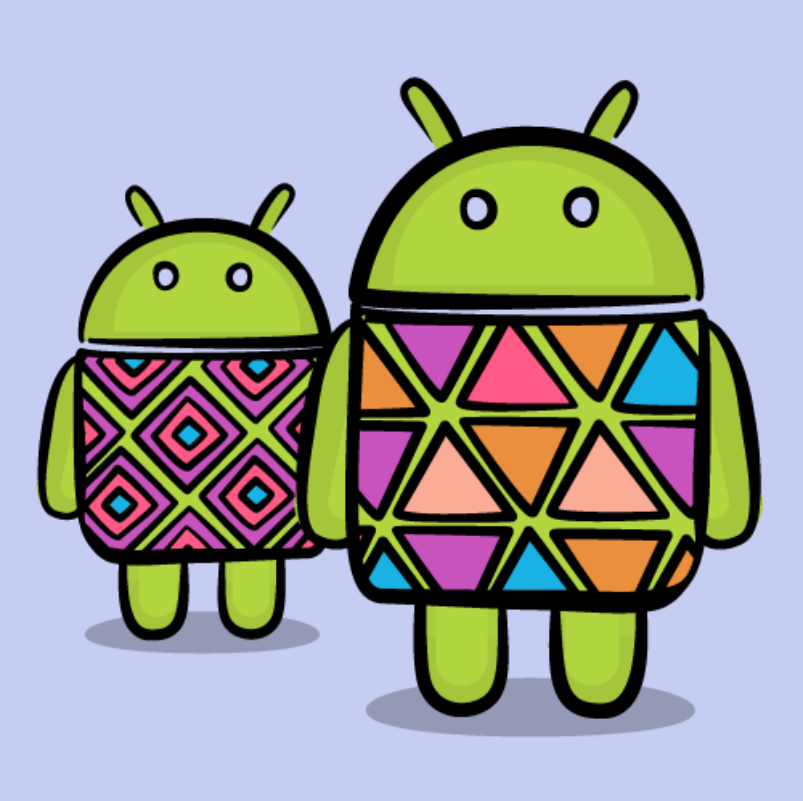
- Comment les objets peuvent être combinés
- Indépendance entre les objets et les connexions

Comportement

- Comment les objets communiquent
- Encapsulation de processus (ex : observer/observable)

Deux exemples de patron de conception

Adapter
Classification : structure



Observable / Oberserveurs
Classification : comportement

A QUOI SERVENT ADAPTERS ?

- ✓ **Rappel de Pattern**
- ✓ **Afficher une liste de Strings dans une ListView**
- ✓ **Afficher une liste de <??> dans une ListView**

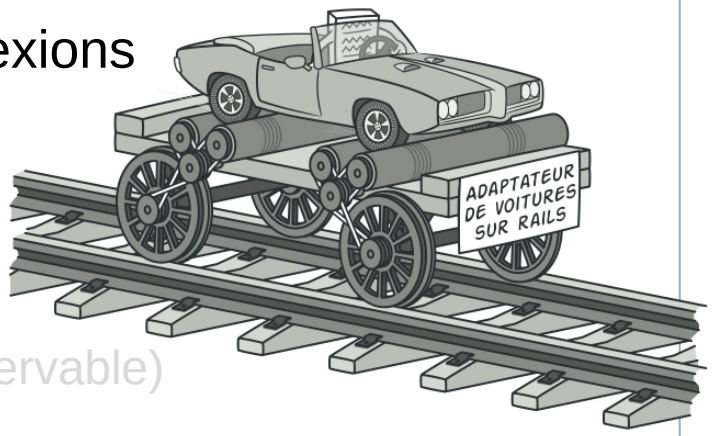
CLASSIFICATION DES PATRONS DE CONCEPTION

Création

- Comment un objet peut être créé
- Indépendance entre la manière de créer et la manière d'utiliser

Structure

- Comment les objets peuvent être combinés
- Indépendance entre les objets et les connexions



Comportement

- Comment les objets communiquent
- Encapsulation de processus (ex : observer/observable)

LA PROBLÉMATIQUE QU'IL CHERCHE À RÉSOUDRE

Structure

- Comment les objets peuvent être combinés
- Indépendance entre les objets et les connexions

Il permet de convertir l'interface d'une classe en Il permet de convertir l'interface d'une classe en une autre interface que le client attend.

