Projet d'informatique

Emma Geoffray et Jonathan Burkhard

2 avril 2014

MakeFile

```
CXX = g++-4.8
CC  = $(CXX)

CXXFLAGS = -std=c++11  # C++11

# Partie commentÃ©e : choisissez les options que vous voulez avoir
#                    en dÃ©commentant la/les lignes correspondantes
#
# CXXFLAGS += -pedantic -Wall         # pour les purs et durs
# CXXFLAGS += -g                      # pour debugger
# CXXFLAGS += -pg                     # pour profiler
# LDFLAGS  += -pg                     # pour profiler
# CXXFLAGS += -O2                     # pour optimiser la vitesse

all:: testSysteme

Vecteur.o: Vecteur.cc Vecteur.h

Particule.o: Particule.cc Particule.h

TXTNeon.o: TXTNeon.cc TXTNeon.h

TXTArgon.o: TXTArgon.cc TXTArgon.h

Enceinte.o: Enceinte.cc Enceinte.h

Systeme.o: Systeme.cc Systeme.h

testSysteme.o: testSysteme.cc Systeme.h

testSysteme: testSysteme.o Systeme.o Enceinte.o Particule.o TXTNeon.o TXTArgon.o Vecteur.o
```

Classe Dessinable :

```
#ifndef PRJ_DESSINABLE_H
#define PRJ_DESSINABLE_H

class Dessinable
{
   public :
   virtual void dessine() const = 0;
};

#endif // PRJ_DESSINABLE_H
```

Classe Particule

```
#ifndef PRJ_PARTICULE_H
#define PRJ_PARTICULE_H


#include "Vecteur.h"
#include "Dessinable.h"

class Particule : public Dessinable
{
   private :

   Vecteur position;
   Vecteur vitesse;
   double masse;

   public :

   Particule();
   Particule(Vecteur position, Vecteur vitesse, double masse);

   std::ostream& afficher(std::ostream& sortie) const;

};

std::ostream& operator<<(std::ostream& sortie, Particule const & p);

#endif // PRJ_PARTICULE_H
```

```
#include "Particule.h"

/* DÃ©finition des constructeurs : */

Particule::Particule()
   : position(Vecteur(0,0,0)), vitesse(Vecteur(0,0,0)), masse(0) {}

Particule::Particule(Vecteur position, Vecteur vitesse, double masse)
   : position(position), vitesse(vitesse), masse(masse) {}

/* MÃ©thodes utiles : */

ostream& Particule::afficher(ostream& sortie) const
{
   return sortie << "pos␣:␣" << position << "␣;␣v␣:␣" << vitesse
                 << "␣;␣m␣:␣" << masse;
}

/* Fonctions utiles : */

ostream& operator<<(ostream& sortie, Particule const& p)
{
   return p.afficher(sortie);
}
```

Classe Argon

```
#include "Particule.h"

class TXTArgon : public Particule
{
   private:
   TXTArgon(Vecteur position, Vecteur vitesse);

   void dessine() const override;
};
```

```cpp
#include "TXTArgon.h"

TXTArgon::TXTArgon(Vecteur position, Vecteur vitesse)
 : Particule(position, vitesse, 39.948) {}

void TXTArgon::dessine() const
{
    cout << "particule␣TXTAr␣:␣" << *this << endl;
}
```

Classe Neon

```cpp
#include "Particule.h"

class TXTNeon : public Particule
{
    private :
    TXTNeon(Vecteur position, Vecteur vitesse);

    void dessine() const override;
};
```

```cpp
#include "TXTNeon.h"

TXTNeon::TXTNeon(Vecteur position, Vecteur vitesse)
 : Particule(position, vitesse, 20.1797) {}

void TXTNeon::dessine() const
{
    cout << "particule␣TXTNe␣:␣" << *this << endl;
}
```

