GL-Modélisation Orientée Objet

DOUHA DJAMEL
D.DOUHA@UNIV-BATNA2.DZ
FACULTÉ DE L'INFORMATIQUE ET DES MATHÉMATIQUE
UNIVERSITÉ DE BATNA2

Chapitre 4: Modélisation Statique

1. Diagramme de classes
   1. Notion d'objet,
   2. Classe,
   3. Détails et types de relation
      1. Relation d'association,
      2. Relation d'agrégation et de composition,
      3. Relation de dépendance,
      4. Relation réflexive,
      5. Qualification et Contraintes.
   4. Classe abstraite,
   5. Interface.
2. Diagramme d'objets.
   1. Notation et Exemple.

1. Diagramme de classes

- **Objet** : Définit une représentation abstraite d'une entité du monde réel ou virtuel, dans le but de la piloter ou de la simuler.
  - ***L'abstraction*** est le processus qui consiste à représenter des objets qui appartiennent au monde réel dans le monde du programme que l'on écrit. Il consiste essentiellement à extraire des variables pertinentes, attachées aux objets que l'on souhaite manipuler, et à les placer dans un modèle informatique convenable.

→ Objet = État + Comportement + Identité

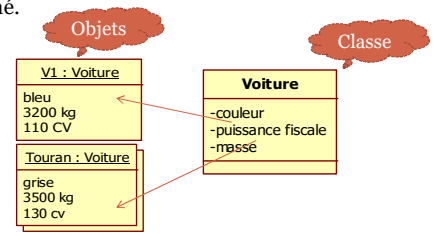
1. Diagramme de classes

- **L'identité:**
  - Chaque objet possède une identité qui caractérise son existence propre,
  - L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état.
  - Permet de distinguer deux objets dont toutes les valeurs d'attributs sont identiques.

1. Diagramme de classes

5

- **État:** Regroupe les valeurs instantanées de tous les attributs d'un objet,
  - Un attribut est une information qui qualifie l'objet qui le contient,
  - Chaque attribut peut prendre une valeur dans un domaine de définition donné.



1. Diagramme de classes

6

- L'état d'un objet à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différent s attributs,
- L'état évolue au cours du temps, il est la conséquence de ses comportements passés,
- Une voiture roule, la quantité de carburant diminue, les pneus s'usent, … etc,
- Certaines composantes de l'état peuvent être constantes.

1. Diagramme de classes

7

- **Le comportement**
  - Regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet,
  - Chaque atome (partie) de comportement est appelé opération,
  - Les opérations d'un objet sont déclenchées suite à une stimulation externe, représentée sous la forme d'un message envoyé par un autre objet,
  - L'état et le comportement sont liés,
  - Le comportement à un instant donné dépend de l'état courant et l'état peut être modifié par le comportement.

1. Diagramme de classes

8

- **Le comportement**
  - **Exemple** : il n'est pas possible de faire atterrir un avion que s'il est en train de voler: le comportement *Atterrir* n'est valide que si l'information *En vol* est valide.
  - Après l'atterrissage, l'information *En vol* devient invalide et l'opération *Atterrir* n'a plus de sens.

1. Diagramme de classes

9

- **Une classe** décrit une abstraction *des objets* ayant :
  - des *propriétés* similaires,
  - un *comportement* commun,
  - des relations identiques avec les autres objets,
  - une sémantique commune.

1. Diagramme de classes

10

- Une classe possède des fonctionnalités tel que:
  - Définir la structure générale des objets qu'elle crée en indiquant quelles sont les variables d'instance; «Patron: Template» à des objets,
  - Contenir et donner l'accès à l'ensemble des objets qu'elle a créés; «conteneur» d'objets,
  - Contenir des méthodes que ses objets peuvent utiliser puisque tous les objets d'une classe utilisent les mêmes méthodes; sert de «réceptacle». Il serait inutile de les dupliquer dans ces objets eux-mêmes.