L'**adaptateur** fait fonctionner ensemble des classes qui n'auraient pas pu fonctionner sans lui, à cause d'une incompatibilité d'interfaces.

WIKIPÉDIA
L'encyclopédie libre

ANDROID ET ADAPTERS

Il permet de convertir l'interface d'une classe en Il permet de convertir l'interface d'une classe en une autre interface que le client attend.

L'**adaptateur** fait fonctionner ensemble des classes qui n'auraient pas pu fonctionner sans lui, à cause d'une incompatibilité d'interfaces.

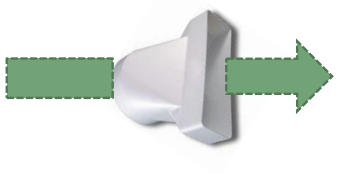
Avec quoi cela s'applique ?

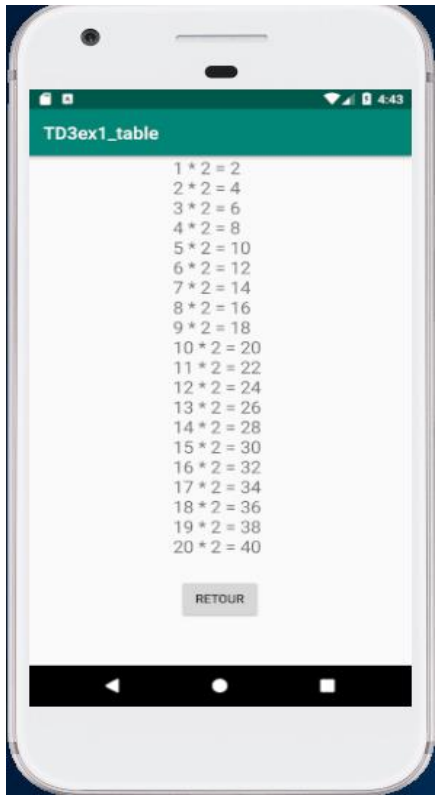
- Spinner
- ListView

Que doit-on adapter ?

- Les collections (ArrayList, Set…) → ListView

```
List<String> list = new ArrayList<>();
for (int i = 1; i <= 20; i++) {
    list.add( i + " " + "* " + table + " = " + (table * i) );
}
```



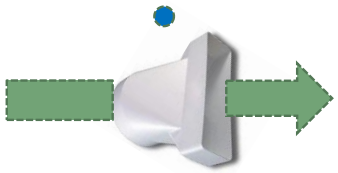


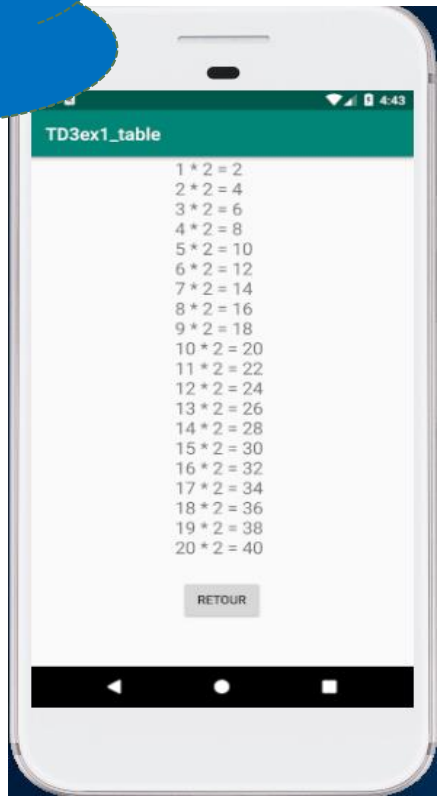
ANDROID ET ADAPTERS

Cet adapter là est de type
« ArrayAdapter » et s'appelle
« android.R.layout.simple_list_item_1 »

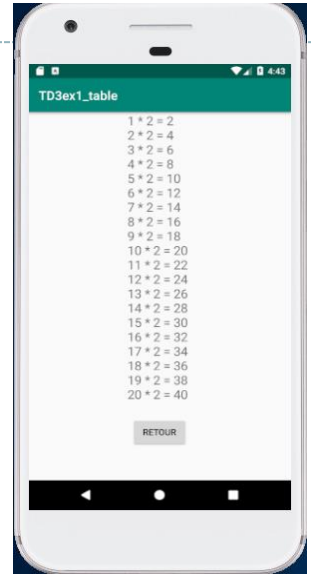
- **Que doit-on adapter ?**
  - Les collections (ArrayList, Set…) → ListView

```
List<String> list = new ArrayList<>();
for (int i = 1; i <= 20; i++) {
    list.add( i + " " + "* " + table + " = " + (table * i) );
}
```





