Langages de programmation

TP 10

Objectifs

- Relations entre classes (suite)
- Diagrammes UML

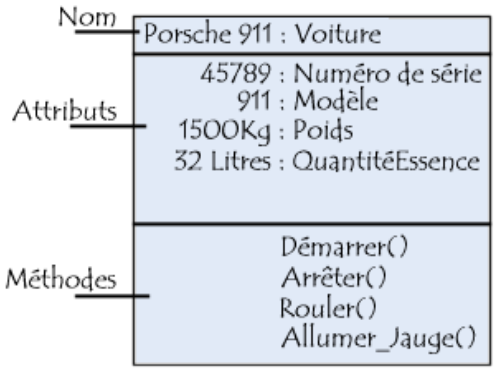
Théorie

Les diagrammes UML

UML (Unified Modeling Language, que l'on peut traduire par « langage de modélisation unifié ») est une notation permettant de modéliser un problème de façon standard.

Dans notre cas, UML est un formalisme de modélisation objet.

Avec la méthode UML, un objet est par exemple représenté de la façon suivante:



Théorie

Importance des diagrammes UML

Un diagramme UML permet de définir le problème à haut niveau sans rentrer dans les spécificités d'un langage, parce que les langages orientés objet constituent chacun une manière spécifique d'implémenter le paradigme objet

Un diagramme UML représente ainsi un outil permettant de définir un problème de **façon graphique**, afin de le présenter à tous les acteurs d'un projet

Un diagramme UML est une méthode **d'analyse du problème** (afin de couvrir toutes les facettes du problème), d'autre part un langage permettant **une représentation standard stricte des concepts abstraits** (la modélisation) afin de constituer un langage commun.

Théorie

Modélisation des classes en UML:

UML propose une manière de représenter les objets de façon graphique, sous forme **de rectangle, dans lequel le nom de l'objet est souligné:**

- Le premier contient le nom donné à la classe (non souligné).

- Les attributs d'une classe sont définis par un nom, un type (éventuellement une valeur par défaut, c'est-à-dire une valeur affectée à la propriété lors de l'instanciation) dans le second compartiment.

- Les opérations dans le troisième compartiment du rectangle.

Théorie

La visibilité des attributs en UML:

Les niveaux de visibilité des éléments de la classe définissent les droits d'accès aux données selon que l'on y accède par une méthode de la classe elle-même, d'une classe héritière, ou bien d'une classe quelconque. Il existe trois niveaux de visibilité:

- **publique (public):** Les fonctions de toutes les classes peuvent accéder aux données ou aux méthodes d'une classe définie avec le niveau de visibilité public. Il s'agit du plus bas niveau de protection des données

- **protégée (protected):** l'accès aux données est réservé aux fonctions des classes héritières, c'est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées

- **privée (private):** l'accès aux données est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé

Théorie

La notation UML permet de représenter le niveau de visibilité des attributs de façon graphique en faisant précéder le nom de chaque attribut par un caractère représentant la visibilité:

+ défini un attribut public

défini un attribut protégé

- défini un attribut privé

Théorie

Relations entre les classes:

1. L'association

C'est une simple relation d'utilisation

2. La dépendance

Relation d'utilisation unidirectionnelle => la modification d'un objet dont on dépend cause la mise a jour de l'autre élément

3. L'agrégation

Une association particulière où une classe est une partie d'une autre classe (un ensemble d'éléments)

4. La composition

Une agrégation forte (si un élément est détruit => ses composants sont aussi détruits)

