Programmation Orientée Objet

ISIMA - ZZ2 - 2009

Christophe Duhamel
Andréa Duhamel

ISIMA Coordonnées

- Bureau: B108
- Téléphone : 04 73 40 76 62
- e-mail: andrea@isima.fr
- 12:30-13:30 sans rendez-vous

2

ISIMA Plan général

- Le génie logiciel
  - évolution de la programmation au cours des âges
  - pourquoi faire ?
- La notion d'objet
  - définitions
  - relations fondamentales sur les objets

3

ISIMA I.e - Évolution des logiciels

- idée générale de programme
- époque héroïque
- premiers langages évolués
- notion de sous programme
- programmation modulaire
- types abstraits de données
- objets

4

ISIMA Idée général de programme (avant 1940)

- (820) le mathématicien El Khawarizmi a publié à Bagdad un traité intitulé La science de l'élimination et de la réduction.

- (1814-1852) Ada Lovelace, a défini le principe des itérations successives dans l'exécution d'une opération.
  - Elle a nommé "algorithme" le processus logique d'exécution d'un programme.
  - Ada est la première programmeur (une femme !) pour ces travaux dans la machine analytique de Babbage.

5

ISIMA Époque héroïque (40-50)

- Ordinateurs peu nombreux (armée…)
  - peu de choses sont demandées aux ordinateurs
  - interface rudimentaire voire inexistante
  - petits programmes
  - langage machine puis assembleur
  - un seul concepteur ou une équipe réduite

6

ISIMA Premiers langages évolués (50)

- Démocratisation des ordinateurs (universités)
  - nécessité de formalismes plus simples
    - premiers langages structurés : FORTRAN
    - règne du « spaghetti »
  - programmes plus ambitieux
    - premières équipes de travail
    - découpage en plusieurs programmes reliés
    - premiers problèmes liés à la communication
      - découpage
      - interfaçage
  - première idée : abstraction de l'implémentation au bénéfice de la description des interfaces

7

ISIMA Sous-programmes (60)

- Motivation
  - factoriser les tâches répétitives
- Fonctionnent sous la forme d'une boîte noire
  - connaissance du seul prototype
    - paramètres
    - fonctionnalités
- Abstraction procédurale

  Tout appel de sous programme apparaît atomique
- Problèmes :
  - nommage des identificateurs
  - collisions possible entre les identificateurs de l'utilisateur et ceux des sous programmes.

8

ISIMA Modules (70)

- Langages de cette époque: Pascal, Prolog, SmallTalk, C, ...
- Apparition de gros logiciels de gestion
  - chaque module est lié à une fonctionnalité
    - gestion client, gestion compte, etc.
  - communication par messages

9

ISIMA Modules (70)

- évolution naturelle des sous-programmes
  - entité indépendante regroupant les sous-programmes et les données sur lesquelles ils travaillent
    - séparation partie publique / partie privée
    - la partie publique définit l'interface (contrat d'utilisation)
  - problème majeur
    - impossible de dupliquer un module
    - une seule copie des données sur lesquelles un module est capable de travailler
    - danger du découpage : faire des modules trop petits réduit l'efficacité

10

ISIMA Types abstraits de données (80)

- Exemple typique du `package` en ADA
  - première apparition du schéma modèle/instance
    - structure de données avec traitements associés
    - autant d'instances que nécessaire (duplication de données)
  - notion d'interface
    - exporte les opérations sur la structure de donnée accessibles à l'utilisateur
- Abstraction de données
  - cacher l'implémentation des données
  - seule l'interface est connue
  - on ne connaît une donnée que par les opérations disponibles dessus

11

ISIMA I – Le Génie Logiciel

- Introduction
- Pourquoi avons-nous besoin du génie logiciel ?
- Problèmes liés à la conception de logiciels
- Qualités d'un bon logiciel

12

ISIMA I.a - Introduction

- But : améliorer la qualité des logiciels
- Point de vue des utilisateurs
  - logiciel répondant bien aux besoins
  - fiable et sûr
  - bonne prise en compte des potentialités du parc
  - bonne documentation
- Point de vue de la maintenance
  - logiciel facile à faire évoluer et à maintenir
  - bonne documentation

13

ISIMA I.b – Un réel besoin (1/3)

- Quelques statistiques:
  - Microsoft office Mac : 30 millions de lignes de code
  - Chaque développeur a à sa charge … 428 000 lignes
  - Mac OS X Tiger : 86 Millions de lignes. J'imagine en incluant les frameworks de développement et les applications integrées!
  - Windows XP et Vista : 40 millions, ~68 millions
  - Un système complet GNU/Linux avec Gnome: 15 à 18 millions de lignes.
  - OpenOffice.org, dans sa version 1.1.3 : 5 millions de lignes de code.
  - BLAST (**Basic Local Alignment Search Tool**): http://www.ncbi.nlm.nih.gov/ieb/ToolBox/CPP_DOC/doxyhtml/group__AlgoBlast.html

Source: Association c-net de l'Université de Savoie (Christian Corsano)

14

ISIMA I.b - Un réel besoin (2/3)

- Constat alarmant
  - seul 1/4 des logiciels commandés sont encore utilisés un an après leur livraison
- Problèmes utilisateurs
  - fonctionnalités manquantes ou incomplètes
    - non respect du cahier des charges
  - mauvaise ergonomie
  - bogues
  - documentation inadaptée
    - trop succincte
    - trop technique

15

ISIMA I.b - Un réel besoin (3/3)

- Problèmes techniques et financiers
  - livré trop tard
  - dépassements de budget
    - trop d'aller retours entre le fournisseur et le client
  - mauvaise adéquation parc/logiciels
    - demande des machines trop puissantes
    - n'exploite pas les machines présentes
  - évolutivité douteuse
    - code source médiocre
    - découpage du logiciel incompréhensible
    - impossible de vérifier que le logiciel fonctionne normalement
    - portabilité impossible
    - documentation technique inexploitable

