ECE PARIS
ÉCOLE D'INGÉNIEURS

Robin FERCOQ
2019-2020

Conception et Programmation Orientée Objet C++

POO - C++

Sommaire général du semestre

COURS	Semaine suivante →	**TPs**
1. *Intro, concepts, 1 exemple*		1. *Organisation objet des données*
2. *Modélisation objet / UML*		2. *Diagrammes de classe UML*
3. *C++ pratique 1*		3. *C++ pratique, E/S, string, vector*
4. *C++ pratique 2*		4. *C++ pratique, type &, surcharge*
5. *Classes & C++ : bases*		5. *Date : une classe simple en C++*
6. *Classes & C++ : compléments*		6. *UML et C++, associations*
7. *Conteneurs & C++ : la STL*		7. *Gestion de collections complexes*
8. *Héritage / polymorphisme*		8. *Collections polymorphes*
9. *Abstraction / design patterns*		9. *Modèle composite et graphismes*
10. *Exceptions, flots, fichiers ...*		10. *Persistance / fichiers / except.*
11. *Templates côté développeur*		11. *Développement de templates*
12. Compléments		12. *Soutenance de **projet** ...*

COURS 12

A) Static variables/attributs/méthodes
B) Complément conteneurs/itérateurs
C) Exemples de prog. fonctionnelle

COURS 12

A) Static variables/attributs/méthodes
B) Complément conteneurs/itérateurs
C) Exemples de prog. fonctionnelle

static variables/attributs/méthodes

- *Les variables **statiques** sont des variables définies localement (à une fonction ou une méthode)*
- *se comportent **comme une variable globale** :*
  - *valeur initialisée par défaut (comme une globale)*
  - ***valeur conservée d'un appel à l'autre** :*
    la variable n'est pas détruite en fin d'appel

C ou C++

```
void fonction()
{
    static int compteur = 0;

    std::cout << "appel numero " << ++compteur << std::endl;
}

int main()
{
    fonction();
    fonction();
    fonction();
    ...
```

```
appel numero 1
appel numero 2
appel numero 3
```

static variables/attributs/méthodes

- *Utile en débogage, générer des identifiants…*
- *Pas possible d'accéder depuis l'extérieur de la fonction à la ressource static pour la reset ou autres opérations*
- *Attention aux bricolages : bon potentiel d'abus !*

C ou C++

```
/// HORRIBLE: bricolage, un seul pacman, pas accès aux données...

void bougerPacMan(int depx, int depy)
{
    static int posx=200, posy=100;

    posx += depx;   posy += depy;

    std::cout << "le pacman est en x=" << posx
                              << " y=" << posy << std::endl;
}

int main()
{
    bougerPacMan(+50, +10);
    bougerPacMan(  0, +20);
    bougerPacMan(-20,   0);
```

```
le pacman est en x=250 y=110
le pacman est en x=250 y=130
le pacman est en x=230 y=130
```

static variables/attributs/méthodes

- *L'abus de static indique une confusion fonction / classe*
- *Utiliser une fonction comme une classe ? => **classe** !*

C++ only !

```
/// Correct
class PacMan
{
    public :
        PacMan() = default;
        void bouger(int depx, int depy)       {
            m_posx += depx;   m_posy += depy;
            std::cout << "le pacman est en"
                      << " x=" << m_posx
                      << " y=" << m_posy << std::endl;   }
    private :
        int m_posx = 200;  /// Init. par défaut des attributs
        int m_posy = 100;
};

int main()
{
    PacMan monPacMan;
    monPacMan.bouger(+50, +10);
    monPacMan.bouger(  0, +20);
    monPacMan.bouger(-20,   0);
```

```
le pacman est en x=250 y=110
le pacman est en x=250 y=130
le pacman est en x=230 y=130
```

static variables/attributs/méthodes

- *Exemple de cas d'usage légitime : détecter 1er passage*
- *Supposons un besoin de vitesse sur une fonction mathématique lourde mais sans besoin de précision*

C++14

```
/// Calcul du sinus d'un angle "au degré près" (pas très précis)
double slowSin(unsigned int deg)
{
    const double M_PI = 3.14159265358979323846;
    return sin(deg*M_PI/180);
}

int main()
{
    std::vector<double> vals(100000000);
    std::generate(vals.begin(), vals.end(),
                  [v = -10] () mutable { return v+=10; });

    std::vector<double> result;
    std::transform(vals.cbegin(), vals.cend(),
                   std::back_inserter(result), slowSin);

    for (size_t i=0; i<10; ++i)
        std::cout << "sin(" << vals[i] << ")="
                  << result[i] << std::endl;
```

100 millions d'appels à sinus : ~5s

```
sin(0)=0
sin(10)=0.173648
sin(20)=0.34202
sin(30)=0.5
sin(40)=0.642788
sin(50)=0.766044
sin(60)=0.866025
sin(70)=0.939693
sin(80)=0.984808
sin(90)=1
```