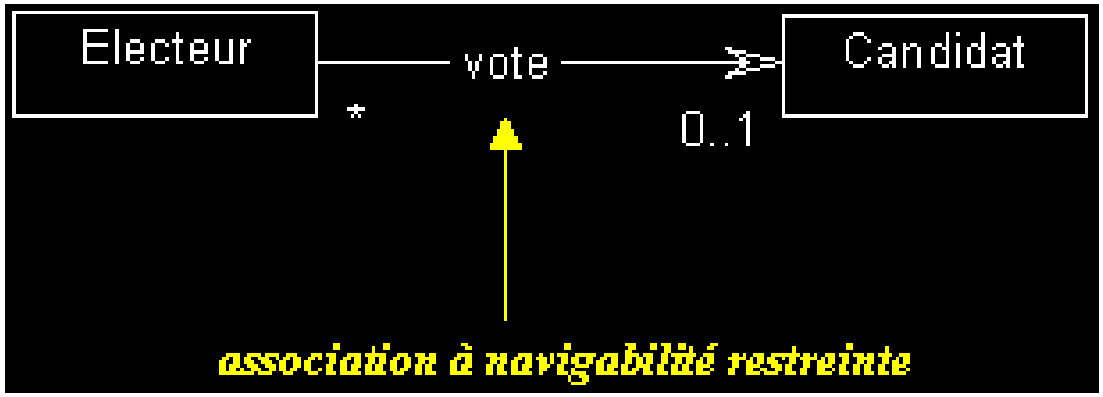
5. L'héritage (Spécialisation & Généralisation)

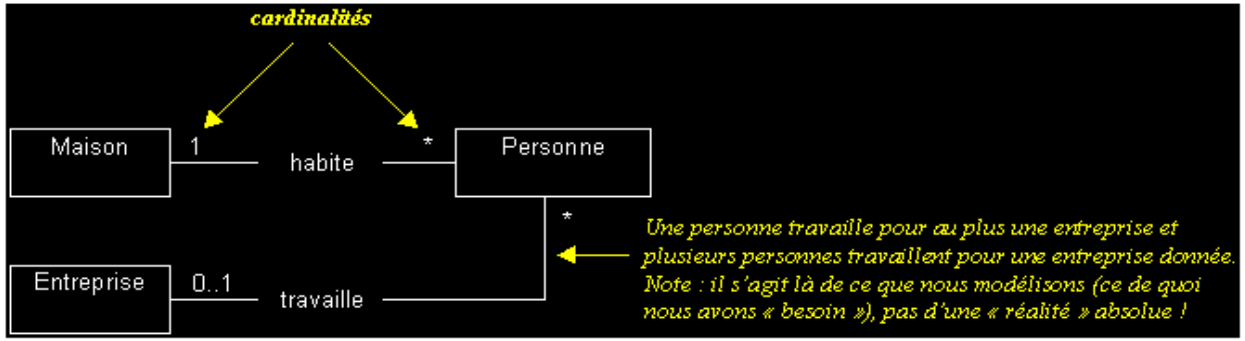
une classe hérite des caractéristiques d'une autre classe

