4) Compléments sur l’instanciation

4.1) Et si les attributs ne sont pas d’un type primitifs ?

Il faut continuer d’appliquer le mécanisme d’instanciation aux attributs eux-mêmes.

Exemple :

CoupleComplexe

NombreComplexe c1

NombreComplexe c2

CoupleComplexe cc 🡨 créer CoupleComplexe()

CC c1 : null

C2 : null

Pour le créer complètement

cc.c1 🡨 créer NombreComplexe()

cc.c2 🡨 créer NombreComplexe()

* Instancier l’objet lui-même, instancier ensuite ses attributs non primitifs

Et si les attributs des attributs ne sont pas primitifs ?

En pratique, on ne procède pas comme ça pour instancier :

On va alors doter une classe de constructeurs chargée=s de créer correctement les objets en mémoire.

4.2)Chaines et tableaux

Chaînes – type java String (type objet)

Java réalise tout seul l’instanciation, par exemple dans String s = ‘‘Toto’’ ;

Tableaux :

Il faut instancier en donnant leur taille typeTableau nomTableau 🡨 créer typeTableau[taille]

Ex : tableau d’entier tabInt 🡨créer tableau[20] de entier

Instancie un tableau de 20 entier

En Java int[] tabInt = new int[20] ;

En java, le type tableau possède un attribut nommé length qui donne sa taille.

Chapitre 4 : Les Constructeurs

1. Notion de constructeur

Par quel mécanisme est assurée la création en mémoire des objets, et quelle valeur initiale ont les attributs ?

Réponse : chaque appel à créer (new en Java) invoque une méthode spéciale (qui est un constructeur) qui :

-réserve l’espace mémoire

-initialise les attributs

Un constructeur peut être implicite ou explicite.

1. Constructeur Implicite

Sa définition n’apparaît pas dans la classe ! Il existe pourtant car java en a besoin pour créer les objets.

Exemple : on m’a défini aucun constructeur pour NombreComplexe et pourtant java sait le créer en mémoire.

Que fait le constructeur implicite ?

Il déclare une variable pour chaque attribut et l’initialise à une ‘‘valeur nulle’’

Valeur nulle : - entier : 0 -booléen : faible

-réel : 0.0 -caractère : ‘0000’

-chaîne : ‘‘’’ -type objet : constante null

Exemples

NombreComplexe c 🡨 créer NombreComplexe()

C pReelle 0.0

pImage 0.0

Pour faire plus que ça, il faut prendre le contrôle de l’instanciation par des constructeurs explicites.

1. Ecrire des constructeurs explicites

On peut écrire soi-même les constructeurs d’une classe : ils deviennent alors explicites.

Constructeur explicite : code chargé de créer et initialiser les attributs, éventuellement en fonction de valeurs reçues en paramètre.

3.1) Syntaxe

PDL++ à une méthode nommée constructeur(…) , sans type de retour.

Java : méthode possèdant exactement le même nom que la classe, sans type de retour (même ^pas void)

C’est cette méthode qui est appelée par créer (new)

Le constructeur explicite peut admettre des paramètres.

Exemple 1 : nombre complexe

NombreComplexe

Réel pReelle, pImag

Constructeur(reel pReelle, reel pImag)

Reel module()

Reel conjugue()

PDL++

Classe NombreComplexe{

Réel pReelle, pImage

Constructeur(reel pReelleI,reel pImagI)

pReelle 🡨 pReelleI

pImage 🡨 pImagI

fin

reel module()

…

Reel conjugue()

…

}

Programme principal()

NombreComplexe c 🡨 créer NombreComplexe(5,8)

Fin

C pReelle 5

pImag 8

En java

Public class NombreComplexe{

Double pReelle, pImag ;

NombreComplexe(double pReelleI, double pImagI){

pReelle = pReelleI ;

pImage = pImagI ;

}

Double module(){…}

Double conjugue(){…}

}

3.2) Instanciation des oAttributs objets

Ex classe CoupleComplexe{

NombreCompleexe c1,c2

Constructeur()

C1 🡨 créer NombreComplexe(0,0)

C2 🡨 créer NombreComplexe(0,0)

Fin

}

TRES IMPORTANT

Si on écrit un constructeur explicite, il n’y a plus de constructeur implicite.

