

Introduction à la programmation orientée-objet

Yannick Prié
UFR Informatique – Université Lyon 1

Master SIB M1 – 2006-2007
UE1.2 *Organisation de l'information documentaire et programmation*

Objectifs du (petit) module

- Découvrir ce qu'est la programmation orientée-objet
 - pour des étudiants qui ont déjà appris la programmation impérative
 - Cours → 3 x 1h
 - TP → 3 x 2h
- Découvrir le langage Python

Organisation du module

- CM1
 - Objets / classes
 - Attributs
 - Méthodes
 - Collaboration d'objets
- TP1
 - Prise en main de l'environnement et du langage python (syntaxe python), manipulation d'objets
- CM2
 - Hiérarchie de classe et héritage
 - Relations entre classes
 - Conception objet
- TP2
 - Réalisation d'un petit programme impliquant plusieurs objets et classes.
- CM3 + TP3
 - Conception et réalisation d'un programme un peu plus ambitieux

Objets

- Objets du monde
 - Objets « concrets », plus ou moins coopératifs : cette pierre, ma télévision, ta voiture
 - Objets « abstraits », « conceptuels » : mon compte bancaire, le langage de programmation que j'utilise
- Classes d'objets
 - les pierres, les télévisions, les langages de programmation, les comptes bancaires, etc.
- Objets et classes d'objets sont toujours considérés dans un contexte

Abstraction

- Objets
 - tout ce qui nous permet de réfléchir, parler, manipuler des concepts du domaine, avec
 - un certain nombre de propriétés les caractérisant
 - un certain nombre de comportements connus
 - des classes d'objets avec des propriétés et des comportements similaires
- Abstraction
 - passage du particulier au général
 - « abstraire » des propriétés, comportements

En informatique

- Programme classique
 - structures de données (tableau, arbre, etc.)
 - opérations sur ces structures de données (fonctions)
- Difficultés
 - faire *évoluer* structures de données et fonctions en même temps
 - *réutiliser* des structures/fonctions en les spécialisant
 - ...

Ideée objet

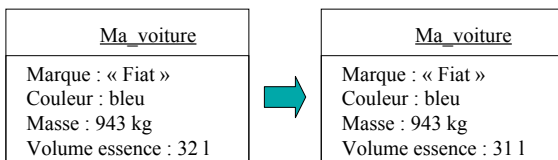
- Regrouper dans un composant
 - des caractéristiques qui concernent une entité informatique
 - structure de données
 - ensemble d'attributs
 - variables avec nom, type, valeur
 - les opérations liées à cette entité
 - ensemble de fonctions
 - appelées *méthodes*
 - avec : nom, valeur de retour, paramètres

Objet informatique

- Etat
 - Ce qu'est l'objet à un instant donné
- + Comportement
 - Comment l'objet réagit aux sollicitations
- + ...

Etat d'un objet

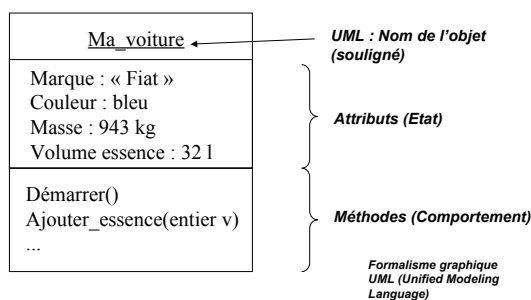
- Ensemble des valeurs des attributs de l'objet à un instant donné
- L'état d'un objet change pendant sa vie



Comportement d'un objet

- Actions et réactions possibles
 - ensemble d'opérations / méthodes
 - exemple automobile
 - démarrer, rouler, stopper, ajouter_essence
- Stimulation
 - demander à un objet d'effectuer une méthode = lui envoyer un message
 - Par exemple dans un programme
 - `ok = ma_voiture.démarrer()`
 - `vol = ma_voiture.ajouter_essence(15)`
- L'état dépend des opérations effectuées
 - Ex. `ma_voiture.volume_essence` si `ma_voiture.rouler()` a été appelée
- Les opérations dépendent de l'état courant
 - Ex. `ma_voiture.démarrer()` ne marchera pas si `ma_voiture.volume_essence == 0`