1. Diagramme de classes

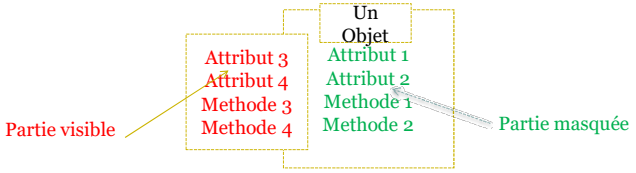
11

- **Caractéristiques** d'une classe
  - Un objet créé par une classe sera appelé une *instance* de cette classe ce qui justifie le terme «*variables d'instances*»,
  - les *valeurs des variables d'instances* sont propres à chacune de ces instances et les caractérisent,
  - Les généralités sont contenues dans la classe et les particularités sont contenues dans les objets,
  - Les objets sont construits à partir de la classe, par un processus appelé **instanciation** : tout objet est une instance de classe.
- **types de classes**
  - Classe concrète : elle peut être instanciée.
  - Classe abstraite : elle ne peut pas être instanciée, c'est une classe qui ne donne pas directement des objets.

1. Diagramme de classes

12

- **Encapsulation**
  - Consiste à masquer les détails d'implémentation d'un objet en définissant une interface,
  - Sépare entre les propriétés externes visibles des autres objets, et les aspects internes propres aux choix d'implantation d'un objet,



  - L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.

1. Diagramme de classes

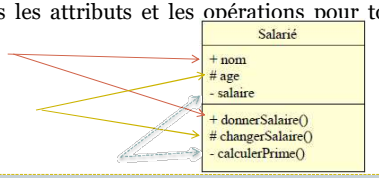
13

- **Encapsulation (suite):** présente un double avantage:
  - facilite l'évolution d'une application car elle stabilise l'utilisation des objets: nous pouvons modifier l'implémentation des attributs d'un objet sans modifier son interface.
  - garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs).
  - les utilisateurs d'une abstraction ne dépendent pas de la réalisation de l'abstraction mais seulement de sa spécification: ce qui réduit le couplage dans les modèles.
  - les données encapsulées dans les objets sont protégées des accès intempestifs.

1. Diagramme de classes

14

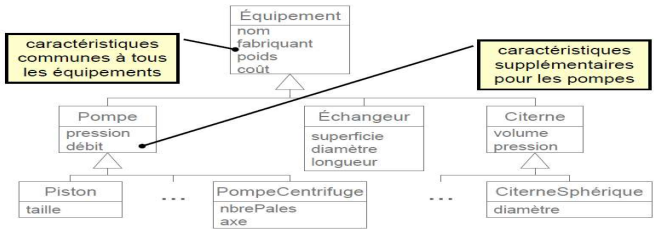
- **Visibilité:** il existe trois niveaux d'encapsulation:
  - Niveau privé: c'est le niveau le plus fort; la partie privée de la classe est totalement opaque.
  - Niveau protégé: c'est le niveau intermédiaire ; les attributs ou les opérations placés dans la partie protégée sont visibles par les classes dérivées de la classe fournisseur. Pour toutes les autres classes, les attributs ou les opérations restent invisibles.
  - Niveau publique: ceci revient à se passer de la notion d'encapsulation et de rendre visibles les attributs et les opérations pour toutes les classes.



1. Diagramme de classes

15

- **Utilisation de l'héritage**
  - dans le sens "**spécialisation**" pour *réutiliser* par modification incrémentielle les descriptions existantes.
  - dans le sens "**généralisation**" pour *abstraire* en factorisant les propriétés communes aux sous-classes,



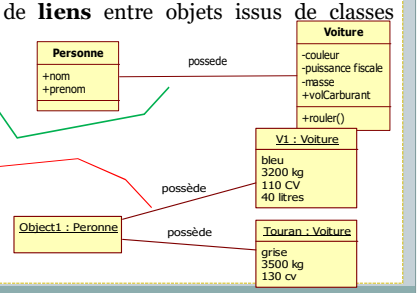
1. Diagramme de classes

16

- **Associations** entre les classes
  - Une **association** exprime une connexion sémantique bidirectionnelle entre deux classes.
  - L'instanciation d'une association, dans un diagramme d'objets ou de collaboration, sous forme de **liens** entre objets issus de classes associées.