16

ISIMA I.c - Problèmes liés aux logiciels

- Les logiciels sont trop complexes
  - difficile de répondre aux besoins des utilisateurs
    - communication développeur ↔ utilisateurs
    - besoins souvent antagonistes
    - beaucoup de non-dit
  - problèmes internes aux développeurs
    - développer, c'est difficile !
    - problèmes spécifiques liés au travail en équipe
      - communication
      - découpage cohérent des tâches

17

ISIMA Les systèmes complexes

- Hiérarchie de systèmes emboîtés
  - choix des composants simples totalement arbitraire
  - respect du principe des interactions :
    - interaction intra >> interaction inter
      - interaction forte → même composant
      - interaction faible → composants éventuellement différents
  - aller du simple vers le complexe
    - construire *ex nihilo* un système complexe est utopique
    - bon principe :
      - concevoir un système initial simple et générique
      - le raffiner peu à peu
      - eXtreme Programming

18

ISIMA Buts du Génie Logiciel

- Résoudre les problèmes liés à la complexité
  - réduire les coûts de développement et de maintenance
  - obtenir des logiciels satisfaisants pour tous
- Comment
  - rationaliser la conception d'un logiciel
  - prévoir son évolution
- Outil fondamental : le principe d'abstraction

  éliminer ce qui n'est pas fondamental pour se concentrer sur le principal
- Problème : subjectivité de l'abstraction

  deux individus ne se focaliseront pas sur les mêmes aspects d'un problème

19

ISIMA I.d - Qualité d'un logiciel

- Deux catégories de critères de qualité :
  - facteurs internes (développeur)
  - facteurs externes (utilisateur)
- Ouvrages de référence
  - Grady Booch
    « *Conception orientée objets et applications* », Addison-Wesley, 1992
  - Bertrand Meyer
    « *Conception et programmation par objets* », InterEditions, 1989
  - deux classifications différentes que nous allons mêler

20

ISIMA Facteurs de qualité (2/3)

- Fiabilité (fondamentale)
  - validité
    - réaliser les tâches prévues dans les spécifications
  - robustesse
    - fonctionner en environnement dégradé
      - capacité à travailler sur des données erronées sans plantage complet
  - intégrité
    - résister aux erreurs de manipulation
      - messages de mise en garde
      - fonctions undo
      - détection des manipulations incohérentes
  - la fiabilité doit être intégrée à la conception !

21

ISIMA Facteurs de qualité (2/3)

- Efficacité
  - tirer le meilleur parti de l'environnement
  - environnement logiciel et matériel
- Modifiabilité
  - extensibilité : ajouter des fonctionnalités
  - réutilisabilité : des composants logiciels
  - portabilité
  - maintenabilité : documentation et qualité du découpage

22

ISIMA Facteurs de qualité (3/3)

- Intelligibilité
  - facilité d'utilisation : ergonomie
  - documentation
  - qualité de la décomposition
  - vérifiabilité
    - facilité de conduire tests (unitaire, intégration, certification)
    - important pour la fiabilité et la modifiabilité
    - critique dans les logiciels 24/24 et temps-réel
- Interropérabilité / compatibilité
  - capacité à dialoguer avec d'autres logiciels
  - formats de fichiers, API, protocoles
- Compromis parfois nécessaires (efficacité !)

23

ISIMA Réutilisabilité (1/2)

- Temps de développement
  - inutile de réinventer la roue à chaque étape
  - omniprésence de certains algorithmes
    - ≠ langages de programmation
    - ≠ structures de données
    - ≠ environnements
- Obstacles
  - peur du « Not written there », frustration programmeur
  - documentation inadéquate
  - problèmes de diffusion
  - problèmes de licence
  - interfaçage avec le code local
  - maintenir les clients dépendants

24

ISIMA Réutilisabilité (2/2)

- Avantages
  - plus un code est utilisé plus il devient fiable
  - détection précoce des bogues et manques
    - réduction du coût de maintenance
    - un module validé n'a plus à être vérifié !
- Réutiliser le code c'est bien ...
  ...réutiliser la conception c'est mieux !
  - éliminer certaines étapes de conception sur des problèmes semblables
  - englober en une unité fonctionnelle unique les travaux de conception et d'implémentation

25

ISIMA Pourquoi méthode orientée objet?

- Qualités d'un système conçu à l'aide d'une méthode orientée objet?
  - **La compréhension du système** qui est facilitée du fait que la différence sémantique entre réalité et système est faible.
  - **Les modifications** que l'on apporte au modèle ont tendance à être localisées puisqu'elles n'ont souvent un impact sur un seule élément, répresenté par un seul objet.

26

ISIMA Logiciel orienté objet (1/3)

- Collection d'objets en interaction
- Abstraction d'exécution

  Un système d'objets doit toujours donner l'impression d'être monolithique

  – un objet cible peut être :
    - sur la machine courante / sur une machine distante
    - actif / en sauvegarde / non encore créé
  – l'objet émetteur n'en n'a pas conscience !
  – middleware : couche logicielle chargée de l'abstraction
  – dialogue via système (IPC, socket, TCP) ou bus objet

27

ISIMA Logiciel orienté objet (2/3)

- 5 grands principes énoncés par Booch
  – décomposabilité
    - partitionner le logiciel en modules d'objets indépendants ne nécessitant que de connaître les interfaces

  – composabilité (réutilisabilité)
    - les différents modules doivent pouvoir s'assembler pour former de nouveaux modules

28

ISIMA Logiciel orienté objet (3/3)

- compréhensibilité (Intelligibilité)
  - basé sur la qualité de la documentation
  - chaque module doit pouvoir être compris par une personne n'appartenant pas à son équipe de développement.
  - les dépendances entre modules doivent être claires
- continuité
  - un changement faible de spécification doit entraîner le minimum de modifications dans les modules
- protection
  - lorsqu'une erreur apparaît dans un module, celui-ci doit s'en rendre compte et avertir ses petits camarades.

