POO AVANCEE – C++ (1)

LIBS C++

```
#include <iostream> // biblio standard input/output
#include <cmath> // biblio pour utiliser les fonctions maths
#include <vector> // biblio pour utiliser des tableaux dynamiques
#include <iomanip> // biblio pour permettre l'utilisation de fixed/setprecision()
#include <fstream> // biblio pour manipuler des fichiers
```

VARIABLES C++

```
// declaration monoligne
string prenom("Albert"), nom("Einstein");

// allocation d'une variable en memoire (on ne met jamais du parenthese vide)
bool isValid;
int age;

// declarer une constante
double const pi(3.14);
```

NUMBER C++

```
// N chiffre apres virgule
cout << fixed << setprecision(N) << myDouble;
```

MATHS C++

```
// min/max value
min(1, 0);
max(100, 110);

// square root
sqrt(625);

// arroud value
ceil(a);
floor(a);

// power value
pow(base, exp);
```

LOOP C++

```
// for each
for (<type>& tmp : array){
  cout << tmp;
}

// for classic
for (<type> i(0); i < array.size(); i++) {
  cout << array[i];
```

```
}
```

POO AVANCEE – C++ (2)

FONCTION C++

```
// utilisation d'une variable de reference par "&"
void substitution(int& a, int& b) { // ex: echange de valeur entre a et b
  int tmp(a);
  a = b;
  b = tmp;
}


// passage par reference constante – utile pour eviter une copie de valeur en
creant une reference tout en evitant de modifier l'original
void myFunction(string const& text){
  ...
}


// valeur par defaut
int rectangle(int largeur = 20, int hauteur = 10);
```

ARRAY C++

```
// creer un tableau
int arrStatic[5] = {10, 5, 78, 56, 0}; // statique avec des valeurs par defaut
vector<string> arrDynamic; // dynamique sans taille (evolue dans le temps)
int arr2d[col][row]; // multidimensionnels


// longueur d'un tableau (array.length)
array.size();


// passer un tableau en argument d'une fonction
int myFunc(int array[]); // simple array
int myFunc(int& array[5]); // statique
int myFunc(vector<double>& array); // dynamique


// ajouter un element a la queue d'un tableau
array.push_back(e);


// supprimer un element a la queue d'un tableau
array.pop_back();
```

POO AVANCEE – C++ (3)

FILE SYSTEM C++

```
// ecriture dans un fichier avec "ofstream"
ofstream myFile("/data/file", ios::app) // app = append | permettant de concatener chaque nouvelle ecriture


  if (myFile) { // tester si l'ouverture a ete faite
    myFile << "new line";
  } else {
    cout << "error !";
  }


// lecture dans un fichier avec "ifstream"
ifstream myFile("data/file");

  if (myFile) {


    // par caractere avec get()
    char c;

    while (myFile.get(c)) {
      cout << c << endl;
    }


    // par ligne avec getline()
    string line;

    while (getline(myFile, line)) {
      cout << line << endl;
    }

  } else {
    cout << "error !";
  }


// fermeture d'un fichier
myFile.close();
```

POO AVANCEE – C++ (4)

Pour déclarer un pointeur il faut, comme pour les variables, deux choses : un type et un nom

```
// Un pointeur qui peut contenir un variable de type int
int *pointeur(nullptr);

// Un pointeur qui peut contenir l'adresse d'un nombre à virgule
double *pointeurA(nullptr);

// Un pointeur qui peut contenir l'adresse d'un nombre entier positif
unsigned int *pointeurB(nullptr);

// Un pointeur qui peut contenir l'adresse d'une chaîne de caractères
string *pointeurC(nullptr);

// Un pointeur qui peut contenir l'adresse d'un tableau dynamique de nombres entiers
vector<int> *pointeurD(nullptr);

// Un pointeur qui peut contenir l'adresse d'un nombre entier constant
int const *pointeurE(nullptr);
```