Théorie

L'association

exprime une connexion sémantique entre deux classes

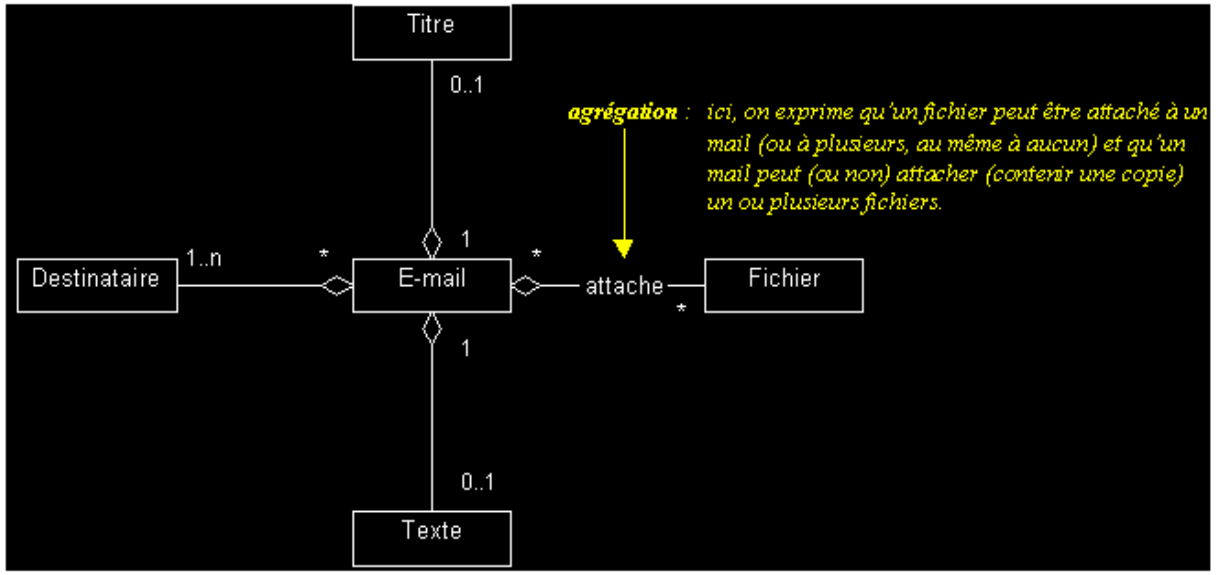




Théorie

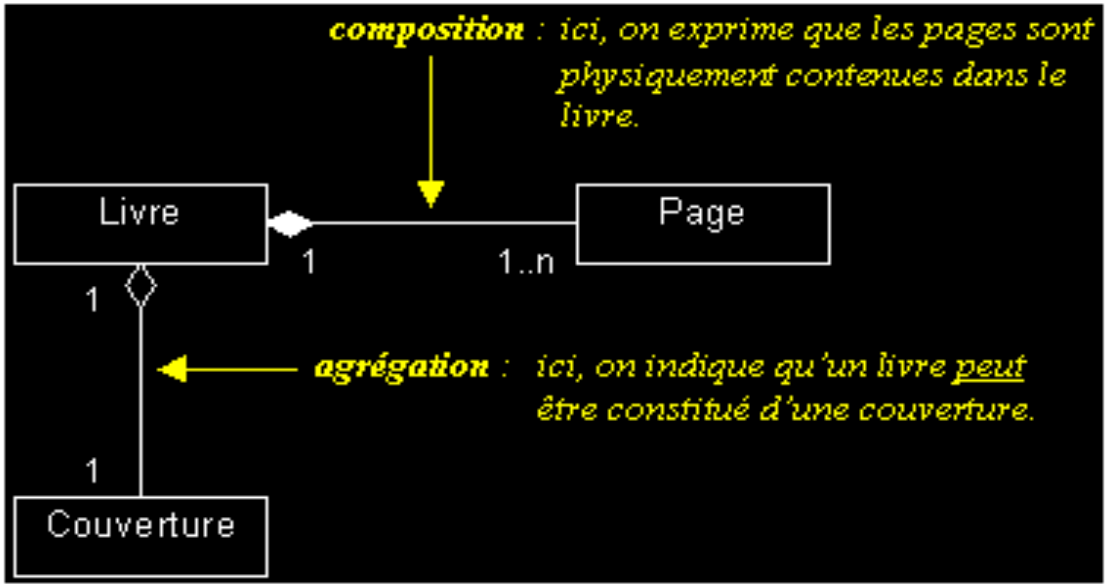
L'agrégation

une association non symétrique, qui exprime un couplage fort et une relation de subordination.



Théorie

L'agrégation
une agrégation forte



Théorie

La composition - Java

```
class Car {
        private Engine engine;
        public Car () {
                engine = new Engine();
        }
        void move() {
                engine.work();
        }
}
```

Composition

- Une instance de la classe “engine” est créée directement dans le constructeur de la classe Car, avec le mot clé “new”
- L’objet “engine” sera détruit sans la classe Car