Association :relation sémantique entre classes qui définit un ensemble de liens

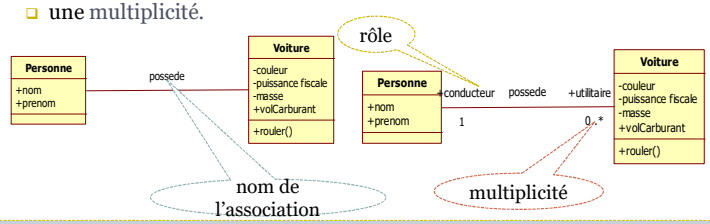
Lien : relation en objets qui définit une connexion sémantique ➔ instance d'une association entre instances de classes



1. Diagramme de classes

17

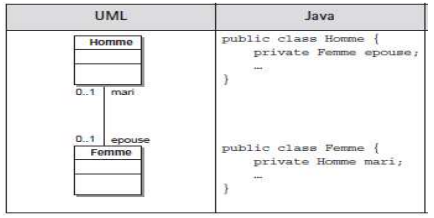
- **Caractéristique:** Une association est caractérisée par:
  - Une **association** exprime une connexion sémantique bidirectionnelle entre deux classes,
  - un nom, qui peut être omis notamment quand les rôles des classes sont spécifiés,
  - un rôle pour chacune des classes qui participent à l'association.
  - une multiplicité.



1. Diagramme de classes

18

- **Association (UML et code Java)**
  - Par défaut, une association est navigable dans les deux sens,
  - Exemple en Java:

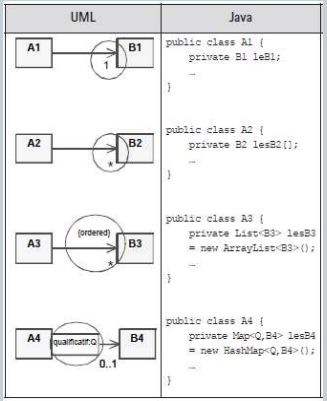


  - La réduction de la portée peut aussi être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre classe.

1. Diagramme de classes

19

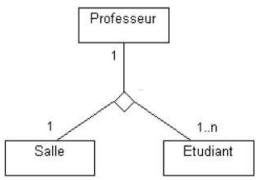
- **Association à navigabilité restreinte**
  - Le principe est appelé aussi, la réutilisation de code par délégation: la classe A délègue des tâches à une autre classe B.
  - En général, ce choix se fait dans la phase de conception.
  - Exemple UML to Java:

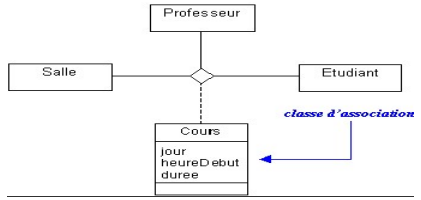


1. Diagramme de classes

20

- **Association n-aire**
  - Il s'agit d'une association qui relie plus de deux classes.
  - Elle peut souvent être promu au rang de classe d'association.



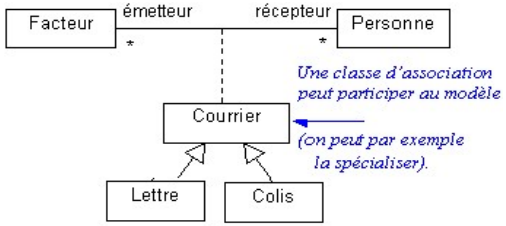


1. Diagramme de classes

21

Association n-aire

- Par le biais du principe d'héritage, on peut aussi la spécialiser:
- Classes spéciales: Lettre et Colis.
- Classe générique: Courrier.

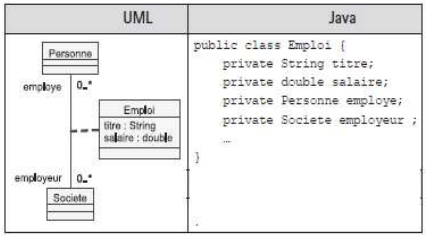


1. Diagramme de classes

22

Classe d'association

- Elle s'agit d'une association qui relie plus de deux classes.
- Elle s'agit d'une classe qui réalise la navigation entre les instances d'autres classes.

