Isima

PARTIE I

Rappels sur la programmation objet

Bruno Bachelet

Loïc Yon

Les objets

- Définitions
  - Objet
  - Classe
- Formalisme UML
- Relations entre classes
  - Héritage
  - Agrégation
  - Association

Définitions

- Objet
  - Entité cohérente rassemblant des données et le code travaillant sur ces données
  - Données = attributs
  - Code = méthodes
- Classe
  - Fabrique à objets, i.e. une donnée qui décrit des objets
  - Représente une catégorie d'objets

Exemple d'instanciation

Classe

Instances

Véhicule
- nombreVéhicules : Entier - marque : Chaîne - puissance : Entier - vitesseMaximale : Entier - vitesseCourante : Entier
+ créerVéhicule() + détruireVéhicule() + démarrer() + accélérer(taux: Réel) + avancer() + reculer()

Instanciation

marque = "Lotus"
puissance = 11
vitesseMaximale = 230
vitesseCourante = 170

marque = "Peugeot"
puissance = 7
vitesseMaximale = 150
vitesseCourante = 70

2

nombreVéhicules

Membres d'une classe

- Attributs d'instance: une valeur par objet
  - Exemple: vitesse / couleur d'un véhicule
- Attributs de classe: une valeur par classe
  - Partagés par tous les objets de la classe
  - Exemple: nombre de véhicules présents à un instant donné
- Méthode d'instance: agit sur un objet particulier
  - Exemple: accélérer, freiner
- Méthode de classe: agit sur toute la classe
  - Exemple: ajouter un nouveau véhicule

Principes fondateurs

- Encapsulation
  - Protection des attributs
  - Interface de communication
- Héritage
  - Relation de généralisation / spécialisation
  - Factorisation de données et comportement
- Polymorphisme
  - Réponse spécifique à un message commun

Encapsulation

- Séparation forte entre interface et implémentation
- Interface: partie visible d'un objet
  - Ensemble de messages paramétrables
  - Communiquer avec un objet = envoi de messages
  - Dans la pratique: appel direct de méthode
- Implémentation: partie cachée d'un objet
  - Attributs
  - Code des méthodes
- Intérêt: principe d'abstraction
  - Modification de l'implémentation d'un objet sans effet visible
  - Tant que l'interface n'est pas modifiée
    ⇒ aucune conséquence pour l'utilisateur