Théorie

L'agrégation – Java

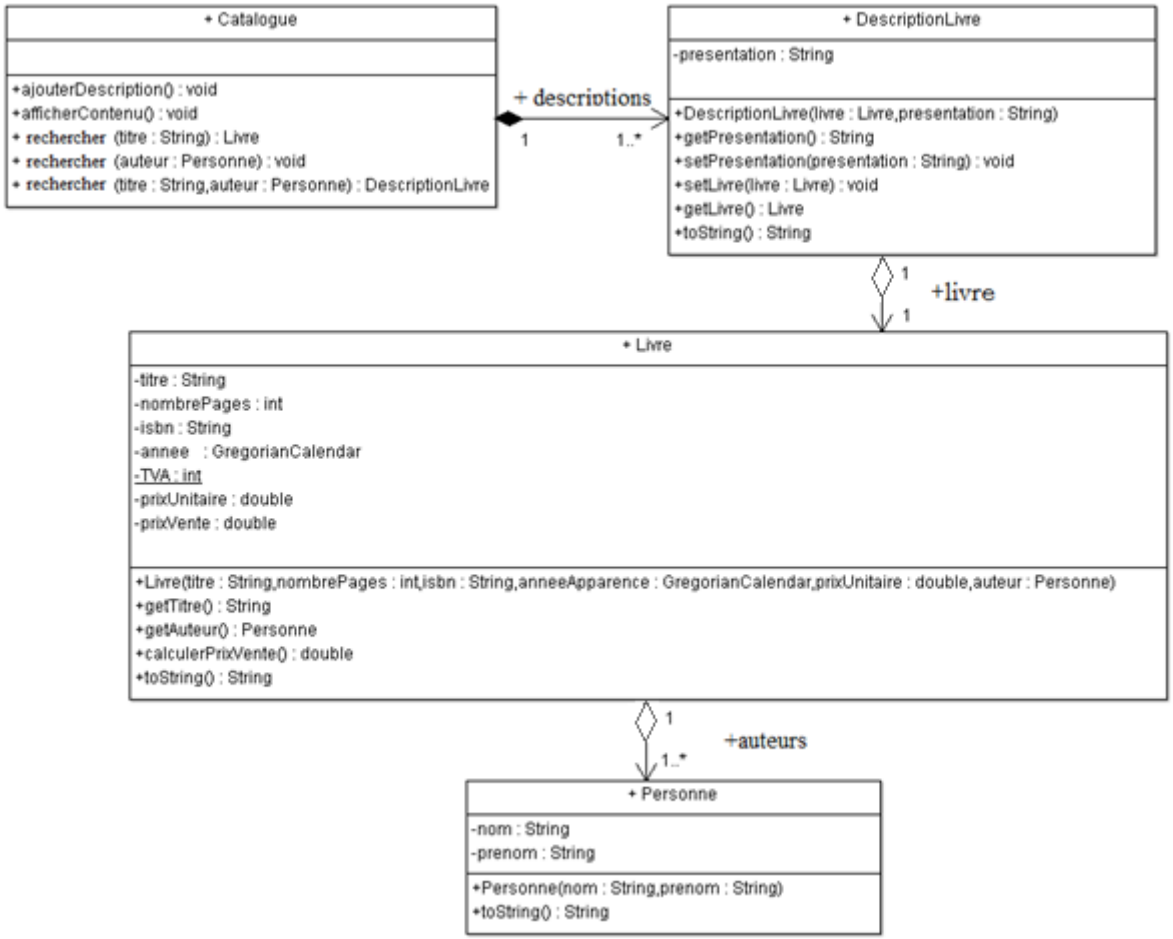
```
class Car {
        private Engine engine;
        public Car (Engine engine) {
                this.engine = engine;
        }
        void move() {
                if (engine != null)
                        engine.work();
        }
}
```

Agrégation

- Une instance de la classe "engine" existe a l'extérieur, elle est prie de là
- L'objet "engine" ne sera pas détruit sans la classe Car

Problèmes en classe

1. La diagramme de classes de ce problème est la suivante:



Problèmes en classe

Dans une librairie on vend des livres. Chaque livre contient les informations suivantes:

- auteur
- description
- nombre de pages
- ISBN unique, année d'apparition et le prix unitaire. Au prix unitaire nous ajoutons 10% (la valeur TVA) et nous obtenons le prix de vente pour chaque livre. Les descriptions des livres sont contenues dans un catalogue offert pour consultation aux clients.

Problèmes en classe

Ecrivez un programme qui exécute les fonctions suivantes:

- ajoute les descriptions de tous les livres dans le catalogue
- recherche par un ou deux critères (titre, auteur) et affiche les livres qui répondent à ces critères
- affiche le contenu du catalogue
- Ajouter les constructeurs, accesseurs et mutateurs nécessaires.

Devoir

1. Mettez en œuvre la classe Voiture. Une voiture est caractérisée par:

- Code unique (integer)
- Marque (string)
- Modèle (string)
- Prix à la production (double)
- Pays de production (string)
- Date de fabrication (GregorianCalendar " dd/mm/yyyy ")

Décidez les plus appropriés méthodes pour la classe.

