Conception et Programmation objet Avancées

Adaptateur et Observateur

Sylvie Coste

IUT de Lens

2021–2022

Plan

Adaptateur

Observateur

Étude du problème

Encore un problème de canard

On reprend notre simulateur de mare aux canards. La mare est un lieu où les canards viennent cancaner et voler.
Tous les canards doivent implémenter les deux méthodes *cancaner()* et *voler()*.

Arrivée des dindons

Des dindons sauvages viennent fréquenter votre mare. Ils volètent par intermittence et ne cancanent pas mais glougloutent. Vous devez cependant trouver une solution pour qu'ils cancanent et volent dans votre simulateur !

- Comment modéliser ce problème ?

Diagramme de classes : la mare

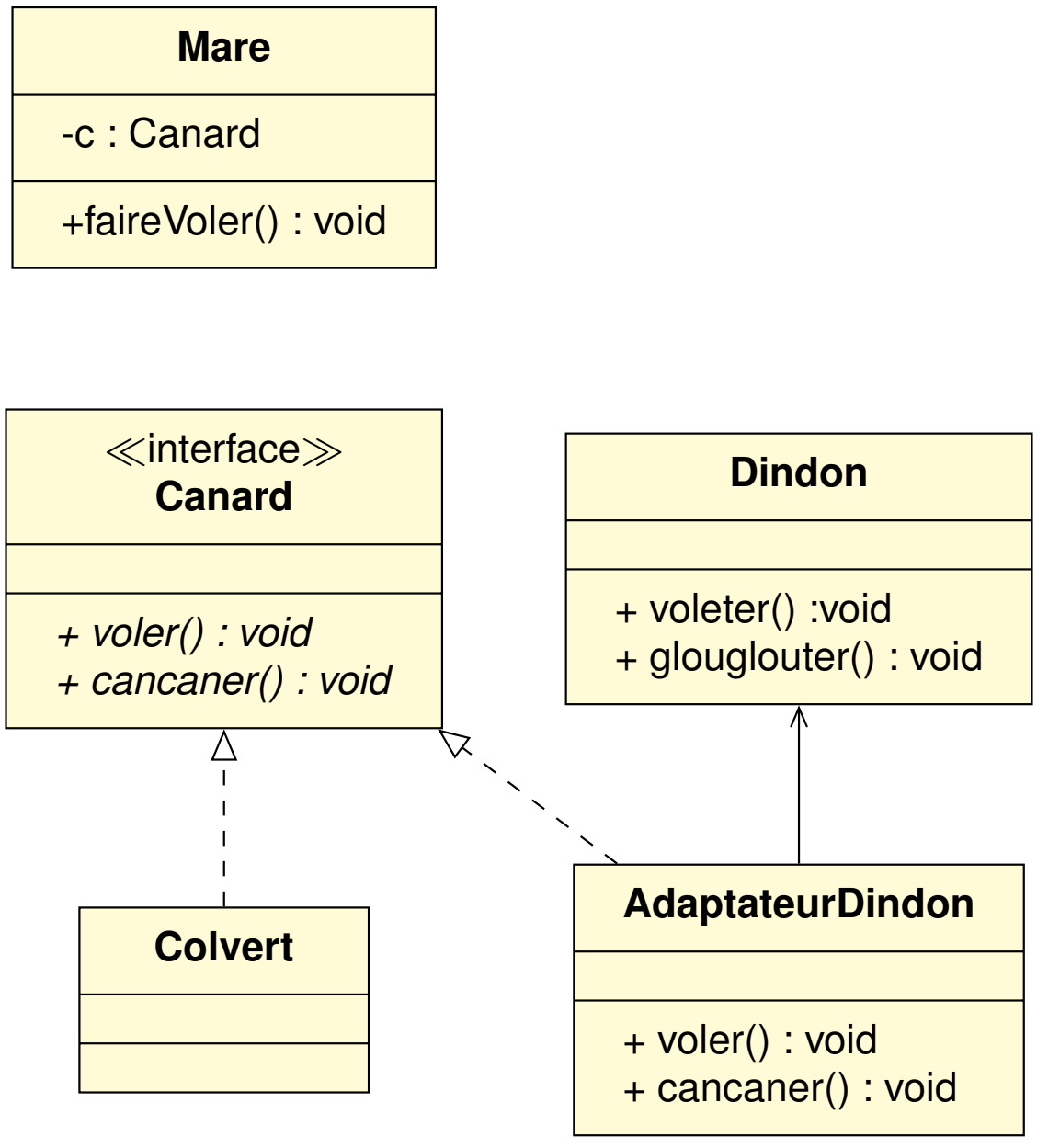
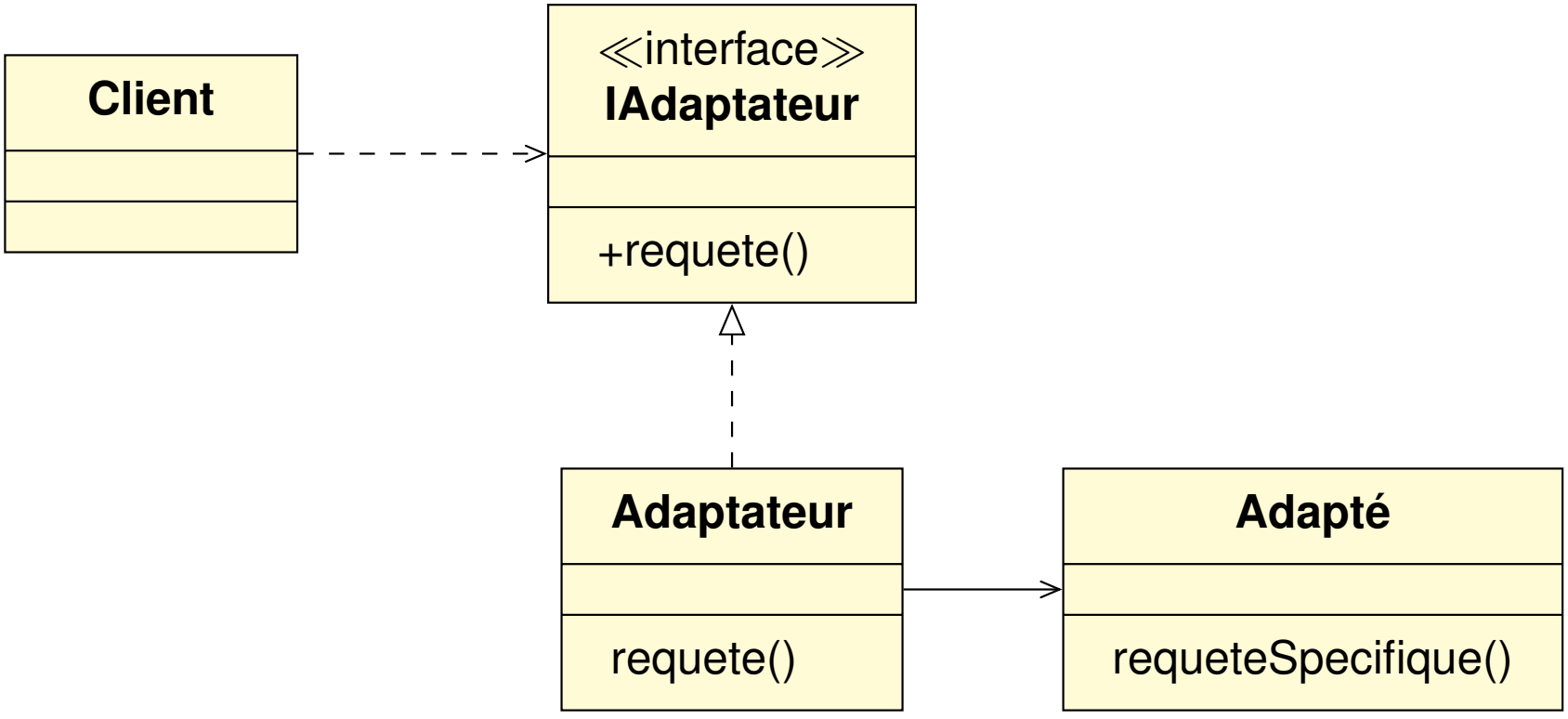


Diagramme de classe : cas général



Design pattern adaptateur

Objectif

- Faire correspondre à une interface donnée un objet (ou une classe) que l'on ne contrôle pas

Problème

- Un système a les bonnes données et les bonnes méthodes, mais pas les bonnes interfaces

