M1 info

GINF41B2 (Conception et Programmation Orientée Objet)

Cours #6
Classes abstraites et interfaces

Pierre Tchounikine

Plan

- Classes abstraites
  - point de vue technique
  - utilisation
- Interfaces
  - point de vue technique
  - utilisation : réalisation de l'héritage multiple
  - utilisation : réalisation de contrats de services

Classes abstraites
(point de vue technique)

Classe abstraite

- Une classe abstraite est une classe qui n'est pas destinée à créer des objets

  « classe non instanciable »

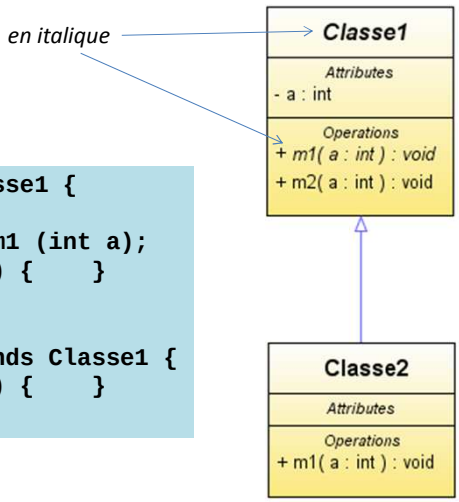
  ⇒ destinée à (faciliter) la création de sous-classes

un chat est un félin et un félin est un animal ;
dans la nature, il y a des chats, mais pas de félins ni d'animaux !

- Une classe abstraite peut comporter
  - *des champs*
  - des méthodes entièrement définies (« normales »)
  - des méthodes abstraites = signature + type de retour

    « méthode retardée », « méthode différée »

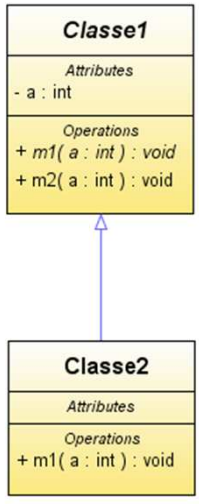
Classe abstraite : syntaxe




```
public abstract class Classe1 {
    private int a;
    public abstract void m1 (int a);
    public void m2 (int a) {    }
}

public class Classe2 extends Classe1 {
    public void m1 (int a) {    }
}
```

5/53
Cours P. Tchounikine

Classe abstraite : instances



Classe1 a1= new Classe1(); // légal ? ⟶ NON !
Classe2 b2 = new Classe2(); // légal ? ⟶ oui
Classe1 a2;
a2=b2; // légal ? ⟶ oui
Classe1 a3=new Classe2(); // légal ? ⟶ oui

```
Classe1 is abstract; cannot be instantiated
```

possibilité de référencer les objets via le type (la classe) abstrait(e)

6/53
Cours P. Tchounikine

Classe abstraite : règles de base

- Une classe qui a une méthode abstraite est une classe abstraite

  (« abstract » facultatif, mais il est conseillé de le mettre)

- Est-ce qu'une méthode « private » peut être abstraite ?

  NON ! sinon, comment en définir l'implantation ?

- Une classe abstraite peut
  - être une sous-classe d'une classe non-abstraite
  - ne définir que des méthodes abstraites (c'est même souvent le cas)
  - avoir des sous-classes qui sont elles-mêmes abstraites (→ définir une partie des méthodes abstraites de ses sous-classes)

classes abstraites : pas d'instances ➡ le fait que certaines méthodes soient (encore) abstraites ne pose pas de problème

classes concrètes : instances possibles ➡ il faut que toutes les méthodes soient définies

Classes abstraites
(utilisation)

Classe abstraite : à quoi ça sert ?

