CONCEPTS DE BASE DE L'ORIENTE OBJET

TABLE DES MATIERES

OBJETS A LA BASE

- **Bien comprendre les concepts sous-jacents**

- **Bien comprendre la notion d'objet**

  => comprendre le modèle objet

  => comprendre les mécanismes des langages orientés objet

MODELE OBJET

Caractéristiques majeures de ce modèle :

- **abstraction**
- **encapsulation**
- **modularité**
- **hiérarchisation**

Si un modèle ne possède pas l'un de ces éléments, il n'est pas orienté objet.

CARACTERISTIQUES MAJEURES

- **Abstraction :**

  Faire ressortir les caractéristiques externes essentielles d'une entité pour la distinguer des autres.

- **Encapsulation :**

  Cacher les détails qui ne font pas partie des caractéristiques essentielles d'une entité.

- **Modularité :**

  Décomposer un programme en un ensemble de modules cohérents et faiblement couplés pouvant être compilés séparément.

- **Hiérarchisation :**

  ranger ou ordonnancer les abstractions

  héritage =>
  généralisation / spécialisation ("genre de")

  agrégation =>
  composition par regroupement ("partie de")

AVANTAGES DU MODELE OBJET

- **Exploiter la puissance des langages de programmation orientés objet**

- **Favoriser la réutilisation de composants logiciels et même d'architectures complexes**

- **Produire des systèmes basés sur des formes stables qui résistent mieux aux changements**

- **Penser plus près des modèles "naturels"**

Le modèle objet est l'ossature conceptuelle des méthodes orientées objet

OBJET : DEFINITIONS

- **quelque chose qui existe dans le temps et l'espace**

- **élément individuel identifiable, soit réel, soit abstrait, avec un rôle bien défini dans le domaine du problème**

- **chose physique, relation, événement, situation, idée, règle, tâche, rôle joué, ...**

- **abstraction informatique d'un objet du monde réel**

=> quelque chose du domaine du problème dont on parle en lui attribuant des propriétés, ou qui doit être manipulé

RQ : ne pas confondre objet et attributs de l'objet

=> ce qui caractérise l'objet : taille, position, valeur, adresse, état de l'objet, ...

LANGAGES ORIENTES OBJET

Objective-C : langage d'implémentation de la machine Next (et de Nextstep).

Object-Pascal : langage d'implémentation du Macintosh.

C++ : couche-objet développée à partir du langage C.

SMALLTALK, Simula
Eiffel, FLAVORS, CLOS, KRL, LOOPS, SPOKE, Ada95, Cobol-objet, ...

JAVA

=> étudier la mise en oeuvre du modèle objet

MECANISMES COMMUNS

Un véritable langage orienté objet se doit de fournir :

- **définition de classe**
- **définition d'objet**
- **héritage**
- **communication entre objets**

Un mécanisme d'encapsulation est aussi considéré comme étant nécessaire

Les 3 grands atouts des langages orientés objet :

- **modularité**
- **réutilisabilité**
- **flexibilité**

CLASSE

Description d'un modèle d'objets ayant tous la même structure et le même comportement.

Chaque classe a deux composantes :

- **composante statique : une liste de champs nommés (données) qui caractérisent l'état des objets de la classe pendant l'exécution**

- **composante dynamique : les méthodes qui manipulent les champs; chacune identifiée par un nom appelé le sélecteur et une liste de paramètres.**

Les méthodes caractérisent le comportement commun des objets de la classe : ce sont les actions que l'on peut appliquer à chaque objet de la classe

En pratique :

- **les attributs des objets correspondent aux champs,**
- **les attributs des méthodes correspondent aux paramètres.**

VUES EXTERNE ET INTERNE D'UNE CLASSE

=> mise en oeuvre des mécanismes d'abstraction / encapsulation

- **vue externe : l'interface de la classe**
  - **=> déclaration des champs**
  - **=> déclaration des méthodes**

  selon les langages, jusqu'à trois parties :
  - **=> public : déclarations visibles de tous les clients**
  - **=> privée ou protégée : déclarations invisibles des clients**

- **vue interne**
  - **=> déclarations de données internes, d'opérations internes**
  - **=> implémentation des méthodes**

RQ : opération /= méthode
opération = service utile à l'implémentation des méthodes ou sous-programme libre

EXEMPLE DE CLASSE

Classe : **VOL_D_AVION**

Champs :
- **Nombre_Total_De_Places**
- **Nombre_De_Places_Occupées**
- **Compagnie_Aérienne**
- **Aéroport_De_Départ**
- **Aéroport_D_Arrivée**
- **Horaires**

Méthodes :
- **Prendre_Réservation ( )**
- **Annuler_Réservation ( )**
- **Destination ? ( )**
- **Heure_Départ ? ( )**
- **Heure_Arrivée ? ( )**

RQ : ici tout est en accès libre

INSTANCE / OBJET

Une instance est un représentant physique d'une classe : c'est un OBJET. Tout objet est une instance de classe.

- <u>Champs :</u>

- **la liste des champs d'une instance est celle définie pour la classe,**

- **les valeurs des champs sont particulières à l'instance,**

- **les champs d'une instance sont rémanents.**

- <u>Méthodes :</u>

- **la liste des méthodes fournies par l'instance est celle de la classe.**

exemple :

VOL_512 :
VOL_D_AVION (100, 92, "Air Canada", Dorval, ...)

