

# PolyWinSdl

## Contenu

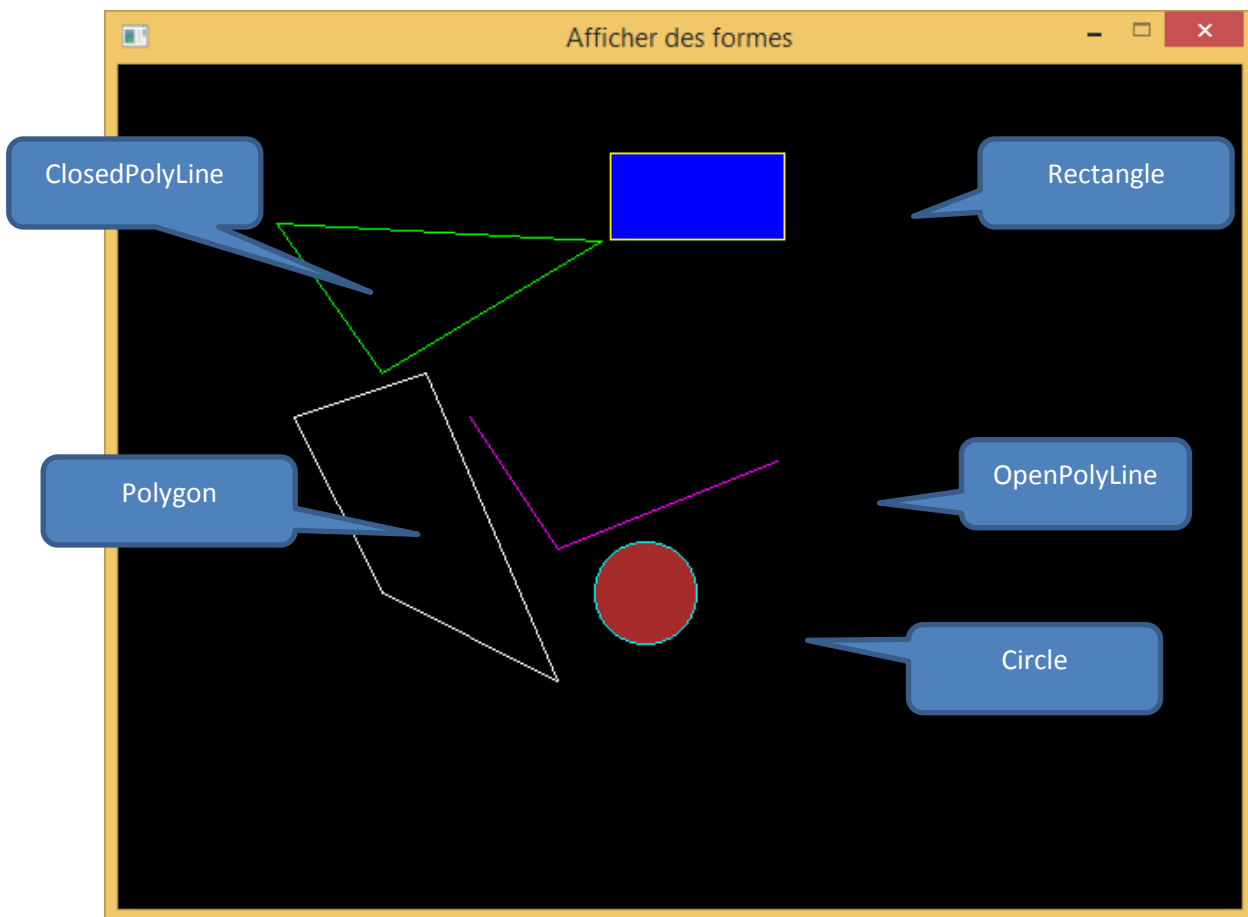
|                                  |   |
|----------------------------------|---|
| PolyWinSdl .....                 | 1 |
| Avant de commencer .....         | 2 |
| L'application PolyWinSDL.....    | 2 |
| Architecture de PolyWinSDL ..... | 3 |
| Diagramme de classes.....        | 4 |
| Projet Shape .....               | 5 |
| Classe Shape.....                | 5 |
| Classe OpenPolyLine .....        | 5 |
| Classe ClosedPolyLine .....      | 5 |
| Classe Rectangle.....            | 5 |
| Classe Circle.....               | 5 |
| Classe ShapeFactory.....         | 5 |
| Classe Color .....               | 5 |
| Classe Point .....               | 5 |
| Projet Animation Render .....    | 6 |
| WindowRender .....               | 6 |
| WindowEvent.....                 | 6 |
| Projet AppSDL .....              | 6 |
| Par où commencer ? .....         | 7 |