static variables/attributs/méthodes

- *Plutôt que de recalculer à chaque fois la valeur de la fonction f(x) en un ensemble de x finis, on peut pré-calculer tous les f(x) dans une Lookup Table (LUT)*

```
/// Calcul du sinus d'un angle "au degré près" (pas très précis)
double fastSin0(unsigned int deg)
{
    static const double lut[360] = {
        0.000000, 0.017452, 0.034899, 0.052336, 0.069756, 0.087156, 0.104528, 0.121869, 0.139173, 0.156434,
        0.173648, 0.190809, 0.207912, 0.224951, 0.241922, 0.258819, 0.275637, 0.292372, 0.309017, 0.325568,
        0.342020, 0.358368, 0.374607, 0.390731, 0.406737, 0.422618, 0.438371, 0.453990, 0.469472, 0.484810,
        0.500000, 0.515038, 0.529919, 0.544639, 0.559193, 0.573576, 0.587785, 0.601815, 0.615661, 0.629320,
        0.642788, 0.656059, 0.669131, 0.681998, 0.694658, 0.707107, 0.719340, 0.731354, 0.743145, 0.754710,
        0.766044, 0.777146, 0.788011, 0.798636, 0.809017, 0.819152, 0.829038, 0.838671, 0.848048, 0.857167,
        0.866025, 0.874620, 0.882948, 0.891007, 0.898794, 0.906308, 0.913545, 0.920505, 0.927184, 0.933580,
        0.939693, 0.945519, 0.951057, 0.956305, 0.961262, 0.965926, 0.970296, 0.974370, 0.978148, 0.981627,
        0.984808, 0.987688, 0.990268, 0.992546, 0.994522, 0.996195, 0.997564, 0.998630, 0.999391, 0.999848,
        1.000000, 0.999848, 0.999391, 0.998630, 0.997564, 0.996195, 0.994522, 0.992546, 0.990268, 0.987688,
        0.984808, 0.981627, 0.978148, 0.974370, 0.970296, 0.965926, 0.961262, 0.956305, 0.951057, 0.945519,
        0.939693, 0.933580, 0.927184, 0.920505, 0.913545, 0.906308, 0.898794, 0.891007, 0.882948, 0.874620,
        0.866025, 0.857167, 0.848048, 0.838671, 0.829038, 0.819152, 0.809017, 0.798636, 0.788011, 0.777146,
        0.766044, 0.754710, 0.743145, 0.731354, 0.719340, 0.707107, 0.694658, 0.681998, 0.669131, 0.656059,
        0.642788, 0.629320, 0.615661, 0.601815, 0.587785, 0.573576, 0.559193, 0.544639, 0.529919, 0.515038,
        0.500000, 0.484810, 0.469472, 0.453990, 0.438371, 0.422618, 0.406737, 0.390731, 0.374607, 0.358368,
        0.342020, 0.325568, 0.309017, 0.292372, 0.275637, 0.258819, 0.241922, 0.224951, 0.207912, 0.190809,
        0.173648, 0.156434, 0.139173, 0.121869, 0.104528, 0.087156, 0.069756, 0.052336, 0.034899, 0.017452,
        0.000000, -0.017452, -0.034899, -0.052336, -0.069756, -0.087156, -0.104528, -0.121869, -0.139173, -0.156434,
       -0.173648, -0.190809, -0.207912, -0.224951, -0.241922, -0.258819, -0.275637, -0.292372, -0.309017, -0.325568,
       -0.342020, -0.358368, -0.374607, -0.390731, -0.406737, -0.422618, -0.438371, -0.453990, -0.469472, -0.484810,
       -0.500000, -0.515038, -0.529919, -0.544639, -0.559193, -0.573576, -0.587785, -0.601815, -0.615661, -0.629320,
       -0.642788, -0.656059, -0.669131, -0.681998, -0.694658, -0.707107, -0.719340, -0.731354, -0.743145, -0.754710,
       -0.766044, -0.777146, -0.788011, -0.798636, -0.809017, -0.819152, -0.829038, -0.838671, -0.848048, -0.857167,
       -0.866025, -0.874620, -0.882948, -0.891007, -0.898794, -0.906308, -0.913545, -0.920505, -0.927184, -0.933580,
       -0.939693, -0.945519, -0.951057, -0.956305, -0.961262, -0.965926, -0.970296, -0.974370, -0.978148, -0.981627,
       -0.984808, -0.987688, -0.990268, -0.992546, -0.994522, -0.996195, -0.997564, -0.998630, -0.999391, -0.999848,
       -1.000000, -0.999848, -0.999391, -0.998630, -0.997564, -0.996195, -0.994522, -0.992546, -0.990268, -0.987688,
       -0.984808, -0.981627, -0.978148, -0.974370, -0.970296, -0.965926, -0.961262, -0.956305, -0.951057, -0.945519,
       -0.939693, -0.933580, -0.927184, -0.920505, -0.913545, -0.906308, -0.898794, -0.891007, -0.882948, -0.874620,
       -0.866025, -0.857167, -0.848048, -0.838671, -0.829038, -0.819152, -0.809017, -0.798636, -0.788011, -0.777146,
       -0.766044, -0.754710, -0.743145, -0.731354, -0.719340, -0.707107, -0.694658, -0.681998, -0.669131, -0.656059,
       -0.642788, -0.629320, -0.615661, -0.601815, -0.587785, -0.573576, -0.559193, -0.544639, -0.529919, -0.515038,
       -0.500000, -0.484810, -0.469472, -0.453990, -0.438371, -0.422618, -0.406737, -0.390731, -0.374607, -0.358368,
       -0.342020, -0.325568, -0.309017, -0.292372, -0.275637, -0.258819, -0.241922, -0.224951, -0.207912, -0.190809,
       -0.173648, -0.156434, -0.139173, -0.121869, -0.104528, -0.087156, -0.069756, -0.052336, -0.034899, -0.017452,
    };
    return lut[deg%360];
}
```

360 valeurs précalculées « en dur »

static variables/attributs/méthodes

- *Plutôt que de recalculer à chaque fois la valeur de la fonction f(x) en un ensemble de x finis, on peut pré-calculer tous les f(x) dans une Lookup Table (LUT)*

C ou C++

```
/// Calcul du sinus d'un angle "au degré près" (pas très précis)
double fastSin0(unsigned int deg)
{
    static const double lut[360] = {
        0.000000, 0.017452, 0.034899, 0.052336, 0.069756, ...
        0.173648, 0.190809, 0.207912, 0.224951, 0.241922, ...
        0.342020, 0.358368, 0.374607, 0.390731, 0.406737, ...
        0.500000, 0.515038, 0.529919, 0.544639, 0.559193, ...
        0.642788, 0.656059, 0.669131, 0.681998, 0.694658, ...
        0.766044, 0.777146, 0.788011, 0.798636, 0.809017, ...
        ... etc ...
    };

    return lut[deg%360];
}
```

100 millions de consultations d'un tableau : ~0.71s

static variables/attributs/méthodes

- *Véloce mais pas pratique !* *(alternatives => includes, rc files …)*
- *Détection static de 1er passage => générer au 1er appel*

C ou C++

```
double fastSin1(unsigned int deg)
{
    const double M_PI = 3.14159265358979323846;
    const int nbSamples = 360;
    static bool firstCall = true;
    static double lut[nbSamples];

    if (firstCall)
    {
        for (size_t d=0; d<360; ++d)
            lut[d] = sin(d*M_PI/180);
        firstCall = false;
    }

    return lut[deg%nbSamples];
}
```

Ce calcul est déroulé une seule fois au 1er appel

100 millions test + consultations d'un tableau : ~0.85s

static variables/attributs/méthodes

- *Avec les **constexpr** on peut demander au compilateur d'**exécuter un algorithme à la compilation***
- *Cette approche est (presque) aussi rapide que "en dur"*

C++ 17

```
double fastSin2(unsigned int deg)
{
    constexpr auto M_PI = 3.14159265358979323846;
    constexpr size_t nbSamples = 360;

    constexpr auto makeLut = []() constexpr {
        std::array<double, nbSamples> lut{};

        for (size_t d=0; d<lut.size(); ++d)
            lut[d] = sin(d*M_PI/180);

        return lut;
    };

    static const auto lut = makeLut();

    return lut[deg%nbSamples];
}
```

Ce calcul est déroulé par le compilateur !