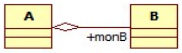


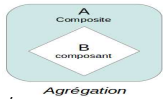
1. Diagramme de classes

24

Relation d'agrégation ou agrégation faible

- Une agrégation est un cas particulier d'une association non symétrique exprimant une relation de contenance ou subordination (dépendance). Les agrégations n'ont pas besoin d'être nommées : implicitement elles signifient *contient*, *appartient à* ou *est composé de* ,





Agrégation

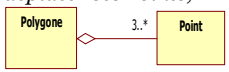
- Donc, elle exprime une relation de type: « ***contenant*** / ***contenu*** » ou « ***ensemble*** / ***élément*** », ou encore « ***agrégat*** / ***agrégé***» ou « ***composite*** / ***composant*** »
- Sur la figure, ***A*** est le composite et ***B*** le composant. Dans une agrégation, le composant peut être ***partagé entre plusieurs composites*** ce qui entraîne que, lorsque le composite ***A*** sera détruit, le composant ***B*** ne le sera pas forcément.

1. Diagramme de classes

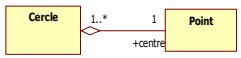
25

Une agrégation (suite)

- Par analogie à la généralisation et la spécialisation qui se repère par la relation « ***est un*** », *l'agrégation* se repère par la relation « ***est une partie de*** » ou « ***a un*** ».
- Par exemple, *déplacer un Polygone revient à déplacer ses Points,*



- Par exemple, *déplacer un Polygone revient à déplacer ses Points*. Il est important de noter que l'agrégation ***autorise*** qu'un élément de l'ensemble appartienne à un ***autre ensemble,***
- Un cercle possède un seul centre.



1. Diagramme de classes

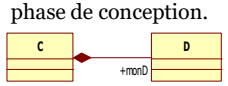
26

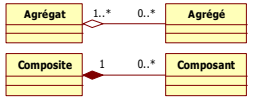
- Règles permettant de choisir une agrégation :
  - Est-ce une partie de ?
  - Les opérations appliquées à l'ensemble sont-elles appliquées à l'élément ?
  - Les changements d'états sont-ils liés ?
  - Est-ce un élément agrégé peut être lié à d'autres classes?
  - Est-ce la suppression de l'ensemble n'entraîne pas celle de l'élément?

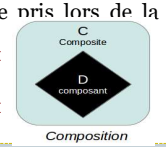
1. Diagramme de classes

28

- **Relation de composition**: Une composition est une agrégation avec une relation plus forte signifiant «*est composée d'un*».
- Contrairement à l'agrégation, le cas de la composition, une relation exprimant une relation de couplage encore plus fort : un élément appartient ***exclusivement*** à un ensemble.
  - La destruction du composite entraîne la destruction de toutes ses parties et il est responsable du cycle de vie de ses parties,
  - Une partie ne peut appartenir qu'à un seul composite (agrégation non partagée),
  - Le choix entre composition et agrégation peut être pris lors de la phase de conception.







Composition

1. Diagramme de classes

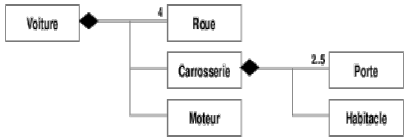
29

- **Composition ou agrégation forte (suite)**
  - Les éléments de l'ensemble « appartiennent » de manière exclusive à l'ensemble, contrairement à l'agrégation qui autorise qu'un élément appartienne à plusieurs ensembles,
  - Les composants sont créés par le composé,
  - Un élément n'existe pas sans être dans un ensemble.
- Ces deux règles impliquent, que le cycle de vie d'un élément (de la création jusqu'à la destruction) est contraint par le cycle de vie de l'ensemble.

1. Diagramme de classes

30

- **Exemples**
  - La destruction de la voiture provoque la destruction des roues. Cet exemple interdit donc la revente ou la récupération des roues de manière séparée comme dans les casses de voiture, alors il aurait fallu mettre une agrégation.

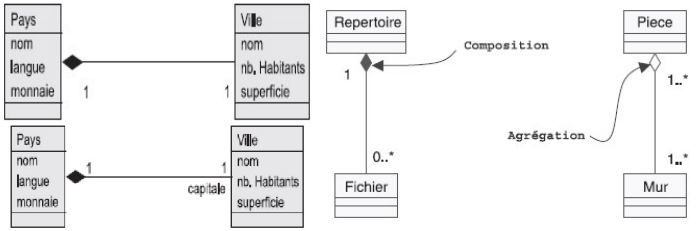


1. Diagramme de classes

31

Exemples

- Un mur peut appartenir à deux pièces. Un répertoire contient des fichiers,
- Exprimer le fait qu'un pays contient des villes, dont une seule joue le rôle de capitale.