Solution

- Délégation (objet) ou héritage (adaptateur de classe)

Conséquence

- Des objets existants peuvent être intégrés à de nouvelles structures de classes sans être limités par leur interface

Les constituants du DP adaptateur

IAdaptateur

- Définit les méthodes requises

Adapté

- Fournit les méthodes mais n'implémente pas l'interface

Adaptateur

- Adapte les méthodes de la classe `Adapté` pour qu'elles correspondent à la cible de l'`IAdaptateur`

Indications d'utilisation du DP adaptateur

- Une API (interface de programmation applicative) développée par un tiers contient le code recherché mais la signature des méthodes ne correspond pas
- D'anciennes classes doivent être utilisées sans réécrire tout leur code

Le code de l'adaptateur

La classe AdaptateurDindon

```
public class AdaptateurDindon implements Canard {
  private Dindon dindon;

  public AdaptateurDindon() {
          this.dindon = new Dindon();
  }

  public void cancaner() {
    dindon.glouglouter();
  }

  public void voler() {
    for(int i=0; i < 5; i++) {
      dindon.voleter();
    }
  }
}
```

Le code de l'adaptateur

Une partie du code de la Mare

```
public class Mare {
  public static void main(String[] args) {
    Canard canard = new Colvert();
    Canard adaptateurDindon = new AdaptateurDindon();

    System.out.println("\nCanard ");
    canard.cancaner();
    canard.voler();

    System.out.println("\nAdaptateurDindon ");
    adaptateurDindon.cancaner();
    adaptateurDindon.voler();
  }
}
```