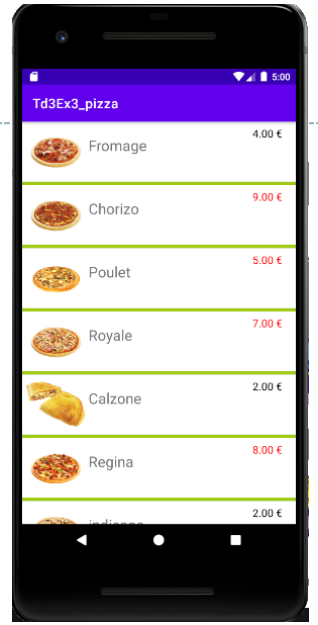
ANDROID ET ADAPTERS




```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_affichage);
    Intent intent = getIntent();
    int table =intent.getIntExtra(VALUE_TABLE,0);

    List<String> list = new ArrayList<>();
    for (int i = 1; i <= 20; i++) { list.add( i + " " + "* " + table + " = " + (table * i) );  }
    ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, list);
    ((ListView)findViewById(R.id.listView)).setAdapter(adapter);
}
```

Cet adapter là est de type
« ArrayAdapter » et s'appelle
« android.R.layout.simple_list_item_1 »

ANDROID ET ADAPTERS




```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_affichage);
    Intent intent = getIntent();
    int table =intent.getIntExtra(VALUE_TABLE,0);

    List<String> list = new ArrayList<>(); new PizzaList<>();
    for (int i = 1; i <= 20; i++) { list.add( i + " " + "* " + table + " = " + (table * i) );  }
    ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, list);
    ((ListView)findViewById(R.id.listView)).setAdapter(adapter);
}
```

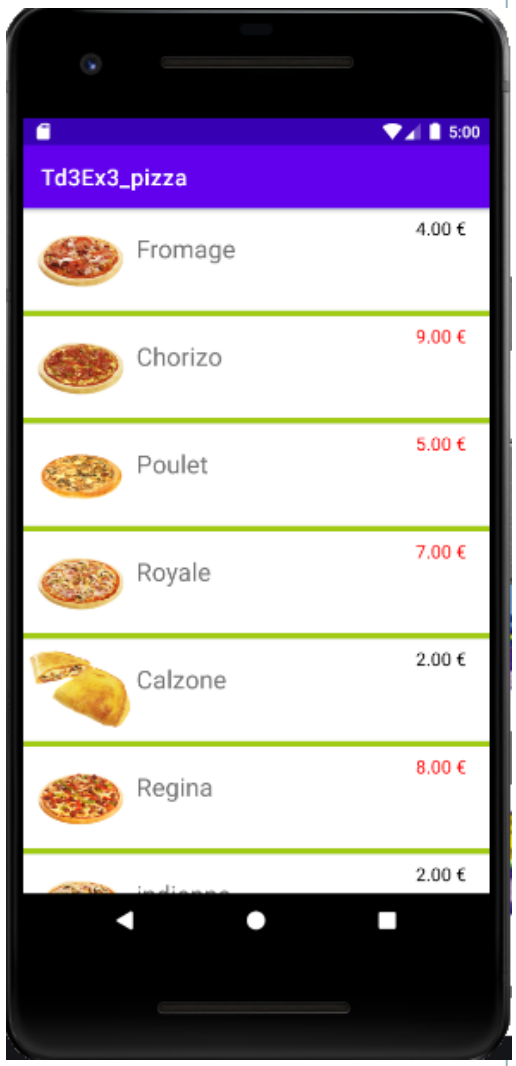
Si on veut afficher quelque chose de différent, cet adapter ne peut pas le faire !!!

ANDROID ET ADAPTERS

Et si on veut adapter autre chose qu'une collection de Strings ?

Ce n'est pas le même adapteur !!!