100 millions de consultations d'un tableau : ~0.75s

static variables/attributs/méthodes

- *constexpr : exemple de philosophie « zero overhead »*
- *Une abstraction ne devrait pas coûter plus qu'un code équivalent sans l'abstraction*

FAQ What is the zero-overhead principle?

The zero-overhead principle is a guiding principle for the design of C++. It states that: *What you don't use, you don't pay for* (in time or space) and further: *What you do use, you couldn't hand code any better.*

In other words, no feature should be added to C++ which would make any existing code (not using the new feature) larger or slower, nor should any feature be added for which the compiler would generate code that is not as good as a programmer would create without using the feature.

static variables/attributs/méthodes

- *Dans une **classe** on peut déclarer un **membre static***
- *Pour un **attribut** cela indique une donnée unique de la classe plutôt que pour chaque objet de la classe*
- *Pour une **méthode** cela indique une action appelable sans partir d'un objet cible (pas de this)*
- *Les seuls attributs accessibles à une méthode static sont des attributs statics : il n'y a pas d'objet !*
- *Utilisations :*
  - *Manager la classe*
  - *Générer des identifiants uniques*
  - *Actions de classe en l'absence d'objet cible (fonctions auxiliaires, fabriques d'objets...)*

static variables/attributs/méthodes

- *Exemple d'usage classique : comptage d'instances*

```
/// **** pacMan.h ****
class PacMan
{
    public :
        PacMan(int posx, int posy)
            : m_posx{posx}, m_posy{posy} {
            ++s_nbPacMan;                 }

        ~PacMan()           {
            --s_nbPacMan;   }

        static void printHowMany() {
            std::cout << "number of PacMan = "
                      << s_nbPacMan << std::endl; }

    private :
        int m_posx, m_posy;
        static size_t s_nbPacMan;
};


/// **** pacMan.cpp ****
size_t PacMan::s_nbPacMan = 0;
```

static variables/attributs/méthodes

- *Exemple d'usage classique : comptage d'instances*

C++ 17

```
/// **** pacMan.h ****
class PacMan
{
    public :
        PacMan(int posx, int posy)
            : m_posx{posx}, m_posy{posy} {
            ++s_nbPacMan;                 }

        ~PacMan()           {
            --s_nbPacMan;   }

        static void printHowMany() {
            std::cout << "number of PacMan = "
                      << s_nbPacMan << std::endl; }

    private :
        int m_posx, m_posy;
        inline static size_t s_nbPacMan = 0;
};


/// **** ~~pacMan.cpp~~ **** pas besoin de définition en .cpp (header only)
```

static variables/attributs/méthodes

- *Exemple d'usage classique : comptage d'instances*

```
void application()
{
    std::vector<PacMan*> dynaPac(3, nullptr);
    PacMan autoPac{10,20};

    PacMan::printHowMany();

    dynaPac[0] = new PacMan{30, 40};
    dynaPac[1] = new PacMan{50, 60};
    dynaPac[2] = new PacMan{70, 80};
    PacMan::printHowMany();

    delete dynaPac[0];
    delete dynaPac[2];
    PacMan::printHowMany();
}

int main()
{
    PacMan::printHowMany();
    application();
    PacMan::printHowMany();
    return 0;
}
```

```
number of PacMan = 0
number of PacMan = 1
number of PacMan = 4
number of PacMan = 2
number of PacMan = 1
```

Fuite mémoire repérée

static variables/attributs/méthodes

- *Exemple plus complexe de « registering » d'instances*

```
class PacMan {
    public :
        using Id = unsigned long long;

        PacMan(int posx, int posy)
            : m_posx{posx}, m_posy{posy}, m_id{s_nextId++} {
            s_allPacMan[m_id] = this; }

        ~PacMan() { s_allPacMan.erase(m_id); }

        static PacMan* getById(Id id) {
            auto it = s_allPacMan.find(id);
            return it!=s_allPacMan.end() ? it->second : nullptr; }

        static void printHowMany() {
            std::cout << "number of PacMan = "
                      << s_allPacMan.size() << std::endl; }

        static void printById(Id id) {
            PacMan* pc = getById(id);
            if (pc) std::cout << pc->m_posx << " "
                              << pc->m_posy << std::endl; }

        /// Attention dangereux avec des automatiques
        static void deleteAll() {
            while ( s_allPacMan.size() )
                delete s_allPacMan.begin()->second; }
    private :
        int m_posx, m_posy;
        Id m_id;

        inline static Id s_nextId;
        inline static std::map<Id, PacMan*> s_allPacMan;
};
```

approche atypique
! static ≈ globales !

static variables/attributs/méthodes

- *Exemple plus complexe de « registering » d'instances*

```
int main()
{
    new PacMan(10, 20);
    new PacMan(30, 40);
    new PacMan(50, 60);
    PacMan::printHowMany();

    PacMan::printById(0);
    PacMan::printById(1);
    PacMan::printById(2);

    delete PacMan::getById(1);
    PacMan::printHowMany();

    PacMan::deleteAll();
    PacMan::printHowMany();

    return 0;
}
```

approche atypique
! static ≈ globales !

```
number of PacMan = 3
10 20
30 40
50 60
number of PacMan = 2
number of PacMan = 0
```

static variables/attributs/méthodes

- *! confusion classe / groupe d'instances de la classe !*

```
class PacManPack
{
    public :
        using Id = unsigned long long;

        ~PacManPack() {
            for (auto p : m_allPacMan)
                delete p.second;   }

        void addPacMan(PacMan* pm) {
            m_allPacMan[m_nextId++] = pm; }

        PacMan* getById(Id id) {
            auto it = m_allPacMan.find(id);
            return it!=m_allPacMan.end() ? it->second : nullptr; }

        void deleteById(Id id) {
            PacMan* pc = getById(id);
            if (pc) {
                delete pc;
                m_allPacMan.erase(id); } }

        void printById(Id id) {
            PacMan* pc = getById(id);
            if (pc) pc->print(); }

        void printHowMany() {
            std::cout << "pack has " << m_allPacMan.size()
                      << " PacMan" << std::endl; }

    private :
        Id m_nextId = 0;
        std::map<Id, PacMan*> m_allPacMan;
};
```