Représentation d'un objet



Accès aux attributs/méthodes

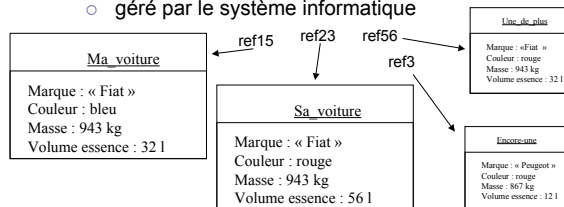
- Accès depuis un autre objet
 - Attribut/méthode publics
 - tout objet peut y accéder
 - Attribut/méthode privés
 - aucun autre objet ne peut y accéder
 - seul l'objet lui-même peut utiliser ses attributs et méthodes
 - comme un programme « indépendant »
 - Attribut/méthode protégé
 - accès limité

Objet informatique

- **Etat**
Ce qu'est l'objet à un instant donné
- **+ Comportement**
Comment l'objet réagit aux sollicitations
- **+ Identité**
Ce qui identifie l'objet

Identité d'un objet

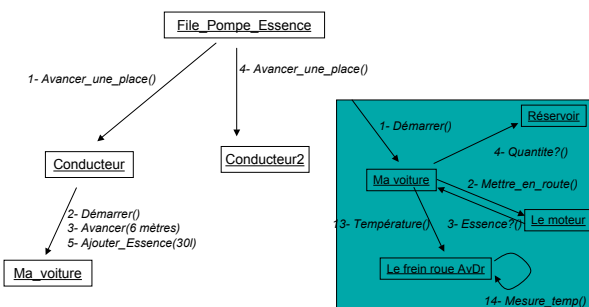
- Existence propre de l'objet
 - identification non ambiguë
 - indépendante de l'état
 - géré par le système informatique



Exécution OO

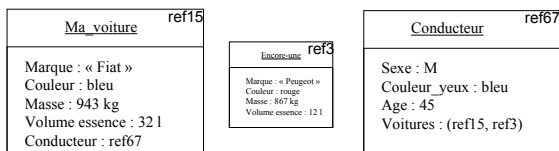
- Une « société d'objets » en mémoire
 - avec leurs états et leurs comportements
- Ces objets se « passent la main » en se rendant mutuellement des services
 - Un objet A envoie un **message** à un objet B pour lui demander de faire quelque chose
 - B exécute ce que A lui demande
 - il rend le service
 - B rend ensuite la main à A
- Remarque
 - pour rendre son service à A, B a pu demander un service à un autre objet C

Collaboration et envoi de messages



Liens entre objets

- Pour pouvoir envoyer un message à un objet, il faut le « connaître »
 - Ex. l'objet **Conducteur** connaît l'objet **Ma voiture**
- Connaître un objet revient à avoir une référence qui lui correspond
 - Certains attributs d'un objet sont des références vers d'autres objets



En bref

- Cohérence interne des objets
 - données + traitements
- Faible couplage entre l'objet et l'environnement
 - envoi de messages
- Insertion dans un scénario de communication par envoi de messages
 - objets acteurs : à l'origine d'une interaction
 - objets serveurs : répondent à la sollicitation
 - objets agents : les deux

Que nous manque-t-il ?