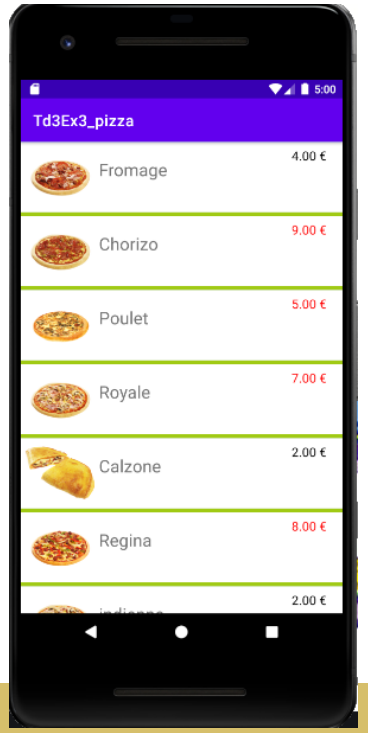
```
List<String> list = new ArrayList<Pizza>();
list.add(new Pizza(« reine »));
 …
```



ANDROID ET ADAPTERS

Personnalisez votre Adapter

- Il étend Adapter (ou une fille de Adapter)
- Il surcharge public int getCount()
- Il a un constructeur (dans lequel on passe une instance du modèle)
- Il surcharge
  - **public View getView(int position, View convertView, ViewGroup parent)**
  - public int getCount()
  - public Object getItem(int position)
  - public long getItemId(int position)



On peut adapter n'importe quel modèle

- Imaginons un objet Pizza
  - Nom
  - Photo
  - prix

EXEMPLE D'ADAPTER PERSONNALISÉ

```
public class Pizza {
    private String name;
    private float price;
    private int picture;

    public Pizza(String name, float price,  int picture)
{
        this.name = name;
        this.price = price;
        this.picture =picture;
    }

    public String getName() {
        return name;
    }
    public float getPrice() {
        return price;
    }
    public int getPicture(){ return picture; }

}
```

```
public class ListPizza extends ArrayList<Pizza>{
    public  ListPizza(){
        add(new Pizza("Fromage", 4, R.drawable.pizza3));
        add(new Pizza("Chorizo", 9, R.drawable.pizza2));
        add(new Pizza("Poulet", 5, R.drawable.pizza1));
        add(new Pizza("Royale", 7, R.drawable.pizza7));
        add(new Pizza("Calzone", 2, R.drawable.pizza4));
        add(new Pizza("Regina", 8, R.drawable.pizza5));
        add(new Pizza("indienne", 2, R.drawable.pizza6));
        add(new Pizza("Speciale", 2, R.drawable.pizza8));
        add(new Pizza("Végetarienne",7,
R.drawable.pizza9));
    }
}
```

EXEMPLE D'ADAPTER PERSONNALISÉ

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Récupération de la liste de pizzas (peuplée)
        ListPizza pizzas = new ListPizza();

        //Création et initialisation de l'Adapter pour les diplomes
        PizzasAdapter adapter = new PizzasAdapter(getApplicationContext(), pizzas);

        //Récupération du composant ListView
        ListView list = findViewById(R.id.listView);

        //Initialisation de la liste avec les données
        list.setAdapter(adapter);

    }
```

EXEMPLE D'ADAPTER PERSONNALISÉ

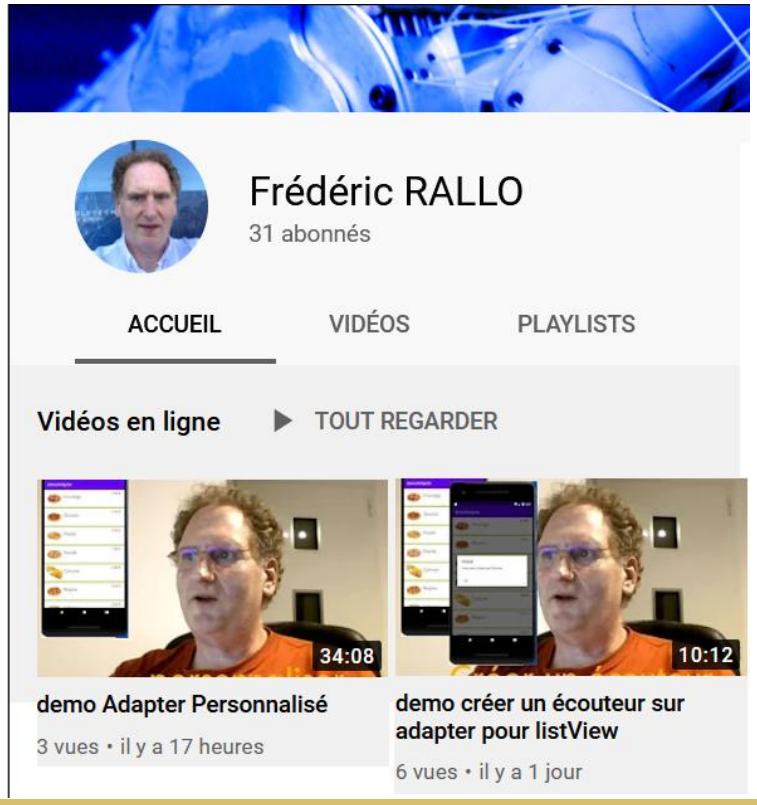
```
public class PizzasAdapter extends BaseAdapter {
    private ListPizza pizzas;