Exemple d'utilisation

```
int main()
{
    int ageUtilisateur(16);

    int *ptr(nullptr);

   // On met l'adresse de 'ageUtilisateur' dans le pointeur 'ptr'
    ptr = &ageUtilisateur;

   // Affiche l'adresse de la variable pointEe
          cout << ptr;

   // Acceder a la valeur de la variable pointEe grace a l'etoile "*"
          cout << *ptr

   // Allocation dynamique
    string *nom(nullptr);
          nom = new string("rakoto"); // avec initialisation

   // Liberation de memoire
          delete nom;
          nom = 0; // pointer le pointeur vers rien

    return 0;
}
```

POO AVANCEE – JAVA (1)

LIBS JAVA

```
import java.util.*; // import all utilities
```

NUMBER JAVA

```
// n chiffre apres virgule
System.out.format("%.nf", d);
```

STRING JAVA

```
// convertir une string en array
String[] arr = str.split("");

// recuperer la valeur ascii d'un caractere
int ascii = (int) c;

// recuperer un caractere a partir d'un index i dans une chaine de caractere
str.charAt(i);

// taille d'une chaine de caractere
str.length();

// case
str.toLowerCase() // minuscule
str.toUpperCase() // majuscule
```

MATHS JAVA

```
// pi
Math.PI

// min/max value
Math.min(1, 0);
Math.max(100, 110);

// square root
Math.sqrt(625);

// arround value
Math.round(a);
Math.floor(a);
Math.ceil(a);

// absolute value
Math.abs(a);

// power value
Math.pow(base, exp);

// random value
Math.random();
```

LOOP JAVA

```
// for each
for (<type> tmp : array) {
  System.out.println(tmp);
}

// for classic
 for (int i = 0; i < array.length; i++) {
   System.out.println(array[i]);
 }
```

POO AVANCEE – JAVA (2)

ARRAY JAVA (basics)

```
// creer un tableau statique
int[] array = {14, 13, 5, 0};
String[] array = {"rakoto", "randria", "rasoa"};

// creer un tableau a 2D
int[][] number = {
      {12 , 10, 5},
      {7 , 0, 3},
      {17, 2, 7},
};

// longueur d'un tableau statique
array.length;
```

/* ----- both methods ----- */

NOTE: both = stack | list

```
// longueur
both.size();

// parcours des elements
for (<Type> e : both) {
  System.out.println(e);
}

// parcours d'un tableau d'objet specifique
List<Developpeur> dev = new ArrayList<Developpeur>();
dev.add(dev1);
dev.add(dev2);
for (int i = 0; i < dev.size(); i++) {
System.out.println(dev.get(i).nom);
}
```

/* ----- func args ----- */

```
// passer un pile (stack) dans une fonction
public static <type> <func_name>(Stack<Type> <arg_name>) {...}

// passer une liste (ArrayList) dans une fonction
public static <type> <func_name>(List<Type> <arg_name>) {…}
```

POO AVANCEE – JAVA (3)

ARRAY JAVA (list)

```
// ajouter un nouvel element au queue d'une liste
list.add(e);

// concatener 2 liste existante
list3.addAll(list1);
list3.addAll(list2);

// ajouter un nouvel element au index i donnE
int i = 0; // au premier
int i = list.size(); // au dernier
int i = 1; // au deuxieme
int i = list.size()-1; // avant dernier
list.add(i, e);

// recuperer un element existant a partir d'un index i donnE
list.get(i);

// modifier un element existant a partir d'un index i donnE
list.set(i);

// supprimer un element existant a partir d'un index i donnE
list.remove(i);

// supprimer tout les elements dans une liste donnEe
list.removeAll(liste);

// verifier si une liste est vide ou non
list.isEmpty();

// verifier l'existance d'un element dans une liste existante
list.contains(e);

// trier une liste existante par ordre croissant (String | Integer)
Collections.sort(list);

// inverser une liste existante (a combiner avec sort() pour un tri par ordre
decroissant)
Collections.reverse(list);

// copier une liste vers une autre liste
Collections.copy(<source_list>, <dest_list>);

// permuter 2 valeurs d'une liste a partir de l'index i et j
Collections.swap(list, i, j);
```