- Soient 2 objets :
 - même structures de données (attributs)
 - même comportement (opérations)
- Il faut les décrire abstraitement de la même manière → classes

Ma R12	Ta 205
Marque : Renault Couleur : bleu	Marque : Peugeot Couleur : rouge
Démarrer()	Démarrer()

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

19

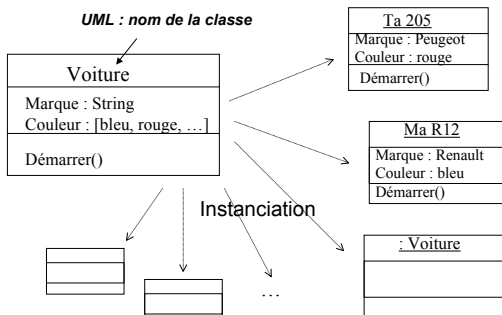
Regroupement en classes

- Les objets sont regroupés dans une *classe*
- Une classe est une *abstraction* décrivant les propriétés communes des objets qui en sont des *instances*
- Une classe décrit une infinité d'instances
- Un objet sait toujours de quelle classe il fait partie

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

20

Classification



SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

21

Dans un programme OO

- On définit des classes
 - leur attributs, privés et publics
 - leurs méthodes, privées et publiques
- On instancie des objets à partir des classes
- On lance/gère la collaboration
 - envoi de messages à des objets
- Exécution du programme : objets
 - qui s'envoient des messages
 - qui changent d'état

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

22

Pour la suite

- Concepts objets
 - CM2 / CM3
- Découverte du langage OO Python
 - Fin de ce cours / TP1

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

23

Python

- Langage
 - Interprété
 - Interactif
 - Portable
 - Orienté-objet
- Inventé en 1990 par Guido Van Rossum
- Avec pour objectifs
 - Simplicité et puissance
 - Programmation modulaire
 - Lisibilité du code
 - Développement rapide d'application
 - Facilité de fonctionnement avec d'autres langages

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

24

Installation et utilisation

- Disponible sur <http://www.python.org/download> (gratuit, Open Source)
- Utilisation
 - interactive (shell)
 - batch (programme)
- Editeur intégré
 - IDLE : Integrated DeveLopment Environment

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

25

Éléments du langage

- Les espaces sont importants
 - Indentation d'un bloc (tabulation)
- Commentaires
 - *# mon commentaire*
- Booléens
 - Tout ce qui vaut 0, null, vide, est de longueur nulle, etc. est évalué à Faux
 - Le reste est évalué à Vrai

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

26

Nombres

- décimal e.g. 631, 3.14
- octal e.g. 0631
- hexadécimal e.g. 0xABC
- complexe e.g. 1 + 3j
- long e.g. 1222334455664455L

- Opérateurs arithmétiques et logiques standards
- Division entière : $1/2 = 0$

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

27

Chaînes (1/2)

- Concaténation
 - "Hello" + "World" -> "HelloWorld"
- Répétition
 - "UMBC" * 3 -> "UMBCUMBCUMBC"
- Indexation
 - "UMBC"[0] -> "U"
- Tranches
 - "UMBC"[1:3] -> "MB"
- Taille
 - len("UMBC") -> 4

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

28

Chaînes (2/2)

- Comparaison
 - "UMBC" < "umbc" -> 0
- Recherche
 - "M" in "UMBC" -> 1
- Possibilité d'utiliser des « simples quotes » ou des triples guillemets
 - e.g. 'UMBC' ou ""Du texte "compliqué" avec des caractères spéciaux ""
- Non mutable
 - on ne modifie pas, on recopie
 - c1 = c1 + c3 -> on ne modifie pas le c1 original

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

29

Listes

- Ensemble de valeurs quelconques
 - Ma_liste = [124, "Python", [32, "bonjour"]]
 - Quelques opérations
 - Opérations de chaînes (accès, tranche, etc.)
 - Ajout `ma_liste.append(342)`
 - Insertion `ma_liste.insert(2, "toto")`
 - Inversion `ma_liste.reverse()`
 - Tri `ma_liste.sort()`
 - ...
- Tuples = listes non mutables
- Mon_tuple = (124, " Python ", "bonjour")

