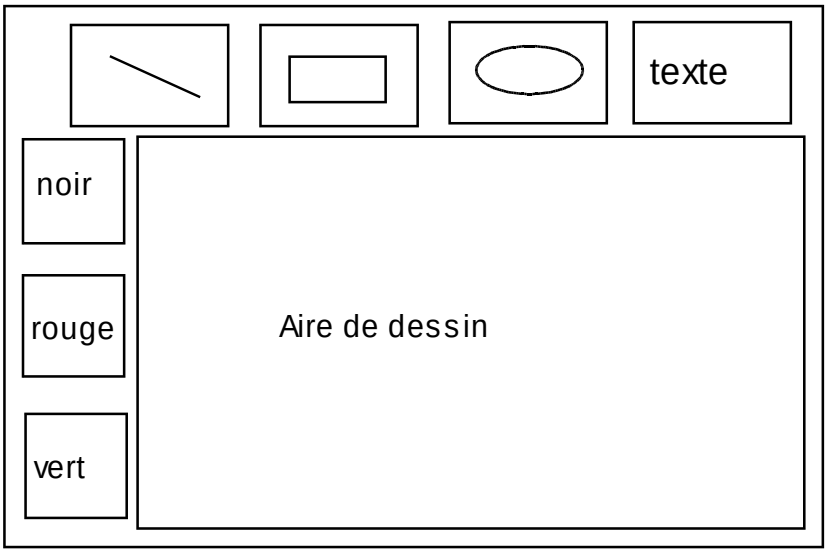
ED Notions orientés objets pour les IHL

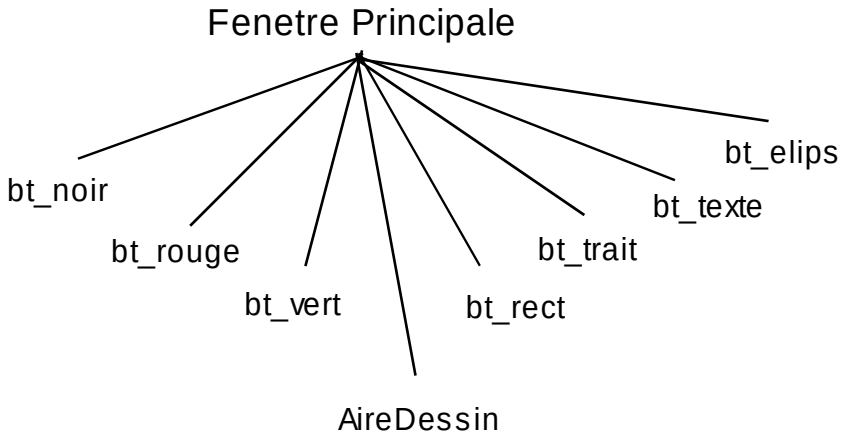
Arborescence des composants graphiques, notions orientées objets
exercice 1 : arborescence d'un interface graphique

But de l'Exercice
Montrer que construire un interface c'est "mettre des boîtes dans des boîtes". Que ces boîtes s'appellent des widgets, des fenêtres, etc. peu importe. Cette architecture a le mérite de bien ranger les choses. Elle est aussi utile pour personnaliser l'interface par un utilisateur (par exemple sous Xt/Motif). La fin de l'exercice a pour but de mettre en évidence des termes français très utilisés dans les IHM.