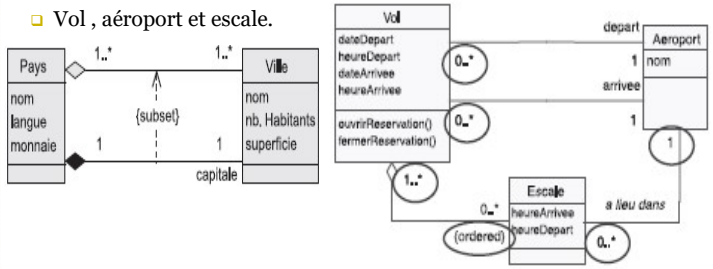


1. Diagramme de classes

32

Exemples

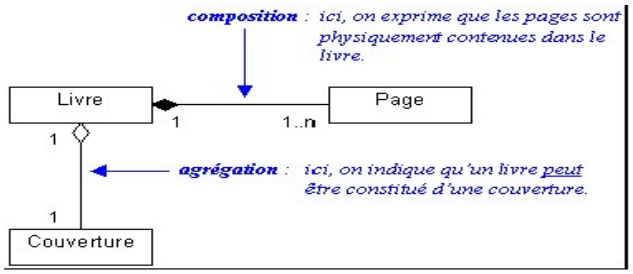
- une contrainte pour exprimer le fait que la capitale d'un pays est forcément une de ses villes. Et qu'une ville peut appartenir à plusieurs pays dans un souci de généralité.
- Vol , aéroport et escale.



1. Diagramme de classes

33

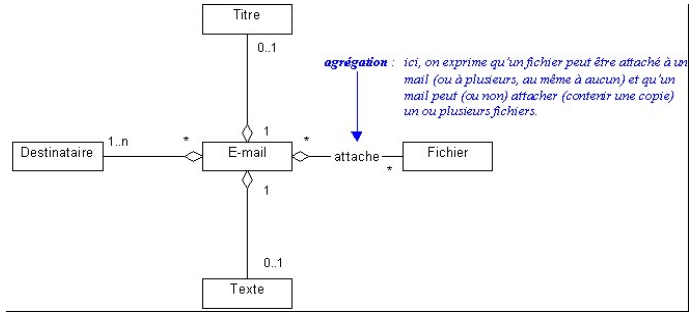
Exemples



1. Diagramme de classes

34

Exemples

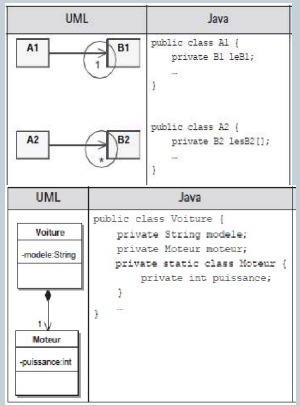


1. Diagramme de classes

35

Passage au code java

- La sémantique des agrégations n'est pas fondamentalement différente de celle des associations simples, elles se traduisent donc comme indiqué précédemment en Java,
- La seule contrainte est qu'une association ne peut contenir de marque d'agrégation (composition) qu'à l'une de ses extrémités.

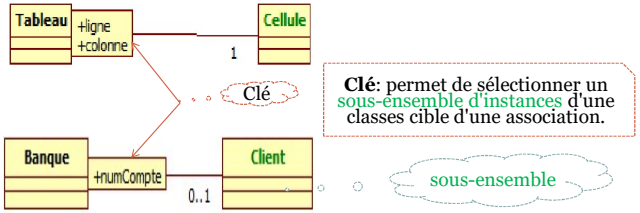


1. Diagramme de classes

36

Qualification

- Permet de sélectionner un *sous-ensemble d'objets*, parmi l'ensemble des objets qui participent à une association,
- La restriction de l'association est définie par **une *clé***, qui permet de sélectionner les *objets ciblés.*



1. Diagramme de classes

37

Contrainte sur une association: Les contraintes sont des expressions qui précisent le rôle ou la portée d'un élément de modélisation.