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

30

Dictionnaires

- Ensemble de couples clé/valeur
 - Mon_dict = {"Guido": "Python", "Ullman": "ML"}
 - Quelques opérations
 - Insertion Map["Ritchie"] = "C"
 - Accès Map["Guido"]
 - Effacement del Map["Ullman"]
 - Itération keys() values() items()
 - Présence has_key("Guido")
- Les valeurs peuvent être n'importe quoi
- Les clés doivent être non mutables

Variables

- Pas besoin de les déclarer
- Déduction du type à l'initialisation
ex. $F = 2 * 4.5 \rightarrow f$ de type float
- Variables globales et locales

Références

- $a = b$
 - ne fait pas de copie de b
 - a et b réfèrent au même objet

E.g.

```
>>> a = [1,2,3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

Contrôle du flot

- if condition : statements
[elif condition : statement]
[else : statement]
- while condition : statements
- for var in sequence : statements
- break
- continue

Exemple

- (suite de Fibonacci)

```
Prompt >>> a = 0
>>> b = 1
>>> while b < 1000 :
...     print b
...     a, b = b, a + b
Suite instruction } Bloc d'instructions
Tabulation
```

Fonction range()

- Très utilisé dans les boucles for
- Exemples

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Fonctions et procédures

- Forme générale

```
def name(arg1, arg2, ...):
    Statements
    return # from procedure    OU
    return expression # from function
```
- Exemple

```
>>> def fib(n): # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while b < n:
...         print b
...         a, b = b, a+b
...     return a
```

Documentation

Print

- Pour afficher n'importe quoi
 - directement

```
>>> toto = 99
>>> print toto , 5
99 5
```
 - En formattant

```
>>> chaine = "Bonjour"
>>> print "J'affiche %s et %i" % (chaine, toto)
J'affiche bonjour et 99
```

Modules

- Sortes de librairies
- Fichier contenant des définitions et des instructions Python
- Les fichiers ont un suffixe
 - mon_module.py
- Le module a un nom
 - mon_module
- On peut importer le contenu d'un module
 - import mon_module
- Certains modules sont livrés avec Python
 - exemple : sys

Paquetages

- Collections de modules
- Espace de nom permettant d'accéder à des modules
 - A.B réfère au module B dans le paquetage A
- Pour importer le module B
 - import A.B
 - puis utiliser A.B.xxx pour accéder aux objets, fonctions
 - from A.B import *
 - puis utiliser directement and use only the module name
 - possibilité d'importer seulement quelques éléments
 - from A.B import xxx, yyy, zzz

Classes et objets

- Classes

```
class MaClasse:
    instructions
class MaClasse(ClasseBase1, ClasseBase2):
    instructions
```

Héritage (cf. CM2)
- Objets
 - x = MaClasse()
 - Crée une nouvelle instance de la classe MaClasse, et l'assigne à la variable x

Exemple de classe

```
class Pile:
    "Une structure de données bien connue."
    def __init__(self):
        self.items = []
    def push(self, x):
        self.items.append(x)
    def pop(self):
        x = self.items[-1]
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0
```

Constructeur (cf. CM2)

une méthode prend toujours self comme argument

self = l'instance elle-même

Exceptions

Mécanismes de gestion des erreurs

```
try:
    Print 1/x
except ZeroDivisionError, message:
    print "Can't divide by zero"
    print message

