Polymorphisme :

class Sportif

{

private:

string \_prenom;

string \_nom;

int \_anneeNaissance;

public:

Sportif(string, string, int);

virtual ~Sportif();

string getPrenom();

string getNom();

int getAnneeNaissance();

void setPrenom(string);

void setNom(string);

void setAnneeNaissance(int);

virtual string toString(); //methode virtuelle pour pouvoir etre redefinie dans les classes filles

virtual int getAge(); //methode virtuelle pour pouvoir etre redefinie dans les classes filles

};

class SportifMenteur : public Sportif

{

private:

string \_tuteur;

public:

SportifMenteur(string, string, int, string);

~SportifMenteur();

string getTuteur();

void setTuteur(string);

string toString(); //redefinition de la methode toString

int getAge(); //redefinition de la methode getAge

};

Lien entre 2 classes :

class Individu

{

private:

    string nom;

    string prenom;

    Voiture\* maVoiture;

public:

    Individu(string, string);

    ~Individu();

    string toString();

    string toStringAndLink();

    string getNom();

    string getPrenom();

    Voiture\* getVoiture();

    void setVoiture(Voiture\* laVoiture);

    void setNom(string);

    void setPrenom(string);

    void majVoiture(Voiture\* voiture);

    void supprimerLien();

};

string Individu::toStringAndLink()

{

if (this->getVoiture() != nullptr)

{

return this->toString() + this->getVoiture()->toString();

}

else

{

return this->toString();

}

}

Individu::~Individu()

{

    supprimerLien();

    delete maVoiture;

}

void Individu::setVoiture(Voiture \*laVoiture)

{

supprimerLien();

this->majVoiture(laVoiture);

laVoiture->majPilote(this);

}

void Individu::supprimerLien()

{

    this->maVoiture = nullptr;

}

string Voiture::toStringAndLink()

{

if (this->getPilote() != nullptr)

{

return this->toString() + ", pilote : " + this->getPilote()->toString();

}

else

{

return this->toString();

}

}

Voiture::~Voiture()

{

supprimerLien();

delete monPilote;

delete this;

}

void Voiture::setPilote(Individu \*lePilote)

{

supprimerLien();

this->majPilote(lePilote);

lePilote->setVoiture(this);

}

void Voiture::supprimerLien()

{

this->monPilote = nullptr;

}

class Voiture

{

private:

    string marque;

    string plaque;

    Individu\* monPilote;

public:

    Voiture(string, string);

    ~Voiture();

    string toString();

    string toStringAndLink();

    string getMarque();

    string getPlaque();

    Individu\* getPilote();

    void setPilote(Individu\* lePilote);

    void setMarque(string);

    void setPlaque(string);

    void majPilote(Individu\* pilote);

    void supprimerLien();

};

Classe generique :

template <class T, class A = allocator<T>>

class List

{

public:

    typedef T value\_type;

    typedef value\_type \*iterator;

    List();

    void push\_back(const value\_type &val);

    iterator begin();

    iterator end();

    void listeSimple();

};

template <class T1, class T2>

class Pair

{

public:

    typedef T1 fisrt\_type;

    typedef T2 second\_type;

    Pair(const fisrt\_type &, const second\_type &);

    void exemplePair();

    void listePaires();

};

// Exemple de modèle de fonction qui retourne le max. de 2 éléments

template <class T> // La classe T est un paramètre formel

T maxi (T& t1, T& t2) { // ‘maxi’ est une fonction modèle

return (t1 > t2 ? t1 : t2);

}

// Définition d’un opérateur (ex. operator>) dans une classe (ex. Produit)

bool Produit::operator> (const Produit& unProduit) const {

bool resultat;

resultat = (this->prix > unProduit.prix);

return resultat;

}