Utiliser une classe comme un groupe ?
=> classe groupe

static variables/attributs/méthodes

- *! confusion classe / groupe d'instances de la classe !*

```
void application()
{
    PacManPack pack;

    pack.addPacMan( new PacMan(10, 20) );
    pack.addPacMan( new PacMan(30, 40) );
    pack.addPacMan( new PacMan(50, 60) );
    pack.printHowMany();

    pack.printById(0);
    pack.printById(1);
    pack.printById(2);

    pack.deleteById(1);
    pack.printHowMany();

    /// Le pack est une variable automatique !
}

int main()
{
    PacMan::printHowMany();
    application();
    PacMan::printHowMany();
    return 0;
}
```

Destruction du pack
=> libération des PacMan

Utiliser une classe comme un groupe ?
=> classe groupe

```
number of PacMan = 0
pack has 3 PacMan
10 20
30 40
50 60
pack has 2 PacMan
number of PacMan = 0
```

COURS 12

A) Static variables/attributs/méthodes
B) Complément conteneurs/itérateurs
C) Exemples de prog. fonctionnelle

Complément conteneurs/itérateurs

- *Le conteneur **std::initializer_list** permet de déclarer des **listes de données littérales** de taille arbitraire*
- *Pratique pour définir des séquences de valeurs pour automatiser des séquences répétitives avec littérales*

```
struct Coords
{
    int x, y;
};

int main()
{
                std::initializer_list<int> // type déduit !

    for (auto val : {10, 20, 30, 40} )
        std::cout << val << std::endl;


                                std::initializer_list<Coords>

    for (const auto& param : {Coords{10, 20}, {30, 40}, {50, 60}} )
        std::cout << param.x << " " << param.y << std::endl;
```

```
10
20
30
40
10 20
30 40
50 60
```

Complément conteneurs/itérateurs

- *Le conteneur **std::initializer_list** permet de déclarer des **listes de données littérales** de taille arbitraire*
- *Pratique pour définir des séquences de valeurs pour automatiser des séquences répétitives avec littérales*

```
struct Coords
{
    int x, y;
};

int main()
{
    PacManPack pack;


    pack.addPacMan( new PacMan(10, 20) );
    pack.addPacMan( new PacMan(30, 40) );
    pack.addPacMan( new PacMan(50, 60) );


    for (const auto& param : {Coords{10, 20}, {30, 40}, {50, 60}} )
        pack.addPacMan( new PacMan(param.x, param.y) );
```

Complément conteneurs/itérateurs

- *Le conteneur **std::initializer_list** permet de déclarer des **listes de données littérales** de taille arbitraire*
- *Peut être passé en paramètre, permet de recevoir des arguments en quantité variable, init. de conteneurs ...*

```
class Etudiant
{
    public :
        Etudiant(std::string nom, std::initializer_list<float> notes)
            : m_nom{nom}, m_notes{notes} {}

    private :
        std::string m_nom;
        std::vector<float> m_notes;
};

int main()
{


    Etudiant etuA{"Paul Dupont"   , {12.0, 8.5, 17.0} };
    Etudiant etuB{"Martine Martin", {16.0, 9.5} };
```

Initialisation directe d'un vector par un initializer_list reçu en paramètre

Complément conteneurs/itérateurs

- *Le conteneur **std::initializer_list** permet de déclarer des **listes de données littérales** de taille arbitraire*

```
struct Coords
{
    int x, y;
};

class PacManPack
{
    public :

        PacManPack(std::initializer_list<Coords> params) {
            makeMany(params); }

        void makeMany(std::initializer_list<Coords> params) {
            for (const auto& param : params)
                addPacMan( new PacMan(param.x, param.y) ); }
    ...

};

int main()
{

    PacManPack pack{{10, 20}, {30, 40}, {50, 60}};

    pack.makeMany( {{70, 80}, {90, 100}} );
```

Complément conteneurs/itérateurs

- *Le conteneur **std::array** est une alternative aux tableaux natifs de **tailles fixes**...*
- *Avec des perfs. équivalentes, il transmet sa taille, a une sémantique par valeur, s'intègre avec les itérateurs… interface d'un conteneur standard, sans le coût d'un vector*

```
void modifTab(double tab[5]) {
    Tab[0] *= 2;              }

void modifArray(std::array<double, 5>& arr) {
    Arr[0] *= 2;                             }

int main()
{
    double tab[5] = {1.2, 2.3, 3.4, 4.5, 5.6};
    modifTab(tab);
    for (size_t i=0; i<sizeof(tab)/sizeof(*tab); ++i)
        std::cout << tab[i] << " ";
    std::cout << std::endl;

    std::array<double, 5>  arr = {1.2, 2.3, 3.4, 4.5, 5.6};
    modifArray(arr);
    for (size_t i=0; i<arr.size(); ++i)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
```

```
2.4 2.3 3.4 4.5 5.6
2.4 2.3 3.4 4.5 5.6
```

Complément conteneurs/itérateurs

- *Pour être complet il faut mentionner l'allocation **new[ ]** et **delete[ ]** dans le cas des tableaux dynamiques*
- *Compte tenu des problèmes de l'allocation dynamique il est **généralement** très préférable d'adopter vector !*

```
int main()
{
    size_t taille;
    std::cout << "Taille de votre tableau dynamique SVP : ";
    std::cin >> taille;

    double* tab = new double[taille];
    /// Utiliser tab ...
    for (size_t i=0; i<taille; ++i) tab[i] = 2*i;
    /// ...
    /// Libérer avec delete[]
    delete[] tab;

    std::vector<double> vec(taille);
    /// Utiliser vec ...
    for (size_t i=0; i<vec.size(); ++i) vec[i] = 2*i;
    /// ...
    /// Libération automatique (pas de new, pas de delete)

    return 0;
}
```