POO AVANCEE – WARN (1)

C++ WARN

– La fonction math de la **valeur absolue** en c++ s'ecrit **fabs()** mais non pas abs()

– On utilise **type var()** pour l'initialisation et **=** pour une affectation

– L'utilisation d'une variable de type **string** nécessite l'appel d'un nouveau bibliothèque **#include <string>**

– Seul le prototype ***.hpp** doit contenir les valeurs par defaut (pas la definition de la fonction)

– Les **valeurs par défaut** doivent se trouver a la fin de la liste des paramètres (a doite)

– Pour les tableaux dynamiques, le **type** n'est pas le premier mot de la ligne, contrairement aux autre variables, on utilise une notation avec un chevron **<type>**, on ecrit la taille entre **(size)** et non [size]

– La methode **array.push_back()** ne fonctionne que sur un array de type **vector** (dynamique)

– Il faut toujours déclarer un **pointeur** en lui donnant la valeur **0** ou **nullptr** pour éviter d'utiliser n'importe quel adresse mémoire**.**

– Apres avoir libérer une case mémoire utilisé par un pointeur grâce a **delete,** il faut aussi pointer ce pointeur vers **rien,** i–e **0** (au risque de modifier la valeur d'une variable utilisée par une autre programme)

– Les attributs d'une classe héritée devrait toujours en mode d’accès **protected**

– Une variable de type **byte** a besoin de la bibliothèque **#include <cstddef>** et la fonction **to_integer<int>(b)** pour être affiché

POO AVANCEE – WARN (2)

JAVA WARN

– La longueur d'un tableau dynamique ne s'obtient pas de la même façon qu'en tableau statique (array**.**length)**.** Pour ce faire**,** on doit appeler la méthode **array.size()**

– Pour effectuer la lecture d'un caractère (char)**,** on utilise la méthode **sc.next().charAt(0)**

– Dans un static array**,** le dernier élément s'obtient par la méthode **array.length – 1** tandis que un ArrayList s'obtient en appelant la méthode **list.size()**

– Le trie d'un ArrayList par la méthode **Collections.sort(list)** ne peut être faite qu'en dehors d'un sysout

– On utilise le mot clé **final** pour déclarer une variable comme constante

POO AVANCEE – CORR – 2016 | 2019 (1)

/* ----- calling ----- */

```
#include <iostream>
#include <cstddef> // biblio dependance typage byte

using namespace std;

class A
{
public:
  A();
  void f(int n, float x);
  void g(byte b);
};

// constructeur
A::A()
{
  cout << "objet initialisE !" << endl;
}

// methode 1
void A::f(int n, float x)
{
  cout << "n = " << n << endl;
  cout << "x = " << x << endl;
}

// methode 2
void A::g(byte b)
{
  cout << "b = " << to_integer<int>(b) << endl;
}

int main()
{
  // soit ces declarations
  A a;
  int n;
  byte b;
  float x;
  double y;

// calling
```

a.f(n, x); // => correct car les variables en entrer respecte le typage des arguments lors de la declaration de la fonction membre f();

a.f(b + 3, x); // => incorrect car dans la premiere argument b+3, seul une variable ou valeur de type entier sera acceptEe selon la regle defini pour la fonction f() alors qu'on a ici une addition sur 2 operandes dont le type est different (byte et integer)

a.f(n, y); // => correct car il est autorisé à passer un argument en tant que paramètre de type différent dans une fonction s'il existe une séquence de conversion implicite entre les 2 types comme le double et float.