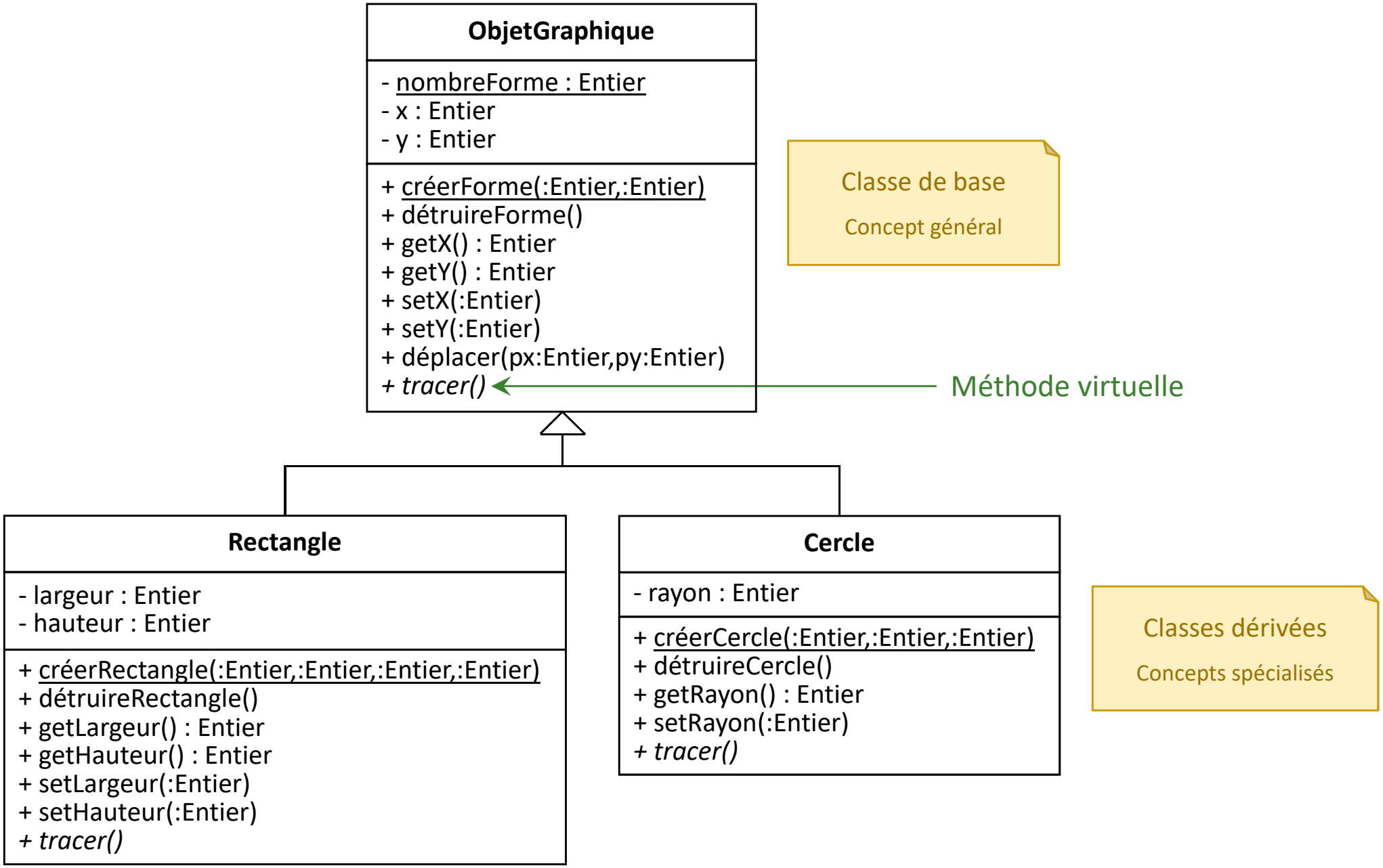
Relations fondamentales entre classes

- 3 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 unanimement reconnues

Héritage

- Concept naturel de généralisation / spécialisation
  - Classes organisées sous forme d'arborescence
- 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

Exemple d'héritage



Classe dérivée

- Reprend les caractéristiques de la classe mère
  - Attributs et méthodes
  - Sauf généralement les constructeurs (cas particulier)
- Ajoute / modifie les siennes
  - Attributs et méthodes
- Peut répondre à de nouveaux messages
- Peut répondre différemment aux messages de la classe mère
  - Grâce au polymorphisme
- Attention à l'héritage des membres de classe
  - Ils ne sont pas «dupliqués» dans la classe fille
  - Ils sont uniques et propres à la classe mère
  - Mais sont accessibles depuis la classe fille

Utilisation de l'héritage

- Principe de substitution de Liskov
  - Partout où un objet de la super-classe est utilisé
    on peut le remplacer par un objet d'une sous-classe
- 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 l'arbre d'héritage
  - «Programmation différentielle»

Classe abstraite

- Classe abstraite = classe qui ne peut pas être instanciée
  - Définit en général au moins une méthode abstraite
    - Sans implémentation
  - Peut définir des attributs

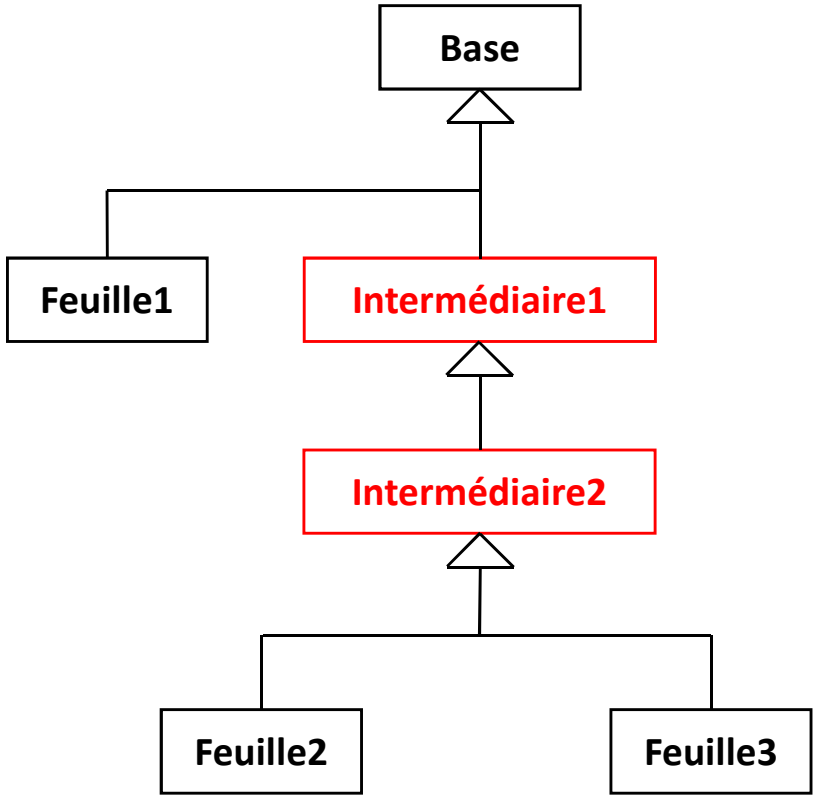
- Utilisée comme super-classe d'une hiérarchie
  - Exemple: *Véhicule*, *ObjetGraphique*
    - Aucun intérêt (ou sens) d'avoir des instances
    - Instances créées dans les classes dérivées
    - Support pour le polymorphisme