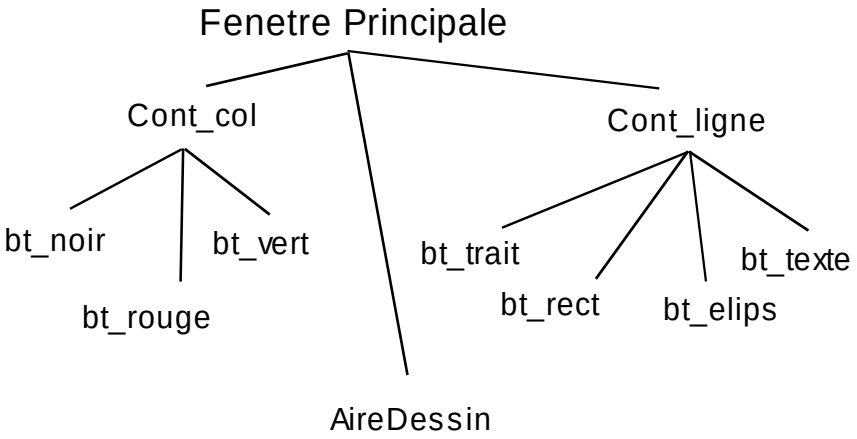
On se propose de créer l'interface d'un éditeur de dessin de la manière suivante :



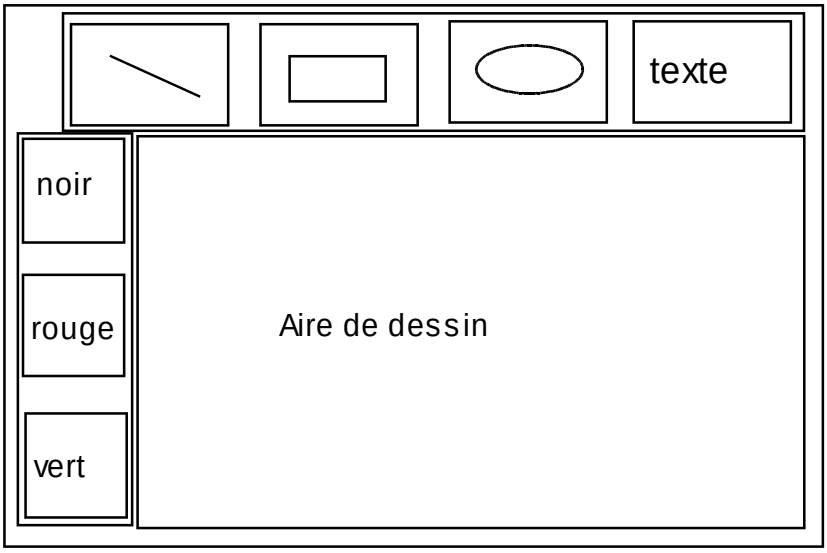
1°) Définir l'arborescence des éléments graphiques de cet interface. Que faut il ajouter pour gérer certains choix sous forme de boutons radios : on pourra penser à faire des regroupements.
Il faut déterminer les différents objets graphiques qui apparaissent dans l'interface. Essentiellement il y a un objet AireDeDessin et des boutons poussoir. La fenêtre de l'application est le dernier élément graphique. Ces objets sont rangés sous forme arborescente :



Avec une telle solution tout est placé au même niveau. Comme le suggère l'énoncé il faut penser à faire des regroupements. Par exemple regrouper les couleurs dans un objet qui dispose ses objets contenus en colonne et les différentes formes de dessins dans un objet qui dispose ses objets contenus en ligne. On obtient une arborescence :



correspondant aux objets emboîter de la manière suivante :



Insistons que ce rangement n'est pas seulement la pour "faire joli" et avoir des choses bien rangées. Il est indispensable de le connaître au moins en partie si un utilisateur veut personnaliser l'environnement de cette application (changer les libellés pour les mettre en anglais par exemple). Il est indispensable de le connaître exactement pour un programmeur car il représente l'essentiel de ce qui devra être codé.

2°) Que manque t il à ce programme pour en faire un logiciel acceptable ? Que proposez vous pour remédier à ces manques ?

On peut penser à développer les propositions déjà existantes : mettre plus de boutons couleurs. On peut envisager d'utiliser un éditeur de couleurs mais dans ce cas il sera appeler et s'affichera dans une fenêtre non modale. Mêmes remarques pour les formes de dessins. Il faut aussi concevoir que l'aire de dessin n'est pas directement liée à une application Paint mais que l'application permet de lancer plusieurs aires de dessin et permmetre des possages de l'une à l'autre par copier coller par exemple. Il faudra alors développer des outils permettant de faire du Couper/Copier/Coller entre aire de dessin.