    //Le contexte dans lequel est présent notre adapter
    private Context context;

    //Un mécanisme pour gérer l'affichage graphique depuis un layout XML
    private LayoutInflater mInflater;


    public PizzasAdapter(Context context, ListPizza pizzas) {
        this.context = context;
        this.pizzas = pizzas;
        mInflater = LayoutInflater.from(this.context);
    }

    public int getCount() { return pizzas.size(); }

    public Object getItem(int position) { return pizzas.get(position); }

    public long getItemId(int position) {  return position; }

           ...
```

EXEMPLE D'ADAPTER PERSONNALISÉ

```
...

public View getView(int position, View convertView, ViewGroup parent) {
        LinearLayout layoutItem;

        //(1) : Réutilisation des layouts
        layoutItem = (convertView == null ? mInflater.inflate(R.layout.pizza_layout, parent, false) : convertView);

        //(2) : Récupération des TextView de notre layout
        TextView tvName = layoutItem.findViewById(R.id.pizzaName);
        TextView tvPrice = layoutItem.findViewById(R.id.pizzaPrice);
        ImageView pizzaPicture = layoutItem.findViewById(R.id.pizzaPicture);

        //(3) : Renseignement des valeurs
        tvName.setText(pizzas.get(position).getName());
        tvPrice.setText(Float.toString(pizzas.get(position).getPrice())+"0 €");  //TODO change uggly text format
        pizzaPicture.setImageResource(pizzas.get(position).getPicture());

        //(4) Changement de la couleur du fond de notre item
        tvPrice.setTextColor( pizzas.get(position).getPrice() >= 5 ? Color.RED : Color.BLACK);

