# 3 IF 2014/2015

# Andrea ACCARDO

# Cyril CANETE

# Compte Rendu

# TP C++ n°4 : Héritage, polymorphisme

## Document de conception

### Spécification complètes des classes

Nôtre application a été conçue pour permettre à l’utilisateur de créer, manipuler, modifier et supprimer des objets géométriques simples (rectangles, cercles, lignes et poly-lignes). Il est possible d’utiliser une série de commandes pour réaliser autant d’actions :

1. C nom X Y R : il crée un cercle de centre (X, Y) et de rayon R ;
2. R nom X1 Y1 X2 Y2 : il crée un rectangle ayant pour extrémité inferieure le point (X1, Y1) et pour supérieure (X2, Y2) ;
3. L nom X1 Y1 X2 Y2 : il crée une ligne passante pour les points (X1, Y1) et (X2, Y2) ;
4. PL nom X1 Y1 X2 Y2 … Xn Yn (n ≥ 1) : il crée un poly-ligne passant, dans l’ordre, pour tout point.
5. S nom X1 Y1 X2 Y2 : il crée une sélection rectangulaire ayant pour extrémité inferieure le point (X1, Y1) et pour supérieure (X2, Y2) ;
6. DELETE nom1 nom2 … nomn : il supprime tous les objets ayant le nom donné ( dans le cas où un nom soit invalide le commande n’a aucun effet) ;
7. MOVE nom δX δY : il déplace l’objet identifié de δX selon l’axe des abscisses et de δY selon l’axe des ordonnées.
8. LIST : il affiche tout objets existant (sauf les sélections) ;
9. UNDO : il annule la dernière action (sauf tout action concernant une sélection, qui sera ignorée) ;
10. REDO : il rétablit la dernière action annulée (sauf si pendant ce temps une autre action a été faite) ;
11. LOAD nomFichier : il charge depuis un fichier tous les actions contenu dedans (sauf les sélections) ;
12. SAVE nomFichier : il sauvegarde un fichier contenant tous les informations permettant de rétablir tous les objets existants ;
13. CLEAR : il supprime tous les objets ;
14. EXIT : il termine le programme ;

Ici on peut trouver tous les tests effectués et leurs relatives adresses :

|  |  |
| --- | --- |
| 1. *Description*: Création d’un cercle | Test\_Cercle :   1. run : Syntaxe correcte ; 2. Test\_InvalideCenter/run : Valeur X et/ou Y non valides ; 3. Test\_InvalideRadius/run : Valeur R non valide ; 4. Test\_NameMissing/run : Nom manquante. |
| 1. *Description*: Création d’une ligne | Test\_Line :   1. run : Syntaxe correcte ; 2. Test\_InvalidePoints/run : Valeurs X1, Y1, X2 et/ou Y2 non valides ; 3. Test\_NameMissing/run : Nom manquante. |
| 1. *Description*: Création d’un rectangle | Test\_Rectangle:   1. run : Syntaxe correcte ; 2. Test\_InvalidePoints/run : Valeurs X1, Y1, X2 et/ou Y2 non valides ; 3. Test\_NameMissing/run : Nom manquante. |
| 1. *Description*: Création d’un poly-ligne | Test\_Polyline:   1. run : Syntaxe correcte ; 2. Test\_InvalidePoints/run : Valeurs X1, Y1, X2, Y2, … Xn et/ou Yn non valides ; 3. Test\_NameMissing/run : Nom manquante. |
| 1. *Description :* Création d’une sélection | Test\_Selection:   1. run : Syntaxe correcte ; 2. Test\_InvalidePoints/run : Valeurs X1, Y1, X2 et/ou Y2 non valides ; 3. Test\_NameMissing/run : Nom manquante. |
| 1. *Description*: Affichage des objets existants | Test\_List:  run : Affichage correcte (ignore les sélections). |
| 1. *Description*: Suppression de plusieurs objets | Test\_Delete:   1. run : Syntaxe |
| 1. *Description*: Déplacement d’un objet. | Test\_Move:   1. run : Syntaxe correcte ; 2. Test\_InvalidePoints/run : Valeurs δX et/ou δY non valides ; 3. Test\_NameMissing/run : Nom manquante. |
| 1. *Description*: Annulation de la dernière commande | Test\_Undo:  run : annule correctement toute commande sauf celles qui concernant les sélection. |
| 1. *Description*: Rétablir la dernière commande annulée | Test\_Redo:  run : rétablit correctement la dernière commande sauf si après le UNDO on a exécuté un autre commande. |
| 1. *Description*: Suppression de tous les objets | Test\_Clear:  run : suppression de tous les objets (remise à zéro). |
| 1. *Description*: Sauvegarde d’un fichier | Test\_Save:   1. run : Syntaxe correcte ; 2. Test\_NameMissing/run : Nom manquante ; 3. Test\_ExistentFile/run : Fichier déjà existant. |
| 1. *Description*: Chargement d’un fichier | Test\_Load:   1. run : Syntaxe correcte ; 2. Test\_NameMissing/run : Nom manquante ; 3. Test\_FileMissing/run : Fichier inexistant. |