- Définir une classe abstraite C1 avec …
  - un ensemble de méthodes implémentées $\mathbb{M}1$ (éventuellement vide)
  - un ensemble des méthodes abstraites $\mathbb{M}2$

… permet de garantir que **toutes les sous-classes de C1 présenteront les comportements $\mathbb{M}1+\mathbb{M}2$**

(tous les objets effectifs créés sauront répondre aux messages)

permet de raisonner sur (de manipuler) les objets de type C1 sans se préoccuper de leur type effectif (de l'implantation de leurs méthodes)

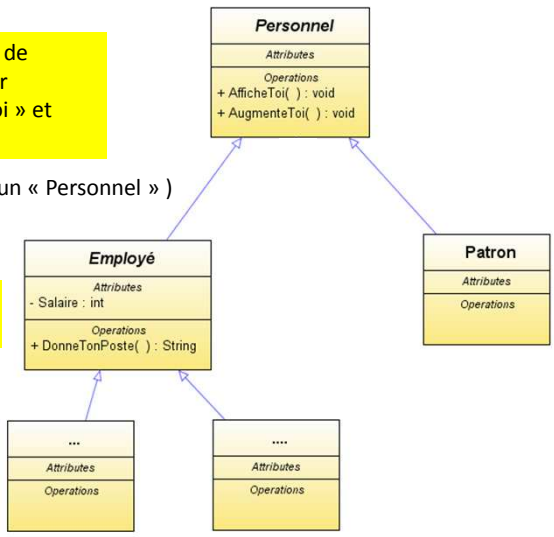
(plus que) pratique pour exploiter la puissance du polymorphisme

Classe abstraite : exemple

un objet créé comme un sous-type de Personnel ne peut pas ne pas savoir répondre aux messages « AfficheToi » et « AugmenteToi »

(mais aucun objet ne sera créé comme un « Personnel »)

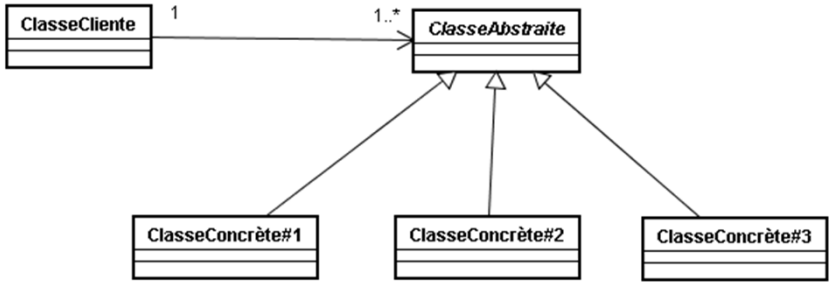
un Employe (Trader ? Guichetier ? autre ?) saura donner son poste



NB : toujours pas terrible comme modélisation

Classe abstraite : schéma fréquent

Cf. Design Pattern « Facade »



- déclaration d'un tableau d'objets abstraits (type statique)
- remplissage avec des objets concrets (type dynamique)
- exploitation du polymorphisme

- il peut y avoir plusieurs niveaux de classes abstraites avant les classes concrètes
- la profondeur des niveaux peut être dissymétrique

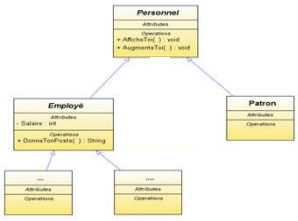
Classe abstraite et GL

CLASSES ABSTRAITES

Travail au niveau de la modélisation générale

Scénarios généraux

…



(une certaine forme de)
cohérence assurée

IMPLANTATION

Détails de modélisation

Implantation des comportements

…

Interfaces
(point de vue technique)

Interface
(comme un cas particulier de classe abstraite)

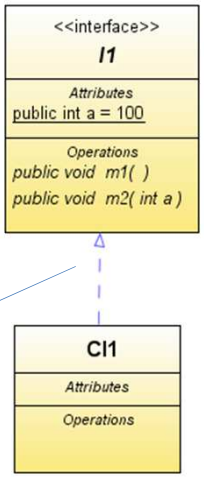
- Une interface est une classe abstraite
  - sans champ propre
  - dont toutes les méthodes sont abstraites (et donc publiques)

Une interface peut comporter
- *des constantes*
- des méthodes abstraites = signature + type de retour

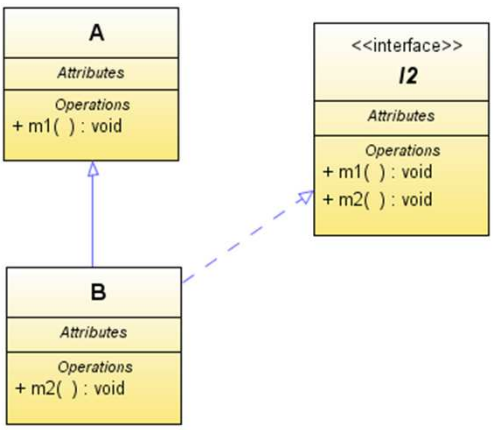
- Une interface a les propriétés d'une classe abstraite
  - définit un type (les messages acceptés)
  - ne permet pas de créer des objets
  - doit être implantée (implémentée) par une classe, i.e., une classe associée à l'interface doit définir le comportement (le code) des méthodes
  - permet de déclarer des variables via lesquelles référencer des objets de la classe implantant l'interface