Dans le Paint proposé par l'énoncé, il manque de charger, sauvegarder les dessins effectués, Quitter l'application, Proposez des tailles et des fontes différentes etc. bref une foule de choses. Certaines peuvent être obtenues par des menus déroulant, activés par des (cascade) boutons dans une (la) barre de menus. D'autres particularités peuvent être obtenues par des menus jaillissants (pop-up).

Exercice 2 : Ensemble de classes graphiques

But de l'exercice

Faire découvrir ce qu'il est nécessaire d'avoir dans tout système graphique avancé c'est à dire possédant un système de classes d'objets graphiques. Le système proposé est un ensemble de classes orientées objets. Bien mettre en évidence que si certaines codes sont entièrement donnés par ce système de classes (faire apparaître le bouton enfoncé lorsqu'on appuie dessus par exemple), un système graphique avancé doit aussi fournir des mécanismes permettant de lancer du code (que ce bouton lance la sauvegarde d'un fichier ou permette de quitter l'application sera implanté par le programmeur). Ranger ensuite ces classes sous forme arborescence en utilisant l'héritage (orientation objet).
Indiquer la différence entre cette arbre et l'arbre de l'exercice précédent. Peut être faut il commencer par cette dernière question ?!

On veut créer une bibliothèque de classes d'objets graphiques permettant de développer des interfaces.

1°) Définir les classes d'"objets de base" en précisant les données et méthodes contenues dans ces classes et ranger ces classes dans une arborescence.

Ces classes sont les classes de libellé, bouton poussoir, de fenêtre de saisie de texte, de boite de liste, barre de défilement, échelle ou jauge, etc. Elles sont appelées classes d'objets de base dans cet exercice mais on appelle ces classes, les classes de contrôle dans le monde Windows et Macintosh et classes Primitive dans le monde Motif.

Pour chaque classe on peut envisager les données et méthodes suivantes :

classe libellé
    la chaîne à afficher

sa font associée
la couleur d'affichage
les coordonnées x et y de son affichage par rapport à son contenant
méthodes internes
dessiner les parties qui ont été masquées lorsqu'elles passent en avant plan.
méthodes interfaces
A priori cette classe ne sert qu'à afficher du texte et ne possède pas de méthodes d'interface
fin classe

classe bouton poussoir
la chaîne à afficher
sa font associée
la couleur d'affichage
les coordonnées x et y de son affichage par rapport à son contenant
méthodes internes
dessiner les parties qui ont été masquées lorsqu'elles repassent en avant plan.
méthodes permettant de faire paraître le bouton enfoncé ou relâché
méthodes interfaces
mécanisme permettant de lancer des procédures du programmeur lorsqu'on appuie sur le bouton.
fin classe

classe zone de texte
la chaîne à afficher
sa font associée
la couleur d'affichage
les coordonnées x et y de son affichage par rapport à son contenant
Nombre de lignes et de colonnes de l'objet
méthodes internes
dessiner les parties qui ont été masquées lorsqu'elles repassent en avant plan.
mécanisme de copier/couper-coller par l'intermédiaire d'un presse papier
méthodes interfaces
mécanisme permettant de saisir et d'afficher le texte tapé par l'utilisateur
fin classe

