République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Biskra

Département d'Informatique

Notes de cours de la matière : Programmation Orientée Objet

Proposé par : Pr. Laid Kahloul

Année Universitaire 2019-2020

Résumé :

L'objectif de ce cours est de présenter aux étudiant de L2 les notions liées à la programmation orientée objet (POO). La POO est une approche de programmation basée sur le paradigme objet à l'opposition du paradigme impératif dont les étudiants ont déjà examiné dans les modules algorithmique en MI et en S1 de L2. Dans le paradigme objet, un programme informatique n'est plus (un algorithme + des structure de données) comme proposé Niklaus Wirth, mais c'est plutôt un ensemble d'objets en interaction. C'est cette interaction qui permet au programme de réaliser les tâches attendues par son programmeur. Dans ce paradigme, deux concepts clés sont à assimilés: (1) le concept d'objet comme une entité réelle ou virtuelle doté d'un état et qui exécute un comportement, et (2) le concept de classe qui est un concept plutôt abstrait regroupant un ensemble d'objet partageant des caractéristiques communs et pouvant exécuter ou subir des actions communes.

Dans ce cours, l'étudiant est censé tout d'abord connaitre et découvrir la philosophie et les motivations du paradigme objet, ensuite il doit apprendre et maîtriser un langage orienté objet. Le module propose des travaux pratiques qui peuvent être faits en langages C++ ou en langage Java.

Introduction Générale :

Rappel sur des concepts de base et motivation de la POO

Ce premier chapitre contient un rappel sur les concepts d'algorithme, d'algorithmique, de programme et de langage de programmation. Les éléments à traiter dans ce chapitre sont les suivants.

1) Introduction: objectif du module, La notion de l'informatique, apprendre quoi? La partie hard, la partie soft
2) La notion d'algorithme, l'activité d'algorithmique
3) La notion de programme, l'activité de programmation
4) Le langage de programmation :
   - 5.1. Développement : langage binaire, langage assembleur, langages de haut niveau (principe, avantages, abstraction,) : fonctions, procédures (60 Fotran) , typage (70 Pascal), modules (80 ada), paramétrage, …
   - 5.2. Usage des langages : scientifiques, pédagogiques et apprentissage, pour le business, pour le hardware (système), etc
   - 5.3. Qualité de langages de programmation : langage structuré/non structuré, langage interprété/compilé, non typé/faiblement typé/fortement typé
   - 5.4. Paradigme de langages de programmation : impératif (exemple), fonctionnel (exemple), Parallèle, déclaratif (exemple), OO (exemple), Aspect (exemple), Agent (exemple).
5) Pourquoi la POO ?
   - 6.1. Limites de la Programmation non OO : manque d'abstraction, manque de réalisme, manque de maintenabilité, manque de fiabilité, manque de réutilisation,
   - 6.2. Apparition d'approche de développement favorisant le développement orientée Objet
   - 6.3. Mise en place de plusieurs langages et surtout, plate-forme favorisant la programmation Orientée Objet

Ce chapitre était présenté dans le cours (Amphi).

Type Abstrait de données : Un fondement théorique de la POO

Dans ce chapitre, on fait une présentation du fondement théorique de la programmation orientée objets. Il s'agit de présenter le concept de type abstrait de données, comment le spécifier syntaxiquement et comment présenter sa sémantique axiomatique. Ce chapitre exige des connaissances dans l'algèbre et la logique équationnelle. Ce chapitre présenté dans le cours (amphi) traite les éléments suivant :