Interface : syntaxe

```
public interface I1 {
    public static final int a = 100;
    public void m1 ();
    public void m2 (int a);
}

public class CI1 implements I1 {
    public void m1() {    }
    public void m2(int a) {    }
}
```



Interface : implémentation via héritage



Dérivation d'interfaces

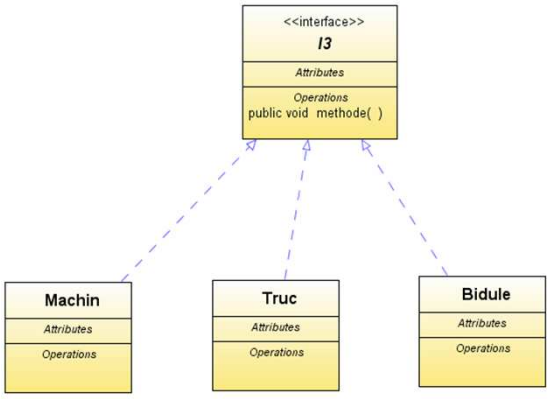
- La dérivation d'interface correspond à un simple emboitement des déclarations

```
interface I1 {
// constantes c1I1, c2I1, etc.
// méthodes m1I1, m2I1, etc.
}
interface I2 extends I1 {
// constantes c1I2, c2I2, etc.
// méthodes m1I2, m2I2, etc.
}
```

```
interface I2 {
// constantes c1I1, c2I1, etc.  + c1I2, c2I2, etc.
// méthodes m1I1, m2I1, etc. + m1I2, m2I2, etc.
}
```

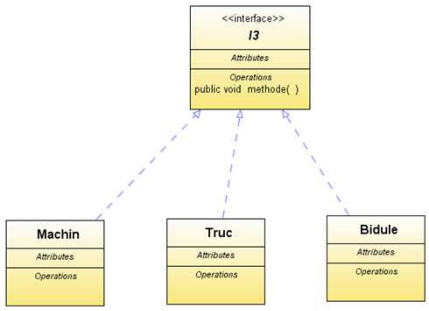

Particularité des interfaces

- Une interface peut être implémentée par plusieurs classes



Machin, Truc et Bidule doivent redéfinir `methode()`

Exemple

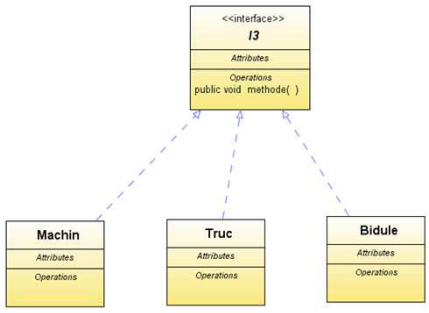



```
I3 o;
Machin m = new Machin();
Truc t = new Truc();
Bidule b = new Bidule();

o=m; // légal ?
m=t; // légal ?
o=t; // légal ?
o=b; // légal ?
```

oui
non
oui
oui

Exemple

```
I3 o;
Machin m = new Machin();
Truc t = new Truc();
Bidule b = new Bidule();
```




```
I3 tab []={m,t,b};
for (int i=0; i<tab.length; i++)
 {tab[i].methode(); }  // légal ?
```

oui !!