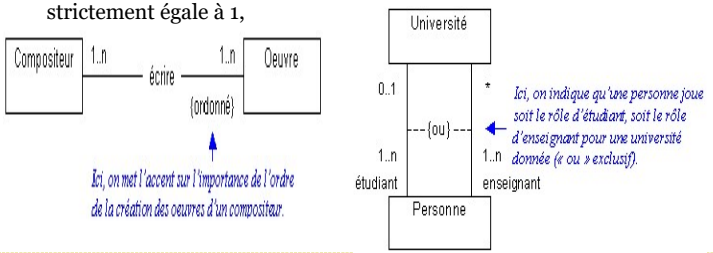
- elles permettent d'étendre ou préciser sa sémantique,
- sur une association, elles peuvent par exemple restreindre le nombre d'instances visées,
- les contraintes peuvent s'exprimer en langage naturel ou graphiquement; il s'agit d'un texte encadré d'accolades.

1. Diagramme de classes

38

Contrainte sur une association

- ***{ordred}*** ou Ordonné: introduit l'importance dans l'ordre de création des instances,
- ***{xor}*** ou (***ou***) : Introduire explicitement la contrainte prédéfinie {xor} entre les deux associations qui portent une multiplicité strictement égale à 1,

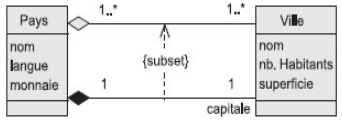


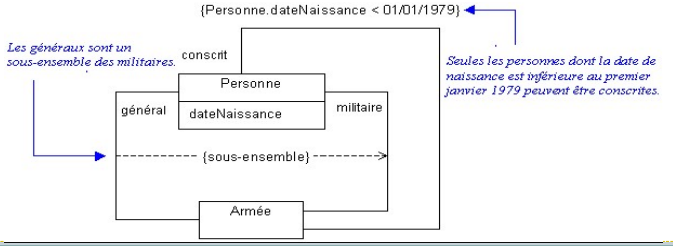
1. Diagramme de classes

39

Contrainte sur une association

- ***{subset}*** ou Sous-ensemble .





1. Diagramme de classes

40

Contrainte sur une association

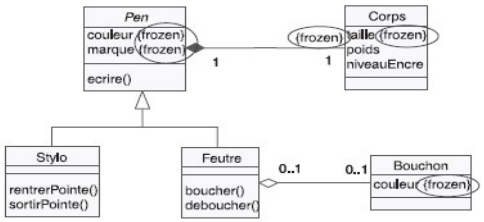
- ***{Frozen}*:** Cette contrainte permet d'ajouter une information détaillée, mais qui peut être intéressante, sur un diagramme de classes :
- Pour un *attribut*, le fait que sa valeur ***ne change jamais*** pendant la vie d'un objet (par exemple, la marque d'un Pen).
- Pour une *association*, le fait qu'un lien entre deux objets ***ne puisse plus jamais être modifié*** après sa création (par exemple, le lien de composition entre Pen et Corps, mais pas celui entre Feutre et Bouchon).
- Par défaut, les *attributs* et les associations ne sont pas {frozen}.

1. Diagramme de classes

41

Contrainte sur une association

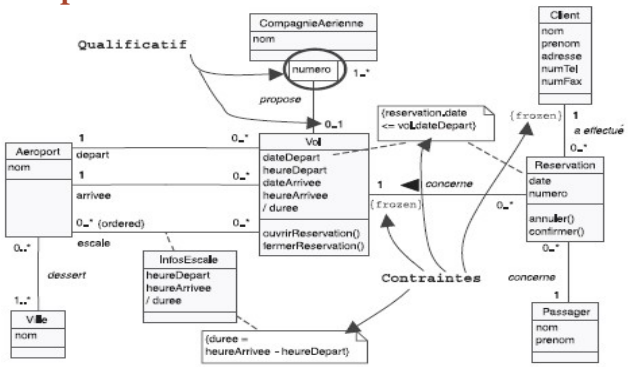
- Pour un *attribut*, le fait que sa valeur ***ne change jamais*** pendant la vie d'un objet (par exemple, la marque d'un Pen).
- Pour une *association*, le fait qu'un lien entre deux objets ***ne puisse plus jamais être modifié*** après sa création (par exemple, le lien de composition entre Pen et Corps, mais pas celui entre Feutre et Bouchon).



