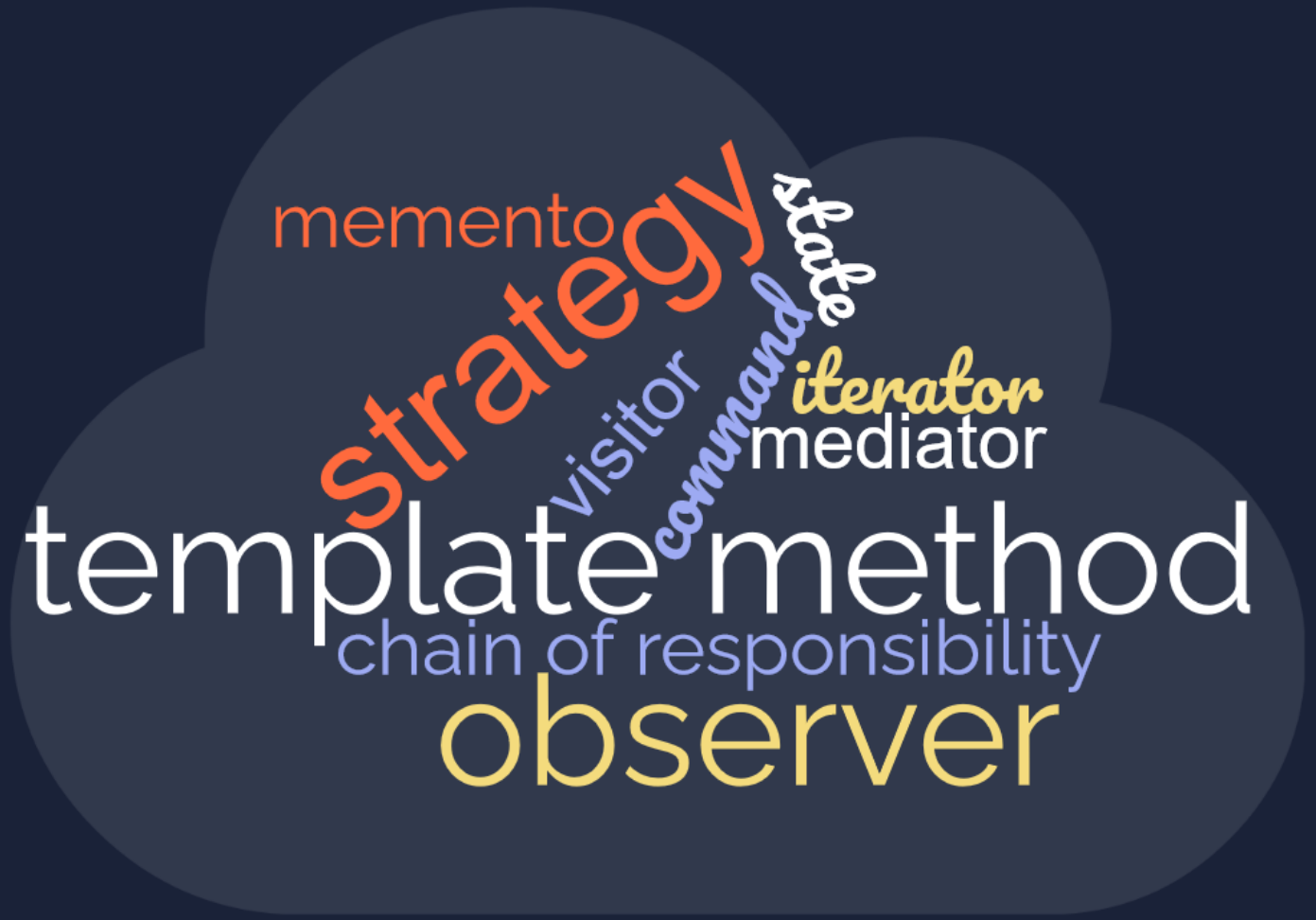


Les design patterns comportementaux

Les design patterns comportementaux

- Formes de comportement pour décrire :
  - des algorithmes
  - des comportements entre objets
  - des formes de communication entre objet

Les design patterns comportementaux

- Chain of responsability
  - Permet de passer un objet à travers une chaîne d'objet jusqu'à ce qu'un objet approprié puisse le traiter.
- Command
  - Prend une action à effectuer et la transforme en un objet autonome qui contient tous les détails de cette action.
- Interpreter
  - Transforme un texte source en objets. Utilisé par les parsers, analyse syntaxique
- Iterator
  - Parcours une collection d'objet
- Mediator
  - Encapsule une logique d'interaction entre objets en évitant que ceux-ci n'aient à se référer les uns les autres.