La notion d'interface permet de manipuler un tableau d'objets qui ne sont pas de mêmes types (pas d'héritage) tout en utilisant (quand même) le polymorphisme

Interface

(en tant que notion sécifique)

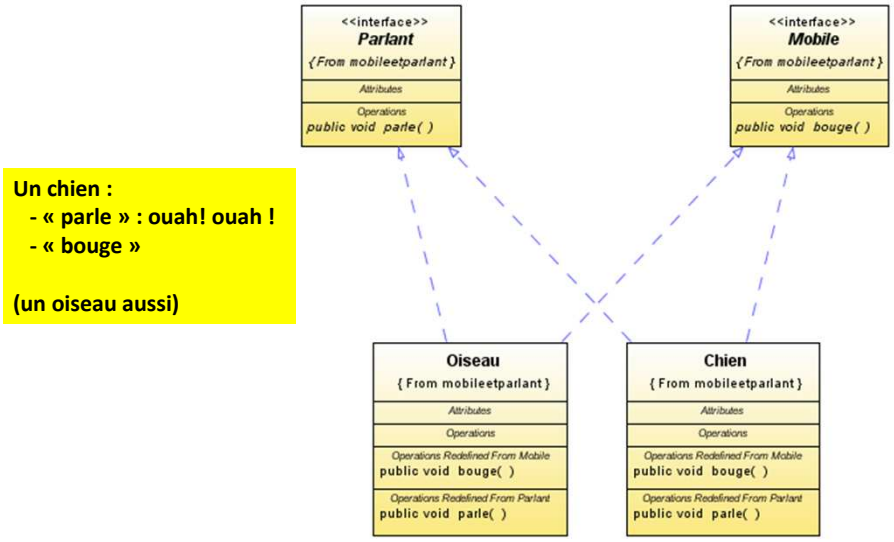
- Une interface définit une spécification
  - *des constantes*
  - des méthodes abstraites = signature + type de retour
- Cette interface peut être mise en œuvre par des classes différentes et sans relations (types différents)
- Il est possible de référencer les objets des classes implantant les interfaces via une référence du type = interface

Exemple : tableau hétérogène de Machin, de Truc et de Bidule

Faisable par des classes abstraites et de l'héritage ?
Parfois, mais c'est vilain d'utiliser l'héritage pour des objets sans relation « est-un » !

Interfaces : réalisation de l'héritage multiple

Interfaces et héritage multiple



Interfaces et héritage multiple

```
public interface Mobile {
public void bouge();
}
```

```
public interface Parlant {
public void parle();
}
```

```
public class Chien implements Mobile, Parlant {
public void bouge ( ) {System.out.println(":: ::");   }
public void parle ( ) {System.out.println("ouah ouah");   }
}
```

```
public class Oiseau implements Mobile, Parlant {
public void bouge ( ) {System.out.println("v v v");   }
public void parle ( ) {System.out.println("cui cui"); }
}
```

Utilisation

```
Mobile zoo[ ] = {new Oiseau(), new Chien()};

for (int i=0; i<zoo.length; i++) {
            zoo[i].bouge( );
            ((Parlant)zoo[i]).parle( );
      }
```

```
v v v
cui cui
:: ::
ouah ouah
```

modélisation (et donc appel) dissymétriques : PAS BEAU !!

25/53
Cours P. Tchounikine

Interface « de typage »

```
public interface Animal {
}
```

Ajout d'une interface de typage
« Animal » pour ranger les animaux

```
public class Chien implements Animal, Mobile, Parlant {…}

public class Oiseau implements Animal, Mobile, Parlant {…}
```

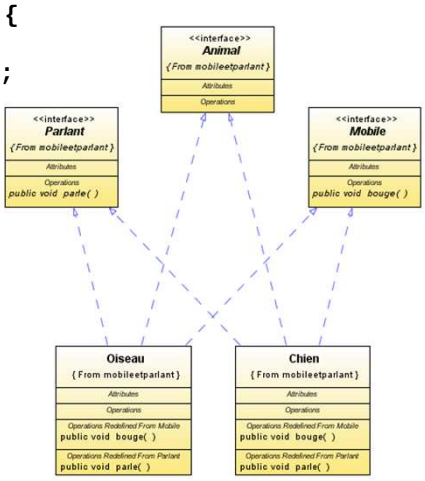
26/53
Cours P. Tchounikine

Utilisation

```
Animal zoo[ ] = {new Oiseau(), new Chien(),new Chien()};
for (int i=0; i<zoo.length; i++) {
       ((Mobile)zoo[i]).bouge( );
       ((Parlant)zoo[i]).parle( );
       }
```

```
v v v
cui cui
:: ::
ouah ouah
:: ::
ouah ouah
```

Remarque
Dans ce cas, on aurait pu utiliser une **classe** Animal plutôt qu'une interface