a.g(b + 1); // => incorrect, meme cas que a.f(b+3,x)

```
  return 0; }
```

POO AVANCEE – CORR – 2016 | 2019 (2)

```
/* ----- indent ----- */

import java.util.*;

class Indent {
  // attrib.s
  protected static int maxId = 0;
  private int id;

  // constructor.s
  public Indent() {
    this.id = ++this.maxId;
    System.out.println("object num " + this.id + " init !");
  }

  // method.s
  public int getIndent() {
    return this.id;
  }

  public int getIndentMax() {
    return this.maxId;
  }

  public static void main(String[] args) {
    Indent a = new Indent();
    Indent b = new Indent();
    Indent c = new Indent();

    // recuperation de l'indentifiant attribuE a chaque objet crEe
    System.out.println("b(id) = " + b.getIndent());
    System.out.println("a(id) = " + a.getIndent());
    System.out.println("c(id) = " + c.getIndent());

    // recuperation du dernier identifiant creE
    System.out.println("last object id = " + a.getIndentMax());
  }
}
```

POO AVANCEE – CORR – 2017 (1)

```
/* ----- vecteur (1) ----- */

#include <iostream>

using namespace std;

class Vecteur {
  // attrib.s
  private:
    float x, y, xp, yp;

  // init constructor.s | destructor | getters | setters | method.s
  public:
    Vecteur(float _x, float _y, float _xp, float _yp); // constructeur
    ~Vecteur(); // destructeur
    float getX(); // getter pour x
    float getY(); // getter pour y
    float getXp(); // getter pour l'extremitE x
    float getYp(); // getter pour l'extremitE y
    void getVector(); // consultation d'un vecteur
    void setX(float _x); // setter pour x
    void setY(float _y); // setter pour y
    void setXp(float _xp); // setter pour l'extremitE x
    void setYp(float _yp); // setter pour l'extremitE y
    void setVector(float _x, float _y, float _xp, float _yp); // modification d'un vecteur
    float det(Vecteur v1, Vecteur v2); // pour calculer le determinant des 2 vecteurs
};

// def constructor.s
Vecteur::Vecteur(float _x, float _y, float _xp, float _yp) {
  this->x = _x;
  this->y = _y;
  this->xp = _xp;
  this->yp = _yp;
}

// def destructor
Vecteur::~Vecteur(){};

// getters
float Vecteur::getX() {
  return this->x;
}

float Vecteur::getY() {
  return this->y;
}


float Vecteur::getXp() {
  return this->xp; }
```

POO AVANCEE – CORR – 2017 (2)

```
/* ----- vecteur (2) ----- */

float Vecteur::getYp() {
  return this->yp;
}

void Vecteur::getVector() {
  cout << "x = " << this->x << endl;
  cout << "y = " << this->y << endl;
  cout << "xp = " << this->xp << endl;
  cout << "yp = " << this->yp << endl;
}

// setters
void Vecteur::setX(float _x) {
  this->x = _x;
}

void Vecteur::setY(float _y) {
  this->y = _y;
}

void Vecteur::setXp(float _xp) {
  this->xp = _xp;
}

void Vecteur::setYp(float _yp) {
  this->yp = _yp;
}

void Vecteur::setVector(float _x, float _y, float _xp, float _yp) {
  this->x = _x;
  this->y = _y;
  this->xp = _xp;
  this->yp = _yp;
}

// method.s
float Vecteur::det(Vecteur v1, Vecteur v2) {
  return v1.getXp()*v2.getYp() - v1.getYp()*v2.getXp();
}

// test
int main() {

  // init
  Vecteur vect1(0, 0, 10, 20);
  Vecteur vect2(0, 0, 12, 18);

  cout << "det = " << vect1.det(vect2, vect1);

  return 0; }
```