29

ISIMA II - Les Objets

- Définitions
  - objet
  - classe
- Notation (UML)
- Relations
  - héritage
  - agrégation
  - association

30

ISIMA II.a - Définitions

- Objet

  entité cohérente rassemblant des données et le code travaillant sur ces données

  – données = attributs + données membres

  – code = méthodes

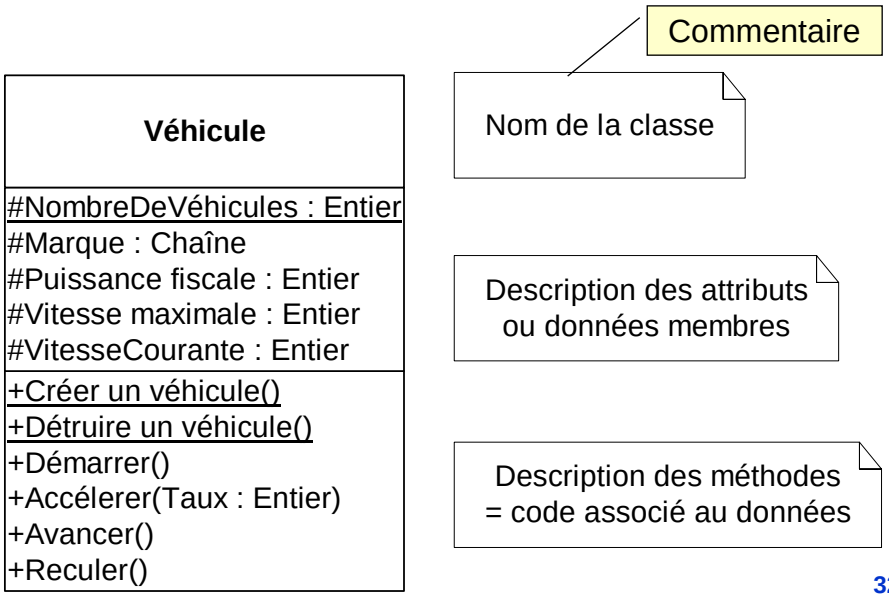
- Classe

  fabrique à objets, donnée qui décrit des objets

  – expression de la notion de classe d'équivalence

31

ISIMA Représentation UML



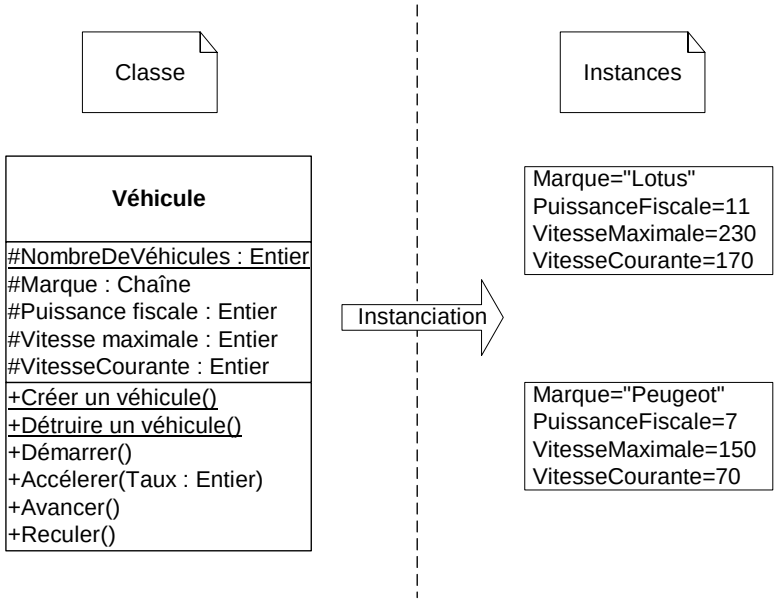
32

ISIMA Instanciation

- Obtenir un objet à partir de la classe
  - la classe décrit un objet
    - données membres
    - méthodes
    - c'est une **méta donnée** *i.e.* une donnée de description des objets
  - l'objet est un état de la classe
    - tous les objets d'une même classe ont la même structure
    - chaque objet conserve des valeurs spécifiques pour les données
    - c'est une *instance* de la classe

33

ISIMA Exemple d'instanciation



34

ISIMA Méta classe

- Méta donnée
  - donnée qui en décrit une autre
  - classe = méta donnée qui décrit les objets
  - méta classe = méta donnée qui décrit une classe
    - classe = instance unique de la méta classe
    - objets = instances multiples de la classe
- Contenu de la méta classe
  - pointeurs vers les méthodes
  - variables de classe
  - méthodes de classe
    - allocation / désallocation mémoire
    - construction des objets, etc.

35

ISIMA Typage fort vs. typage faible

- une classe « est » un type
- deux catégories de langages à objets
  - typage fort (et statique) [C++, Java]
    - un objet est affecté à une classe au moment de la compilation
    - vérification de l'adéquation objet/message à la compilation
    - impossible d'envoyer un message non traitable
    - facile à déboguer et à mettre au point
    - le type est liée au nom (identificateur) de l'objet, donc à son adresse mémoire
  - typage faible (et dynamique) [Smalltalk, Objective C]
    - le type est lié au contenu de l'objet, pas à l'adresse
    - peut modifier le typage de l'objet au cours du temps
    - grande souplesse, notamment lors du prototypage d'une application
    - vérification à l'exécution de l'adéquation objet/message
      - erreur
      - transmission ou délégation vers un autre objet

36

ISIMA Données membres

- attributs d'instance : une valeur par objet
  - exemple : vitesse / couleur d'un véhicule
- attributs de classe : partagés par tous les objets de la classe
  - exemples
    - nombre de véhicules présents à un instant donné
    - gamme de couleurs disponibles

37

ISIMA Données membres

- méthode d'instance : agit sur un objet particulier
  - exemple : accélérer, freiner
- méthode de classe : agit sur toute la classe
  - exemple : savoir le nombre de véhicules créés

38

ISIMA Votre première classe

Quels sont les attributs d'instance?

Quels sont les attributs de classe?

Quelles sont les méthodes de classe?