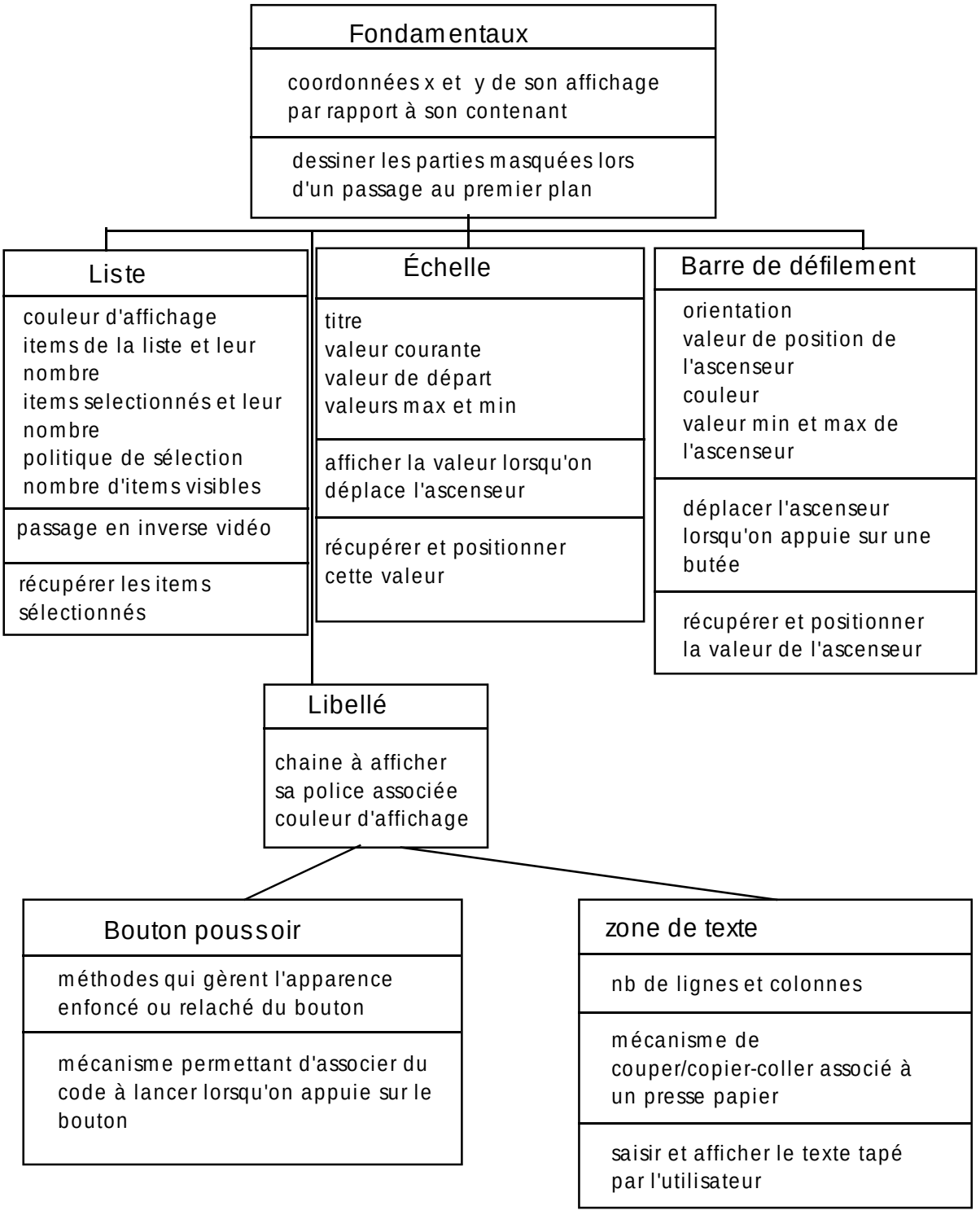
classe de boite de liste
la couleur d'affichage
les coordonnées x et y de son affichage par rapport à son contenant
Les items de la liste
Nombre d'items dans la liste
Les items sélectionnés de la liste
Nombre d'items sélectionnés dans la liste
Nombre d'items visibles
la politique de sélection (peut on sélectionner plusieurs items, un seul, par défilement ou pas, etc.)
méthodes internes
dessiner les parties qui ont été masquées lorsqu'elles repassent en avant plan.