Les design patterns comportementaux

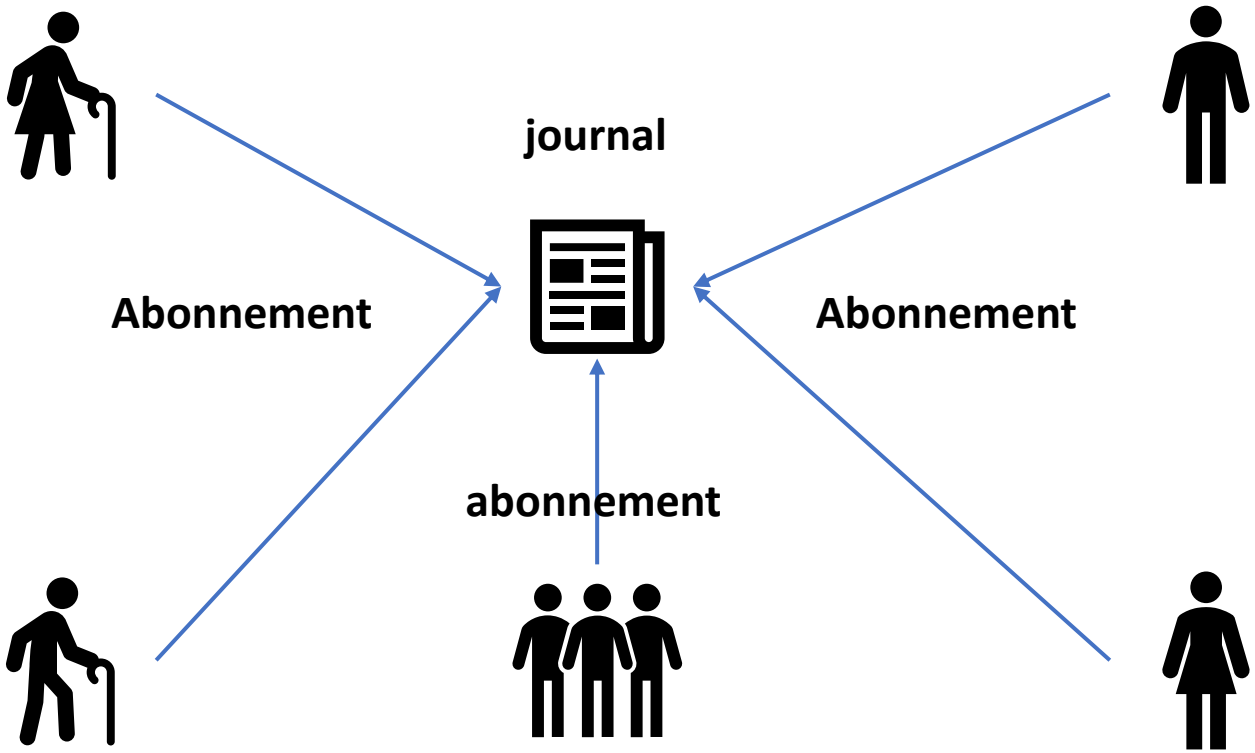
- Memento
  - Stockage temporaire d'un état partiel d'un objet. Permet de le reconstruire par la suite.
- Observer
  - Permet à une classe d'être notifiée du changement d'état d'une autre classe.
- State
  - Permet à un objet de modifier son comportement selon son état.
- Strategy
  - Permet d'encapsuler une famille d'algorithmes interchangeable

Les design patterns comportementaux

- Template
  - Permet de découper un algorithme complexe en petits morceaux. Une classe principale définit la structure et d'autre classes l'implémente.

- Visitor
  - Permet d'ajouter fictivement une méthode à une classe sans pour autant modifier sa structure.

