CORRECTION SESSION NORMALE PROGRAMMATION C++ INF 162 (2020)

Proposez Par : GROUPE GENIUS REPETITION

Par : Mr JoëL_YK, Kratos & Petnga Njemfa Steploic (Le Major)

EXERCICE 1 :

1- *) Encapsulation : C'est un processus consistant à combiner des membres de données et des fonctions dans une seule unité appelée classe.

*) Classe : C'est un modèle à partir duquel on peut créer des objets partageant des attributs et des méthodes communes.

*) Instance : C'est un objet avec un comportement et un état, tous deux définis par la classe.

*) Classe Abstraite : C'est une classe qui possède au moins une méthode déclarée virtuelle, dont le contenu n'a pas été défini.

*) Méthode abstraite : C'est une méthode qui possède une signature, mais pas de Corps.

2- La différence entre **la surcharge fonctionnelle** et **la surcharge de sélection** est que, la surcharge fonctionnelle est assurée par le compilateur, alors que la surcharge par sélection est assurée par le programmeur.

3- **L'intérêt du constructeur par défaut** est d'initialiser par défaut les attributs d'une classe lors de sa création.

4- **Polymorphisme :** C'est la capacité du système à choisir dynamiquement la méthode qui correspond au type de l'objet en cours de manipulation.

EXERCICE 2 :

```
#include <iostream>

using namespace std;

int main()
{
    int tab[10];
    int nbre;
```

```
    cout << "Veuillez saisir votre suite de nombres terminée par le marqueur
-1" << endl;
    cin >> nbre;
    if(nbre == -1){
        cout << "Votre suite est vide" <<endl;
        return 0;
    }
    else{
        int i = 0 , j,k,cpt=0;
        while (nbre!= - 1 && i!=10){
            tab[i] = nbre;
            i++;
            cin >>nbre;
        }
        for(j = 0; j<i; j++ ){
            int verifie = tab[j];
            for(k=0; k<i; k++){
                if(verifie == tab[k] && verifie!=10000){
                  cpt++;
                  tab[k] = 10000;
                }
                else if(verifie==10000){
                    break;
                }
            }
            if(verifie!=10000){
                cout << "Le nombre d'occurrences de " << verifie << " est : "
<< cpt <<endl;
                cpt = 0;
            }
        }
        return 0;
    }
}
```

PROBLEME

PARTIE A :

```
#ifndef POINT_H                    //FICHIER Point.h
#define POINT_H

class Point
{
    private:
        float x; //Abscissse
        float y; //Ordonnee
        static int nombre_points; //Nombre de points
    public:
        float getX(); //Accesseur en consultaion sur l'abscisse
        float getY(); //Accesseur en consultaion sur l'ordonnee
        void setX(float a); //Accesseur en modification sur l'abscisse
        void setY(float b); //Accesseur en modification sur l'ordonnee
        Point(float a, float b); //constructeur a deux arguments
        Point(Point &p); //Constructeur de recopie
        int operator==(const Point &l) const; //Operateur de comparaison
        Point(); //Constructeur par defaut
        ~Point(); //Destructeur
};

#endif // POINT_H
```

```
                                   //FICHIER Point.cpp
#include "Point.h"

//Nombres de points crées
int Point::nombre_points = 0;

//Accesseur en consultaion sur l'abscisse
float Point::getX() {
    return (*this).x;
```

```
}

//Accesseur en consultation sur l'ordonnée
float Point::getY() {
    return (*this).y;
}

//Accesseur en modification sur l'abscisse
void Point::setX(float a){
    this->x = a;
}

//Accesseur en modification sur l'ordonnée
void Point::setY(float b) {
    this->y = b;
}

//constructeur a deux arguments
Point::Point(float a , float b) {
    this->x = a;
    this->y = b;
    nombre_points ++;
}

//Operateur de comparaison
int Point::operator==(const Point &l) const {
    if((this->x == l.x) && (this->y == l.y)) return 1;
    return 0;
}

//Constructeur de recopie
Point::Point(Point &p) {
    this->x = p.x;
    this->y = p.y;
    nombre_points ++;
}

//Constructeur par défaut
Point::Point()
{
    nombre_points ++;
}

//Destructeur
Point::~Point()
```

```
{
   //Socrate
}
```