        //On retourne l'item créé.
        return layoutItem;
    }
}
```

EXEMPLE D'ADAPTER PERSONNALISÉ

- **Complément d'informations**
  - Je connais quelqu'un qui vient d'ouvrir une chaine qui montre comment faire cela….



Deux exemples de patron de conception

Adapter
Classification : structure

Observable / Oberserveurs
Classification : comportement

CLASSIFICATION DES PATRONS DE CONCEPTION

Création

- Comment un objet peut être créé
- Indépendance entre la manière de créer et la manière d'utiliser

Structure

- Comment les objets peuvent être combinés
- Indépendance entre les objets et les connexions

Comportement

- Comment les objets communiquent
- Encapsulation de processus (ex : observer/observable)

Design Pattern Observer

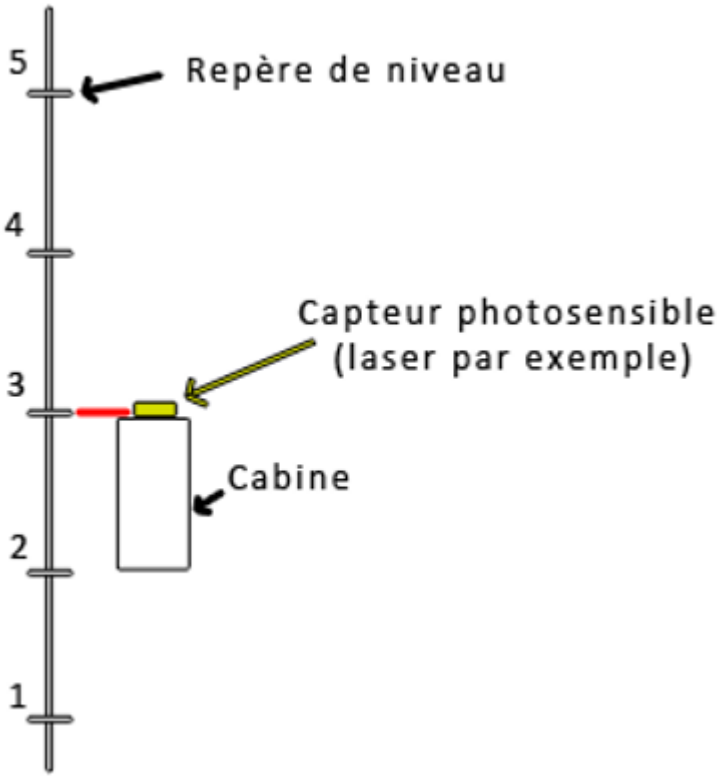
- ✓ Il est utilisé pour envoyer un signal à des modules qui jouent le rôle d'observateurs.

- ✓ En cas de notification, les observateurs effectuent alors l'action adéquate en fonction des informations qui parviennent depuis les modules qu'ils observent (les observables).

Sources : OPENCLASSROOMS

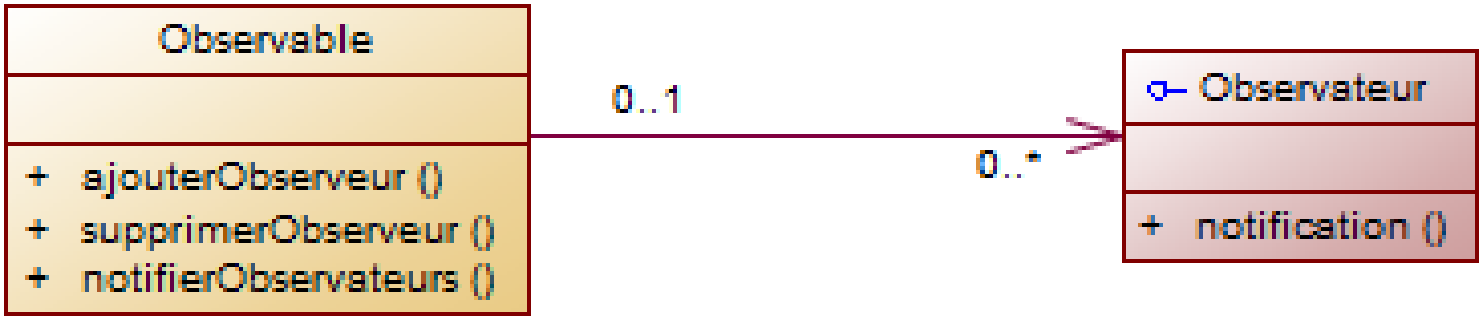
Exemple capteur de l'ascenseur

- ✓ Vous êtes en charge de gérer la bonne position de la cabine de l'ascenseur.
- ✓ La cabine possède l'information suivante :
  - ✓ étage actuel.
- ✓ Le seul moyen pour vous de savoir quand l'ascenseur changera d'étage sera par l'intermédiaire d'un capteur photosensible placé sur l'ascenseur et détectant un repère placé sur chaque étage.



Sources : OPENCLASSROOMS

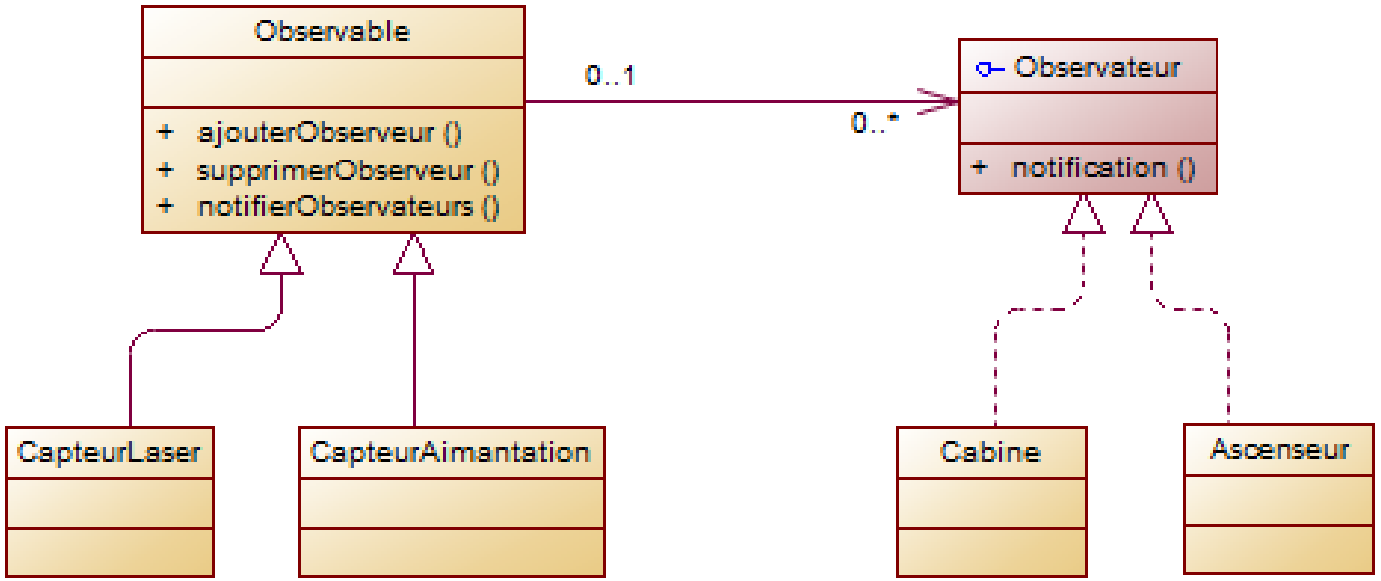
Exemple capteur de l'ascenseur



- ✓ Actions possibles pour l'observateur :
  - ✓ S'inscrire auprès de l'observateur
- ✓ → En retour, l'observateur est assuré de :
  - ✓ Être prévenu lors d'un changement du modèle
  - ✓ C'est la méthode notification() qui sera appelée en call-back

Sources : OPENCLASSROOMS

Exemple capteur de l'ascenseur



En l'appliquant à notre sujet de départ voici le diagramme de classes UML que l'on obtiendrait :

Sources : OPENCLASSROOMS

Exemple capteur de l'ascenseur

