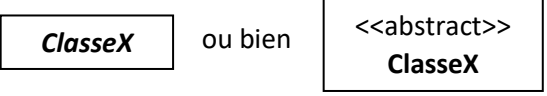
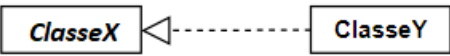
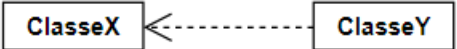


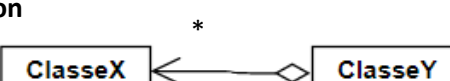


<p>Héritage</p> 	<p>// ClasseY <u>hérite</u> des attributs et méthodes de ClasseX avec leurs modes d'accès (public, private, protected). // ClasseY peut <u>redéfinir</u> les méthodes héritées de ClasseX définies virtual. // ClasseY peut <u>définir</u> de nouveaux attributs et méthodes.</p> <div> <pre>class ClasseX { public: virtual void méthode(){ // Code Selon ClasseX } };</pre> <pre>class ClasseY : public ClasseX { public: // Redéfinition de la méthode héritée virtual void méthode(){ // Code Selon ClasseY } };</pre> </div>
<p>Abstraction</p> 	<p>// ClasseX est une classe abstraite, c.a.d. avec au moins une méthode virtual pure (cf. définie par =0;) // ClasseX n'est pas instanciable.</p> <pre>class ClasseX { public: virtual void méthodeDéclaréeDansClasseX() = 0; //Juste déf/signature (pas de code) };</pre>
<p>Réalisation / Implémentation</p> 	<p>// ClasseY hérite de ClasseX abstraite, avec les mêmes propriétés d'accès (cf. public, private, protected) // et implémente les méthodes virtuelles pures. Instanciation possible de ClasseY.</p> <pre>class ClasseY : public ClasseX { public: void méthodeDéclaréeDansClasseX() { // Code Selon ClasseY } };</pre> <p>// Implémentation (on donne le code) dans ClasseY.</p>
<p>Dépendance</p> 	<p>// ClasseY crée, modifie ou utilise un objet de la ClasseX lors de ses calculs.</p> <pre>class ClasseY { public: void méthodeDeY (ClasseX& unXreçu) { ClasseX unXlocal; unXreçu.uneMéthodeDeX(); unXlocal.autreMéthodeDeX(); } };</pre> <p>// Ici on utilise unXreçu, // ainsi que unXlocal</p>

<p>Association unilatérale</p> 	<p>// Association unilatérale : l'objet ClasseY connaît l'objet ClasseX.</p> <pre>class ClasseY { ClasseX* linkToX; // Déclaration qui implémente l'association unilatérale // 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>class ClasseY { ClasseX* symLinkToX; // Déclaration qui implémente l'association dans un sens // prévoir: init dans le constructeur, setSymLinkToX (ClasseX*), delSymLinkToX (), getSymLinkToX ()... }; class ClasseX { ClasseY* symLinkToY; // Déclaration qui implémente 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>class ClasseY { list<ClasseX*> desComposants; // Les composants existent hors de l'objet. // prévoir: init dans le constructeur, référencerComposantX (ClasseX*), // estIIRéférencéComposantX (ClasseX*), déRéférencerComposantX (ClasseX*), ~ClasseY() ... };</pre>
<p>Composition</p> 	<p>// Composition, dite assemblage fort. L'objet ClasseY connaît les objets ClasseX qui le composent</p> <pre>class ClasseY { list<ClasseX> mesComposants; // Les composants existent dans l'objet // prévoir: init dans le constructeur, ajouterComposantX (ClasseX), supprimerComposantX (ClasseX), // estIIComposantX (ClasseX), ~ClasseY() };</pre>