Observer (Observateur)

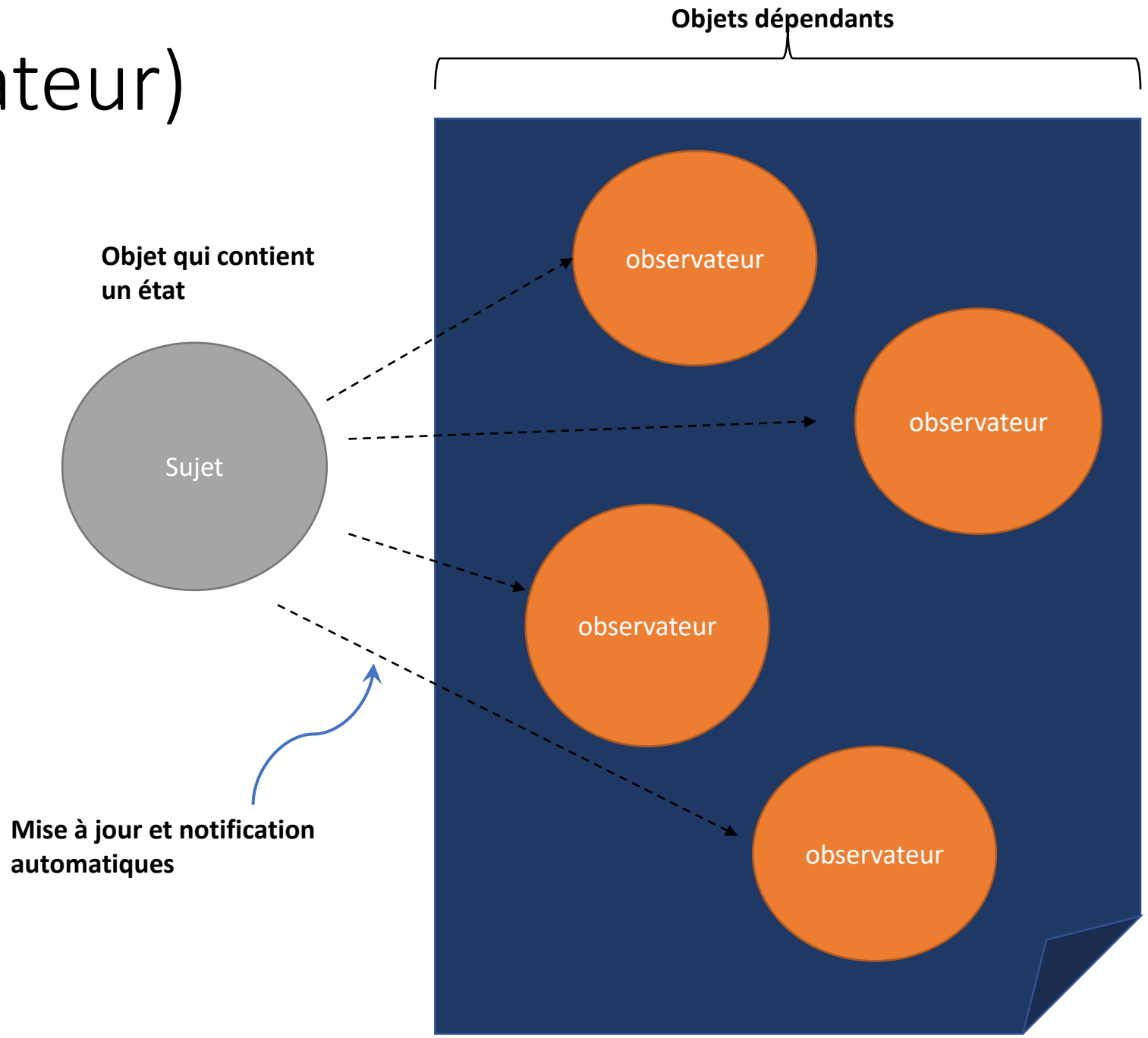


Observer (Observateur)

- **Nom** : Observer (Publish-subscribe)
- **Intention** : le pattern Observer définit une relation 1-n entre objet de manière à ce qu'un objet change d'état, tous ceux qui en dépendent en soient notifiés et soient mis à jour automatiquement.
- **Problème** : Imaginez que vous souhaitez acheter un produit sur un site e-commerce et que vous chercher à savoir quand votre produit est en stock.
  - Comment le savoir sans aller tous les jours vérifier le stock sur le site (perte de temps)