Créer un nouveau contact
Nom
Numéro de téléphone résidentiel
Supprimer la classe contact
Calculer la place disponible
Modifier numéro résidentiel
Numéro de téléphone portable
Prénom
Obtenir prénom
Nombre de contacts

39

ISIMA 3 Principes fondateurs

- Encapsulation
  - protection des attributs
  - interface de communication
- Héritage
  - relation de généralisation / spécialisation
  - factorisation de code
- Polymorphisme
  - répondre de manière spécifique à un message commun

40

ISIMA II.b - Encapsulation (1/3)

- Séparation forte Interface/Implémentation
  - interface : partie visible d'un objet
    - ensemble de messages paramétrés
    - communiquer avec un objet = envoi de messages
    - dans la pratique : appel direct de méthode
  - implémentation cachée
    - attributs
    - quelques méthodes
  - intérêt : boîte noire
    - peut modifier l'implémentation d'un objet sans effet visible de l'extérieur
    - il suffit de ne pas modifier l'interface

41

ISIMA Encapsulation (2/3)

- Abstraction procédurale

  Un appel de message est atomique

  - les objets agissent comme des boîtes noires
  - aucun contrôle sur les traitements associés au message

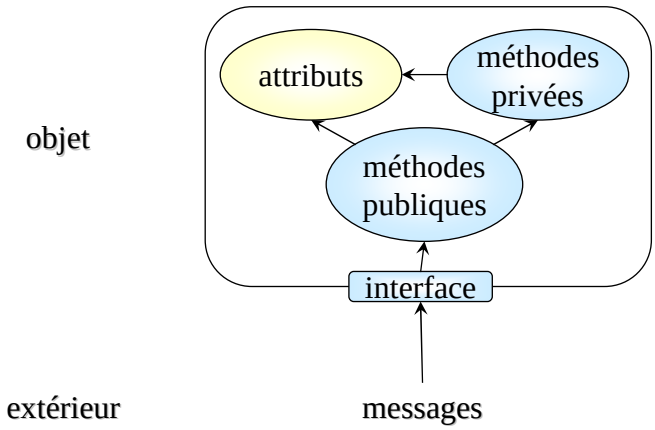
- Abstraction de données

  Un objet n'est accessible que via ses messages

  - aucun renseignement sur la structure interne d'un objet
  - peu importe comment c'est fait…
  - …du moment que c'est fait !

42

ISIMA Encapsulation (3/3)



43

ISIMA Loi de Demeter

- Une méthode accède exclusivement à
  - ses arguments, y compris l'objet cible de la méthode
  - les attributs de l'objet courant
  - les attributs de la classe de l'objet courant
  - les variables globales
  - les variables temporaires créées dans la méthode
- Concept d'interfaces multiples
  - un objet peut avoir plusieurs interfaces (CORBA → IDL)
  - choix de l'interface avant d'accéder aux messages
  - interface en fonction de l'objet appelant

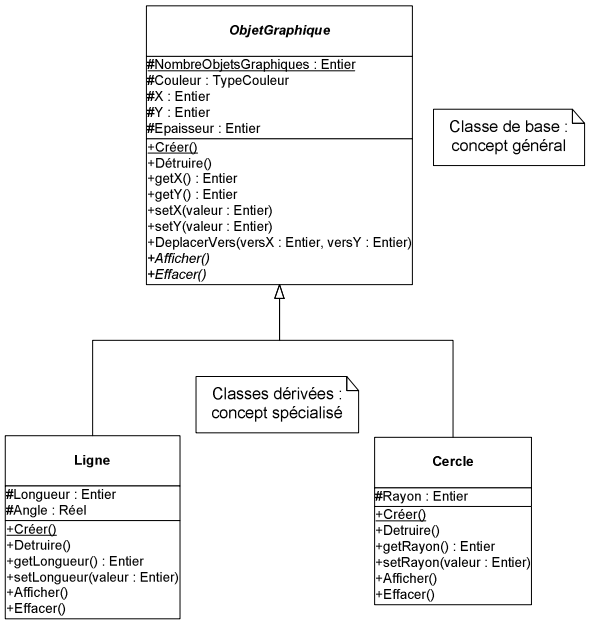
44

ISIMA II.c Héritage

- Concept naturel de Généralisation / Spécialisation
  - classes représentées sous forme d'arbres généalogiques
- Vocabulaire
  - classe spécialisée = sous classe, classe fille / dérivée
  - classe générale = super classe, classe mère
  - la classe spécialisée dérive de sa classe mère
- Idée fondamentale
  - classe B dérivant de classe A
  - B hérite de tous les attributs et méthodes de A
  - B ajoute ses propres attributs et méthodes

45

ISIMA Exemple de hiérarchie



46

ISIMA Classe dérivée

- Reprend les caractéristiques de la classe mère
  - attributs
  - méthodes
- Ajoute les siennes
  - attributs (instances ou classe)
  - méthode (instances ou classe)
- Peut répondre à de nouveaux messages
- Peut répondre différemment aux messages de la classe mère
  - polymorphisme…

47

ISIMA Utilisation de l'héritage

- Construction d'un système *ex nihilo*
  - identifier tous les composants
  - factoriser les caractéristiques communes entre classes
  - généraliser
- Extension d'un système existant
  - identifier les différences avec les classes existantes
  - ajouter les nouvelles classes dans le graphe d'héritage
  - « programmation différentielle »