1. Diagramme de classes

42

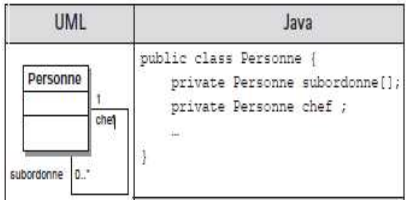
Exemple réservation de vol



1. Diagramme de classes

43

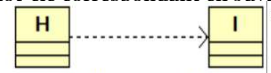
- **Relation réflexive**
  - une personne joue deux rôles : chef et subordonné



1. Diagramme de classes

44

- **Relation de dépendance entre classes**: La dépendance est un concept très général en UML, elle représente relation d'utilisation unidirectionnelle et d'obsolescence.
- une modification de l'élément dont on dépend, peut nécessiter une mise à jour de l'élément dépendant,
- si une dépendance entre une classe **A** et une classe **B** existe. Si **A** possède une méthode prenant comme paramètre une référence sur une instance de **B**, ou si **A** utilise une opération de classe de **B**. Il n'existe pas de mot-clé correspondant en Java.



Généralement, les dépendances ne sont pas montrées dans un diagramme de classes.

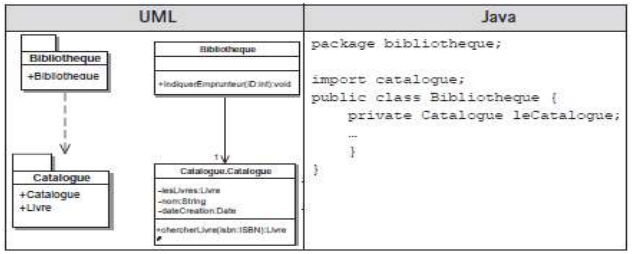
```
public class H {
  void faireQuelqueChose()
  { I i;
    // …
  }
}
public class I {
 }
```

Objet temporaire car il n'existe que pendant la durée de l'exécution de la méthode.

1. Diagramme de classes

45

- **Relation de dépendance (entre packages):**
  - La dépendance entre packages se traduit de façon indirecte par des directives d'importation ou d'usage en Java.
  - une dépendance indique que toute modification de la cible peut causer des modifications de la source (classe ou package).



1. Diagramme de classes

46

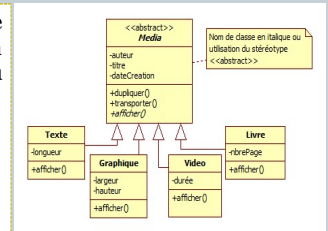
- **Classe abstraite:** UML permet de spécifier une classe de manière incomplète en ne définissant pas certaines opérations.
  - Ces classes dites abstraites, n'étant pas complètement spécifiées, ne peuvent pas être instanciées.
  - Les classes enfants compléteront cette spécification en définissant les opérations non encore définies.
  - Ces classes sont appelées des classes concrètes; toutes les classes sont par défaut complètes, i.e., avec des opérations définies.
  - Les opérations non définies sont dites abstraites et sont écrites en italique dans les diagrammes de classes.
  - Dès qu'une opération est abstraite, sa classe devient abstraite.

1. Diagramme de classes

47

Classe abstraite (exemple)

- Un *Média* peut être transporté, dupliqué, affiché. Le transport et la duplication sont indépendants du type de *Média* (copie de fichiers). Par contre, tout Média peut être affiché et ce n'est pas la même chose pour Livre, Vidéo, Graphique ou Texte. Un Média ne peut pas définir comment il s'affiche tant qu'il ne sait pas ce qu'il est.
- Il n'existe pas d'instance de la classe *Média*. Un Média n'existe qu'en tant que Livre, Texte, Graphique ou Vidéo.



1. Diagramme de classes

48

Interface:

Dans la modélisation UML, les *interfaces* sont des éléments de modèle qui définissent des ensembles d'opérations que d'autres éléments de modèle, tels que des classes ou des composants, doivent implémenter. Un élément de modèle d'implémentation réalise une interface en remplaçant chacune des opérations que l'interface déclare.