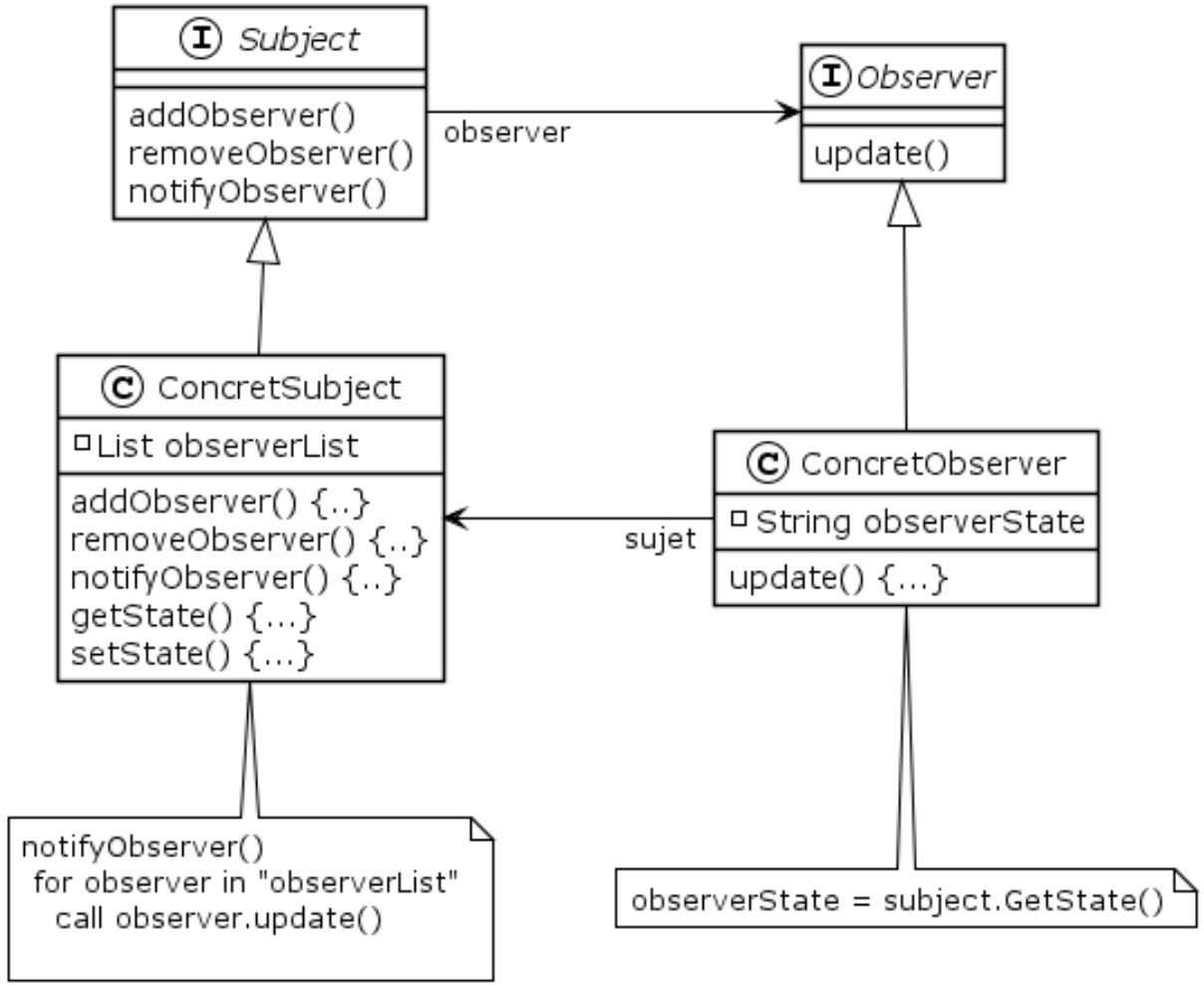
Observer (Observateur)

- Sujet:
  - Enregistre tous les « observateurs"
  - Lors d'un changement, prévient tous les "observateurs« enregistrés
- Observateur:
  - Se déclare au sujet.
  - Peuvent accéder à l'observateur



Observer (Observateur) : Diagramme de classe

- Chaque sujet peut avoir plusieurs observateurs



Observer (Observateur) : Exemple java

- L’agence AFP publie quotidiennement des actualités qui sont reprises par les différents journaux ou télévision qui sont abonnées.
- Le code suivant modélise l’abonnement des chaines de télévisions à l’AFP, puis la publication des nouvelles actualités vers les chaines de télévision.