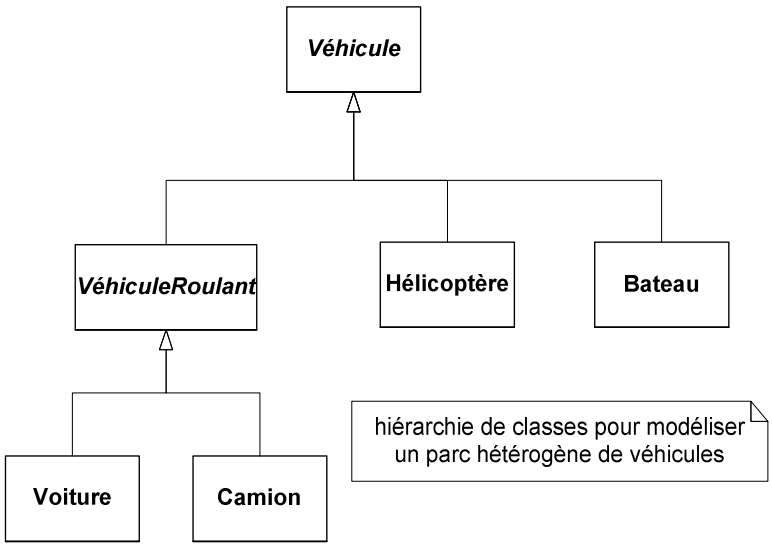
48

ISIMA Exemple : parc de véhicules (1/3)

- Fournir un modèle de parc de véhicules contenant
  - des voitures
  - des camions
  - des bateaux
  - des hélicoptères
- Tous sont des véhicules
  - factorisation au plus haut par une classe Véhicule
  - les voitures et les camions sont des véhicules terrestres pouvant découler d'une même classe

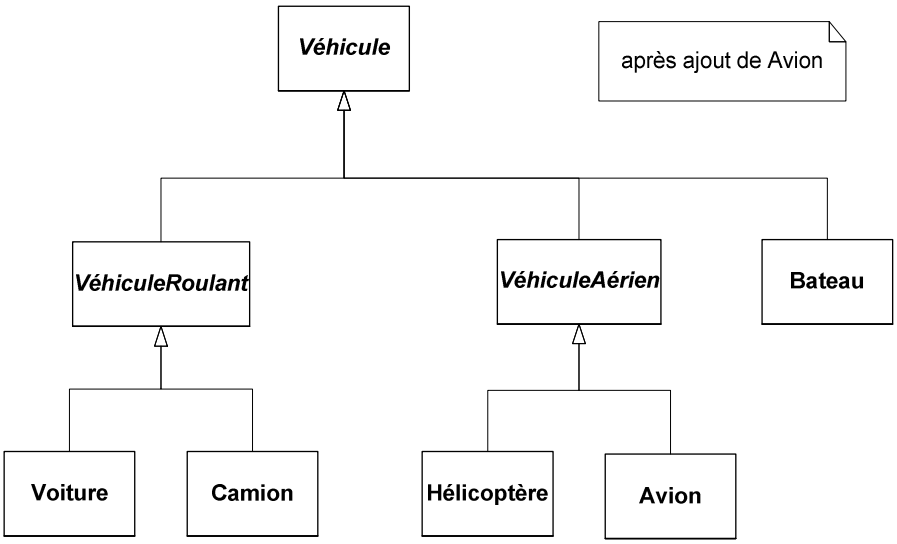
49

ISIMA Exemple : parc de véhicules (2/3)



50

ISIMA Exemple : parc de véhicules (3/3)



51

ISIMA Classe abstraite

- Modélise un concept
  - pas d'instances
  - définit des méthodes abstraites
    - pas d'implémentation
    - Spécifications fonctionnelles
  - peut définir des attributs
- Utilisée comme super classe d'une hiérarchie
  - exemple : Véhicule, Objet Graphique
    - aucun intérêt d'avoir des instances
    - instances créées dans les classes dérivées
    - support pour le *polymorphisme*

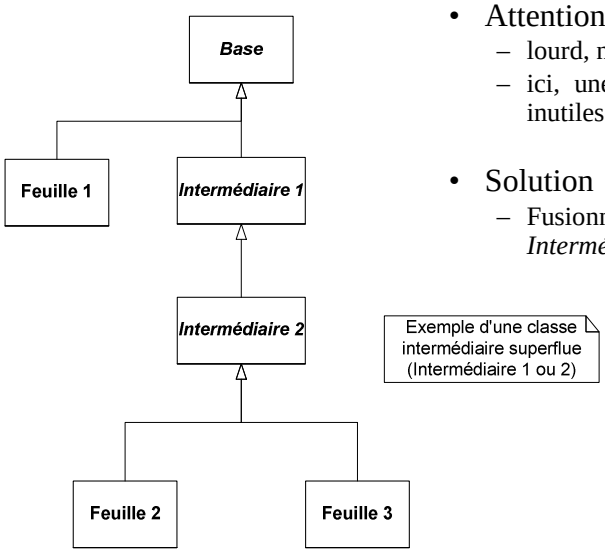
52

ISIMA Avantages de l'héritage

- Partage de code
  - réutilisabilité et fiabilité
  - code des classes les plus hautes dans la hiérarchie utilisé plus souvent : fiabilisation plus rapide
- Modélisation d'un concept naturel
  - certains considèrent qu'on peut se passer de héritage
  - OLE : héritage des interfaces mais pas des implémentations
- Quantité de code source réduite (factorisation)
- Maintenance facilitée
  - modifier l'interface d'une classe fille n'impose pas de recompiler les classes mères
  - modifier l'implémentation d'une classe n'impose pas de recompiler la hiérarchie

53

ISIMA Dangers de l'héritage (1/3)



- Attention à la hiérarchie
  - lourd, nuit à l'efficacité du code
  - ici, une des classes intermédiaires inutiles
- Solution
  - Fusionner *Intermédiaire1* et *Intermédiaire2*

54

ISIMA Dangers de l'héritage (2/3)

- Violation du principe d'encapsulation
  - on récupère tout ce qui vient de la classe mère
  - violation (théorique) : hériter pour accéder aux membres
  - erreurs sémantiques
    - Pingouin dérive de Oiseau mais ne vole pas (utiliser interface)
    - héritage sélectif possible dans certains langages !
- Héritage de construction
  - dériver sans respecter la généralisation/spécialisation
    - dériver Rectangle de Ligne en ajoutant une largeur
  - dériver alors qu'un attribut suffirait
    - dériver des animaux en fonction de la couleur du pelage
    - manque de discrimination fonctionnelle