Complément conteneurs/itérateurs

- *En C++ combien de façons de parcourir un conteneur pour afficher les valeurs dedans ?*

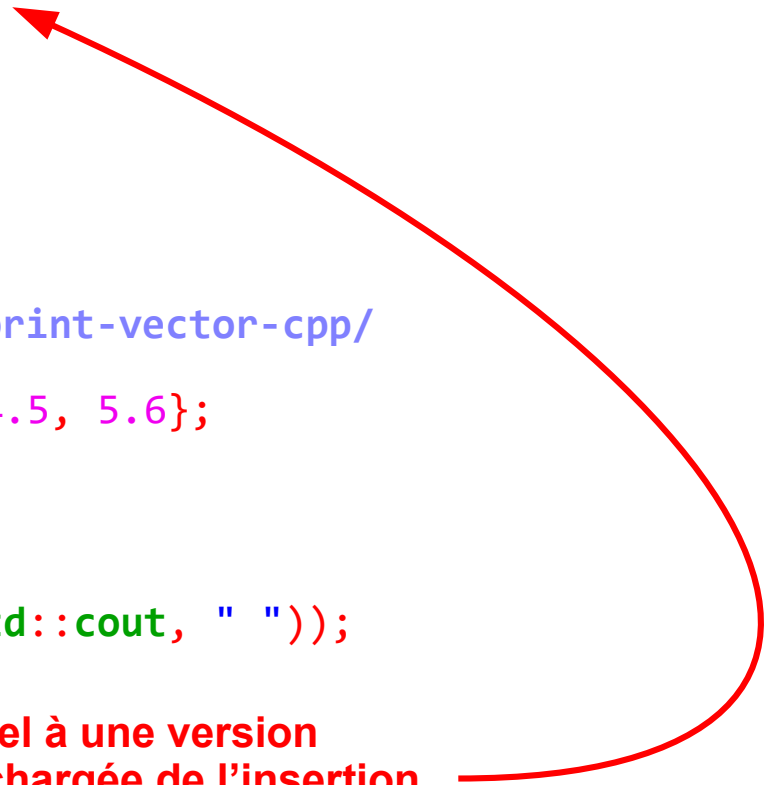
```cpp
int main()
{
    /// voir https://www.techiedelight.com/print-vector-cpp/

    std::vector<double> vec{1.2, 2.3, 3.4, 4.5, 5.6};

    /// range-based for
    for (auto val : vec)
        std::cout << val << " ";
    std::cout << std::endl;

    /// algorithm and lambda
    std::for_each(vec.cbegin(), vec.cend(),
                  [](auto v){std::cout << v << " ";});
    std::cout << std::endl;

    /// stream iterator
    std::copy(vec.begin(), vec.end(),
            std::ostream_iterator<double>(std::cout, " "));
    std::cout << std::endl;

    /// insertion operator overload
    std::cout << vec << std::endl;

    return 0;
}
```

!?

Complément conteneurs/itérateurs

- *En C++ combien de façons de parcourir un conteneur pour afficher les valeurs dedans ?*

```cpp
std::ostream& operator<<(std::ostream& os, const std::vector<double> &input)
{
    for (auto const& i: input) {
        os << i << " ";
    }
    return os;
}

int main()
{
    /// voir https://www.techiedelight.com/print-vector-cpp/

    std::vector vec<double>{1.2, 2.3, 3.4, 4.5, 5.6};


    /// stream iterator
    std::copy(vec.begin(), vec.end(),
            std::ostream_iterator<double>(std::cout, " "));
    std::cout << std::endl;

    /// insertion operator overload
    std::cout << vec << std::endl;

    return 0;
}
```



! Appel à une version surchargée de l'insertion vers un flot de sortie

Complément conteneurs/itérateurs

- *Généralisation de la surcharge de l'opérateur d'insertion vers flot de sortie pour tout type de vecteur*

```
template<typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T> &input)
{
    for (auto const& i: input) {
        os << i << " ";
    }
    return os;
}


class Etudiant
{
    public :
        Etudiant(std::string nom, std::initializer_list<float> notes)
            : m_nom{nom}, m_notes{notes} {}

        friend std::ostream& operator<<(std::ostream& os, const Etudiant& e) {
            os << "[etudiant " << e.m_nom << " : " << e.m_notes << "]";
            return os; }

    private :
        std::string m_nom;
        std::vector<float> m_notes;
};
```

Complément conteneurs/itérateurs

- *Généralisation de la surcharge de l'opérateur d'insertion vers flot de sortie pour tout type de vecteur*

```cpp
template<typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T> &input)
{
    for (auto const& i: input) {
        os << i << " ";
    }
    return os;
}

int main()
{
    std::vector<int> vecInt{2, 3, 4};
    std::vector<float> vecFloat{5.4, 6.3, 7.2};
    std::vector<std::string> vecString{"Hello", "world", "!"};
    std::vector<Etudiant> vecEtudiant{
        {"Paul Dupont"   , {12.0, 8.5, 17.0} },
        {"Martine Martin", {16.0, 9.5} }  };

    std::cout << vecInt << std::endl;
    std::cout << vecFloat << std::endl;
    std::cout << vecString << std::endl;
    std::cout << vecEtudiant << std::endl;

    return 0;
}
```

```
2 3 4
5.4 6.3 7.2
Hello world !
[etudiant Paul Dupont : 12 8.5 17 ] [etudiant Martine Martin : 16 9.5 ]
```

Complément conteneurs/itérateurs

- *Généralisation de la surcharge de l'opérateur d'insertion vers flot de sortie pour tout type de vecteur*

```cpp
template<typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T> &input)
{
    for (auto const& i: input) {
        os << i << (std::is_pod<T>::value ? " " : "\n");
    }
    return os;
}


int main()
{
    std::vector<int> vecInt{2, 3, 4};
    std::vector<float> vecFloat{5.4, 6.3, 7.2};
    std::vector<std::string> vecString{"Hello", "world", "!"};
    std::vector<Etudiant> vecEtudiant{
        {"Paul Dupont"   , {12.0, 8.5, 17.0} },
        {"Martine Martin", {16.0, 9.5} }  };

    std::cout << vecInt << std::endl;
    std::cout << vecFloat << std::endl;
    std::cout << vecString << std::endl;
    std::cout << vecEtudiant << std::endl;

    return 0;
}
```

