



Génie Logiciel 2

Code du module : F321

Présenté par:

Amal HALFAOUI (Epse GHERNAOUT)

a_halfaoui@mail.univ-tlemcen.dz

amal.halfaoui@gmail.com

Université Tlemcen -Licence L3- 2017/2018



Correspondance diagramme de classe code java



- **Forward engineering** : Génération de code, automatique, à partir d'un modèle UML en plusieurs langages
- Utilité:
 - Plus de **productivité** (ex. la structure d'une centaine de classes, d'opérations, etc. générée en un click) – Du 100% code généré dans le domaine du Web, BD, fichier de configuration, etc.
 - Moins **d'erreurs**: un savoir-faire codé dans les générateurs de code
- Le résultat peut **différer d'un outil à un autre**



- UML **pour Java** = UML + contraintes pour permettre génération de code automatique
- Les relations doivent **toutes** avoir un **sens de navigation**
- si on veut spécifier le code Java pour chaque opération; ce dernier doit être dans une Note attachée à celle-ci
- Une classe **ne peut hériter que d'une seule classe** (plusieurs interfaces possibles)

Règles de génération du code Java



R1: classe

À toute classe UML doit correspondre une classe Java portant le même nom que la classe UML.

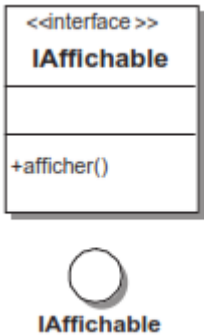
UML	JAVA
A UML class diagram for a class named 'personne'. It is represented as a rectangle with the name 'personne' in the top compartment and an empty bottom compartment.	<pre>class Client{ }</pre>

Classe abstraite

UML	JAVA
A UML class diagram for a class named 'véhicule'. It is represented as a rectangle with the name 'véhicule' in the top compartment and an empty bottom compartment.	<pre>public abstract class véhicule { }</pre>

R2: Interface

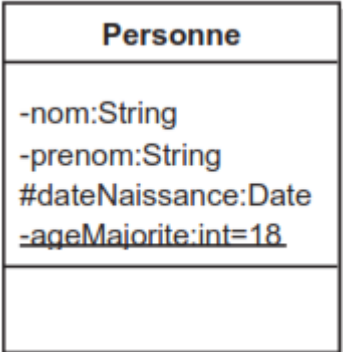
À toute interface UML doit correspondre une interface Java portant le même nom que l'interface UML

UML	JAVA
 The UML diagram shows an interface box with the label '<<interface>>' and 'IAffichable'. It has a single method '+afficher()'. Below the box is a circle with the name 'IAffichable' underneath it. <pre>classDiagram class IAffichable { <<interface>> +afficher() }</pre>	<pre>interface IAffichable { void afficher(); }</pre>



R3: attribut

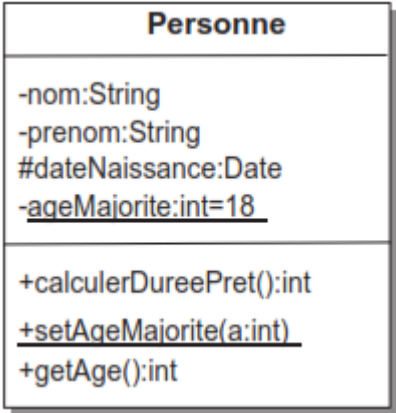
- À toute propriété d'une classe UML doit correspondre un attribut appartenant à la classe Java correspondant à la classe UML.
- Le nom de l'attribut doit être le même que le nom de la propriété.
- Le type de l'attribut doit être une correspondance Java du type de la propriété UML.
- La visibilité des attributs est montrée en les faisant précéder par + pour public, # pour protégé (protected), - pour privé (private).
- Les attributs de classe en UML deviennent des membres statiques en Java .

UML	JAVA
	<pre>abstract public class Personne { private String nom; private String prenom; protected Date dateNaissance; private static int ageMajorite = 18; }</pre>



R4: Opérations




- À toute opération d'une classe UML doit correspondre une opération appartenant à la classe Java correspondant à la classe UML. Les opérations deviennent des méthodes en Java .
- Leur visibilité est définie avec les mêmes conventions que les attributs.
- Les opérations de classe deviennent des méthodes statiques ; les opérations abstraites (en italique) se traduisent par le mot-clé correspondant en Java.
- Les types des paramètres doivent être une correspondance Java des types des paramètres UML.
- Si l'opération UML contient un paramètre de direction return, l'opération Java doit définir un retour qui lui correspond. Sinon , l'opération Java retourne void.

UML	JAVA
 <pre>classDiagram class Personne { -nom:String -prenom:String #dateNaissance:Date -ageMajorite:int=18 +calculerDureePret():int +setAgeMajorite(a:int) +getAge():int }</pre>	<pre>abstract public class Personne { public abstract int calculerDureePret(); public static void setAgeMajorite(int aMaj) { ... } public int getAge() { ... } }</pre>



R5: associations

- Si une classe UML est associée à une autre classe UML et que l'association soit navigable, il doit se trouver un attribut dans la classe Java correspondant à la classe UML.
- Le nom de l'attribut doit correspondre au nom du rôle de l'association.
- Il faut montrer le sens de navigation même si c'est dans les deux sens (bidirectionnelle) sinon il n'est pas possible de générer du code Java.

UML	JAVA
	<pre>public class A1 { private B1 leB1; ... }</pre>
	<pre>public class A2 { private B2 lesB2[]; }</pre>
	<pre>public class A3 { private List<B3> lesB3 = new ArrayList<B3>(); ... }</pre>



R5 associations (suite) :

- **Réflexive** : se traduit par une référence sur un objet de la même classe.
- **Bidirectionnelle** : se traduit simplement par une paire de références, une dans chaque classe impliquée dans l'association. Les noms des rôles aux extrémités d'une association servent à nommer les variables de type référence.

