Cours 3

I. Les boucles

I.1 La boucle TANT QUE *(fondamentale)*

I.1.1 En pseudo-langage : TANT QUE condition FAIRE instructions FIN TANT QUE

répéter les instructions tant que la condition est vraie c'est-à-dire jusqu'à ce que la condition devienne fausse

I.1.2 Exemple : SAISIR(n) TANT QUE n<0 FAIRE AFFICHER_ERREUR puis SAISIR(n) FIN TANT QUE

I.1.3 Caractéristiques :

- nombre de tours inconnu a priori
- test au début ⇒ peut tourner 0 fois
- pb d'initialisation de l'expression booléenne
- pb de modification de l'expression booléenne
- comme pour le `if`, toujours mettre des `{ }`

I.1.4 Syntaxe : `while ( expression_booléenne ) { instructions_tant_que_vraie }`

syntaxe ressemble au `if` mais test ≠ répétition

II. Nouveau type primitif : nombre "réel"

II.1 type entier = sous-ensemble de $\boldsymbol{Z}$: $[-2^{31}, +2^{31}-1]$

II.2 type "réel" = sous-ensemble de $\boldsymbol{D}$ (et non $\boldsymbol{Q}$ ou $\boldsymbol{R}$) ⇒ précision limitée ⇒ nombres à virgule flottante.

min / max et nombre maximum de chiffres significatifs

exemple : bien que max=10^{22}, 1234567 n'est pas représentable si 6 ch.sig.

ligne réelle (voir dessin)

II.3 Java : type primitif sur 8 octets : `double`

II.4 Représentation binaire :

1|11|52 bits → signe | exposant ($<10^{\pm 308}$) | mantisse (16 ch.sig.)

beaucoup plus complexe : voir représentation des nombres réels et pour encore plus d'explications lire floating point numbers

somme d'inverses de puissances de deux différentes ⇒ 0.1 non exact et calculs non exacts.

II.5 Valeurs littérales :

première forme = ±chiffres.chiffres | ±chiffres. | ±.chiffres

deuxième forme = *première forme*E±entier | ±entierE±entier

exemples : 0.5 -4. .32 (=0,32) 1E-6 (=10^{-6}) 2.5E12 (=$2{,}5\text{x}10^{+12}$)

II.6 Opérateurs

II.6.1 unaires : `+ -`

II.6.2 binaires : `+ - * / %`

/ = division réelle ⇔ au moins un des deux nombres est réel

% = définition du modulo étendue aux réels : toujours la relation `a = b × q + r` mais avec q toujours entier

II.7 Fonctions mathématiques (dans la classe Math)

racine carrée = square root = `sqrt`

`abs, sin, cos, tan, log, log10, exp, pow`, ... voir syntaxe d'appel ci-dessous (III.1)

II.8 Comparaisons

- < et > : OK
- `<=, >=, ==, !=` : INTERDIT ! *(problèmes de précision)*
- |x-y|<ε remplace `x==y`
- encore mieux en tenant compte de l'ordre de grandeur : |(x-y)/y|<ε ou |x/y - 1|<ε

II.9 Conversions

- avec perte : `int vE = (int)vR;` *(nouvelle expression entière, ce n'est pas la partie entière, c'est une troncature)*

  Exemples : (int)3.14 → 3 (int)-3.14 → -3 Partie_entière(-3.14) → -4
- sans perte *(car 31 bits < 52)* : `vR = (double)vE;` *(pour la division, nouvelle expression réelle)*
- en `String` : `vS = new String(vR);` ou `vS = "" + vR;`
- possible `String` → `double` ou `int` *(voir syntaxe au III.1 ci-dessous)*

III. Membres de classe (≠ membres d'instance)

III.1 Méthodes de classe
ne nécessitent pas d'instance pour être appelées ⇒ sont appelées directement sur la classe
Exemple : `vR = Math.sqrt(vX);` ou `vR = Double.parseDouble( vChaine );`
ne peuvent pas accéder aux attributs d'instance !
⇒ ne peuvent pas appeler de méthodes d'instance (qui nécessitent une instance) ;
par contre, une méthode d'instance peut appeler une méthode de classe

III.2 Syntaxe :
- On les reconnaît dans la javadoc car signature commence par `static`
- On les déclare en ajoutant `static` après `public`

III.3 Attributs de classe
- On désire compter le nombre de cercles.
- Si attribut d'instance `aNbCercles`, valeur dupliquée dans chaque cercle, et si le constructeur l'incrémente, vaut 1 dans tous les cercles !
- On a besoin d'un attribut partagé par tous les objets de la classe Cercle.

III.4 Syntaxe :
- On les reconnaît dans la javadoc car leur déclaration commence par `static`
- On les déclare en ajoutant `static` après `private`

III.5 Initialisation des attributs statiques
- Mais comment les initialiser avant même qu'un objet soit créé ?
- `static {` instructions `}` à mettre en tête de classe, un peu l'équivalent pour la classe du constructeur pour l'objet : exécuté automatiquement une seule fois au chargement de la classe

III.6 Application : les constantes
- Déjà vu : `private final int MAX = 10;`
- mais même valeur dupliquée dans tous les objets de la classe
- ⇒ constante de classe : `public static final int MAX = 10;` *(aucun risque de modification)*
- Les accès : `MAX` ou `Classe.MAX` mais `objetDeLaClasse.MAX` est possible

IV. **Paquetages** : à lire *(au programme des contrôles ...)*

- *non traité (voir TP3)*

V. Tests unitaires avec JUnit *: A lire (pour les TP)*

Lire le poly :

tout jusqu'à la section 2.3, sections **2.4.1, 2.5**, 3.1, 4, 5.2.0, 6, 7.1, 7.2, 8.1, 8.2.1.1, **8.2.2.1, 13.3**, et annexes 6 & 7 *(à compléter)*

Dernière mise à jour : mardi 13 mars 2012 à 10h19.