PARTIE B :

```
#ifndef SEGMENT_H                    //FICHIER Segment.h
#define SEGMENT_H
#include "Point.h"

class Segment
{
   private:
      Point P1;
      Point P2;
      int code;
      Segment*tab;
      int Taille;
      int libre;
      static int nombreSegments;
   public:
      Segment(); //Constructeur par défaut
      ~Segment(); // Destructeur
      Segment(Point p1 , Point p2 , int taille); //Constructeur a trois arguments
      int estVide(); //Methode qui dit si le tableau de segments est vide ou pas
      int estPlein(); //Methode qui dit si le tableau de segments est plein ou pas
      int ajoutSegment(const Segment &s); //Méthode qui ajoute un segment dans un
tableau
      Segment(Segment &s); //Constructeur de recopie
      friend int operator==(Segment &s1 , Segment &s2); //Méthode Amie de
comparaison de deux segments
      void ecrireFichier(char*nomFichier); //Méthode qui écrit tous les segments d'une
liste dans un fichier
      void lireFichier(char*nomFichier); //Méthode qui lit tous les segments d'un fichier
et les ajoute a une liste
};


#endif // SEGMENT_H
```

```
//FICHIER Segment.cpp
#include "Segment.h"
#include <iostream>
#include <fstream>
#define TRUE 1
#define FALSE 0
using namespace std;

//Nombre de segments crees
int Segment::nombreSegments = 0;

//Constructeur par défaut
Segment::Segment()
{
   nombreSegments ++;
}

// Destructeur
Segment::~Segment()
{
   //dtor
}

//Constructeur a trois arguments
Segment::Segment(Point p1 , Point p2 , int taille){
   this->P1 = p1;
   this->P2 = p2;
   this->tab = new Segment [taille];
   this->code = nombreSegments;
   nombreSegments ++;
   this->Taille = taille;
   this->libre = 0;
}

//Méthode qui dit si le tableau de segments est vide ou pas
int Segment::estVide() {
   if(this->libre == 0) return TRUE;
   return FALSE;
}

//Methode qui dit si le tableau de segments est plein ou pas
int Segment::estPlein() {
   if(this->libre == this->Taille) return TRUE;
```

```
    return FALSE;
}

//Méthode qui ajoute un segment dans un tableau
int Segment::ajoutSegment(const Segment &s) {
    if(this->estPlein() == 1) return FALSE;
    this->tab[this->libre] = s;
    this->libre ++;
    return TRUE;
}

//Constructeur de recopie
Segment::Segment(Segment &s) {
    this->P1 = s.P1;
    this->P2 = s.P2;
    this->Taille = s.Taille;
    this->libre = s.libre;
    this->code = s.code;
    for(int i = 0; i<s.libre; i++) {
        this->tab[i] = s.tab[i];
    }
    nombreSegments ++;
}

//Méthode Amie de comparaison de deux segments
int operator==(Segment &s1 , Segment &s2) {
    if(s1.code == s2.code) return TRUE;
    return FALSE;
}

//Methode qui ecrit tous les segments d'une liste dans un fichier
void Segment::ecrireFichier(char*nomFichier) {
    ofstream Fichier(nomFichier, std::ifstream::out); //Ouverture du fichier
    if(Fichier.is_open()) { //teste si le fichier a bien été ouvert
        for(int i = 0; i<(this->libre); i++){
        Fichier << this->tab[i].P1.getX()
                <<" "
                <<this->tab[i].P1.getY()
                <<" "
                <<this->tab[i].P2.getX()
                <<" "
                <<this->tab[i].P2.getY()
                <<" "
                <<this->tab[i].code
                <<" "
```

```
            <<this->tab[i].Taille
            <<" "
            <<this->tab[i].libre
             <<" "
              <<this->tab[i].tab
              << endl;
        }
    }
    else{
        cout << "Impossible d'ouvrirle fichier" << endl;
    }
}

//Méthode qui lit tous les segments d'un fichier et les ajoute a une liste
void Segment::lireFichier(char*nomFichier) {
    ifstream Fichier(nomFichier,std::ifstream::in);
    if(Fichier.is_open()){
        Segment S;
        char tmp;
        int x1,x2,y1,y2;
        while(!Fichier.eof()){
            if(this->estPlein()== 1) break;
            Fichier >> x1;
            Fichier >> y1;
            Fichier >> x2;
            Fichier >> y2;
            Fichier >> S.code;
            Fichier >> S.Taille;
            Fichier >> S.libre;
            Fichier >> tmp ;

            S.tab = (Segment*)tmp;
            S.P1.setX(x1);
            S.P1.setY(y1);
            S.P2.setX(x2);
            S.P2.setY(y2);
            this->ajoutSegment(S);
        }
        }
```

```
    else {
        cout << "Impossible d'ouvrir le fichier" << endl;
    }
}
```

```
                    //FICHIER main.cpp
#include <iostream>
#include "Point.h"
#include "Segment.h"

using namespace std;
int main()
{
    Point p(2,6);
    Point P1(p);
    Point p2(3,4);
    int V = P1 == p2;
    cout << V << endl;
    Segment S(p,p2,10);
    Segment S1(S);
    Segment S2(p2,p,4);
    S.ajoutSegment(S1);
    S.ajoutSegment(S2);
    S.ecrireFichier("texte.txt");
    Segment P(P1,p2,5);
    P.lireFichier("texte.txt");
    return 0;
}
```