Interfaces : réalisation de contrats de services

Problématique (via un exemple)

- Un office de tourisme propose différents services :
  - trouver un hôtel
  - réserver un hôtel
  - donner un itinéraire

- En fait, un office du tourisme agrège des services fournis par d'autres acteurs
  - trouver un hôtel : centrale de réservation
  - réserver un hôtel : centrale de réservation
  - donner un itinéraire : plan de ville

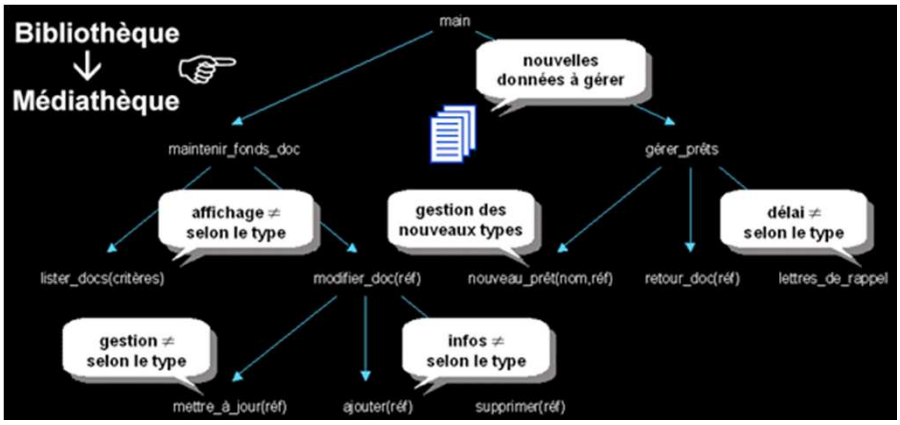
par exemple …

comment aborder ce type de situation avec un maximum de flexibilité ?

Limites de la conception fonctionnelle

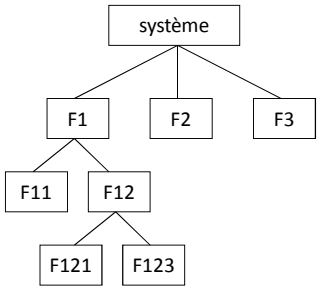
rappel

mais si on décide de gérer modifier les objets / gérer de nouveaux objets …

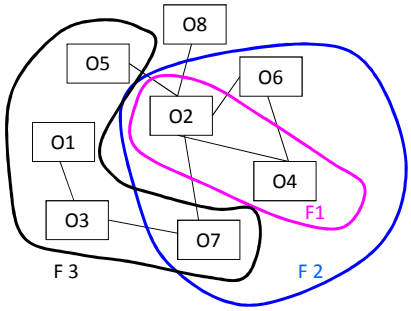


Conception fonctionnelle / objet

rappel



Architecture de fonctions



Réalisation de fonctions par collaboration d'objets

La centrale et le plan de ville

```
public class CentraleRéservation1 {
public String trouveHotel(){
System.out.println("Hotel trouvé par la centrale : Georges V");
return "Georges V";}
public void réserveHotel(String nom){
System.out.println("Hotel " + nom + "réservé par la centrale");}
}

public class PlandeVille1 {
public void donneItinéraire(String adresse){
System.out.println("Plan de ville, pour aller à " + adresse +
" prendre à gauche, puis à droite, puis tout droit");   };
}
```

L'office de tourisme

```
public class OfficedeTourisme1 {

public String trouveHotel(CentraleRéservation1 c){
return(c.trouveHotel());}

public void réserveHotel(String nom,CentraleRéservation1 c){
c.réserveHotel(nom);}

public void donneItinéraire(String nom,PlandeVille1 p){
p.donneItinéraire(nom);
}
```

connaissant une Centrale de réservation et un Plan de ville, un office de tourisme peut trouver un hôtel, le réserver et donner l'itinéraire

Utilisation

```
OfficedeTourisme1 ot = new OfficedeTourisme1();
CentraleRéservation1 c = new CentraleRéservation1 ();
PlandeVille1 p = new PlandeVille1();

String nom = ot.trouveHotel(c);
ot.réserveHotel(nom,c);
ot.donneItinéraire(nom,p);
```

```
Hotel trouvé par la centrale : Georges V
Hotel Georges V réservé par la centrale
Plan de ville, pour aller à Georges V prendre à gauche,
puis à droite, puis tout droit
```