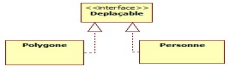
- Une **relation de réalisation** d'interface est un type spécialisé de relation d'implémentation entre un discriminant et une interface fournie. La relation de réalisation d'interface spécifie que le discriminant réalisant doit se conformer au contrat spécifié

1. Diagramme de classes

49

Interface (suite)

- une interface n'est pas une classe, c'est une liste d'opérations,
- une interface, comme une classe abstraite, ne peut pas servir à créer un objet (elle ne peut donner lieu à aucune instance),
- une interface exprime un savoir-faire, un contrat à respecter par les classes qui « réalisent » cette interface,
- une interface permet de décrire le comportement d'une classe, i.e., un savoir-faire sous la forme d'une liste de déclarations d'opérations sans leur définition.
- toutes les opérations d'une interface sont abstraites. Ainsi, une interface est équivalente à une classe abstraite sauf qu'elle ne possède pas d'attribut.



1. Diagramme de classes

50

Interface (suite)

- une classe ***« réalise »*** une interface, i.e., qu'elle définit un corps à toutes les opérations abstraites de l'interface. Une telle classe peut ensuite être utilisée partout où un objet respectant le contrat de l'interface est attendu,
- si une classe ne réalise pas toutes les opérations abstraites de l'interface, alors cette classe est abstraite. Une classe enfant de cette dernière classe peut compléter la concrétisation en définissant les dernières opérations abstraites; la classe enfant devient alors concrète.
- Si vous remplacez, dans votre modèle, des classes ou des composants qui implémentent des interfaces, vous n'avez pas besoin de refaire la conception de votre application si les nouveaux éléments de modèle implémentent les mêmes interfaces.

1. Diagramme de classes

51

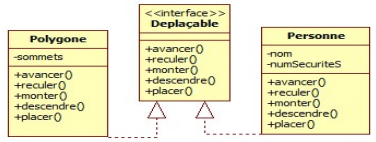
Interface (exemple)

- le schéma suivant montre l'utilisation d'une interface.
- une Personne et un Polygone réalisent les opérations déclarées dans l'interface Déplaçable.

```
void manipuler(Déplaçable élément) {
    ...
    élément.avancer(8);
    élément.monter(4);
    ...
};

Personne jean;
Polygone poly;

obj.manipuler(jean); // licite car Personne implante Déplaçable
obj.manipuler(poly); // licite car Polygone implante Déplaçable
```



2. Diagramme d'objets

52

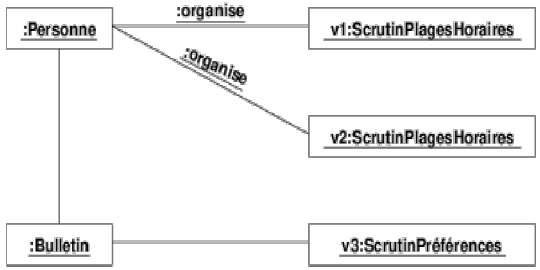
Diagramme d'objets

- Un diagramme d'objets est particulièrement utile quand vous voulez décrire comment les objets dans le système « travaillent » ensemble dans un scénario donné. Un diagramme d'objets représente une configuration donnée.
- La notation UML de l'objet est très simple. Un objet est un rectangle avec le nom de l'objet et le nom de la classe séparés par le signe « : » et soulignés. Le nom de l'objet est facultatif : sans nom d'objet, l'objet est dit « anonyme ».
- Lorsque deux classes sont reliées par deux associations, le nom de l'instance d'association dessinée doit être précisée pour lever l'ambiguïté. Clairement, un diagramme d'objets doit respecter les contraintes d'un diagramme de classes : par exemple, ne pas tracer de liens entre deux objets dont les classes ne sont pas reliées dans le diagramme de classes.

2. Diagramme d'objets

53

- Exemple



Références

54

Références

- UML 2 par la pratique études de cas et exercices corrigés. Pascal Roques Eyrolles 2006.
- Cours POO UML/C++ : les relations entre classes. tvaira@free.fr
- Analyse et Conception de Systèmes
- Informatiques Orientés objets en UML. Abdelhak-Djamel SERIAI http://www.lirmm.fr/~seriai
- http://www-inf.it-sudparis.eu/COURS/CSC4002/EnLigne/Cours/CoursUML/5.18.33.html
- http://www-inf.telecom-sudparis.eu/cours/CSC4002/EnLigne/Cours/CoursUML/3.4.html