CHARTER_DU_SOIR :
VOL_D_AVION (30, 15, "Air Alliance", ...)

HERITAGE SIMPLE

- **Une sous-classe est une spécialisation d'une classe appelée sa superclasse**

  => une classe hérite des caractéristiques de sa superclasse : elle en partage les champs et les méthodes

- **Spécialisation d'une classe :**

  - **enrichissement des champs et/ou des méthodes par ajout**

et/ou

  - **substitution pour donner une nouvelle définition à une méthode héritée si celle-ci n'est pas adaptée.**

EXEMPLE D'HERITAGE SIMPLE

La classe des vols d'avions internationaux peut être définie comme une sous-classe de la classe VOL_D_AVION

Classe :	**VOL_D_AVION_INTERNATIONAL**
Superclasse :	**VOL_D_AVION**
Champs :	**Pays_De_Départ** **Pays_D_Arrivée** **Pays_De_Transit**
Méthodes :	**Pays_Etapes ? ( )**

HERITAGE MULTIPLE

Eviter la duplication d'informations : donner à une classe la possibilité d'hériter directement de plusieurs superclasses.

La relation d'héritage est transitive :

=> dans le cas de l'héritage simple on a un arbre d'héritage

=> dans le cas de l'héritage multiple on a un graphe d'héritage (graphe orienté sans circuit)

Une classe hérite de l'union des champs et des méthodes de ses superclasses.

L'héritage simple et multiple accroît la modularité des programmes et facilite la mise au point (et la maintenance?).

EXEMPLE D'HERITAGE MULTIPLE

Classe	VELOS	ENGINS_A_MOTEUR
Champs	Longueur Hauteur Nombre_De_Roues Couleur	Puissance_Du_Moteur Carburant_Du_Moteur Régime_Du_Moteur Niveau_Du_Réservoir Couleur
Méthodes	Choisir_Couleur () Choisir_Taille () Nombre_De_Roues ? ()	Positionner_Regime () Remplir_Reservoir () Etat_Reservoir ? () Choisir_Couleur ()

héritage

Classe	VELOMOTEURS
Superclasse	VELOS ENGINS_A_MOTEUR
Champs	Nb_De_Vitesses
Méthodes	Positionner_Regime () Choisir_Couleur ()

redéfinition explicite soit on redéfinit, soit on dit laquelle est héritée

VISIBILITE DANS L'HERITAGE

Tiraillement entre les principes d'abstraction, encapsulation et hiérarchie :

- **pour une classe donnée, il y a deux genres de clients :**
  - **les clients "vrais" (relation client-serveur)**
  - **les sous-classes**

- **avec l'héritage l'encapsulation peut être violée :**
  - **partie public : accessible par tous les clients**
  - **partie protégée : accessible seulement par la classe et ses sous-classes**

  => encapsulation totale si partie privée : accessible seulement par la classe elle-même

Classe : **VOL_D_AVION**
Champs : **Protégée :**
...
Public :
Compagnie_Aérienne
Méthodes : **Public :**
...

CLASSE ABSTRAITE

=> classe dont on ne définit en général pas d'instance

objectif :

forcer les sous-classes à enrichir la structure et le comportement

=> en particulier en complétant l'implémentation de méthodes volontairement vides ou incomplètes

ex : méthode virtuelle et virtuelle pure en C++

Classe :	**QUADRILATERE**
Champs :	**Distance_AB**
	Distance_AC
	Distance_CD
	Distance_DB
Méthodes :	**Calculer_Perimetre ()**
	Dessiner () : virtuelle

Classe :	**RECTANGLE**
Superclasse :	**QUADRILATERE**
Méthodes :	**Dessiner ()**

GENERICITE

Paramétrer pour rendre plus général donc plus réutilisable (objets, opérations, données, ...)

Paramètres : **type_des_éléments**
Classe : **LISTE**
Méthodes : **AJOUTER_ELEMENT ()**
RETIRER_ELEMENT? ()

Classe : **LISTE_VOL : LISTE [VOL_D_AVION]**

COMMUNICATION ENTRE OBJETS

Des objets clients adressent des requêtes aux objets serveurs : envoi d'un message demandant l'exécution d'une méthode.

=> la programmation en langage orienté objet : décrire un ensemble d'objets et leurs relations client / serveur

Un message est constitué de 3 parties :

- **l'objet récepteur du message**
- **la méthode requise**
- **les arguments appropriés pour la méthode**

exemple :

VOL_512 . PRENDRE_RESERVATION ( )

ENVOI DE MESSAGE

Il existe essentiellement trois manières d'implémenter le mécanisme d'envoi de message :

- **la liaison statique : liaison traitée à la compilation**
  => équivalent à un appel de sous-programme
  => cas des langages fortement typés

- **la liaison dynamique : liaison traitée au moment de l'exécution**
  => parcourir de bas en haut le graphe d'héritage
  => mécanisme très souple mais coûteux
  => problème pour l'héritage multiple avec surcharge
  => cas des langages faiblement typés

- **les méthodes virtuelles : faire profiter les langages fortement typés des avantages de la liaison dynamique mais sans surcoût**
  => rechercher la méthode par une simple indirection (adresses de sous-programmes)

Polymorphisme :
= héritage + liaison dynamique
= liaison statique + méthodes virtuelles