55

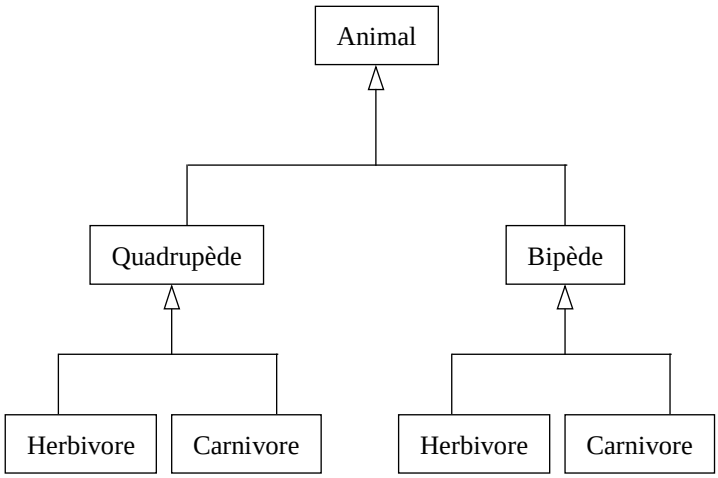
ISIMA Dangers de l'héritage (3/3)

- Les problèmes de covariances
- Considérons les cas d'un modèle d'animaux qu'on s'intéresse à décrit la digestion et la démarche:
  - Le régime alimentaire : herbivore et carnivore
  - La démarche : bipède et quadrupède
- Première idée: créer une modèle qui dérive d'abord la démarche et ensuite le régime alimentaire.
- Deuxième idée : créer une modèle qui dérive d'abord le régime alimentaire et ensuite la démarche.

56

ISIMA Problème de covariance

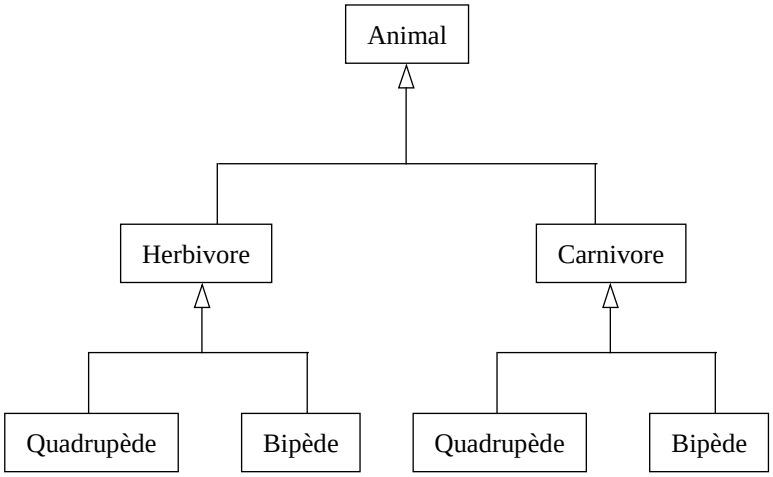
- Première idée



57

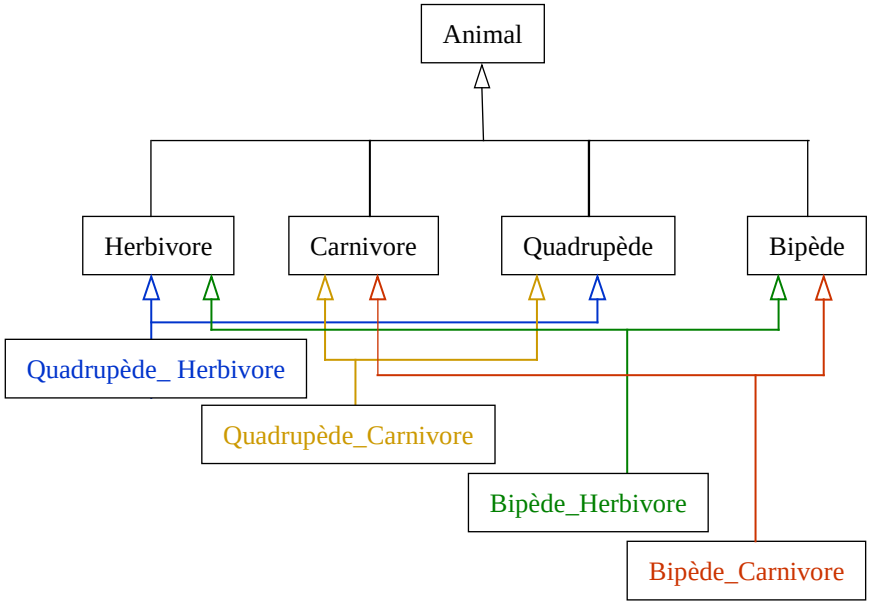
ISIMA Problème de covariance

- Deuxième idée



58

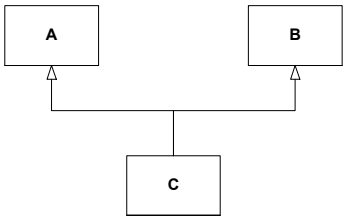
ISIMA Solution par héritage multiple



59

ISIMA Héritage multiple

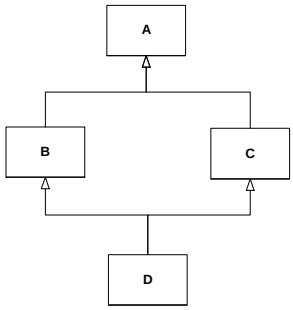
- Plusieurs super classes pour une même sous-classe
  - modélise des principes naturels
    - appartenance à plusieurs catégories



  - risques de collision de noms sur les classes mères
    - préfixage du nom des membres dans certains langages
  - se transforme vite en héritage en diamant !