Le GPS apparait …

il faut créer une classe GPS

```
public class GPS1 {
  public void donneItinéraire(String adresse){
        System.out.println("GPS, pour aller à " + adresse + " :
        partir du point (123, 456) et aller vers le Nord");  }
   }
```

il faut modifier le code de OfficeTourisme1 en ajoutant une méthode

```
public void donneItinéraire(String nom,GPS1 g){
g.donneItinéraire(nom);
}
```

à la place du donneItinéraire de PlandeVille ou en surcharge

et bien sur modifier le code de Main

Utilisation

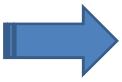
```
OfficedeTourisme1 ot = new OfficedeTourisme1();
CentraleRéservation1 c = new CentraleRéservation1 ();
PlandeVille1 p = new PlandeVille1();

String nom = ot.trouveHotel(c);
ot.réserveHotel(nom,c);
ot.donneItinéraire(nom,p); // donneItinéraire de PlandeVille
ot.donneItinéraire(nom,new GPS1()); // donneItinéraire de GPS
                                    // création d'un GPS anonyme
```

```
Hotel trouvé par la centrale : Georges V
Hotel Georges V réservé par la centrale
Plan de ville, pour aller à Georges V prendre à gauche,
puis à droite, puis tout droit
GPS, pour aller à Georges V : partir du point (123, 456)
et aller vers le Nord
```

Ca marche, mais c'est pas beau

modélisation / code fragile

- Chaque fois que l'on ajoute un nouveau moyen de trouver un hôtel ou un itinéraire, il faut
  - le modéliser et le réaliser **normal !**
  - modifier la classe OfficedeTourisme pour le prendre en compte
  - modifier le main **discutable** **pas normal !**
- Solution : utiliser l'abstraction



définition de services abstraits

Définition des interfaces

```
public interface TrouveurdeHotel {
public String trouveHotel();
}
```

définition des services à un niveau abstrait

```
public interface RéserveurdeHotel {
public void réserveHotel(String nom);
}
```

```
public interface DonneurdeItinéraire {
public void donneItinéraire(String adresse);
}
```

Réalisation des interfaces

public class CentraleRéservation2 implements TrouveurdeHotel, RéserveurdeHotel{

même code que précédemment

```
public String trouveHotel(){
System.out.println("Hotel trouvé par la centrale : Georges V");
return "Georges V";
}
public void réserveHotel(String nom){
System.out.println("Hotel " + nom + " réservé par la centrale");
}}
```

public class PlandeVille2 implements DonneurdeItinéraire {…}

public class GPS2 implements DonneurdeItinéraire{…}

L'office de tourisme abstrait

public class OfficedeTourisme2 {

```
// j'utilise des services sans savoir comment ils sont réalisés

public String trouveHotel(TrouveurdeHotel t){
return(t.trouveHotel());}

public void réserveHotel(String nom,RéserveurdeHotel r){
r.réserveHotel(nom);}

public void donneItinéraire(String nom,DonneurdeItinéraire d){
d.donneItinéraire(nom);
}
}
```

connaissant un Trouveur d'hôtel, un Réserveur d'hôtel et un Donneur d'itinéraire, un office de tourisme peut trouver un hôtel, le réserver et donner l'itinéraire

Utilisation

```
OfficedeTourisme2 ot = new OfficedeTourisme2();
TrouveurdeHotel t = new CentraleRéservation2();
RéserveurdeHotel r = new CentraleRéservation2 ();
DonneurdeItinéraire d = new PlandeVille2 ();

String nom = ot.trouveHotel(t);
ot.réserveHotel(nom,r);
ot.donneItinéraire(nom,d);
```

même chose que précédemment

```
Hotel trouvé par la centrale : Georges V
Hotel Georges V réservé par la centrale
Plan de ville, pour aller à Georges V prendre à gauche, puis à droite, puis tout droit
```


Le GPS apparait …

il faut créer une classe GPS

```
public class GPS2 implements DonneurdeItinéraire {
  public void donneItinéraire(String adresse){
        System.out.println("GPS, pour aller à " + adresse + " :
        partir du point (123, 456) et aller vers le Nord");   }
   }
```

il n'est pas nécessaire de modifier le code de OfficeTourisme2

et il faut bien sur modifier le code de Main

(vu que c'est là que l'on déclare les variables dans ce cas)