POO AVANCEE – CORR – 2017 (3)

```
/* ----- gestion des utilisateurs (1) ----- */

import java.util.*;

abstract class Personne {
  // attrib.s
  protected int id;
  protected String nom;
  protected String prenom;
  protected double salaire;

  // constructor.s
  Personne(int _id, String _nom, String _prenom, double _salaire) {
    this.id = _id;
    this.nom = _nom;
    this.prenom = _prenom;
    this.salaire = _salaire;
  }

  // method.s
  public double calculerSalaire() {
    return this.salaire;
  };
}

class Developpeur extends Personne {
  // attrib.s
  private String specialite;

  // constructor.s
  Developpeur(int _id, String _nom, String _prenom, double _salaire) {
    super(_id, _nom, _prenom, _salaire);
  }

  // method.s
  public double calculerSalaire() {
    return this.salaire + ((this.salaire * 20) / 100);
  }
}

class Manager extends Personne {
  // attrib.s
  private String service;

  // constructor.s
  Manager(int _id, String _nom, String _prenom, double _salaire) {
    super(_id, _nom, _prenom, _salaire);
  }

  // method.s
  public double calculerSalaire() {
    return this.salaire + ((this.salaire * 35) / 100);
  }}
```

POO AVANCEE – CORR – 2017 (4)

```
/* ----- gestion des utilisateurs (2) ----- */

public class Main {

  public static void main(String[] args) {

    // creation des 2 developpeurs
    Developpeur dev1 = new Developpeur(1, "nom1", "prenom1", 2000);
    Developpeur dev2 = new Developpeur(2, "nom2", "prenom2", 1500);

    // creation des 2 managers
    Manager manag1 = new Manager(3, "nom3", "prenom3", 700);
    Manager manag2 = new Manager(4, "nom4", "prenom4", 2000);

    // affichage
    List<Developpeur> dev = new ArrayList<Developpeur>();
    List<Manager> manag = new ArrayList<Manager>();
    dev.add(dev1);
    dev.add(dev2);
    manag.add(manag1);
    manag.add(manag2);

    System.out.println("~ Liste des developpeurs ~\n");
    for (int i = 0; i < dev.size(); i++) {
      Developpeur d = dev.get(i);
      System.out.println("id: "+d.id);
      System.out.println("nom: "+d.nom);
      System.out.println("prenom: "+d.prenom);
      System.out.println("salaire: "+d.calculerSalaire()+"\n");
    }

    System.out.println("\n-------------------\n");

    System.out.println("~ Liste des managers ~\n");
    for (int i = 0; i < manag.size(); i++) {
      Manager m = manag.get(i);
      System.out.println("id: "+m.id);
      System.out.println("nom: "+m.nom);
      System.out.println("prenom: "+m.prenom);
      System.out.println("salaire: "+m.calculerSalaire()+"\n");
    }
  }
}
```

POO AVANCEE – CORR – 2017 (5)

/* ----- point de couleur (1) ----- */

```
#include <iostream>
#include <string>

using namespace std;

// creation de la class point
class Point
{
protected:
  int x, y;
  Point(int _x, int _y);

public:
  ~Point();
  void affichage();
};

// constructeur de la class point
Point::Point(int _x, int _y)
{
  this->x = _x;
  this->y = _y;
  this->affichage();
}

// methode permettant de tracer l'utilisation de la class point
void Point::affichage()
{
  cout << "Classe Point utilisEe !" << endl;
}

// destructeur de la class point
Point::~Point(){};

// creation de la class couleur
class Couleur
{
protected:
  string color;
  Couleur(string _color);

public:
  ~Couleur();
  void affichage();
};

// constructeur de la class couleur
Couleur::Couleur(string _color)
{
  this->color = _color;
  this->affichage();
}

// methode permettant de tracer l'utilisation de la class couleur
void Couleur::affichage()
{
  cout << "Classe Couleur utilisEe !" << endl;
}
```

POO AVANCEE – CORR – 2017 (6)

