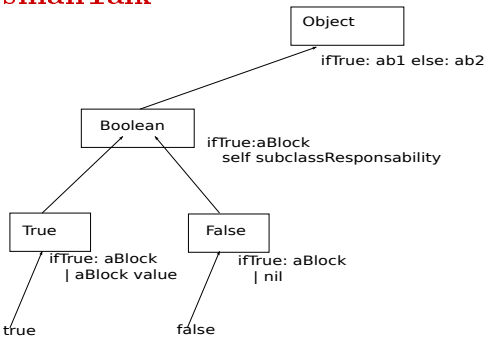
Principes de la POO :

1. Objet
2. Méthode
3. Classe
4. Hiérarchie de classes

Exemple : définition de la structure conditionnelle en smallTalk



Plan de cours

1. Introduction
2. Concepts de base (objet, classe, envoi de message, hiérarchie)
3. Illustrations et exemples (Java, Simula, SmallTalk
4. Etude comparative de LO
5. TP (Java, Eclipse)

Histoire d'Ada Le département de la défense américain (Dod) est :

- Plus gros consommateur de logiciels
- 1968 - 73 : Le coût des SI augmente et le coût du matériel chute
- 73 : 7,5 milliards de dollars pour les Si
- 75 : 450 langages différents
- Réutilisabilité et partage de code inexistants
- 1983 : naissance d'Ada en réponse à la crise du logiciel et pour résoudre les pbs liés au dev de gros logiciels
- non issu d'un projet académique ou de recherche interne

Idée : Un seul langage incorporant tous les bons concepts de GL.

Paradigmes de programmation

- impérative
- fonctionnelle
- logique
- objet
- par contraintes

Les domaines révolutionnés par la POO

- Génie logiciel (analyse, conception, programmation)
- Bases de données
- Intelligence artificielle (représentation des connaissances, programmation par agents)

Quel objet ?

- Des problématiques différentes → Des approches objet différentes
- Une préoccupation commune (approche modulaire de la complexité)

Pour distinguer les approches différentes, on change le voc :

- Génie Logiciel : Objet
- IA : Schéma
- Ingénierie collaborative : Agent
- Bases de données : RDF

Quels languages :

- Java sous éclipse principalement
- Simula
- SmallTalk

Programmation impérative : C'est un paradigme de programmation qui décrit les opérations en termes d'états du programme et de séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme.
Les paramètres sont passés par valeur.
Elle repose sur deux principes :

- Séparation des programmes et des données.
- Priorité donnée aux traitements (alors qu'en fait ce sont les structures de données qui sont les plus stables).

On aimerait donc pouvoir manipuler des objets définis par leurs caractéristiques externes (protocoles) indépendamment de toute représentation interne → Ada (reste impératif car il permet la POO mais ne fanchit pas le pas de le rendre obligatoire).

Remarque :

- Révolution Algol/Ada : associer des données à des programmes (variables locales, paramètres)
- Révolution Objet : associer des programmes à des données

Définition : Manipuler des objets définis par leurs caractéristiques externes (protocoles) indépendamment de toute représentation interne.

Deux manières de faire :

- Langage impératif + Constructions syntaxiques
  - regrouper les données et les programmes
  - protéger (cacher, exporter)
  - Séparer spécification et implémentation
  - $\rightarrow$ on compte sur le bon sens du programmeur
  - exemples : ADA, Pascal
- Langage objet
  - Définir de nouveaux types, et les manipuler (c'est la manipulation qui fait la différence avec les langages impératifs)
  - Différence entre les types prédéfinis et types utilisateurs (ne devrait pas exister dans un langage purement objet)
  - exemples : Simula67, SmallTalk80, Common Lisp objet, Eiffel, C++, Java, Objective C, Ruby, Python, C#, Groovy, Lisaac

Type abstrait :

- Données
- Opérateurs
- Propriétés

Caractéristiques

- Modularité
- Sécurité
- Spécification
- Réutilisabilité

Exemple : Pile d'entiers

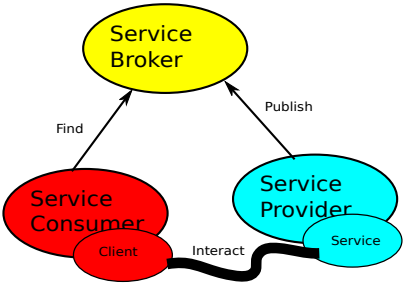
- Pile, Entier, Booléen
- – pile_vide : Pile → Booléen
  – empiler : Pile × Entier → Pile
  – depiler : Pile → Piles × Entier
- – non(pile_vide(empiler(p,e)))
  – depiler(empiler(p,e)) = (p,e)
  – non(pile_vide(p)) ⇒ empiler(depiler(p)) = p

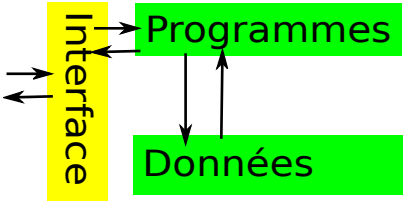
Approche descriptive

- La réalité est composée d'objets (perçu, concret, animé ou non, conçu, abstrait) → connaissance singulière

- Caractéristiques contingentes (peu varier) → attributs
- Caractéristiques essentielles (intrinsèque) → classes différentes
- Objet → Concept (abstraction de classes : fox → voiture → moyen de locomotion)
- Concept → Exemplification (spécialisation de classe : moyen de locomotion → voiture → fox

Approche service (fonctionnelle) : Objet ≡ Composant. L'accent est mis sur les caractéristiques externes, descriptives et comportementales de l'objet.





- Encapsulation des données, boite noire
- Un objet permet de regrouper en une seule entité et de façon indissociable données et procédures d'exploitation.
- Un objet est une instance de classe
  - ses données sont les variables d'instance, attributs variables, données membres. Les données sont accessibles uniquement à traver les procédures (programmation

par état) → c'est rémanent (ie si l'objet se modifie lors d'une procédure, alors l'état initial n'est pas restauré). Remarque : En Java, l'unité d'encapsulation est la classe (les objets d'une même classe peuvent accéder aux champs privés des autres instances de la même classe). Ca devrait être l'objet.

  - Les procédures sont les méthodes, attributs procédures, fonctions membres. Il y a des méthodes exportées (elles définissent l'interface) et des méthodes cachées (utiles uniquement à l'implémentation)

Programmation par état :

- On travaille par référence et non par valeur
- En java, les types primitifs passent par valeur

Exemple : Processus

- Que **fait** un processus ?
  - Suspendre
  - Réveiller
  - Priorité
- Qu'**est** un processus ?
  - Ressources
  - Contexte
  - Programme

- $\rightarrow$ Ces deux points de vue peuvent être mixés par l'héritage multiple, mais nous en général on choisira une des deux approches et on s'y tiendra.

Exemple : NoeudPersonne hérite de Noeud(méthodes de gestion d'un noeud), et de Personne(méthodes de gestion d'une personne. Si l'héritage multiple n'est pas géré, on peut hériter de Noeud uniquement avec une instance de Personne, ou utiliser un objet composite avec un attribut Noeud et un attribut Personne.

Problème : Définir un sous-concept à partir de plusieurs sur-concepts a-t-il un sens ?

Règle : Seules les classes terminales peuvent être instanciées. Toute classe possédant des spécialisations est une classe abstraite (ne peut être instanciée). Les extensions des classes terminales d'une classe générique constituent une partition de l'ensemble des instances relevant de la classe générique (classe de base, abstraite).