Cours 2.1

I. Commentaires et documentation

I.1 `//` jusqu'à la fin de la ligne (ignoré par compilateur)
I.2 `/**` jusqu'à `*/` (multiligne, ignoré par compilateur mais JavaDoc)
I.3 Conseils: `/**` ... `*/` AVANT chaque classe et AVANT chaque méthode
- I.3.1 Classe : `@author` et `@version`
- I.3.2 Méthode : `@param` et `@return`
- I.3.3 Lire ce guide

I.4 `/*` jusqu'à `*/` (multi-lignes ou intra-ligne, ignoré par compilateur, attention ! pas d'imbrication, à conserver pour déboguer)
I.5 Dans BlueJ : bouton implémentation (→ source java) / interface (→ javadoc)

II. Déclarations

II.1 Classe : `public class NomClasse {` attributs(o.i.), constructeurs(o.i.), autres méthodes(o.i.) `}` *(o.i.) = ordre indifférent*
II.2 Attribut : `private Type aNomAttribut;`
- protège de l'extérieur de la classe ==> un autre objet de la même classe peut y accéder

II.3 Fonction :
`public Type nomFonction( paramètres_formels ) { instructions; return expression; }`
II.4 Procédure : `public void nomProcedure( paramètres_formels ) { instructions; }`
II.5 Constructeur : `public NomClasse( paramètres_formels ) { instructions; }`
- rôle = initialiser les attributs ==> naturel : autant de paramètres que d'attributs ==> instructions dans l'ordre des attributs
- mais surcharge ==> plusieurs constructeurs ==> plusieurs moyens d'initialiser un objet (exemple: "origine" ==> 0,0)
- *un constructeur peut en appeler un autre par `this( paramètres );` en première instruction du constructeur*

II.6 Paramètres : `final Type pNom1, final Type pNom2` ou rien entre les parenthèses de la signature d'une méthode
- `final` souvent oublié, mais évite certaines erreurs car le compilateur peut vérifier que le paramètre n'est jamais modifié

II.7 Variable locale : `Type vNom;` en tête du coprs d'une méthode ou juste avant d'en avoir besoin
II.8 Initialisation : `Type vNom = valeurInitiale;`
- Impossible pour les paramètres et ne pas utiliser pour les attributs (rôle du constructeur)

II.9 Différence déclaration / définition
- II.9.1 Classe : toujours définition
- II.9.2 Méthode : déclaration = juste signature
- II.9.3 Variable = définition = initialisation

II.10 Durée de vie / visibilité
- II.10.1 Attribut : durée de vie de l'objet / interne à la classe
- II.10.2 Paramètre (formel) : pendant l'exécution de la méthode / interne à la méthode
- II.10.3 Variable locale : depuis sa déclaration jusqu'à la fin du bloc / interne au bloc

III. Instructions simples

III.1 une instruction (se termine par un `;`), `{` des instructions `}` = un bloc = une instruction composée
III.2 affectation (ou assignation ou <---) non symétrique : `vNom= expression;`
- initialisation ≠ déclaration + affectation

III.3 retour de résultat (fonction uniquement) : `return expression;`
- instruction ≠ expression (notamment opérations) → *voir plus loin la définition d'une expression*

III.4 Fin de méthode (procédure uniquement) : `return;` (facultatif)
III.5 Affichage (non graphique ==> terminal,console / linux,dos,BlueJ) : `System.out.println( uneString );`

mais conversion automatique en `String` des `int` et `boolean`
et concaténation par l'opérateur + : `"vI=" + vI` vaut `"vI=12"` si vI vaut 12
`println` ≠ `print`

IV. Objets et références

IV.1 création d'un nouvel objet : `vC2= new Cercle();` ou `vC2= new Cercle( paramètres_effectifs );`
(appel automatique du constructeur) si `Cercle vC2;` auparavant (voir dessin) §
référence, pointeur, adresse / reference type = type objet / référence spéciale = `null`

IV.2 classe **(cf co1.2-V.3.3) spéciale : ne nécessite pas d'appel de constructeur
`String vM;` puis `vM= "mot";` remplace `vM= new String("mot");` ← paramètre effectif
très utilisée, beaucoup de méthodes (exemple: `length()`), opérateur +

IV.3 accès à un attribut (dans la même classe) : `vC2.aNomAttribut` (sur l'objet c2) ou `this.aNomAttribut`
`this` = référence spéciale, à l'objet courant (objet sur lequel a été appelée la méthode)

IV.4 appel de méthode : `vC2.move( paramètres_effectifs );` ou `vS= vC2.surface();` (sur l'objet vC2)
`this.move( paramètres_effectifs );` ou `vS= this.surface();` (`this` = l'objet courant)

IV.5 appels successifs : `this.aChaine.length()` ou `this.getChaine().length()`

V. Recopies

V.1 Type primitif
`int vE1=12; int vE2; vE2=vE1; vE1=vE1*2; vE2`? (voir dessin) §

V.2 Type objet
`Circle vC1=new Circle(0,0,12); Circle vC2; vC2=vC1; vC1.doubleSize();`
`vC2.getDiameter()`? (voir dessins, clonage) §

V.3 Passage de paramètre ("par recopie")

V.3.1 type primitif ==> recopie de valeur ==> aucun danger
```
void fois2( int pN ) { pN = pN*2; affiche(pN); pN=0; }
```
ailleurs : `int vI=12; fois2(vI); affiche(vI);`

V.3.2 type objet ==gt; recopie de référence ==gt; aucun danger pour la référence mais pour l'objet !
```
void fois2( Circle pC ) { pC.doubleSize(); affiche(pC.getDiameter());
pC=null;}
```
ailleurs : `Circle vC=new Circle(0,0,12); fois2(vC); affiche(vC.getDiameter());`

Lire le poly : sections 4, 5.2.0, 7.1, 7.2, et 8.1