Observer (Observateur) : Exemple

- Sujet
  (Observable)

```java
public interface Subject {

    /**
     * ajoute un nouvel observateur
     * @param observer
     */
    void addObserver(Observer observer);

    /**
     * supprime un observateur
     * @param observer
     */
    void removeObserver(Observer observer);

    /**
     * notifie tous les observateurs d'un nouvel évènement
     */
    void notifyObservers();


}
```

Observer (Observateur) : Exemple

- Observateur (Observer)

```java
public interface Observer {
    /**
     * mets à jour les données de l'observateur
     */
    void update(News news);

}
```

Observer (Observateur) : Exemple

- Sujet : classe concrète

```java
public class AgenceFrancePresse implements Subject {

    private List<News> newsList = new ArrayList<>();

    private List<Observer> observers = new ArrayList<>();

    private News lastNews = null;

    public AgenceFrancePresse() {
    }

    public AgenceFrancePresse(List<Observer> observers) {
        this.observers = observers;
    }

    @Override
    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    @Override
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    public void addNews(News news){
        this.lastNews = news;
        notifyObservers();
    }

    @Override
    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(this.lastNews);
        }
    }
}
```

Observer (Observateur) : Exemple

- observateur : classe concrète

```
public class Journaliste implements Observer{

    private final String agencyName;

    List<News> list = new ArrayList<>();

    public Journaliste(String agencyName) {
        this.agencyName = agencyName;
    }

    @Override
    public void update(News news) {
      list.add(news);
      log.info( msg: "Ajout d'une nouvelle actualité à l'agence "+agencyName +" " + news );
    }

    public void publishNews(){
        log.info( msg: "publication des actualités de "+agencyName+ " : "+ list);
    }
}
```

Observer (Observateur)

- Le sujet ne sait qu'une chose à propos de l'observateur : il implémente une interface particulière (DIP)
- Il n'est pas nécessaire de modifier le sujet pour ajouter de nouveaux types d'observateur (OCP)
- Modifier un sujet ou un objet n'impacte pas l'un ou l'autre

Couplage faible entre des objets

TP : Observateur

- Utiliser l'exemple donnée et le réécrire avec l'implémentation du JDK
- https://docs.oracle.com/en/java/javase/11/docs/api/java.desktop/java/beans/PropertyChangeSupport.html

- NB : https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Observer.html **est deprecated, ne plus utiliser.**

Template Method (Patron de méthode)

Template Method (Patron de méthode)

- **Nom** : Template Method
- **Intention**
- définition d’un squelette d’algorithme dont certaines étapes sont fournies par une classe dérivée.
- Utiliser lorsque il est nécessaire d’implémenter une fois pour toute les parties invariantes d’un algorithmes et de laisser aux sous-classes de soin d’implémenter les parties dont le comportement est conçu comme modifiable.

Template Method (Patron de méthode)

- **Problème** : Imaginez que vous souhaitez créer un programme pour piloter une machine à café. Cette dernière produit du café, du thé, du chocolat.

Template Method (Patron de méthode)

Comment faire un thé à la menthe ?

Étape 1. Faire bouillir de l'eau.
Étape 2. Mettre vos feuilles de **menthe**
Étape 3. Ajouter votre cuillerée de gingembre, votre cuillerée de poivre et votre clou de girofle.
Étape 4. Verser le thé dans une tasse

Comment faire un café Latté macchiato ?

Étape 1. Faire bouillir de l'eau.
Étape 2. Filtrer le café avec l'eau bouillante
Étape 3. Ajouter votre cuillerée de chocolat en poudre, votre lait et du sucre vanillé.
Étape 4. verser le café dans une tasse

Template Method (Patron de méthode)

- **Problème** : Comment éviter que la classe en charge de la fabrication du café et celle du thé ne comporte de code dupliqué ?