- Classe abstraite pure: modélise un «concept»
  - Toutes les méthodes sont abstraites
  - Similaire à une interface

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
- Quantité de code source réduite (factorisation)
- Maintenance facilitée
  - Héritage = code factorisé
  - Modification de l'implémentation d'une classe sans impact
    - Sur la hiérarchie d'héritage
  - Modification de l'interface d'une classe sans impact
    - Sur ses ancêtres

Dangers de l'héritage *(1/2)*



Exemple d'une classe intermédiaire superflue

- Attention à la hiérarchie
  - Trop lourde, elle peut nuire à l'efficacité et à la compréhension du code
  - Ici, une classe intermédiaire peut être inutile
    - Classification trop fine
    - Couches de programmation différentielle
- Solution
  - Fusionner *Intermédiaire1* et *Intermédiaire2*

Dangers de l'héritage *(2/2)*

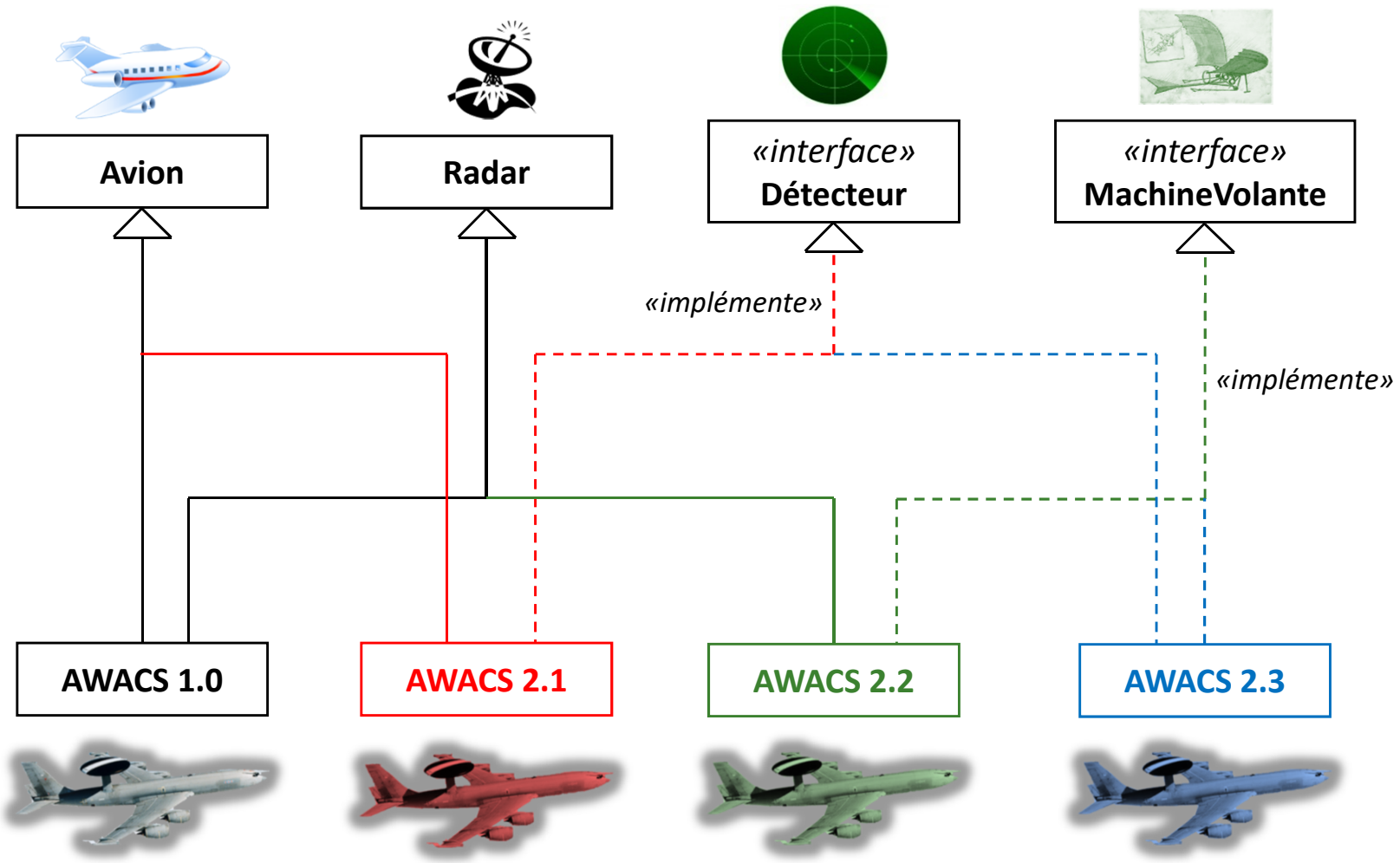
- Violation du principe d'encapsulation
  - Accès aux membres protégés de la classe mère
    - 2 niveaux d'interface
  - Violation (théorique): hériter pour accéder aux membres
  - Problèmes de maintenabilité