/* ----- point de couleur (2) ----- */

```
// destructeur de la class couleur
Couleur::~Couleur(){};

// creation de la class derivEe pointdecouleur avec un heritage multiple par la class de base point et couleur
class PointDeCouleur : public Point, public Couleur
{
public:
  // constructeur paramétré pour la class pointdecouleur
  PointDeCouleur(int _x, int _y, string _color) : Point(_x, _y), Couleur(_color)
  {
    this->x = _x;
    this->y = _y;
    this->color = _color;
    this->affichage();
  }

  // getters & setters pour les points x, y et la couleur
  int getX()
  {
    return this->x;
  }

  int getY()
  {
    return this->y;
  }

  string getColor()
  {
    return this->color;
  }

  void setX(int _x)
  {
    this->x = _x;
  }

  void setY(int _y)
  {
    this->y = _y;
  }

  void setColor(string _color)
  {
    this->color = _color;
  }

  // methode permettant de tracer l'utilisation de la class point
  void affichage()
  {
    cout << "Classe PointDeCouleur utilisEe !" << endl;
  }
  // destructeur pour la class pointdecouleur
  ~PointDeCouleur(){};
};

int main()
{
  PointDeCouleur i(5, -2, "magenta");
  cout << "x = " << i.getX() << " | y = " << i.getY() << " | couleur = " << i.getColor() << endl;

  return 0;}
```

POO AVANCEE – CORR – 2016 (1)

```
/* ----- rectangle ----- */

#include <iostream>

using namespace std;

class Rectangle {

  // attrib.s
  private:
    float largeur, hauteur;

  // init constructor.s | destructor | method.s
  public:
    Rectangle(float _largeur, float _hauteur);
    ~Rectangle();
    float calcPerimetre();
    float calcSurface();
    void affichage();
};

// def constructor.s
Rectangle::Rectangle(float _largeur, float _hauteur) {
  this->largeur = _largeur;
  this->hauteur = _hauteur;
}

// def method.s
float Rectangle::calcPerimetre() { // 2 (largeur + hauteur)
  return (this->largeur + this->hauteur) * 2;
}

float Rectangle::calcSurface() { // largeur x hauteur
  return (this->largeur * this->hauteur);
}

// def destructor
Rectangle::~Rectangle() {};

void Rectangle::affichage() {
  cout << "Largeur = " << this->largeur << " | " << "Hauteur = " << this->hauteur
<< endl;
  cout << "Perimetre = " << this->calcPerimetre() << endl;
  cout << "Surface = " << this->calcSurface() << endl;
}

int main() {

  Rectangle rectangle(12.5, 56);
  rectangle.affichage(); // resultat

  return 0;}
```

POO AVANCEE – CORR – 2016 (2)

```
/* ----- livre ----- */

import java.util.*;

public class Livre {
  // attrib.s
  private static int maxIdLivre = 0;
  private int idLivre;
  private String auteur;

  // constructor.s
  public Livre(String _auteur) {
    this.idLivre = ++this.maxIdLivre;
    this.auteur = _auteur;
    System.out.println("Livre num " + this.idLivre + " crEe !");
  }

  // method.s
  public String getAuthor() {
    return this.auteur;
  }

  public static void main(String[] args) {
    Livre livre1 = new Livre("Rakoto");
    Livre livre2 = new Livre("Rabe");

    System.out.println("L'auteur du premier livre est " +
livre1.getAuthor());

    System.out.println("L'auteur du deuxieme livre est " +
livre2.getAuthor());
  }
}
```

POO AVANCEE – CORR – 2015 (1)

```
/* ----- point ----- */

#include <iostream>

using namespace std;

class Point
{
protected:
   static int pointNumber; // pour preserver la derniere valeur de l'identifiant
d'un objet point crEe
public:
   int id; // pour stocker l'identifiant d'un objet point crEe
   Point(); // declaration de la constructeur
   ~Point(); // declaration de la destructeur
};

// init pointNumber
int Point::pointNumber = 0;

// definition de la constructeur
Point::Point()
{
   this->id = ++this->pointNumber; // pre-incrementation pour la premiere valeur a
1
   cout << "\nPoint numero " << this->id << " crEe !";
}

// definition de la destructeur
Point::~Point()
{
   cout << "\nPoint numero " << this->id << " detruit !";
}

int main()
{
   // tableau d'objet pour les 4 points
   const int size(4);
   Point *point = new Point[size]; // utilisation du mot cle new pour appeler explicitement chaque
destructeur de l'objet

   // instanciation de l'objet
   for (int i(0); i < size; i++) {
      point[i];
   }

   // destruction de l'objet
   for (int i(0); i < size; i++) {
      point[i].~Point();
   }

   return 0;
}
```