60

ISIMA Héritage en diamant



- Duplication des informations de A dans D
  - une fois via B, une fois via C
- Cas fréquent dans les bibliothèques
- Mécanismes de protection spécifiques (C++)

61

ISIMA Interface

- Alternative à l'héritage multiple
- Interface
  - ensemble de méthodes virtuelles
  - similaire à une classe abstraite sans attribut
  - définit une fonctionnalité (clonable en Java)
- Vocabulaire
  - une classe implémente une interface
- Héritage simple vs. interfaces multiples
  - hériter du concept primordial
  - implémenter des interfaces

62

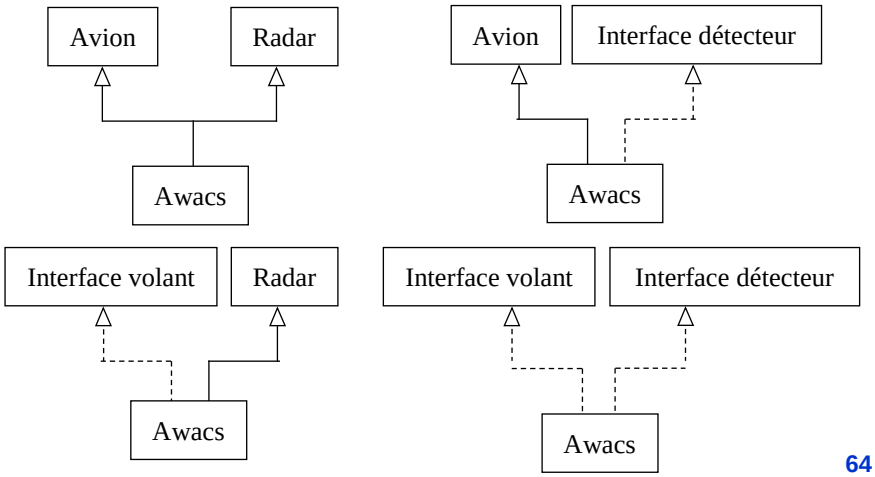
ISIMA Exemple : AWACS

- Avion radar ou un radar avion ?

63

ISIMA Exemple : AWACS

- choix du modèle dicté par
  - les besoins du logiciel
  - la culture du concepteur (Boeing = AWACS2.1, ESW = AWACS2.2)



64

ISIMA II.d - Polymorphisme (1/4)

- Une même méthode prend plusieurs formes
- Forme faible : la surcharge
  - même nom de méthode
  - paramètres différents (nombre, type)
  - exemple-type : surcharge des opérateurs
  - pas vraiment un concept objet !
- Forme forte : le polymorphisme dynamique
  - même nom de méthode
  - paramètres identiques (nombre, type)
  - comportement différent le long d'une hiérarchie
  - exemple-type : affichage

65

ISIMA Polymorphisme (2/4)

- Exemple : les objets graphiques
- Méthode à différencier selon les objets : afficher()
- Code ancienne mode :

```
pour chaque objet graphique :
  switch (TypeObjet)
    case CERCLE
        afficherCercle(Objet)
    case LIGNE
        afficherLigne (Objet)
  fin switch
fin Pour
```

- Lourd et peu extensible

66

ISIMA Polymorphisme (3/4)

- Des méthodes spécifiques
- Définir le comportement propre à chaque classe
- Hériter du code de la classe mère si nécessaire
- Code nouvelle sauce :

```
pour chaque objet graphique :
  [objet Afficher]
fin Pour
```

- Simple et ne dépend pas du nombre de classes

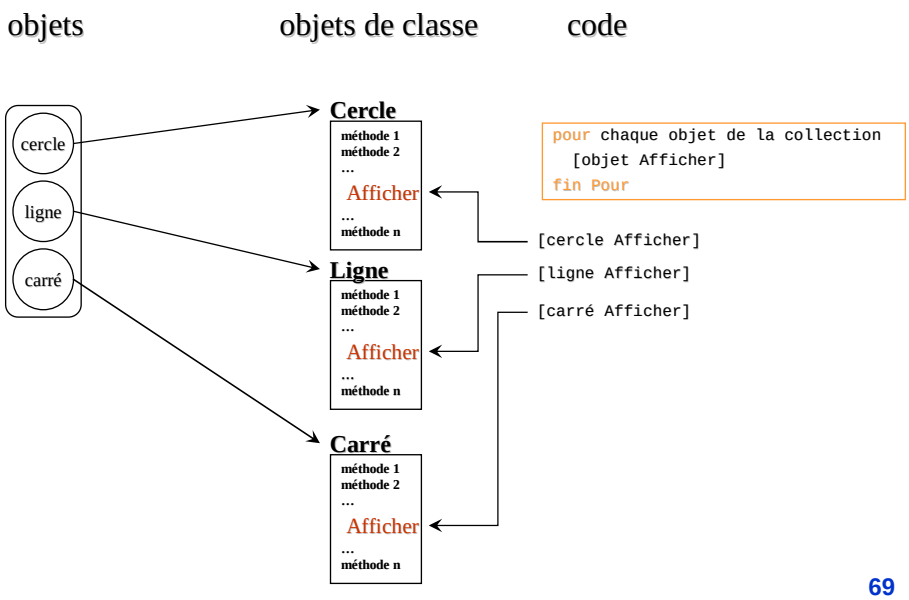
67

ISIMA Polymorphisme (4/4)

- Souvent utilisé dans les agrégats
- Nécessite
  - une hiérarchie de classes (cf. héritage)
  - une méthode virtuelle (pour avoir la TMV)
- Utilise la compatibilité descendante des pointeurs
  - classe B dérivant de A
  - ptr1 : pointeur sur une instance de A
  - ptr2 : pointeur sur une instance de B
  - ptr1 ← ptr2 est valide
- Repose sur la liaison différée (*late binding*)
- Les interfaces apportent aussi le polymorphisme

68

ISIMA Mise en oeuvre



69

ISIMA II.d - Relations fondamentales

- 3 grandes relations fondamentales
  - héritage : généralisation/spécialisation
    symbolisé par « est une version spécialisée de » (Is A)
  - agrégation / composition
    symbolisé par « contient », « regroupe » (Has A)
  - association : communication
    symbolisé par « communique avec » (Uses A)
- Il existe d'autres relations mais celles-ci sont quasi unanimement reconnues !