```
interface IObservateur
{
    public void notifier();
}
```

```
class Observable {
    private List<IObservateur> m_observateurs;
    public Observable() {
        m_observateurs = new LinkedList<IObservateur>();
    }

    protected void notifierObservateurs() {
       Iterator<IObservateur> it = m_observateurs.iterator();
        // Notifier tous les observers
       while(it.hasNext()){
           IObservateur obs = it.next();
           obs.notifier();
       }
    }

    void ajouterObservateur(IObservateur observateur) {
        // On ajoute un abonné à la liste en le plaçant en premier (implémenté en pull).
        m_observateurs.add(observateur);
    }

    void supprimerObservateur(IObservateur observateur) {
        // Enlever un abonné a la liste
        m_observateurs.remove(observateur);
    }
}
```

```
class CapteurLaser extends Observable {
    private boolean m_detecteVariation;

    public void bienPositionne(){
        m_detecteVariation = true;
    }

    public void malPositionne(){
        m_detecteVariation = false;
    }

    public void run() {
        while(true) {
            if(m_detecteVariation)
                notifierObservateurs();
        }
    }
}
```

Sources : OPENCLASSROOMS

Exemple capteur de l'ascenseur

```
class Cabine implements IObservateur {
    public void notifier(){
        System.out.println("Cabine a recu la notif");
    }
}
```

```
class Moteur implements IObservateur {
    public void notifier(){
        System.out.println("Moteur a recu la notif");
    }
}
```

```
public class Run {  public static void main(String[] args) {
        Cabine instanceCabine = new Cabine();
        Moteur instanceMoteur = new Moteur();
        CapteurLaser capteurEtage = new CapteurLaser();;

        capteurEtage.ajouterObservateur(instanceCabine);
        capteurEtage.ajouterObservateur(instanceMoteur);

        // On simule manuellement une variation (normalement c'est le thread qui s'en charge)
        capteurEtage.bienPositionne();
        // La cabine et le moteur ont reçu une notification sur leur méthode notifier()

        capteurEtage.supprimerObservateur(instanceMoteur);
        System.out.println("Suppression du moteur dans les abonnes");

        capteurEtage.malPositionne();
    }
}
```

```
Cabine a recu la notif
Moteur a recu la notif
Suppression du moteur dans les abonnes
Cabine a recu la notif
```

Sources : OPENCLASSROOMS

Exemple capteur de l'ascenseur