Exécution

```
OfficedeTourisme2 ot = new OfficedeTourisme2();
TrouveurdeHotel t = new CentraleRéservation2();
RéserveurdeHotel r = new CentraleRéservation2 ();
DonneurdeItinéraire d = new PlandeVille2 ();

String nom = ot.trouveHotel(t);
ot.réserveHotel(nom,r);
ot.donneItinéraire(nom,d);

DonneurdeItinéraire d2 = new GPS2();
ot.donneItinéraire(nom,d2);
```

l'office du tourisme sait donner l'itinéraire grâce au trouveur d'hôtel qu'on lui donne, sans avoir à savoir ce que c'est concrètement

```
Hotel trouvé par la centrale : Georges V
Hotel Georges V réservé par la centrale
Plan de ville, pour aller à Georges V prendre à gauche, puis à
droite, puis tout droit
GPS, pour aller à Georges V : partir du point (123, 456) et
aller vers le Nord
```


La révolution Internet

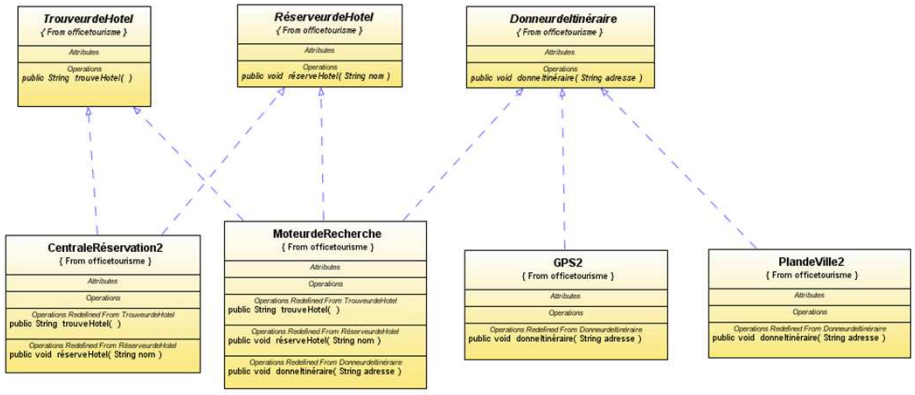
```
public class MoteurdeRecherche implements
TrouveurdeHotel,RéserveurdeHotel,DonneurdeItinéraire{

public String trouveHotel(){
System.out.println("Internet propose : Chez Mimile");
    // Internet c'est sympa mais c'est cheap
return "Chez Mimile";}

public void réserveHotel(String nom){
System.out.println("Internet : hôtel " + nom + " réservé");}

public void donneItinéraire(String adresse){
System.out.println("Internet : pour aller à " + adresse + "
Imprimante bloquée, ne peut pas imprimer Figç%µ1!!239856");  }
}
```

Exécution

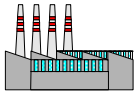
```
OfficedeTourisme2 ot = new OfficedeTourisme2();
MoteurdeRecherche m = new MoteurdeRecherche();

String nom = ot.trouveHotel(m);
ot.réserveHotel(nom,m);
ot.donneItinéraire(nom,m);
```

```
Internet propose : Chez Mimile
Internet : hôtel Chez Mimile réservé
Internet : pour aller à Chez Mimile Imprimante bloquée,
ne peut pas imprimer Figç%µ1!!239856
```


Réalisation multiple d'interfaces



La fabrique (1/2)

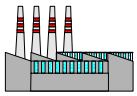


```
import java.util.Random;
public class Fabrique {
// Fabrique des implantations des interfaces,
// i.e., des services

 TrouveurdeHotel fabriqueunTrouveurdeHotel (){
Random randomGenerator = new Random();
int a = new Random().nextInt(10);
System.out.println("a= " +a);
{if (a<3) return new CentraleRéservation2();
 else     return new MoteurdeRecherche();}
}

…
```

utiliser le bon service au bon moment …

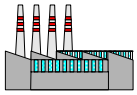
crée et renvoie des objets de types différents mais qui sont tous des « TouveurdeHotel »