## Avant de commencer

1. Installer la librairie **SDL** et **SDL\_DRAW**
  - voir [GitHub\440-H15\Documents\ProceduresEtNormes\installer sdl.pdf](#)
2. Voir l'exemple d'utilisation de la librairie SDL
  - Voir [GitHub\440-H15\Documents\CodeExemples\](#)

## L'application PolyWinSDL

- PolyWinSDL permet d'afficher 5 types de formes.
- Il n'est pas demandé de gérer le remplissage des formes **OpenPolyLine**, **ClosedPolyLine** et **Polygon**.



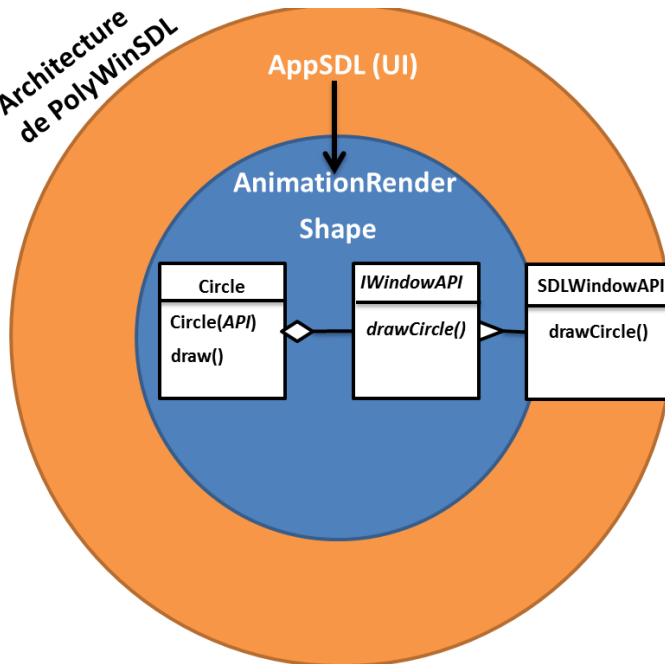
Le code contient un *main* qui crée des formes et affiche l'exemple ci-dessus. Cet exemple devrait s'afficher une fois le laboratoire complété.

## Architecture de PolyWinSDL

La solution contient 4 projets :

- AppSDL
- Shape
- AnimationRender
- UnitTests

Architecture  
de PolyWinSDL



**AppSDL** est le projet exécutable qui dépend de la librairie SDL.

**Shape** est le projet qui contient l'ensemble des formes.

**AnimationRender** est le projet qui gère l'affichage et les événements.

**AppSDL** dépend de **AnimationRender** et **Shape**, mais **AnimationRender** et **Shape** ne peuvent dépendre de **AppSDL**.

**Problème:** **Shape** et **AnimationRender** doivent communiquer avec **AppSDL** (par exemple **Shape** pour demander d'afficher une forme à l'écran).