Classe Enceinte

```cpp
#ifndef PRJ_ENCEINTE_H
#define PRJ_ENCEINTE_H

class Enceinte
{
    private:
    double hauteur;
    double largeur;
    double profondeur;

    public:
/*============================================================================
 * Prototype du constructeur
 *============================================================================*/
    Enceinte();
    Enceinte(double hauteur, double largeur, double profondeur);
/*============================================================================
 * Prototype des mÃ©thodes
 *============================================================================*/
    void display() const;

};

#endif // PRJ_ENCEINTE_H
```

```cpp
#include <iostream>
#include "Enceinte.h"

using namespace std;

/*============================================================================
 * Definition des constructeurs
 *============================================================================*/
   Enceinte::Enceinte()
   :hauteur(20), largeur(20), profondeur(20)
   {}

   Enceinte::Enceinte(double hauteur, double largeur, double profondeur)
   :hauteur(hauteur), largeur(largeur), profondeur(profondeur)
   {}
/*============================================================================
 * Definition des mÃ©thodes
 *============================================================================*/
   void Enceinte::display() const
   {
      cout << hauteur << ",␣" << largeur << ",␣" << profondeur << endl;
   }
```

Classe Systeme

```cpp
#ifndef PRJ_SYSTEME_H
#define PRJ_SYSTEME_H

#include <memory>
#include <vector>
#include "Particule.h"
#include "Enceinte.h"

class Systeme : public Dessinable
{
   private:
   vector<unique_ptr<Particule>> collectionParticule;
   Enceinte enceinte;

/*============================================================================
 * Prototypes du constructeur
 *============================================================================*/
   public:
   Systeme();
   Systeme(double hauteur, double largeur, double profondeur);

   private:
   Systeme(Systeme const& autre) = delete;//suppression du constructeur de copie

/*============================================================================
 * Prototypes des methodes
 *============================================================================*/
   public:
   void ajouterParticule(Particule* particule);
   void dessine() const override;
   bool supprimerParticules();

   private:
   Systeme& operator=(Systeme aCopier) = delete; //suppression de l'operateur = pour un systemme
};

#endif // PRJ_SYSTEME_H
```

```cpp
#include "Systeme.h"

/*================================================================================
 * Definition du constructeur
 *================================================================================*/

Systeme::Systeme()
{
      cout << "creation␣systeme␣avec␣enceinte␣initiale␣!" << endl;
}

Systeme::Systeme(double hauteur, double largeur, double profondeur)
: enceinte(hauteur, largeur, profondeur)
{
   cout << "creation␣systeme␣avec␣une␣enceinte␣personnalisÃⓒe␣!" << endl;
}

/*================================================================================
 * Definition des methodes
 *================================================================================*/

void Systeme::ajouterParticule(Particule* particule)
{
   if(particule != nullptr)
   {
      collectionParticule.push_back(unique_ptr<Particule>(particule));
      cout << "Une␣nouvelle␣particule␣a␣ÃⓒtÃⓒ␣ajoutÃⓒe␣!" << endl;
   }
}

void Systeme::dessine() const
{
   for(auto const& particule : collectionParticule)
   {
      (*particule).dessine();
   }
}

bool Systeme::supprimerParticules()
{
   cout << "Je␣suis␣rentrer␣dans␣supprimerPArticule␣" << endl;
   for(auto& particule : collectionParticule)
   {
      particule.reset();
      cout << "Je␣supprime␣une␣particule␣" << endl;
   }
   cout << "J'ai␣fini␣de␣supprimer" << endl;
   return true;
}
```