La fabrique (2/2)



```
...

 RéserveurdeHotel fabriqueunRéserveurdeHotel (){
int a = new Random().nextInt(10);
System.out.println("a= " +a);
{if (a<3) return new CentraleRéservation2();
 else     return new MoteurdeRecherche();}
}
 DonneurdeItinéraire fabriqueunDonneurdeItinéraire (){
int a = new Random().nextInt(10);
System.out.println("a= " +a);
{if (a<3)          return new PlandeVille2();
 else if (a<7)     return new GPS2();
 else              return new MoteurdeRecherche();}
}
}
```

Utilisation



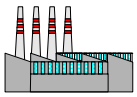
```
OfficedeTourisme2 ot = new OfficedeTourisme2();
Fabrique f = new Fabrique();

TrouveurdeHotel t = f.fabriqueunTrouveurdeHotel();
RéserveurdeHotel r = f.fabriqueunRéserveurdeHotel();
DonneurdeItinéraire d = f.fabriqueunDonneurdeItinéraire();

String nom = ot.trouveHotel(t);
ot.réserveHotel(nom,r);
ot.donneItinéraire(nom,d);
```

l'office du tourisme sait trouver un hôtel (etc.) grâce au trouveur d'hôtel (etc.) que lui fournit la fabrique

```
a=0 ; a=5 ; a=4

Hotel trouvé par la centrale :
Georges V
Internet : hôtel Georges V
reservé
GPS, pour aller à Georges V :
partir du point (123, 456) et
aller vers le Nord
```

```
a=7 ; a=1 ; a=4

Internet propose : Chez Mimile
Hotel Chez Mimile réservé par
la centrale
GPS, pour aller à Chez Mimile :
partir du point (123, 456) et
aller vers le Nord
```

La fabrique (retour sur)



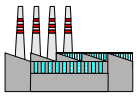
```
import java.util.Random;
public class Fabrique {
// Fabrique des implantations des interfaces,
// i.e., des services

TrouveurdeHotel fabriqueunTrouveurdeHotel (){ }
RéserveurdeHotel fabriqueunRéserveurdeHotel (){}
DonneurdeItinéraire fabriqueunDonneurdeItinéraire (){}
}
```

~~fabrique *ad hoc*~~ → définir une fabrique par service !

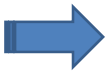
baisser le couplage

50/53
Cours P. Tchounikine

La fabrique (à reprendre et terminer)



- Une fabrique par type de service
- Une fabrique
  - contient une liste de serveurs qui lui permettent d'assurer le service
  - a une méthode « ajouter » qui permet d'ajouter d'autres serveurs



on peut ajouter des serveurs qui assurent le service dynamiquement (à l'exécution), sans modifier la fabrique ni le code de l'utilisateur

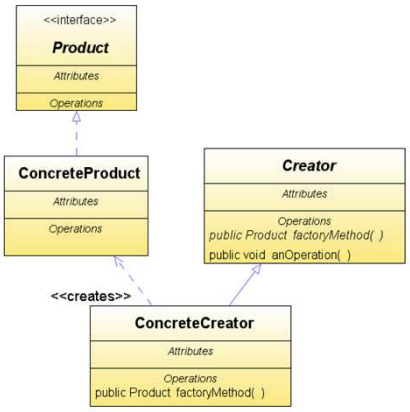
(assurer le service = même chose ou « différemment »)

Pour aller plus loin …

Design Pattern « fabrique »

Factory Method
Provide an interface for creating an object. Subclasses implementing the base Factory interface can decide what concrete object to create at runtime

Product : Defines the public interface that allows clients to create objects through.
Concrete Product : Implements the operations declared in the Product interface.
Creator : Defines the operations used to create objects that implement the Product interface.
ConcreteCreator : Implements the operations declared in the Creator interface



Pour aller plus loin …

Design Pattern « fabrique abstraite »

Abstract Factory
Defines an interface used to create objects in a generic manner, without having to specify concrete classes.

Roles

AbstractFactory : Defines a public interface used to create abstract product objects.

ConcreteFactory : Defines the operations to create concrete product objects.

AbstractProduct : Defines a public interface for a type of product.

ConcreteProduct : Declares a product object that will be instantiated by its related concrete factory. Defines the AbstractProduct interface.

Client : Manipulates the AbstractFactory and AbstractProduct objects.