Template Method (Patron de méthode)

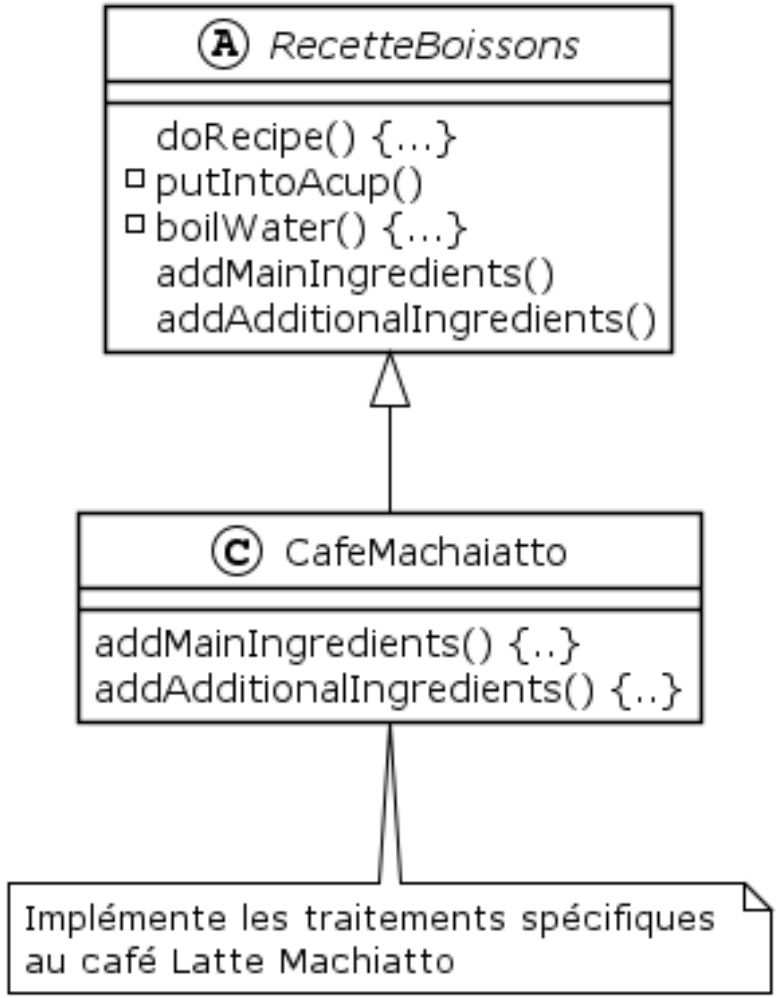
- Abstract classe:
  - Définit des opérations primitives abstraites que les sous-classes concrètes surchargent pour implémenter les étapes de l'algorithme
  - Implémente une template method qui définit le squelette de l'algorithme

- Concrete classe
  - implémente les opérations primitives

```java
public abstract class MachineACafe {

    public  void doRecipe() {
        boilWater();
        addMainIngredients();
        addAdditionalIngredients();
        putIntoACup();

    }

    private void putIntoACup() {
        // implémentation
        log.info( msg: "La boisson a été versée dans une tasse");

    }

    protected abstract void addAdditionalIngredients();

    protected abstract void addMainIngredients();

    private void boilWater() {
        // implémentation
        log.info( msg: "L'eau bouillante est prête à être utilisée'");
    }


}
```

Template Method (Patron de méthode): Diagramme de classe

- Il peut y avoir plusieurs classes concrètes, dans notre exemple une par recette.



Template Method (Patron de méthode)

- Ce patron est utile pour éviter la duplication de code dans plusieurs classes et par conséquent un refactoring important en cas de modification.
- Il n’est pas nécessaire de modifier la classe abstraite pour ajouter de nouveaux comportements nous pouvons le faire en ajoutant des classes concrètes (OCP)

Principe d’Hollywood : Ne nous appelez pas nous vous appellerons.

TP : Template Method

- Créer les classes qui illustrent le template method avec la machine à café :
  - Créer la classe abstraite générique à une recette
  - Créer une classe concrète par recette
    - **Thé à la menthe**
    - **café Latté macchiato**
  - Créer tout élément que vous pensez nécessaire pour le fonctionnement de la machine à café
  - Créer une classe avec une méthode main permettant de tester l'exécution des recettes.
  - Vérifier le bon fonctionnement de votre programme