- Héritage de construction
  - Dériver sans respecter la généralisation / spécialisation
    - Attention à respecter le principe de Liskov
    - Exemple: *Rectangle* hérite *Ligne* pour utiliser son tracé ⇒ erreur !
    - Souvent, l'agrégation est plus adaptée
  - Dériver alors qu'un attribut suffirait
    - Dériver des classes d'animaux en raison de la couleur du pelage
    - Manque de discrimination fonctionnelle

Interface

- Alternative à l'héritage multiple
  - Classe = interface + implémentation
  - Héritage multiple = problèmes dans l'héritage des implémentations
- Interface
  - Ensemble de méthodes abstraites
  - Similaire à une classe abstraite pure sans attribut
  - Définit une fonctionnalité (e.g. *«Comparable»* en Java)
- Vocabulaire
  - Une classe implémente une interface
- Héritage multiple *vs.* interfaces multiples
  - Hériter du concept primordial
  - Implémenter des interfaces

Exemple: AWACS



Pointillés = implémentation d'interface

Polymorphisme *(1/2)*

- Une même méthode prend plusieurs formes

- Forme faible: la surcharge de nom (*«overload»*)
  - Même nom pour plusieurs méthodes
  - Différence sur les paramètres: nombre et type
  - Exemple-type: surcharge des opérateurs
  - Pas vraiment un concept objet

- Forme forte: le polymorphisme dynamique (*«override»*)
  - Méthode redéfinie dans une sous-classe
  - Comportement différent le long d'une hiérarchie
  - Paramètres strictement identiques
  - Différence sur le type véritable de l'objet
  - Exemple-type: affichage d'une liste hétérogène d'objets

Polymorphisme *(2/2)*

- Souvent utilisé dans les agrégats
- Nécessite
  - Une hiérarchie de classes
    - Voir l'héritage
  - Une méthode virtuelle
    - Table des méthodes virtuelles
- Utilise la compatibilité ascendante des pointeurs (et références)
  - Classe *B* dérivant de classe *A*
  - `ptr1`: pointeur/référence sur une instance de *A*
  - `ptr2`: pointeur/référence sur une instance de *B*
  - `ptr1 ← ptr2` est valide
- Repose sur la liaison différée (*dynamic dispatch*)
- Les interfaces apportent aussi le polymorphisme dynamique