Par exemple, l’instruction

NombreComplexe c 🡨 créer NombreComplexe()

Ne compte plus !! Il faut indique 2 valeurs de paramètre.

Heureusement, on peut surcharger les constructeurs

1. Plusieurs constructeurs dans une classe
   1. Surcharge

Comme toute méthode, un constructeur peut être surchargé

Ex : classe NombreComplexe{

Rééel pReelle, pImag

Constructeur(reel pReelleI, reel pImagI)

pReelle 🡨 pReelleI

pImag 🡨 pImagI

fin

Constructeur()

pReelle 🡨0

pImag 🡨 0

fin

double module()

rien conjugue()}

Du coup NombreComplexe c 🡨 créer NombreComplexe() marche à nouveau

En pratique et si ça a du sens, il faut penser à rédiger un constructeur sans paramètre, car il n’existe plus sinon.

* 1. Un constructeur peut en appeler un autre

Exemple : les 2 constructeurs de NombreComplexe

Constructeur(réel pReelleI , réel pImagI) {…}

Constructeur ()

Constructeur(0.0) //appel du constructeur précédent

Fin

En java, l’appel à un autre constructeur se fait par un mot spécial : this(…)

Restrictions

* Un autre constructeur ne peut être appelé que par un autre constructeur (pas depuis une autre méthode)
* ?

Public class NombreComplexe{

Double pReelle,pImag ;

NombreComplexe(double pReelleI, double pImagI){

pReelle = pReelleI ;

pImag = pImagI ;

}

NombreComplexe(){

this(0,0) ;

}

}

Chapitre 5 Références, visibilité des variables

1. Un objet est identifié par une référence
2. Notion de référence

Une variable d’un type objet contient en réalité une référence à un objet situé en mémoire.

C’est un identifiant indiquant de manière non ambigüe de quel objet on parle. Pour fixer les idées, on le voit comme un nom donnée à la partie de la mémoire contenant l’objet.

On utilisera une flèche vers un objet pour représenter une référence.

L’opérateur créer (new en java) appelle un constructeur qui créer l’objet en mémoire, puis retourne une référence à cet objet.

Exemple :

Programme principal()

NombreComplexe

C1 🡨 NombreComplexe()

C2 🡨 NombreComplexe()

Fin

Fin

1. Affectation d’une référence à une autre

Cas primitif (pas de référence) En mémoire

Int n1=5 , n2=8  n1 : 5 n2 : 8

n1 = n2 n1 : 8 n2 : 8

n2 = 10 n1 : 8 n2 : 10

n1 et n2 continuent d’avoir un existence séparée.

Cas objet (avec référence)

NombreComplexe c1 = new NombreComplexe(1,1)

c2 = new NombreComplexe(2,2)

c1 = c2

On obtient un objet avec deux nom différents

C2.pImag 🡨 5

Ecran.afficher(c1.pImag) affiche : 5

En affectant une référence à une autre, on se retrouve avec 2 noms pour un même objet. Ramasse-miettes (garbage collector) Java libère tout seul les espaces mémoire qui ne sont plus référencés.

1. Passage par valeur ou par adresse ?

Passage par valeur : le paramètre n’est pas modifié à son retour dans le code appelant.

Ex : static void swap(int a, int b){

Int temp = a ;

a = b ;

b = temp ;

}

Code appelant programme qui appelle une fonction.

Public static void main(String[] args){

Int a=5, b=8 ;

Swap(a,b) ;

System.out.println(‘’a = ’’+a+’’b = ‘’+b) ;

}

Affiche a = 5, b=8

Le passage par adresse correspondrait à travailler directement avec les variables du programme appelant.

Ça existe en C, en pascal, pas en JAVA

Mais ce n’est pas un problème

Static void swap (NombreComplexe c1, NombreComplexe c2){

Double tempR = c1.pReelle ;

Double tempI = c1.pImag ;

C1.pReelle = c2.pReelle ;

C1.pImag = c2.pImag ;

C2.pReelle = tempR ;

C2.Imag = tempI ;

}

Main : NombreComplexe c1 = new NombreComplexe(1,1)

NombreComplexe C2 = new NombreComplexe(2,2)

Swap(c1,c2) // le swap à bien eu lieu