Détection de type primitif (int, float, char…)
pod = plain old data

```
2 3 4
5.4 6.3 7.2
Hello
world
!

[etudiant Paul Dupont : 12 8.5 17 ]
[etudiant Martine Martin : 16 9.5 ]
```

Complément conteneurs/itérateurs

- *Et qu'est-ce que c'est qu'un stream_iterator ?*
  Un moyen d'itérer sur des (éléments dans des) flots !

```
1.2 2.3 3.4 4.5 5.6
```

```cpp
int main()
{
    std::vector<double> vec{1.2, 2.3, 3.4, 4.5, 5.6};


    /// ostream iterator
    std::copy(vec.begin(), vec.end(), std::ostream_iterator<double>(std::cout, " "));


    /// l'algorithme std::copy ci dessus est équivalent à
    std::ostream_iterator<double> outputIter(std::cout, " ");
    for (auto it=vec.cbegin(); it!=vec.cend(); ++it)
    {
        *outputIter = *it; /// écrire sur outputIter insert dans le flot associé
        outputIter++; /// ceci est inutile dans le cas particulier ostream_iterator
    }

    return 0;
}
```

Complément conteneurs/itérateurs

- *Pour rappel grâce aux istringstream on peut traiter une source textuelle (string) comme un flot entrant*

```
int main()
{
    std::string stringData = "100 260 410 120";

    std::vector<int> vec;

    std::istringstream iss{stringData};
    int val;
    while (iss>>val)
        vec.push_back(val);

    std::cout << vec << std::endl;

    return 0;
}
```

```
100 260 410 120
```

Complément conteneurs/itérateurs

- *Et une istringstream peut devenir itérable avec un istream_iterator… ci dessous version longue*

```cpp
int main()
{
    std::string stringData = "100 260 410 120";

    std::vector<int> vec;

    std::istringstream iss{stringData};
    std::istream_iterator<int> inputIter{iss};
    std::istream_iterator<int> endOfInput;
    while ( inputIter != endOfInput )
    {
        vec.push_back(*inputIter);
        inputIter++;
    }

    std::cout << vec << std::endl;

    return 0;
}
```

```
100 260 410 120
```

Complément conteneurs/itérateurs

- *Ce qui est itérable est utilisable dans les algorithmes*

```cpp
int main()
{
    std::string stringData = "100 260 410 120";
    std::vector<int> vec;

    std::istringstream iss{stringData};
    std::copy(std::istream_iterator<int>{iss},
              std::istream_iterator<int>{},
              std::back_inserter(vec));

    std::cout << vec << std::endl;

    return 0;
}
```

Itérateur en insertion :
équivaut à une séquence de push_back

```
100 260 410 120
```

Complément conteneurs/itérateurs

- *Ré-écriture des opérateurs surchargés génériques sans boucle explicite*

```
template<typename T>
std::ostream& operator<<(std::ostream& os, const std::vector<T> &input)
{
    std::copy(input.begin(), input.end(),
              std::ostream_iterator<T>(os, " "));
    return os;
}


template<typename T>
std::istream& operator>>(std::istream& is, std::vector<T> &output)
{
    std::copy(std::istream_iterator<T>{is},
              std::istream_iterator<T>{},
              std::back_inserter(output));
    return is;
}
```

Complément conteneurs/itérateurs

- *On a encore besoin d'un objet istringstream intermédiaire...*

```cpp
int main()
{
    std::string stringData = "100 260 410 120";
    std::vector<int> vec;

    std::istringstream iss{stringData};
    iss >> vec;

    std::cout << vec << std::endl;

    return 0;
}
```

```
100 260 410 120
```

Complément conteneurs/itérateurs

- *Encapsulons la séquence dans un wrapper générique, l'objectif est de pouvoir passer de string à vector :*

```cpp
template<typename T>
std::vector<T> vecFromString(const std::string& str)
{
    std::vector<T> vec;
    std::istringstream iss{str};
    iss >> vec;
    return vec;
}


int main()
{
    std::string strInt = "2 3 4";
    std::string strFloat = "5.4 6.3 7.2";
    std::string strString = "Hello world !";

    auto vecInt = vecFromString<int>(strInt);
    auto vecFloat = vecFromString<float>(strFloat);
    auto vecString = vecFromString<std::string>(strString);

    std::cout << vecInt << std::endl;
    std::cout << vecFloat << std::endl;
    std::cout << vecString << std::endl;

    return 0;
}
```

```
2 3 4
5.4 6.3 7.2
Hello world !
```

COURS 12

A) Static variables/attributs/méthodes
B) Complément conteneurs/itérateurs
C) Exemples de prog. fonctionnelle

Exemples de prog. fonctionnelle

- *Le polymorphisme permet d'encapsuler des actions spécifiques dans des classes spécialisées avec une interface d'utilisation définie par une classe mère abstraite pure (classe interface, non instanciable)*
- ***Soit l'énoncé suivant*** *: réaliser un programme permettant d'avoir un menu utilisateur avec sélection d'un choix par valeur entière et différentes actions associées* ***sans test ni switch/case***
- *Le principe est de constituer une hiérarchie d'actions de menu avec une classe interface parente, d'avoir un vecteur d'objets spécialisés correspondant aux choix : à chaque indice du vecteur est associé une instance d'une classe spécifique avec l'action désignée*

  => dispatch polymorphe (à la place du switch)

