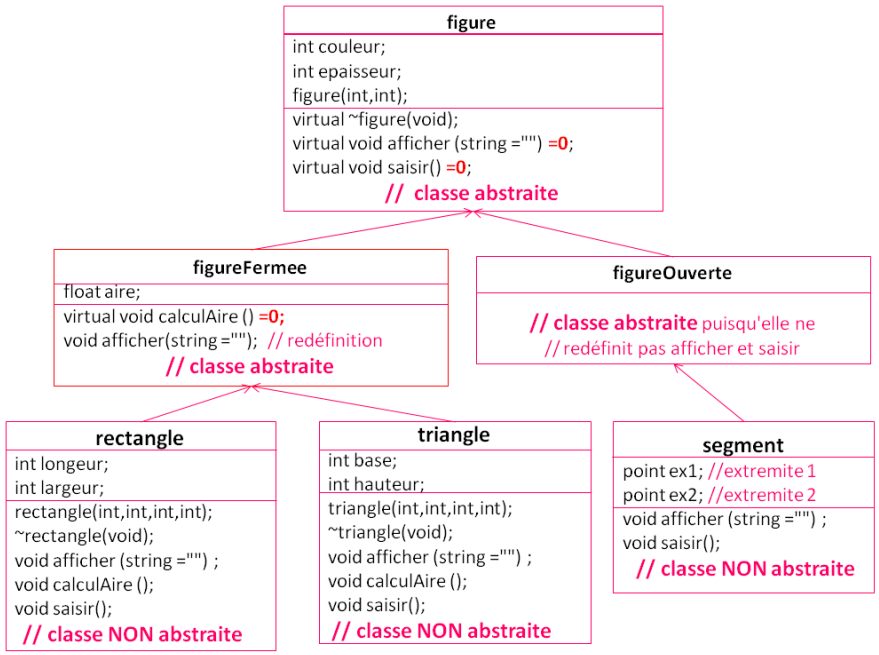


fig.h

```
class fig
{
protected:
    int couleur;
    int epaisseur;
public:
    fig(int =99,int=99);
    virtual ~fig(void);
    virtual void afficher(string ="") =0;
    virtual void saisir() =0;
};
```

fig.cpp

```
#include "fig.h"
fig::fig(int couleur, int epaisseur)
{
    this->couleur=couleur;
    this->epaisseur=epaisseur;
}
fig::~fig(void) { ... }
void fig::saisir()
{ //je peux donner une première définition à saisir
    cout<<"\n saisir couleur et epaisseur "<<endl;
    cin>>couleur>>epaisseur;
}
void fig::afficher(string msg)
{//je peux donner une première définition à afficher
    cout<<msg<<endl;
    cout<<"\n la couleur est "<<couleur<<endl;
    cout<<"\n l'epaisseur est "<<epaisseur<<endl;
}
```

figFermee.h

```
#include "fig.h"
class figFermee :public fig
{
protected:
    float aire;
public:
    figFermee(int =88,int=88);
    virtual ~figFermee(void);
    void afficher(string=""); //redéfinie
    // saisir n'est pas redéfinie
    virtual void calculAire() =0;
// pas de définition pour calculAire
};
```

figFermee.cpp

```
#include "figFermee.h"
figFermee::figFermee(int c, int e):fig(c,e)
{
    aire=0.0;
}
figFermee::~figFermee(void)
{
}
void figFermee::afficher(string msg)
{
    fig::afficher();
    cout<<"\n l'aire est "<<aire<<endl;
}
// pas de définition pour saisir et calculAire
```

rectangle.h

```
#pragma once
#include "figfermee.h"
class rectangle : public figFermee
{
protected:
    int longeur;
    int largeur;
public:
    rectangle(int =66, int=66,int=66,int=66);
    ~rectangle(void);
    void saisir();
    void calculAire();
    void afficher(string ="");
// toutes les méthodes héritées sont redéfinies
};
```

rectangle.cpp

```
#include "rectangle.h"
rectangle::rectangle(int c, int e,int l, int la):
        figFermee(c,e), longeur(l), largeur(la)
{
    calculAire();
}
void rectangle::calculAire()
{
    aire=longeur*largeur;
}
rectangle::~rectangle(void) { ... }
void rectangle::saisir()
{
    figFermee::saisir();
    cout<<"\n saisir longeur et largeur "<<endl;
    cin>>longeur;
    cin>>largeur;
    calculAire();
}
```

rectangle.cpp

```
void rectangle::afficher(string msg)
{
    cout<<msg<<endl;
    figFermee::afficher();
    cout<<"\n la longeur est "<<longeur<<endl;
    cout<<"\n la largeur est "<<largeur<<endl;
}
```

triangle.h

```
#pragma once
#include "figfermee.h"
class triangle : public figFermee
{
    int base;
    int hauteur;
public:
    triangle(int =55, int=55,int=55,int=55);
    ~triangle(void);
    void saisir();
    void calculAire();
    void afficher(string ="");
// toutes les méthodes héritées sont redéfinies
};
```

triangle.cpp

```
#include "triangle.h"
triangle::triangle(int c, int e,int l, int la):
        figFermee(c,e), base(l), hauteur(la)
{
    calculAire();
}
void triangle::calculAire()
{
    aire=(base*hauteur)/2;
}
triangle::~triangle(void) { ... }
void triangle::saisir()
{
    figFermee::saisir();
    cout<<"\n saisir base et hauteur "<<endl;
    cin>>base;
    cin>>hauteur;
    calculAire();
}
```

