Changement d'environnement

Jusqu'à présent, nous avons utilisé l'environnement Geany pour sa simplicité de mise en œuvre, mais quand les projets grossissent, d'autres environnements sont plus adaptés. Pour garder une cohérence avec la suite des TP, nous allons utiliser l'environnement QtCreator.
Après avoir lancé QtCreator, créer un nouveau projet[1] (menus : "Fichier", "Nouveau fichier ou Projet"), puis choisir le modèle "Projet non Qt" et "Projet C++", choisir un nom et un emplacement et valider tout le reste.

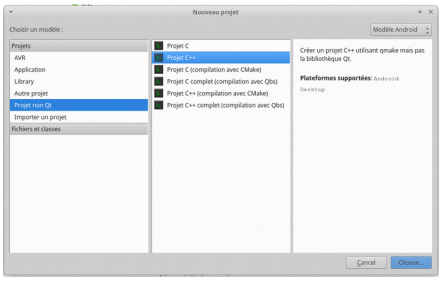

Figure 1 - Nouveau projet c++ non-Qt

1. Vérifier que le programme initial, "*Hello World*", fonctionne bien.
2. analyser comment est stocké ce projet sur le disque.
3. Créer une nouvelle classe c++ (menus : "Fichier", "Nouveau fichier ou Projet"), puis choisir le modèle "C++" et "Classe C++", lui donner un nom (**Position2D**), puis valider le reste. Remarquez ce qui s'est passé !
4. Reprendre la classe avec le code développé au TP1 en faisant des "copier/coller" des bonnes choses aux bons endroits (déclaration dans position2D.h et implémentation dans position2D.cpp), remplacer le main (main.cpp) par celui du TP1. Vérifier le fonctionnement.
5. Faire la même chose avec la classe **Cercle2D** du TP2, vérifier le bon fonctionnement, puis sauver ce projet.
6. En faire une archive (.zip) qui sera sauvée sur votre support personnel et jointe au compte rendu. (cette archive doit permettre de recompiler et d'exécuter le programme sur une autre machine ayant QtCreator installé).

Classe Carre2D

déclarer une classe Carre2D représentant un carré du plan, centré sur une Position2D, avec:

1. un constructeur par défaut qui initialise le centre à (0, 0) et le coté à 1
2. un constructeur permettant de spécifier les deux coordonnées du centre et le coté (si le coté n'est pas spécifié, il sera égale à 1 - argument par défaut)
3. un constructeur permettant de spécifier la position du centre et le coté (si le coté n'est pas spécifié, il sera égale à 1 - argument par défaut)
4. une méthode affiche qui écrit sur le terminal le carré sous la forme [x, y, r]; ex: [`120, 230, 30]`
5. une méthode qui calcule l'aire du carré, prototype: `double Carre2D::aire();`
6. une méthode qui indique si le carré contient, ou pas, une position passée en argument.

1 La présentation, l'ordre et l'intitulé des menus peuvent changer en fonction de la version de QtCreator

prototype: **`bool Carre2D::contient(Position2D);`**

7. la surcharge des opérateurs relationnels (> < ≥ ≤) qui comparent le carré à un autre, passé en paramètre - la comparaison se fera sur leurs aires respectives.
   prototype: **`bool Carre2D::operator >(Carre2D);`**

Marche à suivre :

comme précédemment, déclarer les méthodes de la classe les unes après les autres, en vérifiant au fur et à mesure leur bon fonctionnement dans le programme principal.

Après avoir dupliqué votre programme (enregistrer sous…) dans un autre fichier, remplacer votre programme principal par le suivant :

```
int main(int argc, char **argv)
{
Position2D p1(100, 100), p2;
Carre2D c1, c2(200, 100, 10), c3(-200, 100), c4(p1, 30), c5(p1, -20);

// affichage des deux positions et des cinq cercles initiales
cout<<"p1:";p1.affiche();
cout<<"p2:";p2.affiche();
cout<<"c1:";c1.affiche();
cout<<"c2:";c2.affiche();
cout<<"c3:";c3.affiche();
cout<<"c4:";c4.affiche();
cout<<"c5:";c5.affiche();

cout<<endl;
if (c2.contient(p1))cout<<"c2 contient p1";
else cout<<"c2 ne contient pas p1";
cout<<endl;
if (c1<=c3 && c4>c2) cout<<"la condition est vérifiée !"<<endl;
cout<<endl;

cout<<"après traitement"<<endl;
cout<<"p1:";p1.affiche();
cout<<"p2:";p2.affiche();
cout<<"c1:";c1.affiche();
cout<<"c2:";c2.affiche();
cout<<"c3:";c3.affiche();
cout<<"c4:";c4.affiche();
cout<<"c5:";c5.affiche();

return 0;
}
```

Si vous avez bien respecté le cahier des charges, vous devez impérativement obtenir :

```
p1:(100,100)
p2:(0,0)
c1:(0,0,1)
c2:[200,100,10)
c3:[0,100,1]
c4:[100,100,30]
c5:[100,100,1]

c2 ne contient pas p1
la condition est vérifiée !

après traitement
p1:(100,100)
p2:(0,0)
c1:[0,0,1]
c2:[200,100,10]
c3:[0,100,1]
c4:[100,100,30]
c5:[100,100,1]
```

Représentation UML

En UML, les classes ci-dessus seraient représentées comme suit. La relation de composition représente le verbe "**contient**" : un Cercle2D **contient** une Position2D, un Carre2D **contient** une Position2D. Notez bien la forme et le sens des flèches.

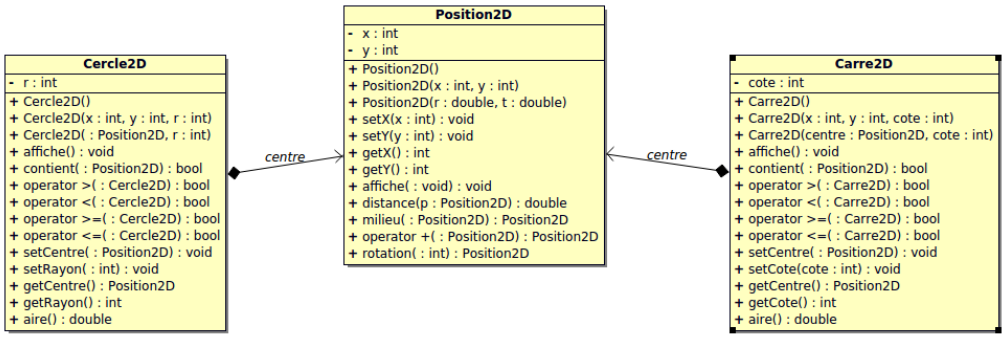

Figure 2 - Représentation UML des classes

Classe Rectangle2D et Hexagone2D

De la même manière, créer et vérifier une classe Rectangle2D représentant un rectangle centré sur une Position2D ainsi qu'une classe Hexagone2D représentant un hexagone du plan.

Chaque binôme devra envoyer par email une archive (.zip) contenant le ou les fichiers constituant le programme complet du TP, et ce, impérativement avant le jour de la séance suivante.

Si le programme qui ne se compile pas (présence d'erreurs de syntaxe) la note de compte rendue ne sera donnée que sur la moitié des points prévus.

Si vous n'avez pas pu terminer le programme durant la séance, charge à vous de le terminer durant la semaine.

La présentation et les commentaires dans les programmes seront très appréciés.

un attribut nom pour chacun des objets des classes ci-dessus - héritage

classe FeuTricolore