Exemples de prog. fonctionnelle

- *Réaliser un programme permettant d'avoir un menu utilisateur **sans test ni switch/case***

```
int main()
{
    Context ctx{3, 3.14, "pi"};

    std::vector<MenuEntry*> menu {
        new MenuQuitter,
        new MenuSaisir,
        new MenuAfficher
    };

    bool terminate = false;
    do
    {
        std::cout << "\n0/ Quitter\n1/ Saisir\n2/ Afficher\n\n";

        int choix;
        std::cin >> choix;

        /// Appel polymorphe !
        menu[choix]->action(ctx, terminate);
    }
    while (!terminate);

    for(auto m : menu) delete m;
    return 0;
}
```

```
struct Context
{
    int x;
    double y;
    std::string z;
};
```

```
0/ Quitter
1/ Saisir
2/ Afficher

2
3 3.14 pi

0/ Quitter
1/ Saisir
2/ Afficher

1
5 2.125 hop

0/ Quitter
1/ Saisir
2/ Afficher

2
5 2.125 hop

0/ Quitter
1/ Saisir
2/ Afficher

0

Process returned 0
```

Exemples de prog. fonctionnelle

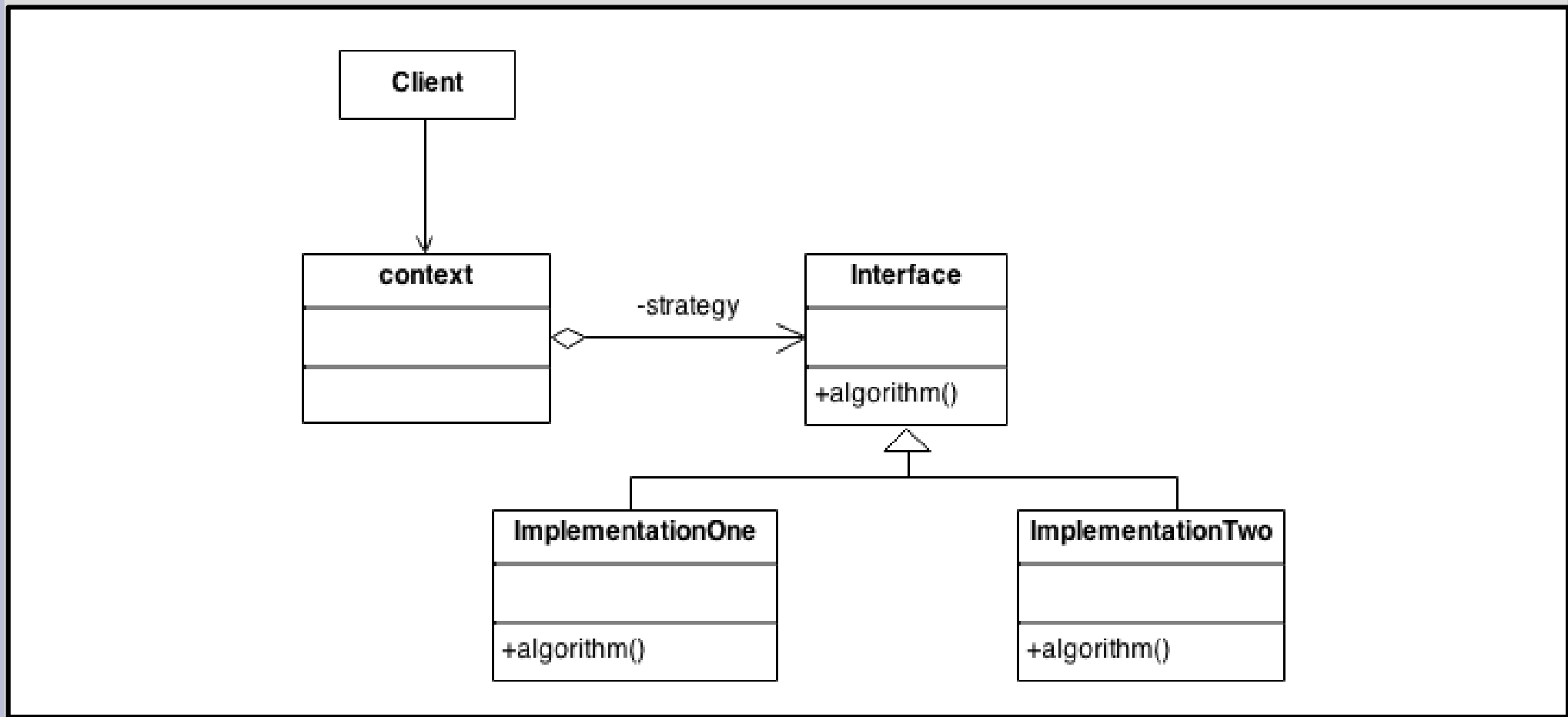
- *Réaliser un programme permettant d'avoir un menu utilisateur* ***sans test ni switch/case***

```
class MenuEntry
{
    public :
        virtual ~MenuEntry() = default;
        virtual void action(Context& ctx, bool& terminate) = 0;
};

class MenuQuitter : public MenuEntry
{
    public :
        void action(Context& ctx, bool& terminate) {
            terminate = true; }
};

class MenuSaisir : public MenuEntry
{
    public :
        void action(Context& ctx, bool& terminate) {
            std::cin >> ctx.x >> ctx.y >> ctx.z; }
};

class MenuAfficher : public MenuEntry
{
    public :
        void action(Context& ctx, bool& terminate) {
            std::cout << ctx.x << " " << ctx.y
                      << " " << ctx.z << "\n"; }
};
```

```
struct Context
{
    int x;
    double y;
    std::string z;
};
```