f = open(file)
try:
    process_file(f)
finally:
    f.close()
print "OK"
```

Lever des exceptions

- Raise ZeroDivisionException
- Raise ZeroDivisionException("can't divide by zero")
- Raise ZeroDivisionException, "can't divide by zero"

- Python permet de définir ses propres exceptions

Domaines d'application

- *Glue language*
- Applications graphiques
- Applications basés sur des protocoles internet
- Applications de bases de données
- Applications web
- Applications multimédia

Introduction à la programmation orientée-objet

Suite du cours

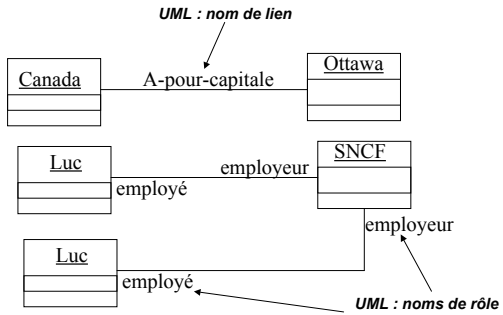
Résumé des épisodes précédents

- Objet = état + comportement + identité
 - Attributs
 - Méthodes
 - (référence)
- Classe
 - Abstraction
 - Définit une infinité d'objets instances

Plan

- Relations entre classes
- Hiérarchies de classes
- Classes et objets
- Initiation à la conception objet
- Eléments objets de Python

Liens entre objets

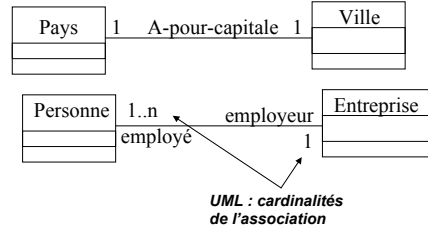


SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Phé 2006-2007

49

Associations entre classes

- Associations simples
 - Liens entre objets → associations entre classes

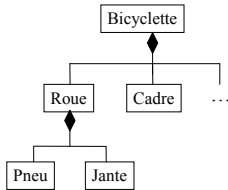


SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Phé 2006-2007

50

Associations entre classe

- Composition
 - Association particulière, dissymétrique
 - La destruction de l'objet composé entraîne celle des objets composants

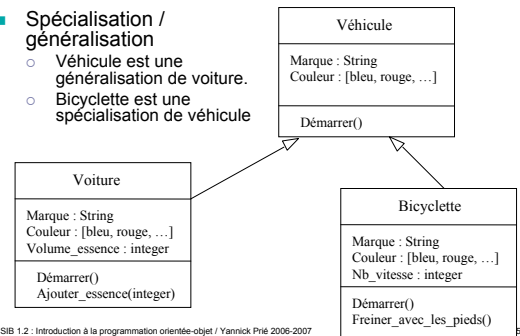


SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Phé 2006-2007

51

Associations entre classes

- Spécialisation / généralisation
 - Véhicule est une généralisation de voiture.
 - Bicyclette est une spécialisation de véhicule



SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Phé 2006-2007

52

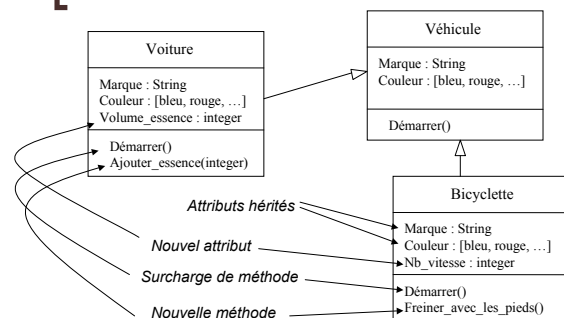
Généralisation / spécialisation

- Mise en place d'une hiérarchie de classes
 - Voiture est une sous-classe de Véhicule
- Partage d'attributs et héritage
 - Une sous-classe hérite des attributs et des méthodes de sa super-classe
 - Héritage multiple : plusieurs super-classes
 - à manipuler avec beaucoup de précautions
- Ajout d'éléments propres
 - Une sous-classe peut ajouter des attributs et méthodes à ceux qu'elle possède par héritage
- Surcharge
 - Une sous-classe peut redéfinir les attributs et méthodes de sa sur-classe