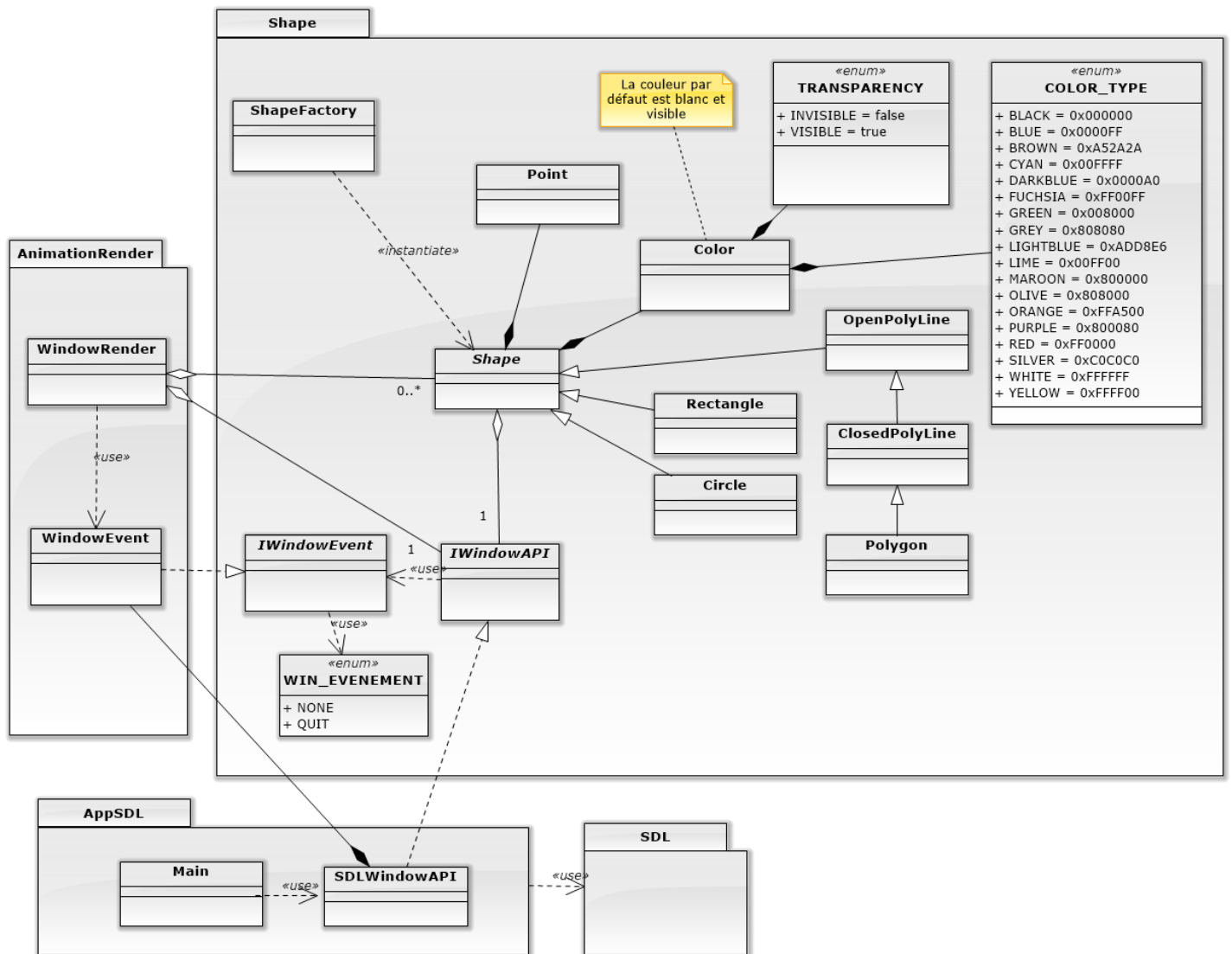
**Solution:** Créer une interface abstraite qui sert de pont pour la communication avec **AppSDL**.

**Exemple de flux pour dessiner un cercle:**

- **AppSDL** crée une instance de **SDLWindowAPI** (qui hérite de **IWindowAPI**)
- L'instance créée de **SDLWindowAPI** est injectée à **Circle** par son constructeur.
- **Circle** (implémenté dans **Shape**) peut se dessiner dans **AppSDL** en appelant **drawCircle** de **IWindowAPI**.

Ainsi, la dépendance directe avec **AppSDL** est coupée. Il est alors possible de tester unitairement **AnimationRender** et **Shape**.

## Diagramme de classes



# Projet Shape

## Classe Shape

- Représente toutes les formes possibles à afficher.
- Est une classe abstraite (on ne peut l'instancier).
- Voir le diagramme de classes pour l'héritage entre les formes.