Classe Vecteur

```
#ifndef PRJ_VECTEUR_H
#define PRJ_VECTEUR_H

#include <iostream>

using namespace std;

class Vecteur
{
   private :
/*============================================================================
 * Definition des attributs
 *===========================================================================*/
   double x;
   double y;
   double z;

   public:
/*============================================================================
 * Prototyopes des constructeurs
 *===========================================================================*/
   Vecteur();
   Vecteur(double x, double y, double z);

/*============================================================================
 * Prototypes des methodes
 *===========================================================================*/
   // Comparateurs == et != :
   bool operator==(Vecteur const&) const;
   bool operator!=(Vecteur const& v) const;
   //MÃ©thodes += et -= :
   Vecteur& operator+=(Vecteur const& v1);
   Vecteur& operator-=(Vecteur const& v1);
   // Multiplication par un scalaire :
   Vecteur& operator*=(double const& scalaire);
   // Produit scalaire :
   double operator*(const Vecteur& v1);
   // Produit vectoriel :
   Vecteur& operator^=(Vecteur const& v1);// ATTENTION XOR a une plus grande prioritÃ© Ã  cause du
   // Vecteur opposÃ© :
   const Vecteur operator-();
   // MÃ©thode afficher :
   std::ostream& afficher(std::ostream& sortie) const;//std est pour pas mettre le using namespace

};

/*============================================================================
 * Prototypes des fonctions
 *===========================================================================*/
std::ostream& operator<<(std::ostream& sortie, Vecteur const& v1); //std est pour pas mettre le us
const Vecteur operator+(Vecteur v1, Vecteur const& v2);
const Vecteur operator-(Vecteur v1, Vecteur const& v2);
const Vecteur operator*(double scalaire, Vecteur v1);
const Vecteur operator*(Vecteur v1, double scalaire);
const Vecteur operator^(Vecteur v1, Vecteur const& v2);// ATTENTION XOR a une plus grande prioritÃ©

#endif // PRJ_VECTEUR_H
```

```cpp
#include "Vecteur.h"

/*==========================================================================
 * Definition des constructeurs
 *==========================================================================*/

Vecteur::Vecteur()
   : x(0), y(0), z(0) {}

Vecteur::Vecteur(double x, double y, double z)
   : x(x), y(y), z(z) {}

/*==========================================================================
 * Definition des mÃ©thodes
 *==========================================================================*/
bool Vecteur::operator==(Vecteur const& v) const
{
   if (x == v.x and y == v.y and z == v.z) { return true; }
   else { return false; }
}

bool Vecteur::operator!=(Vecteur const& v) const
{
   return not(*this == v);
}

Vecteur& Vecteur::operator+=(Vecteur const& v1)
{
   x += v1.x;
   y += v1.y;
   z += v1.z;
   return *this;
}

Vecteur& Vecteur::operator-=(Vecteur const& v1)
{
   x -= v1.x;
   y -= v1.y;
   z -= v1.z;
   return *this;
}

const Vecteur Vecteur::operator-()
{
   return ((*this)*= (-1.));
}

Vecteur& Vecteur::operator*=(double const& scalaire)
{
   x *= scalaire;
   y *= scalaire;
   z *= scalaire;
   return *this;
}

Vecteur& Vecteur::operator^=(Vecteur const& v1)
{
   double x_(x), y_(y);
   x = (y * v1.z - z * v1.y);
   y = (z * v1.x - x_ * v1.z);
   z = (x_ * v1.y - y_ * v1.x);
   return *this;
}

double Vecteur::operator*(const Vecteur& v1)
```

```
{
    return (v1.x*x + v1.y*y + v1.z*z);
}

ostream& Vecteur::afficher(ostream& sortie) const
{
    return sortie << "(" << x << ", " << y << ", " << z  << ")";
}

/*================================================================================
 * Definition des fonctions
 *================================================================================*/

ostream& operator<<(ostream& sortie, Vecteur const& v1)
{
    return v1.afficher(sortie);
}

/* Les mÃ©thodes suivantes sont Ã©crites sur 2 lignes pour Ã©viter la copie
 * inutile faÃ®tes par certain compilateurs */

const Vecteur operator+(Vecteur v1, Vecteur const& v2)
{
    v1 += v2;
    return v1;
}

const Vecteur operator-(Vecteur v1, Vecteur const& v2)
{
    v1 -= v2;
    return v1;
}

const Vecteur operator*(double scalaire, Vecteur v1)
{
    v1 *= scalaire;
    return v1;
}

const Vecteur operator*(Vecteur v1, double scalaire)
{
    v1 *= scalaire;
    return v1;
}

const Vecteur operator^(Vecteur v1, Vecteur const& v2)
{
    v1 ^= v2;
    return v1;
}
```