```
// Exemple tiré de wikipédia
class Signal extends Observable {

  void setData(byte[] lbData){
    setChanged(); // Positionne son indicateur de changement
    notifyObservers(); // (1) notification
  }
}
/*
    On crée le panneau d'affichage qui implémente l'interface java.util.Observer.
    Avec une méthode d'initialisation (2), on lui transmet le signal à observer (2).
    Lorsque le signal notifie une mise à jour, le panneau est redessiné (3).*/

class JPanelSignal extends JPanel implements Observer {

  void init(Signal lSigAObserver) {
    lSigAObserver.addObserver(this); // (2) ajout d'observateur
  }

  void update(Observable observable, Object objectConcerne) {
    repaint();  // (3) traitement de l'observation
  }
}
```

En Java ce design pattern étant tellement utilisé, il existe dans l'API de base.

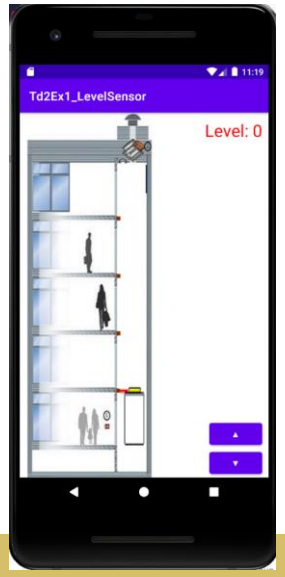
Il n'est dont pas nécessaire d'écrire le code de l'Observateur et de l'Observable, on peut utiliser les interfaces prévues à cet effet :

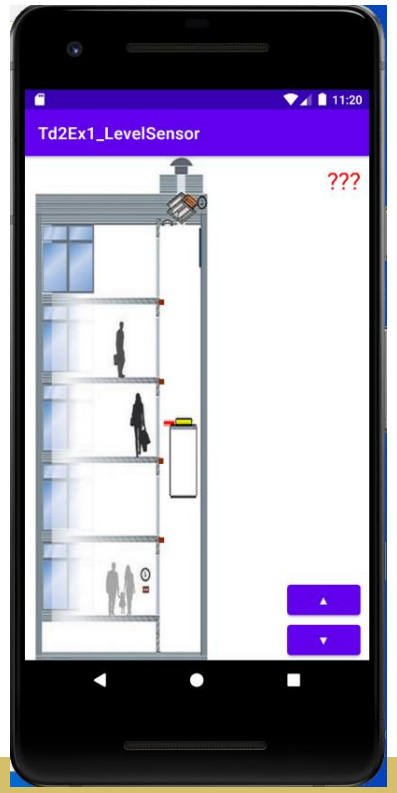
EXEMPLE D'ADAPTER PERSONNALISÉ

Application pratique

- Vous allez reprendre cet exemple dans le cadre d'une application Android

- Vous trouverez en guise d'aide,
  - Un lien vers le site openclassroom qui reprend le pattern
  - Un code source qui montre comment utiliser des animations





EXEMPLE D'ADAPTER PERSONNALISÉ

- **3 classes à créer**

  - 1 observateur : la cabine
    - dispose d'une méthode @Override public void update(Observable o, Object arg) qui met à jour l'attribut currentLevel.
    - Connait le niveau
    - Dispose des accesseurs sur ce niveau

  - 1 contrôleur :
    - Permet à l'utilisateur d'appuyer sur les boutons.
      - affiche l'étage courant ou (« ??? ») entre 2 étages
      - Joue animation de déplacement
    - lorsque l'animation est finie
      - le capteur détecte l'étage atteint.
      - la cabine change la valeur du niveau
    - Est capable d'afficher l'étage lorsque la cabine le lui demande

  - 1 observable : le capteur
    - dispose d'une méthode qui va notifier tous les observateurs de la position courante

EXEMPLE D'ADAPTER PERSONNALISÉ

Conclusion

- 1 observateur : la cabine
  - Il est responsable de traiter le texte qui doit être affiché
  - Il est informé chaque fois que le niveau change
- 1 contrôleur :
  - Ne prend aucune décision.
  - Pour simuler la réalité, on lui demande dans cet exemple de modifier la valeur du niveau dans les capteurs mais dans la réalité, ce sont les capteurs qui détectent
- 1 observable : le capteur
  - Il est responsable d'informer systématiquement tous les observateurs lorsque le niveau change