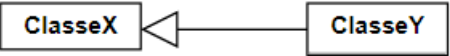
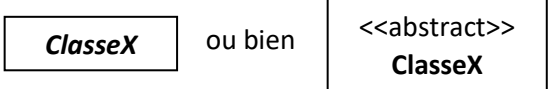
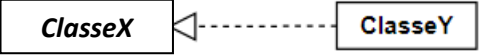
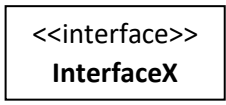
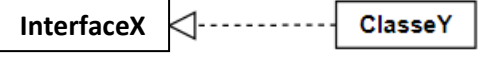
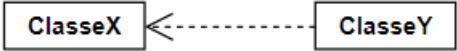
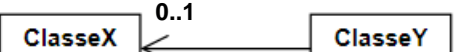





<p>Héritage</p> 	<p>// ClasseY hérite des attributs et méthodes de ClasseX avec leurs modes d'accès (public, private, protected). // ClasseY peut définir de nouveaux attributs et méthodes. ClasseY peut redéfinir les méthodes héritées, sauf si final.</p> <pre> public class ClasseX { public void methode() { // Code Selon ClasseX; } } public class ClasseY extends ClasseX { // Redéfinition de la méthode héritée public void methode() { // Code Selon ClasseY; } } </pre>
<p>Abstraction</p> 	<p>// ClasseX comporte une ou plusieurs méthodes abstraites donc sans code. On ne dispose que de la signature. Les méthodes static, final ou private ne peuvent pas être abstraites. // ClasseX n'est pas instanciable.</p> <pre> public abstract class ClasseX { public abstract void methodeDéclaréeDansClasseX(); // Juste la déf/signature (pas de code). } </pre>
<p>Réalisation / Implémentation d'une abstraction</p> 	<p>// ClasseY hérite de ClasseX abstraite, avec les mêmes propriétés d'accès (cf. public, private, protected) et implémente les méthodes abstraites. Instanciation possible de ClasseY</p> <pre> public class ClasseY extends ClasseX { public void methodeDéclaréeDansClasseX() { // Code Selon ClasseY; // Implémentation (on donne le code) dans ClasseY. } } </pre>
<p>Interface</p> 	<p>// InterfaceX définit une interface. Elle définit la signature de méthodes et n'est donc pas instanciable. // Les méthodes sont implicitement public et abstract, avec un éventuel comportement par défaut. // Les éventuels attributs sont implicitement public, constant et de classe (cad. public static final).</p> <pre> public interface InterfaceX { default public abstract void methodeDéclaréeDansInterfaceX() { // Eventuel Code Par Défaut Selon InterfaceX } public static final int UN=1; } </pre>
<p>Réalisation / Implémentation d'une interface</p> 	<p>// ClasseY implémente les méthodes définies dans InterfaceX</p> <pre> public class ClasseY implements InterfaceX { public void methodeDéclaréeDansInterfaceX () { // Code Selon ClasseY; // Implémentation (si l'éventuel code par défaut ne convient pas). } } </pre>

<p>Dépendance</p> 	<p>// ClasseY crée, modifie ou utilise un objet de la ClasseX lors de ses calculs.</p> <pre>public class ClasseY { public void methodeDeY (ClasseX unXreçu) { ClasseX unXcréé = new ClasseX(); unXcréé.uneMethodeDeX(); // Ici, on utilise unXcréé, unXreçu.autreMethodeDeX(); // ainsi que unXreçu } }</pre>
<p>Association unilatérale</p> 	<p>// Association unilatérale : l'objet ClasseY connaît l'objet ClasseX.</p> <pre>public class ClasseY { ClasseX linkToX; // Cette déclaration permet d'implémenter l'association. // prévoir: init dans le constructeur, setLinkToX (ClasseX), delLinkToX (), getLinkToX ()... }</pre>
<p>Association bilatérale</p> 	<p>// Association bilatérale : l'objet ClasseY connaît l'objet ClasseX et l'objet ClasseX connaît l'objet ClasseY.</p> <pre>public class ClasseY { ClasseX symLinkToX; // Cette déclaration permet d'implémenter l'association dans un sens // prévoir: init dans le constructeur, setSymLinkToX (ClasseX), delSymLinkToX (), getSymLinkToX (), ... } public class ClasseX { ClasseY symLinkToY; // Cette déclaration permet d'implémenter l'association dans l'autre sens // prévoir: init dans le constructeur, setSymLinkToY (ClasseY), delSymLinkToY (), getSymLinkToY (), ... }</pre>
<p>Agrégation</p> 	<p>// Agrégation, dit assemblage faible. L'objet ClasseY connaît les objets ClasseX qu'il réfère</p> <pre>public class ClasseY { ArrayList<ClasseX> desComposantsX; // Les composants existent hors de l'objet. // prévoir: init dans le constructeur, référencerComposantX (ClasseX), // estIIRéférencéComposantX (ClasseX), déRéférencerComposantX (ClasseX), ... }</pre>
<p>Composition</p> 	<p>// Composition, dite assemblage fort. L'objet ClasseY connaît les objets ClasseX qui le composent</p> <pre>public class ClasseY { ArrayList<ClasseX> mesComposantsX; // Les composants existent dans l'objet (créer 1 clone) // prévoir: init dans le constructeur, ajouterComposantX (ClasseX), supprimerComposantX (ClasseX), // estIIComposantX (ClasseX), ... }</pre>