1) Pourquoi la notion de TAD : Il s'agit de décrire abstraitement les données qui seront créées dans le système. l'idée est de faire du raffinement successif, donc au début on ne s'intéresse pas à trop de détails.
2) Définition de type : ensemble de valeurs, on ne désigne pas d'opération spécifique à ce type=> ambigüité possibles, des expressions qui n'ont pas de sens, … Exemple : un poids réel, une longueur réelle, on peut faire la somme des deux ? ça donne quoi ?
3) Définition d'un TAD :
   - Un type de données défini indépendamment de son implémentation.
   - Il ne dépend pas d'un langage de programmation particulier.
   - Il peut être vu comme un ensemble de valeurs à qui on joint les opérations permettant de les manipuler et de les créer.
   - L'implémentation d'un TAD sera faite avec une structure de données qui décrit précisément les détails de réalisation de ces valeurs ainsi que des opérations. Exemple : Tableau, Ensemble, Liste, File, Pile, Arbre, … En principe, les valeurs d'un TAD ne peuvent être accessibles que via les opérations définies dans ce TAD.
4) Description de TAD (Spécification de TAD): langage pour décrire le TAD, spécification algébrique=Syntaxe+sémantique. Avantage : ça donne une idée d'une réalisation fonctionnelle, ou récursive des opérations. (Voir les exemples ci-dessous)
5) Ecriture d'un TAD : Partie syntaxique (opérations) : constructeur, inspecteur+ partie Sémantique (axiomes) : décrivant des propriétés ou invariants toujours satisfaites sur les données du TAD. Donner des exemples. Comment trouver les axiomes : intuition, tout en assurant : la complétude (nombre de constructeurs * nombre d'inspecteurs : composition de l'inspecteur sur le constructeur+ un nombre de pré-condition sur les opérations), la cohérence.

Genre Nom_TAD
Utilise Sortes
Opération:
nom_op:domaine->co-domaine
Sémantique:
Pré-condition:
nom_op ***def_ssi*** condition
Axiomes:
Insp (Cons)=Expression à déduire

Fin

Exemple de TAD:

le TAD booléen

Genre boolean
Utilise Sortes
Opération:
Vrai: ->boolean
Faux :->boolean
Not :boolean->boolean
And :boolean x Boolean->Boolean
Or: Booleanx boolea -> boolean
Sémantique:
Pré-condition:
nom_op ***def_ssi*** condition
Axiomes: b :boolean
not (vrai)==faux
not(faux)==vrai
and(vrai, faux)==and(faux, vrai)=faux
and(vrai,vrai)=vrai
or(vrai,vrai)=or(vrai, faux)==or(faux, vrai)=vrai
or(faux,faux)=faux
and(vrai,b)==and(b,vrai)=b
and(faux,b)==and(faux,b)==faux
or(vrai,b)==or(b,vrai)==vrai

Fin

- le TAD naturel

Genre nat
Utilise boolean
Opération:
0: ->nat
succ :nat->nat
+ :nat x nat->nat
- :nat x nat->nat
x: Boolean x boolea -> Boolean
>=:nat x nat -> Boolean
Sémantique: a,b :nat
Pré-condition:

-(a,b) ***def_ssi*** a>=b

Axiomes:

+(0,succ(0))==+(succ(0),0)==succ(0)

+(b,0)==+(0,b)==b

+(b,succ(0))==Succ(b)

+(succ(a),b)==+(a,succ(b))==succ(+(a,b))

Exemple de code

```
+( succ(a),b){
return succ(+(a,b))
}
```

-(b,0)==b

-(succ(b),b)==succ(0)

-(succ(b),succ(0))==b

-(succ(a),b)==0 if succ(a)=b sinon +(succ(0), -(a,b))

Exemple de code

```
-( succ(a),b){
If (succ(a)=b)return 0;
Else return +(succ(0), -(a,b))
}
```

>=(b,0)==true

>=(0,b)=true if 0=b false otherwise

>=(succ(a),succ(b))=true >=(a,b) fale otherwise

---

x(b,0)==x(0,b)==0

x(b,succ(0))==x(succ(0),b)==b

x(succ(a),b)==x(b,succ(a))==+(b,x(a,b))

Fin

Penser à spécifier : le TAD réel et Le TAD Pile_Entier.

6) TAD enrichi : Motivation et syntaxe

7) TAD générique : Motivation et syntaxe

Exercices:

Exercice 1 :
Soit la description suivante d'une **Pile d'entiers**. Une Pile peut être créée par une opération **Créer_Pile** qui ne prend aucun argument et donne une Pile vide, elle dispose des opérations suivantes : (1) **Pile_Vide** : permettant de voir si une Pile est vide, (2) **Sommet** : permettant de retourner la valeur stockée dans le sommet, (3) **Empiler :** permettant d'empiler un entier donnée, (3) **Dépiler :** permettant de dépiler.

Exercice 2 :
Soit un TAD File d'éléments (type de base est dit: Elem) dont les opérations sont:

1. Créer: permet de créer une file vide
2. Enfiler: permet d'enfiler un élément dans la file sur la queue
3. Défiler: permet de défiler la tête
4. Tête: consulte le tête de la file
5. Queue: consulte la queue de la file
6. FileVide: teste si la file est vide ou non.