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Phé 2006-2007

53

Exemple

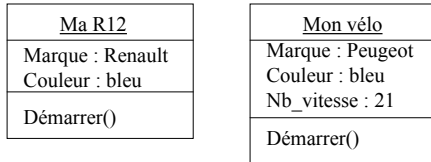


SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Phé 2006-2007

54

Polymorphisme

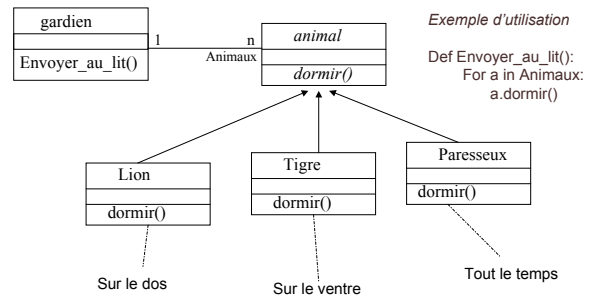
- Une même opération peut se comporter différemment pour différentes classes / objets
 - Suivant l'objet, le langage sélectionne la méthode à utiliser pour la classe en cours
 - Il n'y a pas besoin de connaître toutes les méthodes existantes pour en implanter une nouvelle



SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

55

Exemple polymorphisme



SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

56

Classe abstraite

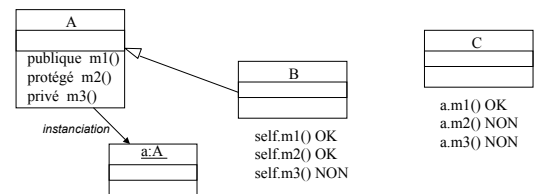
- Classe qui n'est pas utilisée pour l'instanciation, et regroupe des propriétés et comportements
- Une classe dont certaines méthodes seront obligatoirement redéfinies dans les classes utilisées
 - Exemple : animal
 - pas d'instances, mais des instances de sous-classes

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

57

Contrôle d'accès des attributs et méthodes

- 3 types :
 - privé : limitation à la classe
 - public : accès pour toute classe
 - protégé : accès limité aux sous-classes



SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

58

Diagramme de classes

- Regroupement/organisation de l'ensemble des classes de l'application
 - hiérarchie de classe
 - + associations entre ces classes
- Provenance
 - Certaines classes sont livrées avec le système
 - Certaines proviennent de paquets additionnels, récupérés ou achetés
 - Certaines sont fabriquées par le programmeur
- Organisation en paquetages
 - Ensemble de classes utiles

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

59

Définition d'une classe

- Déclaration
 - éventuellement sous-classe d'une ou plusieurs autres classes
- Attributs
 - types simples
 - autres objets
- Méthodes
 - constructeur utilisé à l'instanciation
 - initialiser les attributs
 - Réserver de la mémoire
 - destructeur : utilisé à la destruction
 - libération de la mémoire
 - autres méthodes
 - sélecteurs : renvoient une partie de l'état de l'objet
 - modificateurs : modifient l'état
 - calcul
 - ...

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

60

Exemple python