Plan

Adaptateur

Observateur

Étude du problème

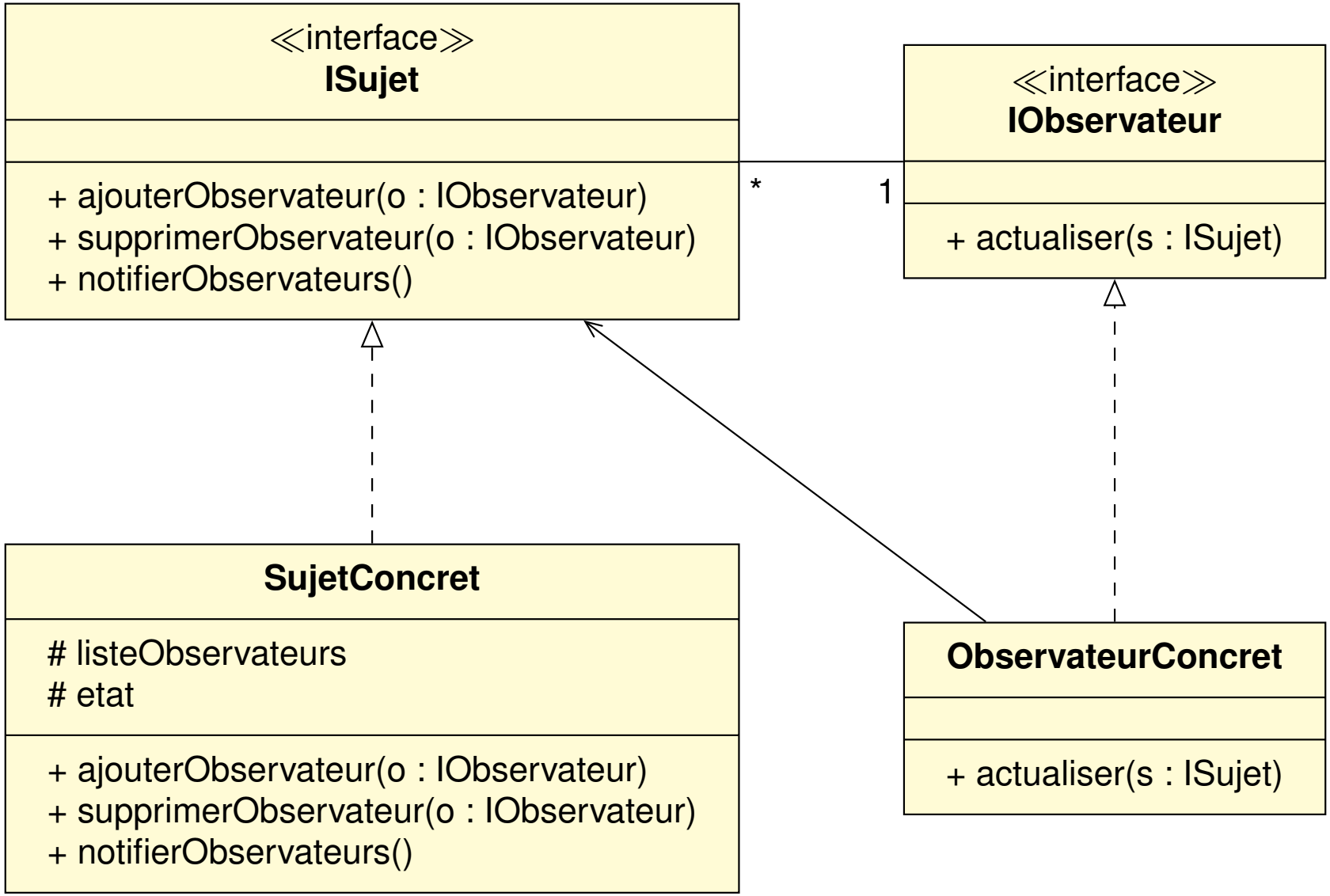
Un problème de GPS

On souhaite réaliser une application permettant à des personnes de se localiser grâce au GPS.

Une personne souhaitant se localiser utilise un récepteur GPS. Ce récepteur reçoit des informations (position, date) d'au moins 3 satellites. Grâce à ces informations, il peut calculer sa position.

Une classe (RecepteurGPS) va stocker les informations du récepteur. Deux classes `AfficheComplet` et `AfficheResume` affichent ces informations dès lors qu'elles sont mises à jour.

Diagramme de classes



Design pattern observateur

Objectif

- Définir une dépendance « un à plusieurs »entre des objets pour que tous ceux qui dépendent d'un objet soient avertis de son changement d'état et mis à jour automatiquement

Problème

- Avertir une liste variable d'objets de la mise à jour

Solution

- Les objets Observateur délèguent la responsabilité de contrôle d'un événement à un objet central, le Sujet

Participants

- Le Sujet connaît les Observateur car ces derniers s'enregistrent auprès de lui. Le Sujet doit avertir les Observateurs dès qu'un événement a lieu

Les constituants du DP observateur (1)

Sujet

- Fournit une interface pour ajouter, supprimer et informer les objets observateurs

SujetConcret

- Connaît ses observateurs (un nombre quelconque d'observateurs peut observer un sujet)
- Mémorise les états qui intéressent les objets observateurs
- Envoie une notification à ses observateurs lorsqu'il change d'état

Les constituants du DP observateur (2)