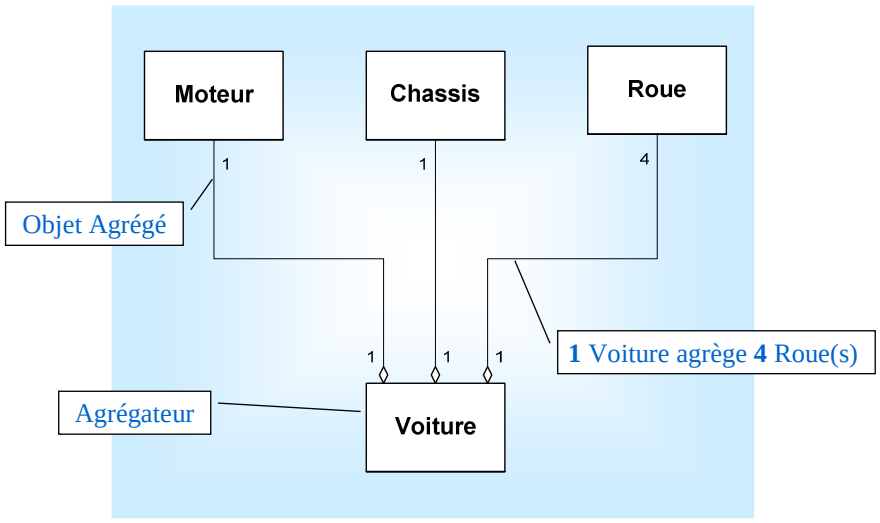
70

ISIMA Agrégation

- Modélisation du groupage
  - « has a », « est composé de »
  - appartenance
- Modèle naturel des conteneurs
- Cardinalité M-N
  - M agrégeants contiennent N agrégés
  - exemple : relation entre Immeuble et Propriétaire
  - souvent M=1
- Agrégation vs. Composition
  - agrégation : agrégé indépendant de l'agrégeant
  - composition : vie de l'agrégé dépend de l'agrégeant

71

ISIMA Représentation UML



72

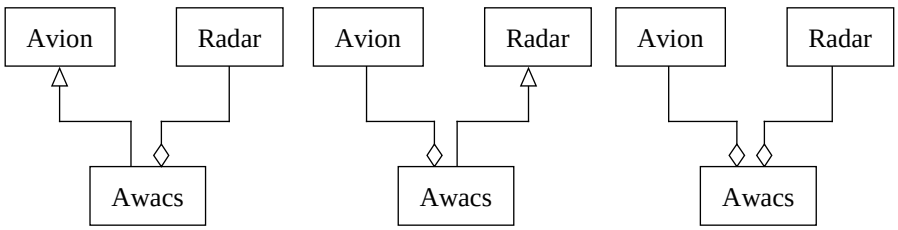
ISIMA Composition vs. héritage

- Utiliser la composition pour remplacer l'héritage
  - une classe encapsule une autre plutôt que d'en hériter
  - évite un accès aux données membres
  - respecte mieux la notion d'interface
  - respect de l'encapsulation
  - permet d'éviter un héritage conceptuellement bancal
  - parfois appelé **délégation** (GoF)

73

ISIMA Composition vs. héritage

- Exemple : AWACS
  - Avion contenant un Radar (AWACS 3.1)
  - Radar porté par un Avion (AWACS 3.2)
  - ensemble comprenant un Avion et un Radar (AWACS 3.3)



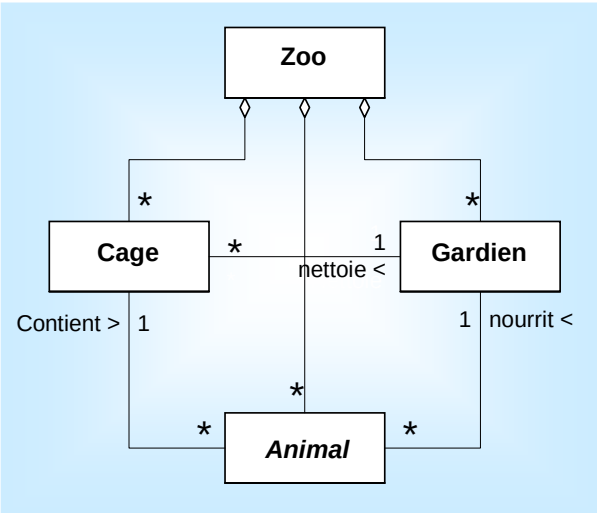
74

ISIMA Association

- Modélise des concepts flous
  - « uses a », est en association avec, communique avec
- Caractéristiques :
  - habituellement nommée (auto documentation)
  - cardinalités
- Association vs. Agrégation
  - concepts voisins, parfois difficiles à différencier
  - souvent implémentation identique
  - l'association n'est pas transitive

75

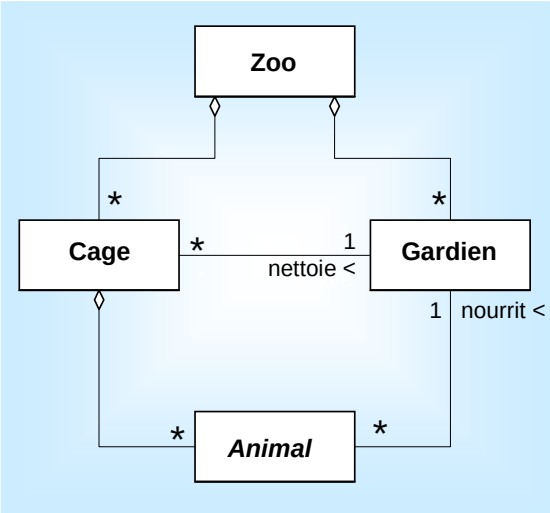
ISIMA Exemple : Zoo (1/2)



Zoo agrège Gardien, Cage et Animal
Les autres relations sont de l'association

76

ISIMA Exemple : Zoo (2/2)



Zoo agrège Gardien et Cage, Cage agrège Animal
Les autres relations sont de l'association

77