Java Cours 8

**Héritage**

**Délégation (agrégation / composition) :**

**Diapo 2**

La variable de classe est équivalente à une constante. Ici on a déjà créé un point ORIGINE. Comme la variable est final elle ne peut plus être modifiée une fois instanciée. Elle est static car elle est liée à la classe point et public pour pouvoir « l’utiliser » Point.ORIGINE (essayer de le faire pour tester voir comment ça marche)

**Diapo 5**

Ici la flèche sur le diagramme UML, c’est pour donner la notion de navigabilité. Les instances de C1 connaissent les instances de C2, mais pas dans le sens inverse. L’attribut de la classe C1 provenant de la classe C2 est écrit sur la flèche et la cardinalité dépend du nombre de ref de C2 que l’on a dans C1.

Pour avoir une cardinalité 0..\* on aurait un tableau ou une liste d’attribut de C2 dans C1.

**Diapo 8**

Il faut lier le centre et le cercle. Pour éviter les effets de bords, il faut dans le constructeur de cercle créer un nouveau point centre et ainsi ce cercle possède son propre centre.

Plus d’effet de bord avec cette façon de faire. Mais le cycle de vie du Point et du Cercle sont liés, si le Cercle est détruit (ou copiés) le centre (qui est un Point) l’est aussi !

**Diapo 9**

Agrégation c’est la diapo 7. Composition c’est la diapo 8.

Agrégation l’objet roue (point) existe en dehors de la classe voiture (cercle).

Composition l’objet roue (point) existe que dans la classe voiture (cercle).

**Héritage :**

**Diapo 12**

Ici on veut ajouter un attribut et une méthode à la classe point. Il faut alors faire un héritage.

L’héritage : Public class Class1 extends Class2

Tout ce que je peux faire dans la classe 2 je peux l’utiliser dans la classe 1.

Remarque : les couleurs c’est comme string c’est immuable.

Les this.x et this.y dans le constructeur de la classe pointGraphique fait référence au x et y de la classe Point (en réalité il faut utiliser super(x,y) pour faire appel au constructeurs de la classe Point. C’est le chainage des constructeurs).

**Diapo 14**

Avec l’héritage, on remonte dans la classe qui possède les méthodes que l’on appelle sur notre objet pour pouvoir les exécuter.

**Diapo 16**

La super-classe est un concept plus général que la sous-classe. La sous-classe est une spécialisation de la super-classe.

**Diapo 18**

Une sous-classe possède tous les attributs et méthodes de **ses** super-classes.

**Diapo 19**

Quand on appelle une méthode d’une super-classe, le « compilateur » parcourt la classe courante et les super-classes pour trouver la méthode à exécuter.

**Diapo 20**

Pour la redéfinition, la méthode de la classe courante a la même signature que la méthode de la super-classe, mais effectue un « truc » différent.

Quand on appelle la méthode dans la classe courante, il prend la première méthode qu’il rencontre donc il utilisera celle redéfinit.

**Diapo 22-23**

Attention surcharge et redéfinition c’est très différent. La surcharge c’est que l’on ajoute un attribut à la méthode déjà créée. Surcharge méthode même nom, mais le nombre et/ou type des attributs différents. Redéfinition (overriding) signature identique des deux méthodes, mais contenu différent.

Exemple utilisons le cas suivant classe B hérite d’A :

Dans le cas de la surcharge : la méthode add(int x) de A est surchargée dans B add (double x). Ainsi B possède deux méthodes add : add(int) et add(double).

Dans le cas de la redéfinition (@override) : la méthode add(int x) de A est redéfinie dans B add(int x), toujours la même signature c’est juste le contenue qui change. B possède alors une seule méthode add.

**Diapo 25**

Super permet de designer la super-classe de notre classe courante. Si on a une redéfinition, d’une méthode et que l’on utilise super.méthode() on utilisera la méthode de la super-classe plus les nouvelles instruction de la méthode de la classe courante.

**Diapo 26**

Héritage simple. Donc l’héritage forme un arbre. Seule la classe Object ne possède pas de super-classe.

**Diapo 27**

Exemple héritage multiple (Smartphone possède un héritage multiple). Quand des attributs ou méthodes viennent de deux branches différentes, il faut résoudre les conflits.

**Diapo 31**

Lorsque l’on fait appel au constructeur de la super-classe, il faut utiliser le mot clé super suivi des paramètres nécessaire pour le constructeur de la super-classe. Si on fait appel dans le constructeur de la classe courante au constructeur de sa super-classe, il faut obligatoirement que super(att1, att2 …) soit la première instruction du constructeur courant.

**Diapo 34-35**

Si on ne met pas de constructeur dans notre classe courante elle va hériter du constructeur par défaut de sa super-classe. Si la super-classe a un constructeur avec paramètres, alors le constructeur par défaut ne sera plus présent. Et donc l’appel implicite au constructeur super() de la super-classe ne peut plus se faire. Erreur de compilation. On doit faire appel au constructeur de la classe mère de façon explicite.

**Diapo 36**

Attention si on utilise dans la classe courante des noms de variables (le type peut être différent) d’instance identiques à ceux d’une super-classe on va les masquer. Pour accéder aux variables de super-classe il faut alors les spécifier directement (voir diapo pour l’accès).

**Diapo 37-41**

Il existe quatre mots clé permettant de gérer la visibilité des méthodes et des attributs d’une classe :

* private : attributs et méthodes seulement accessibles dans la classe courante.
* - (package) : attributs et méthodes accessibles par les classes du même package.
* protected : attributs et méthodes accessibles par les classes du même package et les sous-classes d’un autre package.
* public : attributs et méthodes accessibles par tout le monde.

**Diapo 42**

Pour les classes seulement deux visibilités : public et - (package). Une classe public peut être utilisée par toutes les autres classes. Une classe – (package), ne peut être utilisées que par les classes du même package.

Possible d’avoir plusieurs classes dans un fichier, mais une seule classe en public. Mais il est tout de même plus sur faire un fichier par classe.

**Diapo 44**

Méthode final, impossible de redéfinir cette méthode dans les sous classes.

Toutes les méthodes privées sont implicitement final. Lorsque l’on créé une méthode final il faut bien être sûr de l’utilité dans le cas contraire il vaut mieux ne pas l’ajouter.

**Diapo 45**

Il est possible d’avoir des classes avec le mot clé final. Dans ce cas la classe n’est plus éligible pour l’héritage et ses attributs et méthodes ont automatiquement le statut final aussi.