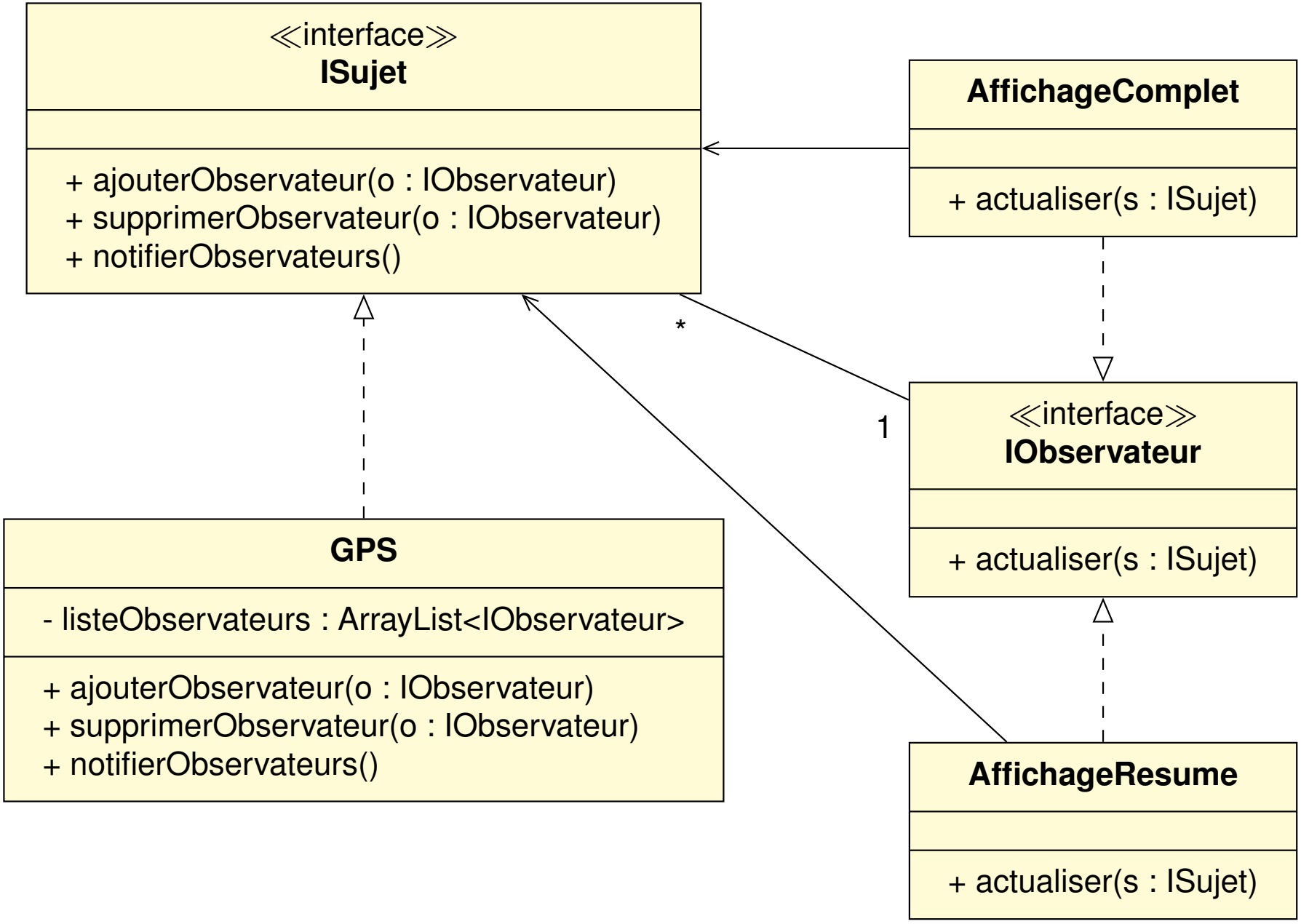
Observateur

- Définit une interface de mise à jour pour les objets qui doivent être notifiés des changements d'un sujet

ObservateurConcret

- Gère une référence sur un SujetConcret
- Mémorise l'état qui doit rester pertinent pour le sujet
- Implémente l'interface de mise à jour de l'Observateur pour conserver la cohérence de son état avec le sujet

Diagramme de classes du GPS



Indications d'utilisation du DP observateur

- Quand la modification d'un objet nécessite d'en modifier d'autres et que l'on ne sait pas combien sont ces autres
- Quand un objet doit être capable de faire une notification à d'autres objets sans faire d'hypothèse sur la nature de ces objets (i.e. ces objets ne doivent pas être trop fortement couplés)
- Quand un concept a plusieurs représentations, l'une dépendant de l'autre. Encapsuler ces deux représentations dans des objets distincts permet de les réutiliser et de les modifier indépendamment

Le code de l'observateur : la classe GPS

```
  private String position;
  private int precision;
  private ArrayList<IObservateur> listeObservateurs;

  public GPS() {
    position="pas connue"; precision=0;
    listeObservateurs = new ArrayList<IObservateur>();}

  public String getPosition() {return position;}
  public int getPrecision() {return precision;}
  public void setMesures(int precision, String position) {
    this.precision = precision;
    this.position = position;
    notifierObservateurs();}

  public void ajouterObservateur(IObservateur o) {
    listeObservateurs.add(o);}
  public void supprimerObservateur(IObservateur o) {
    listeObservateurs.remove(o);}
  public void notifierObservateurs() {
    for (IObservateur o : listeObservateurs)
      o.actualiser(this);}
}
```

Le code de l'observateur

La classe AffichageComplet

```
public class AffichageComplet implements IObservateur {
  public void actualiser(ISujet s) {
    if (s instanceof GPS){
      GPS gps = (GPS) s;
      System.out.println("Position : "+gps.getPosition()+
      " avec une précision de : "+gps.getPrecision()+"/10");
    }
  }
}
```