### Attributs

- **IWindowAPI \* windowAPI** : L'interface à utiliser pour l'affichage.
- **vector<Point> point** : Une forme est composée de points.
- **Color lineColor** : Couleur du contour de la forme.
- **Color fillColor** : Couleur de la forme.

### Méthodes

- **draw** : précise la couleur à utiliser pour dessiner une forme (en appelant **setDrawingColor** de **IWindowAPI**) et la dessine (en appelant **drawLines** de **IWindowAPI**). Cette méthode peut être redéfinie selon la forme à afficher.
- **setLineColor / getLineColor**
- **setFillColor / getFillColor**
- **getPoint(un index)** : retourne un point du vecteur de points.
- **getNumberOfPoints()** : retourne le nombre de points dans une forme.
- **add** : ajout d'un point dans le vecteur de points

## Classe OpenPolyLine

Représente une série de lignes connectées ensemble.

## Classe ClosedPolyLine

Représente une série de lignes connectées ensemble dont le dernier point vient connecter le premier point.

## Classe Rectangle

Représente un rectangle qui s'affiche à partir des coordonnées du coin supérieur gauche. Il contient une longueur et une hauteur.

## Classe Circle

Représente un cercle qui s'affichant à partir de la coordonnée du centre et d'un rayon.

## Classe ShapeFactory

Est une fabrique de formes (*shape*).

## Classe Color

Classe déjà codée.

## Classe Point

## Projet Animation Render

### WindowRender

- C'est la classe qui est responsable de gérer l'affichage des « shapes ».
- Contient une boucle qui exécute les actions suivantes (dans cet ordre) :
  - Gestion des événements (deux événements possibles **NONE** ou **QUIT**)
  - Gestion de l'affichage
    - Efface l'écran
    - Dessine les formes
    - Affiche sur l'écran les formes dessinées

#### Attributs

- **IWindowAPI \* windowAPI** : L'interface à utiliser pour l'affichage.
- **vector <Shape\*> shapes** : Vecteur contenant l'ensemble des formes à afficher.

#### Méthodes

- **attach** : Ajoute une forme (**shape**) au vecteur **shapes**.
- **render**: C'est la boucle pour le rendu. Elle affiche les formes (appelle la méthode **draw** de chacune des formes du vecteur **shapes**). L'évènement **QUIT** permet de sortir de la boucle.
- **putOnTop** : Mets en premier plan une forme. Exemple :



La signature de la méthode est : **void putOnTop(Shape & p);**

Exemple d'appel : **windowRender->putOnTop(monRectangle);**

### WindowEvent

- Représente l'un des deux événements possibles sur une fenêtre (**NONE** : représente aucun événement ou **QUIT** : fermeture de la fenêtre). Classe déjà codée.
- Hérite de IWindowEvent.

## Projet AppSDL

### Main

- Programme exécutable.
- Dépendance avec la librairie graphique SDL.
- Création des formes (shape) et exécution du rendu (AnimationRender).
- Les exceptions sont capturées dans le main. Afin d'afficher le message d'erreur des exceptions dans la console, modifier les propriétés du projet **SDLApp**:

Éditeurs de liens / Système / Sous-système et choisir Console (/SUBSYSTEM:CONSOLE)

### SDLWindowAPI

- Hérite de **IWindowAPI**
- C'est le lien avec la librairie SDL. Possède toutes les méthodes permettant de dessiner et d'afficher des formes ainsi que de gérer les événements de la fenêtre.

## Par où commencer ?

### Par le projet de tests !

Ordre suggéré :

- A. Voir le code de **FakeWindowAPI** (hérite de **IWindowAPI**).
- B. Faire passer les tests du dossier **ShapeTests**
  - 1. pointTests (à compléter)
  - 2. openPolyLineTests
  - 3. rectangleTests
  - 4. circleTests
  - 5. closedPolyLineTests
  - 6. polygonTests
  - 7. shapeTests
- C. Faire passer les tests du dossier **AnimationRenderTests**
- D. Faire passer les tests du dossier **ShapeFactoryTests**