UML	JAVA
	<pre>public class Personne { private Personne subordonne[]; private Personne chef ; ... }</pre>

UML	JAVA
	<pre>public class Homme { private Femme epouse; ... } public class Femme { private Homme mari; ... }</pre>



R6: Classe d'association

- Il s'agit d'une association promue au rang de classe. Elle possède tout à la fois les caractéristiques d'une association et d'une classe et peut donc porter des attributs qui se valorisent pour chaque lien..

UML	JAVA
<pre>classDiagram class Personne class Emploi { - salaire : double - titre : string } class Societe Personne "*" -- "*" Emploi : employe Societe "*" -- "*" Emploi : employeur</pre>	<pre>public class Emploi { private String titre; private double salaire; private Personne employe; private Societe employeur ; ... }</pre>

Règles de génération du code Java



R7: agrégation:

UML	JAVA
<pre>classDiagram class Voiture { - modèle : string } class Moteur { - puissance : integer } Voiture "1" o-- "1" Moteur</pre>	<pre>public class Voiture { private String modèle; private Moteur MonMoteur; public Voiture(Moteur UnMoteur) { MonMoteur = UnMoteur; } }</pre>

R8: Composition:

UML	JAVA
<pre>classDiagram class Voiture { - modèle : string } class Moteur { - puissance : integer } Voiture "1" *-- "1" Moteur : - MonMoteur</pre>	<div><pre>public class Voiture { private String modèle; private Moteur MonMoteur; public Voiture() { //dans le constructeur MonMoteur = new Moteur(); } }</pre></div> <div><u>OU (classe interne)</u><pre>public class Voiture { private String modèle; private Moteur MonMoteur; private class Moteur { //classe interne private int puissance; } }</pre></div>

Règles de génération du code Java



R9 agrégation:

UML	JAVA
<pre>classDiagram class Voiture { - modèle : string } class Moteur { - puissance : integer } Voiture "1" o-- "1" Moteur</pre>	<pre>public class Voiture { private String modèle; private Moteur MonMoteur; public Voiture(Moteur UnMoteur) { MonMoteur = UnMoteur; } }</pre>

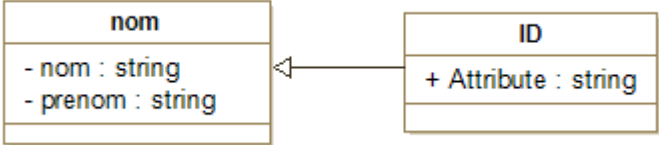
R10 Composition:

UML	JAVA
<pre>classDiagram class Voiture { - modèle : string } class Moteur { - puissance : integer } Voiture "1" *-- "1" Moteur : - MonMoteur</pre>	<div><pre>public class Voiture { private String modèle; private Moteur MonMoteur; public Voiture() { //dans le constructeur MonMoteur = new Moteur(); } }</pre></div> <div><u>OU (classe interne)</u><pre>public class Voiture { private String modèle; private Moteur MonMoteur; private class Moteur { //classe interne private int puissance; } }</pre></div>

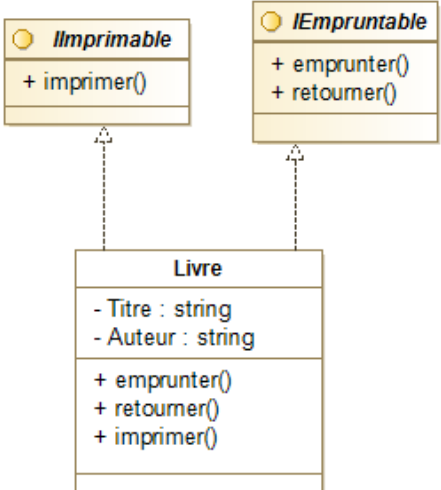
Règles de génération du code Java



R11: Héritage

UML	JAVA
	<pre>public class Adherent extends Personne { private int iD; }</pre>

R12: Réalisation


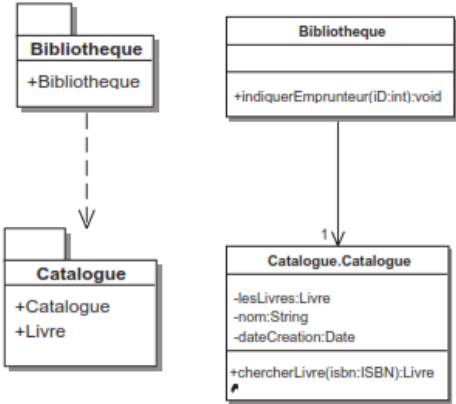
UML	JAVA
	<pre>public class Livre implements IImprimable, IEmpruntable { private String titre; private String auteur; public void imprimer() { ... } public void emprunter() { .. } public void retourner() { .. } }</pre>

Règles de génération du code Java



R13: Dépendance

Une dépendance entre une classe A et une classe B existe par exemple 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 ou en C#.

UML	JAVA	
	<pre>public class Bateau{ public void methode(Port ghazaouet){ ghazaouet.autreMethode(); } }</pre>	<p>OU</p> <pre>public class Bateau{ public void methode(){ Port ghazaouet= new Port(); ghazaouet.autreMethode(); } //l'objet Ghazaouet disparaît à la fin de la méthode }</pre>
	<pre>package bibliotheque; import catalogue; public class Bibliotheque { private Catalogue leCatalogue; ... }</pre>	