passer en inverse vidéo les items sélectionnés
méthodes interfaces
récupérer les items sélectionnés
mécanisme permettant de lancer du code lors d'un double clic sur un item.
fin classe

classe barre de défilement
l'orientation (horizontale ou verticale)
la position d'ascenseur
la couleur
les coordonnées x et y de son affichage par rapport à son contenant
la valeur max et min de l'ascenseur
méthodes internes
dessiner les parties qui ont été masquées lorsqu'elles repassent en avant plan.
déplacer l'ascenseur lorsqu'on appuie sur les butées
méthodes publiques
être sensible à l'appui sur une butée
récupérer et positionner la valeur de l'ascenseur
fin classe

classe échelle ou jauge
les coordonnées x et y de son affichage par rapport à son contenant
le titre
la valeur courante
la valeur de départ
les valeurs max et min
méthodes internes
dessiner les parties qui ont été masquées lorsqu'elles repassent en avant plan.
afficher la valeur lorsqu'on déplace l'ascenseur
méthodes publiques
récupérer cette valeur, la positionner
fin classe

Au vue des différentes valeurs et méthodes de ces classes, on peut les ranger sous forme arborescente. Il faut bien voir que ceci est un exercice d'école et que les implantations véritables (MFC, Motif, Java, Macintosh, etc.) ne suivent pas à la lettre cette arborescence (mais s'en inspire fortement)



2°) Définir des classes d'objets conteneurs (i.e. objets qui peuvent contenir d'autres objets) et des classes d'"objets complexes" (i.e. objets formés à partir d'autres objets). Ranger ces classes dans une arborescence.

Concernant les classes d'objets conteneurs, on peut mettre en évidence au monis deux classes :

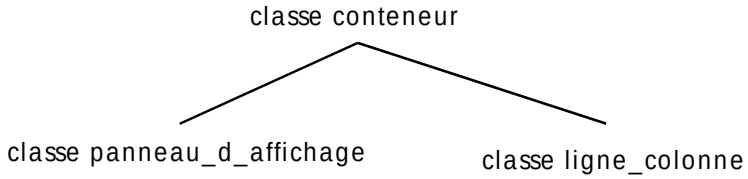
- classe (panneau_d_affichage) des objets qui n'imposent pas la position des objets graphiques contenus

- classe (ligne_colonne) des objets qui imposent de tels positionnements (par exemple en ligne et en colonne.

Classes de boites de dialogues, de fenêtre principale qui seront habillées (par le système cf. Windows, par un WM cf. X Window), de menus

réutilisabilité grace à l'héritage

3°) Construire l'arborescence d'un tel système de classes.
Les données et méthodes communes à plusieurs classes doivent être mises dans une classe de base dont on dérive. Par exemple liste_des_objets_contenus donnée apparaissant dans les deux classes panneau_d_affichage et ligne_colonne doit être mise dans une classe (de base) et dériver ces deux classes de cette classe de base. On a par exemple :



Remarquer qu'on peut enrichir ce graphe en ajoutant certaines classes. Par exemple la classes des menus est une classe dérivée de la classe ligne_colonne.
Toutes les classes contiennent les coordonnées x et y de l'objet par rapport a son objet contenant. Donc toutes les classes vont dériver d'une classe Noyau contenant ces données. Cela assure une cohérence de l'ensemble (un seul arbre) utile par exemple si on veut faire du polymorphisme en C++ (cf. visual C++)

4°) Quelles différences y a t-il entre cette arborescence et celle de l'exercice précédent ?
Dans cet exercice l'arborescence porte sur des classes et la notion être "fils de" dans l'arbre signifie "hérite de" au sens des notions orientées objets. Dans l'exercice précédent, l'arborescence portait sur des instances de ces classes et être "fils de" dans l'arbre de l'exercice précédent signifie "être à l'intérieur de". L'arbre de classes décrit dans cet exercice est en général un ensemble donné ou vendu par une API et est donc statique. L'arbre des instances de l'exercice précédent change d'une interface graphique à une autre et donc d'un programme à un autre.