Agrégation

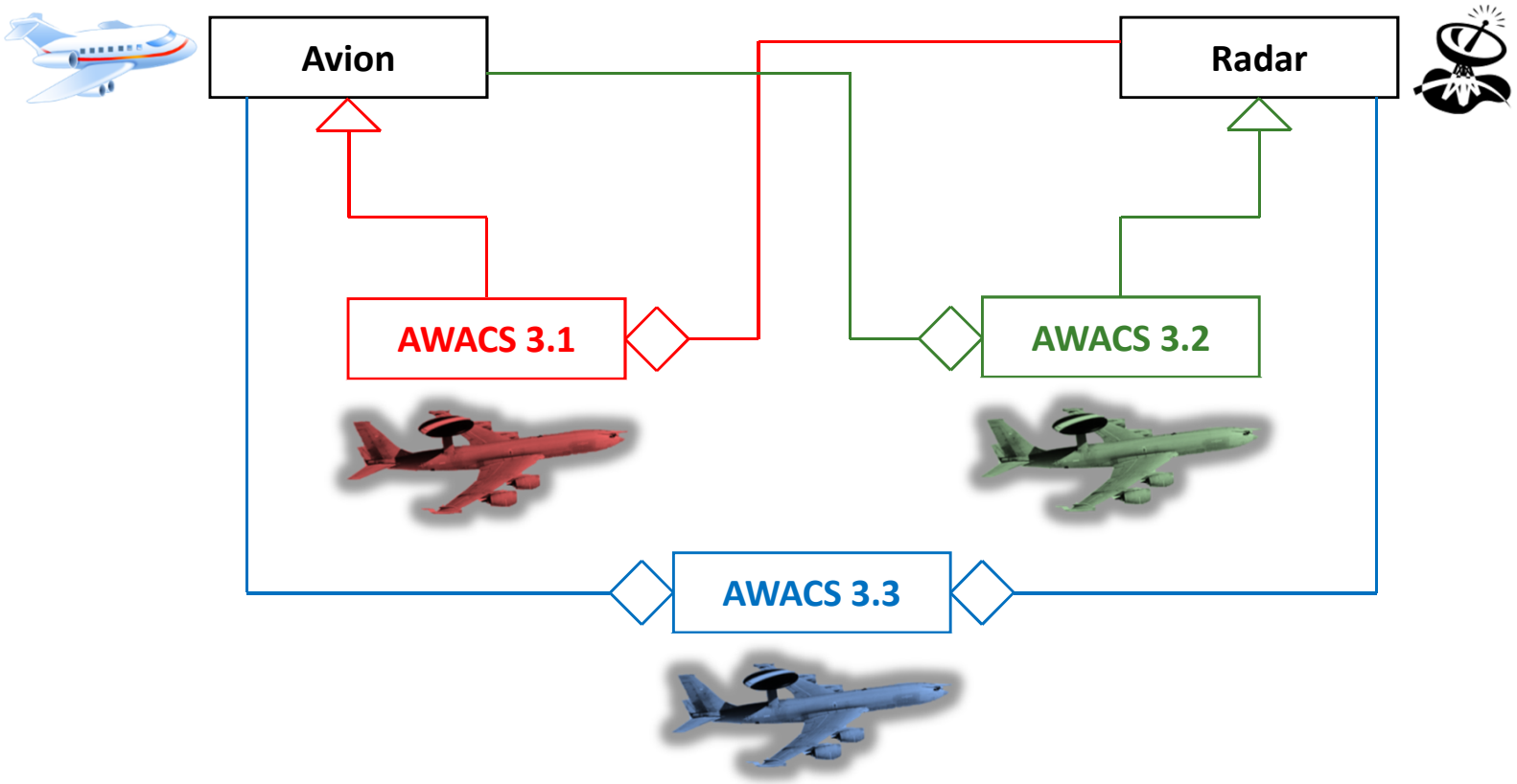
- Modélisation du groupage
  - Appartenance
  - «est composé de»
- Agrégation *vs.* composition
  - Agrégation: agrégé indépendant de l'agrégeant
  - Composition: vie de l'agrégé dépend de l'agrégeant
- Modèle naturel des conteneurs
  - Souvent, cardinalité 1-N
  - 1 agrégeant contient N agrégés

Composition *vs.* héritage *(1/2)*

- Utiliser la composition pour remplacer l'héritage
  - Une classe encapsule une autre plutôt que d'en hériter
  - Attribution de caractéristiques sans lien de type
- Permet d'éviter un héritage conceptuellement bancal
  - Une erreur classique
- Evite un accès aux données membres
  - Respecte mieux la notion d'interface
  - Respect de l'encapsulation
- Parfois appelé «délégation» (cf. *GoF – Gang of Four*)
  - Les messages envoyés à l'objet qui encapsule sont délégués à l'objet encapsulé

Composition *vs.* héritage *(2/2)*

- 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)



Association

- Modélise des relations plus «floues»
  - «utilise», «est associé à», «communique avec»
- Caractéristiques
  - Habituellement nommée
  - Cardinalités M-N
- Association *vs.* agrégation
  - L'agrégation est une forme d'association
  - Pas toujours évident de les différencier
  - Agrégation: notion de parties / décomposition
  - Les moyens d'implémentation sont les mêmes
    - Attributs objets, références ou pointeurs