```
class etudiant:
    "Classe representant un etudiant"

    def __init__(self,n,a):
        self.nom = n
        self.__age = a

    def get_age(self):
        return __age

    def affiche(self):
        print "Nom : %s - age %i" % ( self.nom ,
        self.__age )

    def __repr__(self):
        print "Nom : %s (%i ans)" % ( self.nom ,
        self.__age )

class cours:
    "Classe representant un cours"

    def __init__(self,m):
        self.matiere = m
        self.etudiants = []

    def ajout_etudiant(self,e):
        self.etudiants.append(e)

    def liste_etudiants(self):
        print self.matiere + " : ."
        for e in self.etudiants:
            e.affiche()
```

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

61

Une sous-classe

```
class etudiant_sib(etudiant):
    "Classe representant un etudiant du Mastere SIB"

    def __init__(self,n,a,p):
        etudiant.__init__(self,n,a)
        if p :
            self.prog = 1

    def affiche(self):
        print self.nom
        if self.prog:
            print ("(prend des cours de programmation)")
```

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

62

Instances objets

- Création de l'objet
 - avec des paramètres ou non
 - appel du constructeur adapté
 - allocation mémoire
- Vie de l'objet
 - réception et traitement de messages
 - envoi de messages à d'autres objets
- Mort de l'objet
 - appel du destructeur

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

63

Exemple python

```
>>> e1 = etudiant("Antoine",20)
>>> e2 = etudiant("Antoine",20)
>>> e3 = etudiant("Lydie",24)
>>> e4 = etudiant_sib("Jerome", 22, 1)
>>> c = cours("Systemes d'informations")
>>> c.ajout_etudiant(e1)
>>> c.ajout_etudiant(e2)
>>> c.ajout_etudiant(e3)
>>> c.liste_etudiants()
Systemes d'informations :
Nom : Antoine - age 20
Nom : Antoine - age 20
Nom : Lydie - age 24
Jerome
(prend des cours de programmation)
```

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

64

Critères caractéristiques de l'OO

- Encapsulation données/traitements
- Identité
- Abstraction / classification
- Polymorphisme
- Généralisation / héritage

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

65

Langages orientés-objet

- Plus ou moins récents, objets (différents héritages), lisibles, efficaces, simples, expressifs, modulaires, résistants aux erreurs de codages, interfaçables avec d'autres langages, portables, interprétables/compilables, etc.
- Quelques exemples
 - Smalltalk
 - tout objet
 - C++
 - extension du C
 - JAVA
 - plus haut niveau, très utilisé
 - C# (Microsoft)
 - Clone de Java
 - Python
 - Simple, lisible, très haut niveau
 - ...

SIB 1.2 : Introduction à la programmation orientée-objet / Yannick Prié 2006-2007

66

Initiation à la conception OO

- Quelques transparents
 - Pour donner une idée des grandes lignes de la conception orientée objet de SI
 - Expression des besoins
 - Analyse et conception objet
 - Développement incrémental
 - Gestion de projet
- Mots-clés
 - Réutilisation, abstraction, documentation
 - Attention à l'utilisateur

Objets du monde et objets informatiques

- Objectif :
 - objet informatique de première classe représente un objet du monde réel
 - stabilité au changement
 - langage commun entre
 - l'utilisateur du système
 - le concepteur
 - l'informaticien
- fonder l'analyse du problème (et de la solution) sur des objets partagés

Des besoins aux classes

- Cas d'utilisation
 - classes d'interactions entre système et utilisateur
 - Ex. se connecter au système
 - description par des scénarios = interactions particulières qui décrivent le maximum du fonctionnement du système
 - Ex.
 - sc1 = taper login, mdp, appui « entrée », connexion
 - sc2 = taper login, pas de mdp, appui « entrée », fenêtre erreur, appui « ok »
 - sc3 = taper login, mdp faux trois fois de suite, fenêtre erreur2
 - sc4 = taper login, annulation
 - ...
- Réalisation des scénarios avec des « collaborations d'objet »
 - un ensemble d'objets interagit pour mettre en œuvre le scénario
 - plusieurs scénarios → plusieurs types d'objets qui rendent des services dans divers contextes
- Déduction du diagramme de classes
 - Abstraction à partir des objets nécessaires

Gestion de projet

- Importance du développement incrémental
 - plusieurs cycles pour arriver au système complet
 - prototype qui fonctionne à chaque fin de cycle
 - satisfaction utilisateur / concepteur / développeur
- Attention portée au risque
 - évaluer en permanence ce qu'on sait faire, ce qu'on ne sait pas faire, et le danger associé à chaque risque
 - s'attaquer toujours au plus risqué

Python : le retour

- Déclaration de classes
- Méthodes et attributs privés
- Méthodes de classe
- ...

Remerciements

- Quelques transparents sont directement adaptés du cours « Python » par K. Naik, M. Raju et S. Bhatkar (2002)