POO AVANCEE – CORR – 2015 (2)

```
/* ----- factorielle ----- */

#include <iostream>

using namespace std;

int main()
{
  int number;

  cout << "Entrer un nombre: ";
  cin >> number;

  int fact(number);
  int tmp(number - 1);

  while (tmp > 1)
  {
    fact *= tmp;
    tmp--;
  }

  cout << "Factorielle de " << number << " est: " << fact;

  return 0;
}
```

POO AVANCEE – QUESTION | REPONSE (1)

Question: Quelles sont les ressemblances et les differences entre c++ et java

/* ----- similitudes ----- */

– Ces 2 langages prennent en charge la programmation orientée objet ce qui permet l'utilisation d'une class pour créer un objet en incluant la notion d'héritage, polymorphisme, abstraction et encapsulation

– Au niveau de la syntaxe général comme la condition et boucle, ils ont a peu près la même syntaxe mais java est un peu verbeux que c++

– Les 2 ont les mêmes opérateurs arithmétiques et relationnels

– La syntaxe au niveau des commentaires sont tous identiques. // ... pour un commentaire mono-ligne et /* ... */ pour un commentaire multi-lignes

– L'exécution des programme de ces 2 langages commence a partir de la fonction principale main()

– Ils ont les mêmes type de données primitifs comme char, int, float, double sauf le type booléen qui est boolean en java et bool en c++

– Les 2 langages ont un support multi-threading pour réaliser le multitâche

/* ----- differences ----- */

– La signature d'une méthode en c++ est la précision d'une méthode, du type, de ses arguments et du type de donnée retournée. En Java, ce dernier n'en fait pas partie.

– En java, comme beaucoup de langages a objets, il n'existe pas de destructeur car la libération de la mémoire est gérée automatiquement par le "Garbage Collector" (gestion de la mémoire).

– En c++, une méthode peut être reliée a une class grâce a l'opérateur de portée "::" ce qui n'est pas le cas lors de la création d'une méthode dans une class java

– La directive "new" est obligatoire pour instancier un objet en java

– Java ne supporte pas la surcharge des opérateurs contrairement a c++

– Java ne permet pas l'héritage multiple a cause de la complexité de l'implémentation pour les compilateurs java mais cela a été résolu par la notion d'interface qui permet de récupérer la plupart des fonctionnalités de l'héritage multiple en c++

– Dans le domaine de l'application: c++ est mieux adapté au développement de grands logiciels comme le système de gestion des employés, le système de réservation de passagers tandis que java peut être utilisé pour développer des applications de communication/internet comme protocoles réseau, navigateur web

POO AVANCEE – QUESTION | REPONSE (2)

Question: Pourquoi l'approche objet est elle un approche bien adaptEe au logiciel actuellement ?

L'approche objet aide a mieux organiser la complexité inhérente aux logiciels et celle caractéristique du monde réel a modéliser ce qui donne la possibilité de programmer par composants et facilite la réutilisation du code dans un cadre différent

Question: Qu'est ce qui fait qu'un logiciel "objet" s'avere theoriquement mieux adaptE a la prise en compte d'evolutions fonctionnelles qu'un logiciel programmE en programmation procedurale ?

En programmation orientée objet, les données prennent plus d'importance que les fonctions du programme, ce qui corrige les défauts de la programmation procédurale en introduisant le concept "objet" et "classe" avec les spécificateurs d'accès pour les données afin d'améliorer sa sécurité, les données et les fonctions membres de chaque objet individuel agissent alors comme une seule unité. Cela dit que la programmation orientée objet permet de créer plusieurs instances de l'objet sans aucune interférence et communique entre elles par le biais de messages.