Commentaires

`/*... */`	multi-lignes
`//`	jusqu'à la fin de la ligne
`/** ... */`	commentaires Javadoc
`@author`	auteur
`@version`	version
`@param nom description`	description du paramètre `nom` d'une méthode
`@return description`	description du retour d'une méthode
`@exception typeException`	exception renvoyée par une méthode

Types et références de base

`void`	pas de valeur de retour
`boolean`	**`true`**/**`false`**
`char`	caractère sur 16 bits
`byte`	8 bits
`short`	entier sur 16 bits
`int`	entier sur 32 bits
`long`	entier sur 64 bits
`float`	flottant 32 bits IEEE754
`double`	flottant 64 bits IEEE754
`Object`	un type non primitif
`null`	référence nulle vers un objet

Opérateurs et expressions

`{ ... }`	délimiteurs de bloc
`+, -, *, /, %`	opérations arithmétiques
`++, --`	incrémentation, décrémentation
`>, >=, <, <=`	opérateurs de comparaison
`==, =!`	égalité, inégalité de deux valeurs ou références
`&, |, ^, !`	ET, OU, XOR, négation logique
`&&, ||`	ET conditionnel, OU conditionnel
`&, |, ^`	opérateurs bits à bits (seul. sur des entiers)
`=`	opérateur d'assignation. L'opérande de gauche doit être une variable
`?:`	opérateur conditionnel
`instanceof`	évalue si une référence pointe vers un objet d'une certaine classe ou interface
`(type)`	transtypage vers `type`
`+`	concaténation d'objets de type `String`
`new`	opérateur de création d'instance

Flux de contrôle

`if` `(cond) {`	
`...`	exécuté si `cond` renvoie **`true`**
`} else {`	
`...`	exécuté sinon
`}`	
`switch` `(expression)`	expression doit être de type `integer` ou `String`
`case` `n: ...`	exécuté si `expression` vaut `n`
`case` `m: ...`	exécuté si `expression` vaut `m` ou `n`
`...`	il faut un **`break`** pour sortir du switch
`default``: ...`	si rien ne convient
`while` `(condition) {`	
`...`	exécuté tant que `condition ==` **`true`**
`}`	(peut ne pas être exécuté)
`do` `{`	
`...`	executé tant que `condition ==` **`true`**
`}` **`while`** `(condition);`	(exécuté au moins une fois)
`for``(init; cond; inc) {`	boucle
`...`	`init` déclare et/ou initialise une ou des variables
`}`	la boucle est exécutée tant que `cond ==` **`true`**
	`inc` est exécuté après chaque passage
`return`	finit une méthode (attention au type de retour)

Visibilité

public	accès pour toutes les classes
protected	accès pour les classes du paquetage et les sous-classes
(rien)	accès pour les classes du paquetage
private	accès exclusif pour la classe

Modifieurs

static	attribut ou méthode de classe
final	la classe ne peut avoir de classe fille, la méthode ne peut pas être redéfinie, l'attribut ne peut pas être modifié
final static	constante

Écriture d'une classe

package pack.subp;	la classe appartient au paquetage pack.subp le fichier **.class** correspondant devra se trouver sous $CLASSPATH/pack/subp/
import package2.*;	on importe toutes les classes contenues dans package2

//Declaration de la classe

public ou rien	visibilité de la classe
final	la classe ne peut être spécialisée
abstract	la classe est abstraite
class MaClasse	
extends SupClasse	MaClasse est une sous-classe de SupClasse elle hérite de tous les membres non privés de SupClasse elle peut redéfinir des méthodes de SupClasse elle peut masquer des attributs de SupClasse
implements MonInt	MaClasse implante les méthodes de MonInt les objets de MaClasse sont également du type MonInt

{

//Attributs

visibilite	pas pour les variables locales !
[**final**|**static**]	
Type monAtt;	déclaration. Initialisé avec la valeur par défaut de Type

//Constructeurs

visibilite	
MaClasse (par) {	**this**(par); appelle un autre constructeur de MaClasse **super**(par); appelle un constructeur de SupClasse ces deux expressions doivent être la première instruction du constructeur, sinon Java met **super**(); par défaut
}	

//Methodes

visibilite	
[**final**|**static**]	
abstract	pas de bloc { ... }, mais un ;
Type	type de retour
maMethode(par)	peut être surchargée (paramètres)
throws uneEx	les méthodes qui redéfinissent maMethode doivent lancer la même exception ou un sous-type

{ ... }
}

Types paramétrés

class MaClasse<T,U>	la classe possède les types paramétrés T et U
<T,U> type maMeth(...)	la méthode possède les types paramétrés T et U
<T **extends** C>	borne supérieure de T
<? **extends** C>	*wildcard*