### Architecture globale de l’application

Nous avons ainsi structuré le logiciel :

1. Une classe *Page* constituée par :
   1. Un constructeur et un destructeur de défaut ;
   2. Des méthodes :
      1. *AjoutReferer (const string refererPage)*, qui ajout un référer a cette Page ;
      2. *Afficher (), qui affiche une page : son nom et le nombre de clique qui elle a eu ;*
      3. *GetHit (), qui renvoie l'attribut nombreHits ;*
      4. *InsererDot (ofstream fichier), qui remplit le fichier avec des arcs valus en format graphviz.*
   3. Des attributs :
      1. Un unsigned int indiquant le nombre de hits ;
      2. Un *dictionnaire\** contenant des *unsigned int* ;
      3. Une string contenant le nom de la page.
2. Une classe *Navigation*constituée par :
   1. Un constructeur et un destructeur de défaut ;
   2. Des méthodes :
      1. AjoutPage ( string nomPage, string refererPage ), qui prend deux strings en entrée (respectivement le nom de la page et le page de
      2. AfficheTopPage (), qui affiche les top 10 pages sans prendre paramètre en entrée.
      3. CreeDot (string), qui crée un fichier dot représentant Navigation en prenant le nom du fichier en paramètre ;
      4. comptageTopPage (), méthode privé qui calcule les top 10 pages sans prendre paramètre en entrée, et les renvoie comme un tableau de strings.
   3. Des attributs :
      1. Un *dictionnaire\** contenant des *Pages*.
3. Une classe *LigneLog* constituée par :
   1. Un constructeur qui prend en paramètre une string (que doit avoir le bonne format), un constructeur et destructeur de défaut ;
   2. Des méthodes :
      1. Afficher (), qui affiche toutes les caractéristiques de la LigneLog ;
      2. TestExtension (), qui teste si la LigneLog contient des fichiers image, JavaScript ou CSS et renvoie faux si oui, vrai sinon ;
      3. TestOptions (vector<string>\*), qui prend en paramètre un vecteur et teste si la LigneLog respecte les options contenues dans le vecteur ;
      4. GetReferer ( ), qui return une string contenant le referer ;
      5. GetUrl ( ) qui return une string contenant l’Url Demande.
   3. Des attributs :
      1. Une string ipClient ;
      2. Une string userLogname ;
      3. Une string authenticatedUser ;
      4. Une Date\*\* date ;
      5. Une Requete\*\* request ;
      6. Une int returnCode ;
      7. Une string dataQuantity ;
      8. Une string referer ;
      9. Une string navigateur.
4. Une interface \*\*Data contenant :
   1. La structure Data qui est constituée par :
      1. Un unsigned int Jour;
      2. Une string Mois;
      3. Un unsigned int Annee;
      4. Un unsigned int Heure;
      5. Un unsigned int Minute;
      6. Un unsigned int Seconde;
      7. Une string Fuseau;
   2. La structure Request qui est constituée par :
      1. Une string Action;
      2. Une string UrlDemande;
      3. Une string Protocole;

### \*Structures de données

Navigation : Nous avons choisi un arbre qui stocke des pages en utilisant comme clé leur nom.

Nous avons opté pour cette clé car la recherche par nom et beaucoup plus employée (à chaque lecture de ligne) alors que le max sera calculé une seule fois.

Page : Nous avons choisi un arbre qui stocke des entières (nombre hits) en utilisant comme clé leur référer. Nous avons opté pour cette clé car, dans notre cas, c’était l’unique possible pour identifier le nombre de hits.

LigneLog : Nous avons choisi un vecteur qui stocke des strings (options). Nous avons opté pour cette structure de données car est la plus adapte lorsqu’on ne connais pas la dimension d’une liste d’objet, vu qu’elle modifie sa taille automatiquement.