# UEO15 – Programmation objet

***Durée 2H*** *-Documents autorisés-*

**Cours : interfaces graphiques (3 points)**

1. Expliquer comment créer le lien entre la partie visuelle et la partie fonctionnelle d'une interface graphique.
2. Illustrer ce principe avec un exemple simple utilisant des classes anonymes.

**Exercice 1 : réseau (7 points)**

On cherche à produire un éditeur de figures coopératif. Chaque intervenant est donc sur sa propre machine et édite des figures sur un dessin global partagé.
On suppose que l'on a une architecture client/serveur (**une** application **serveur** et ***n*** applications **clientes)**.

1. On ne veut pas que les figures soient présentes en double dans notre collection. Quelle structure de données faut-il utiliser ?
2. Le serveur possède une liste des clients connectés, donner la déclaration et l'instanciation de cette structure de données.
3. Initialisation
    a. Le serveur créé et initialise une collection (à définir) de figures puis attend les demandes de connexion des clients.
    b. L'application cliente établit une connexion avec le serveur.
    c. Donner le code correspondant à chacune de ces étapes.
4. Echange des données
    a. Le serveur lit les figures ajoutées par chacun des clients puis constitue un nouvel ensemble qu'il envoie vers chacun des clients.
    b. Le client envoie sa propre liste de figures vers le serveur puis récupère la liste complète depuis le serveur et l'affiche.
    c. Donner le code correspondant à chacune de ces étapes

**Exercice 2 : thread (5 points)**

On considère que plusieurs threads peuvent ajouter des figures dans une même liste appelée *listeFig*. Pour cela ils utilisent la méthode *public void add(Figure f).*

1. Donner la déclaration et l'instanciation de *listFig.*
2. Donner le code de la méthode *add()*
3. Comment peut-on éviter qu'il y ait des conflits entre les différents threads lors de l'ajout des figures dans la liste ?
4. On ajoute un thread donc la fonction est de parcourir la liste des figures et de supprimer les figures trop proches. Coder la méthode *filtrer()* qui filtre la liste et renvoie le nombre d'éléments supprimés.

**Exercice 3 : figure récursive (5 points)**

On souhaite définir la notion de figure récursive (type fractale). Une figure récursive est définie par un ensemble de figures et un nombre de répétitions (*nb*). On affiche une figure récursive en affichant les figures qui la composent à différents niveau de résolution (*nb*). On considère pour cela que l'on dispose d'une méthode *réduction()* qui prend en paramètre une figure et renvoie sa réduction à un niveau de résolution juste inférieur.

1. Donner la place de la classe *figureRecursive* dans la hiérarchie des figures.
2. Donner la déclaration des attributs de cette classe.
3. Donner le code de la méthode *public void paint(Graphics g)* de cette classe qui affiche la figure récursive sur les *nb* niveaux.

# UEO15 – Programmation objet

***Durée 2H*** *-Documents autorisés-*

**Cours : interfaces graphiques (3 points)**

1. Expliquer comment créer le lien entre la partie visuelle et la partie fonctionnelle d'une interface graphique.
2. Illustrer ce principe avec un exemple simple utilisant des classes anonymes.

**Exercice 1 : réseau (7 points)**

On cherche à produire un éditeur de figures coopératif. Chaque intervenant est donc sur sa propre machine et édite des figures sur un dessin global partagé.
On suppose que l'on a une architecture client/serveur ( **une** application **serveur** et ***n*** applications **clientes)**.

1. On ne veut pas que les figures soient présentes en double dans notre collection. Quelle structure de données faut-il utiliser ?
2. Le serveur possède une liste des clients connectés, donner la déclaration et l'instanciation de cette structure de données.
3. Initialisation
    a. Le serveur créé et initialise une collection (à définir) de figures puis attend les demandes de connexion des clients.
    b. L'application cliente établit une connexion avec le serveur.
    c. Donner le code correspondant à chacune de ces étapes.
4. Echange des données
    a. Le serveur lit les figures ajoutées par chacun des clients puis constitue un nouvel ensemble qu'il envoie vers chacun des clients.
    b. Le client envoie sa propre liste de figures vers le serveur puis récupère la liste complète depuis le serveur et l'affiche.
    c. Donner le code correspondant à chacune de ces étapes

**Exercice 2 : thread (5 points)**

On considère que plusieurs threads peuvent ajouter des figures dans une même liste appelée *listeFig*. Pour cela ils utilisent la méthode *public void add(Figure f).*

1. Donner la déclaration et l'instanciation de *listFig.*
2. Donner le code de la méthode *add()*
3. Comment peut-on éviter qu'il y ait des conflits entre les différents threads lors de l'ajout des figures dans la liste ?
4. On ajoute un thread donc la fonction est de parcourir la liste des figures et de supprimer les figures trop proches. Coder la méthode *filtrer()* qui filtre la liste et renvoie le nombre d'éléments supprimés.

**Exercice 3 : figure récursive (5 points)**

On souhaite définir la notion de figure récursive (type fractale). Une figure récursive est définie par un ensemble de figures et un nombre de répétitions (*nb*). On affiche une figure récursive en affichant les figures qui la composent à différents niveau de résolution (*nb*). On considère pour cela que l'on dispose d'une méthode *réduction()* qui prend en paramètre une figure et renvoie sa réduction à un niveau de résolution juste inférieur.

1. Donner la place de la classe *figureRecursive* dans la hiérarchie des figures.
2. Donner la déclaration des attributs de cette classe.
3. Donner le code de la méthode *public void paint(Graphics g)* de cette classe qui affiche la figure récursive sur les *nb* niveaux.