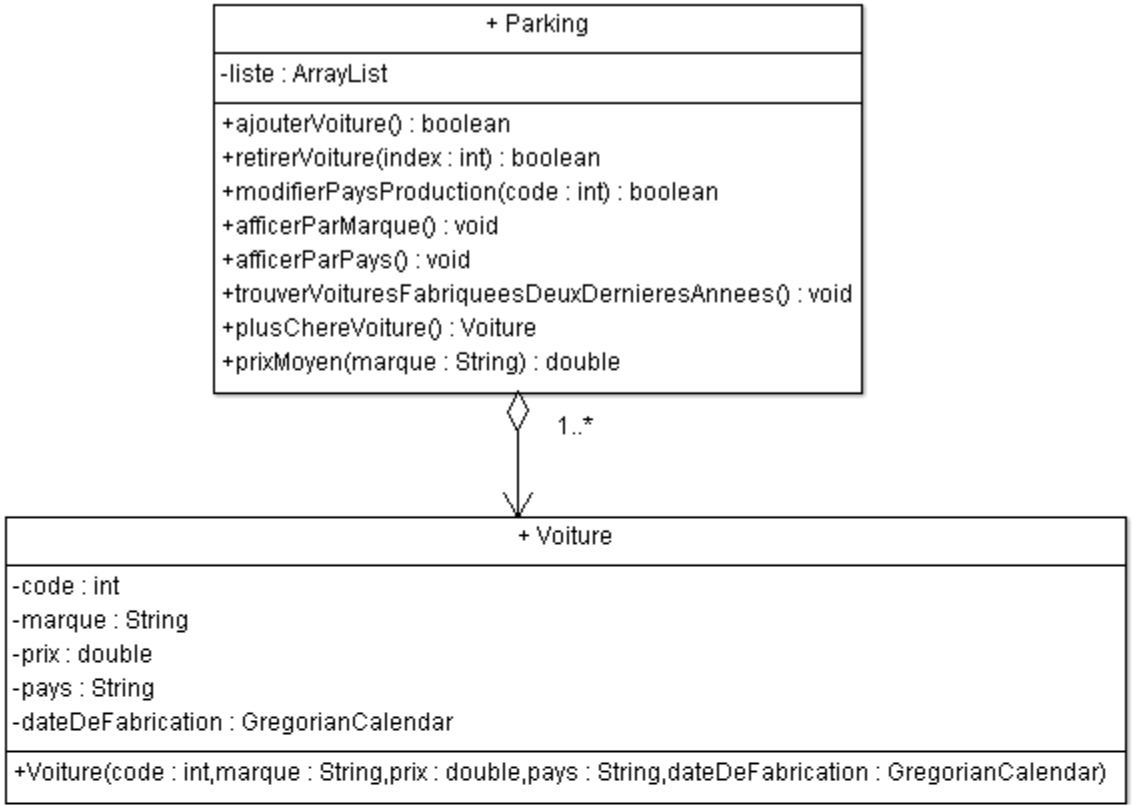
Devoir

Mettez en œuvre la classe Parking. Un parking est caractérisé par le nombre maximum de voitures qu'il peut contenir. Un parking contient un tableau de voitures et les méthodes suivantes:

- Ajoutez une nouvelle voiture
- Retirez une voiture en utilisant son numéro d'index dans le tableau
- Modifiez le pays de production d'une voiture à l'aide de son code unique
- Affichez toutes les voitures d'une marque
- Affichez toutes les voitures qui est produits dans un pays
- Trouvez toutes les voitures fabriquées au cours des deux dernières années
- Trouvez la voiture la plus chère dans le parking
- Trouvez le prix moyen des voitures de la même marque

Créez une classe de test, déclarez le parking avec un nombre maximum de 20 voitures, ajoutez les 5 premières voitures de la même marque, mais produites dans trois pays différents. Puis ajoutez 3 autres voitures d'une deuxième marque et testez les méthodes.

Devoir


+ Parking
-liste : ArrayList
+ajouterVoiture() : boolean
+retirerVoiture(index : int) : boolean
+modifierPaysProduction(code : int) : boolean
+afficerParMarque() : void
+afficerParPays() : void
+trouverVoituresFabriqueesDeuxDernieresAnnees() : void
+plusChereVoiture() : Voiture
+prixMoyen(marque : String) : double
1..*
+ Voiture
-code : int
-marque : String
-prix : double
-pays : String
-dateDeFabrication : GregorianCalendar
+Voiture(code : int,marque : String,prix : double,pays : String,dateDeFabrication : GregorianCalendar)