triangle.cpp

```
void triangle::afficher(string msg)
{
    cout<<msg<<endl;
    figFermee::afficher();
    cout<<"\n la base est "<<base<<endl;
    cout<<"\n la hauteur est "<<hauteur<<endl;
}
```

figOuverte.h

```
#pragma once
#include "fig.h"
class figOuverte :  public fig
{
public:
    figOuverte(int =77,int =77);
    ~figOuverte(void);
};
```

figOuverte.cpp

```
#include "figOuverte.h"
figOuverte::figOuverte(int c,int e):fig(c,e)
{
}
figOuverte::~figOuverte(void)
{
}
```

segment.h

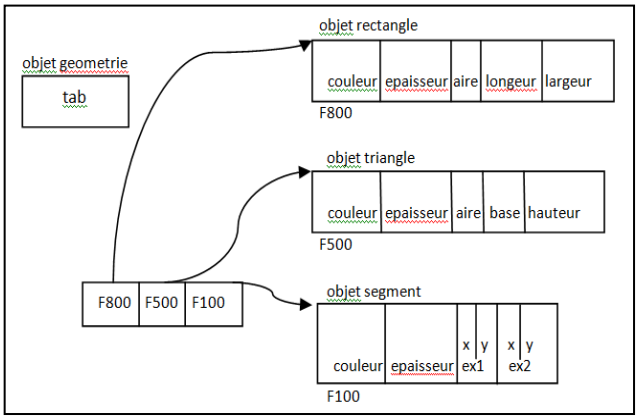
```
#pragma once
#include "figouverte.h"
#include"point.h"
class segment : public figOuverte
{
    point ex1;
    point ex2;
public:
    segment(int =44,int=44,int=44,int=44,int=44,int=44);
    ~segment(void);
    void afficher(string="");
    void saisir();
};
```

segment.cpp

```
#include "segment.h"
segment::segment(int c,int e,int x1, int y1, int x2, int y2):
        figOuverte(c,e), ex1(x1,y1), ex2(x2,y2)
{
}
segment::~segment(void) { ... }
void segment::afficher(string msg)
{
    cout<<msg<<endl;
    figOuverte::afficher();
    ex1.afficher();
    ex2.afficher();
}
void segment::saisir()
{
    figOuverte::saisir();
    ex1.saisir();
    ex2.saisir();
}
```

geometrie.h

```
class geometrie
{
    vector<fig*> tab;
public:
    geometrie(void);
    ~geometrie(void);
    void remplir();
    void afficher(string="");
    geometrie(const geometrie&);
    int taille(){ return tab.size();}
    void ajouter(rectangle, int =0);
    void ajouter(triangle, int=0);
    void ajouter(segment, int=0);
    void supprimer(int =0);
    void ajouter(fig*, int =0);
};
```



geometrie.cpp

```
geometrie::geometrie(void)
{//vide
}
geometrie::~geometrie(void)
{   for(int i=0; i<tab.size(); i++)
        delete tab[i]; // destructeur virtuel
    tab.clear();
}
void geometrie::afficher(string msg)
{
    cout<<msg<<endl;
    // affichage de adresses des objets
    for(int i=0; i<tab.size(); i++)
        cout<<tab[i]<<"  ";
    cout<<endl;
    // affichage des objets
    for(int i=0; i<tab.size(); i++)
        tab[i]->afficher();
}
```

```
void geometrie::remplir()
{   char rep;
    fig* q;
    int choix;
    do
    {   cout<<"\n taper1: rectangle, 2:triangle, 3:segment "<<endl;
        cin>>choix;
        if(choix==1) q=new rectangle();
        else if(choix==2) q=new triangle();
        else if(choix ==3) q=new segment();
        else break;
        q->saisir();
        tab.push_back(q);
        cout<<"\n rajouter ?"<<endl;
        cin>>rep;
    }
    while(rep=='o' || rep=='O');
}
```

```
// geometrie b=a;
geometrie::geometrie(const geometrie &w)
{
    fig *e;
    for(int i=0; i<w.tab.size(); i++)
    {
        if(typeid(*w.tab[i]) == typeid(rectangle))
            e=new rectangle(static_cast <const rectangle&> (*w.tab[i]));
        else if(typeid(*w.tab[i]) == typeid(triangle))
            e=new triangle(static_cast <const triangle&> (*w.tab[i]) );
        else if( typeid(*w.tab[i]) == typeid(segment))
            e=new segment (static_cast <const segment&> (*w.tab[i]));

        tab.push_back(e);
    }
}
```

```
void geometrie::ajouter(fig* q, int ind)
{
    tab.insert(tab.begin()+ind, q);
}
```

```
void geometrie::ajouter(rectangle q, int ind)
{   fig* e=new rectangle(q);
    tab.insert(tab.begin()+ind, e);
}
void geometrie::ajouter(triangle q, int ind)
{
    fig* e=new triangle(q);
    tab.insert(tab.begin()+ind, e);
}
void geometrie::ajouter(segment q, int ind)
{
    fig* e=new segment(q);
    tab.insert(tab.begin()+ind, e);
}
void geometrie::supprimer(int ind)
{
    delete tab[ind];//libérer l'objet
    tab.erase(tab.begin()+ind);
}
```