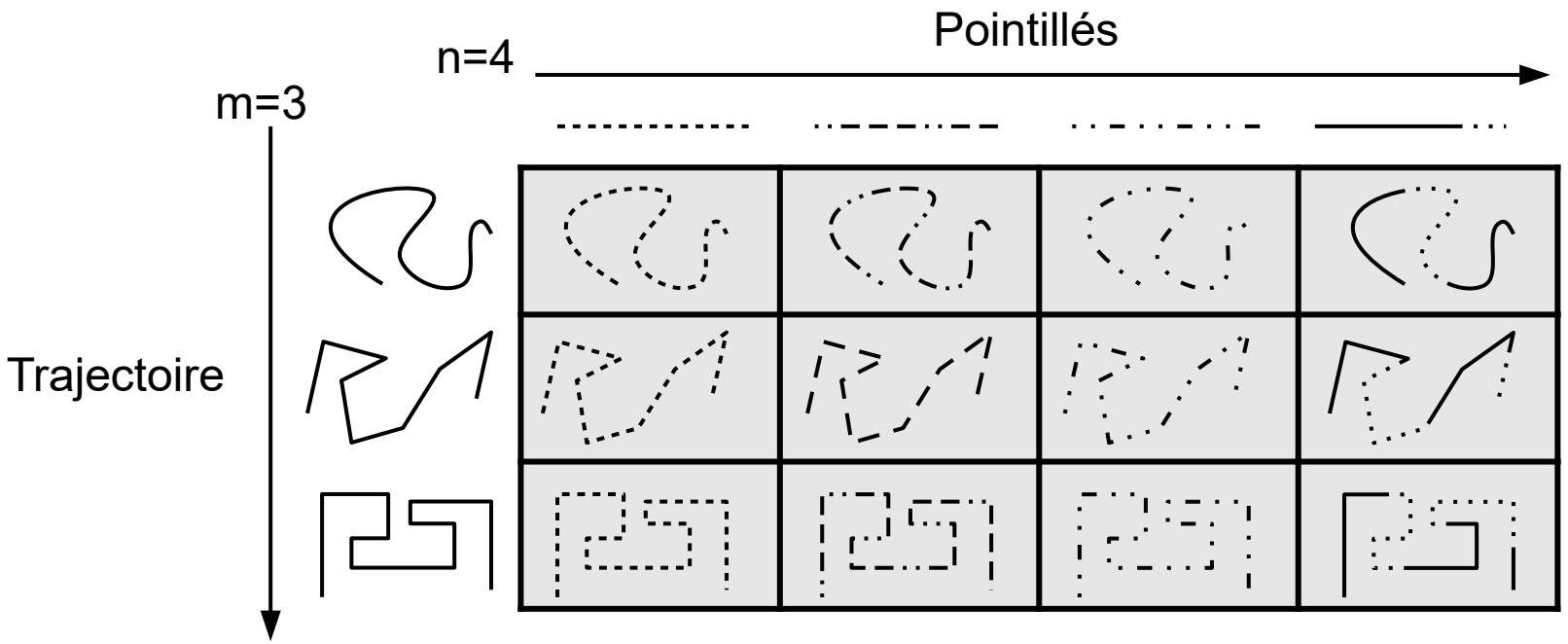
Exemples de prog. fonctionnelle

- Le *pattern strategy* est un usage particulier de délégation : on délègue à une classe qui hérite d'une interface.



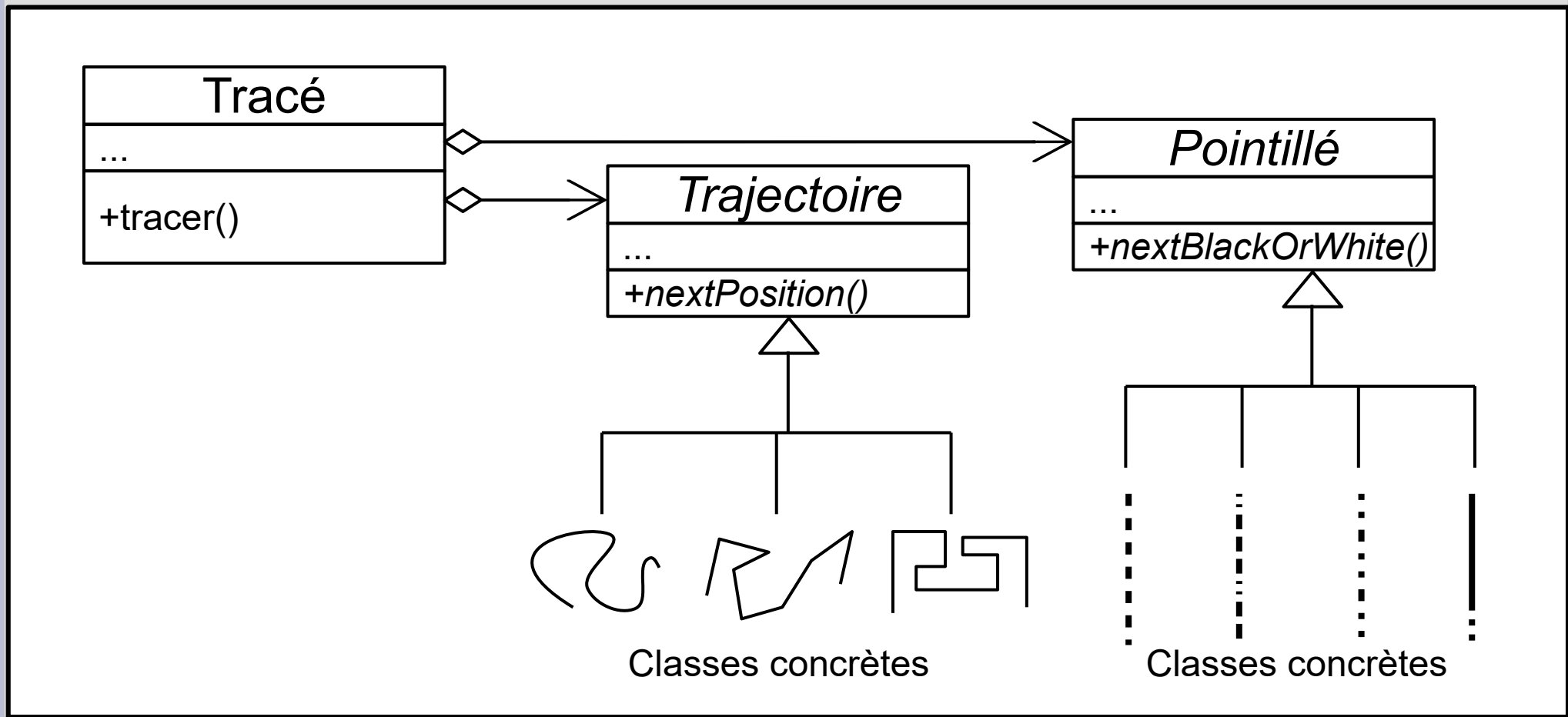
Exemples de prog. fonctionnelle

- On dispose d'une technique puissante pour gérer des **combinatoires** sans multiplier les codes croisés (m+n au lieu de m×n)



Exemples de prog. fonctionnelle

- Néanmoins l'héritage introduit des **couplages** : les classes filles concrètes dépendent des classes mères. Peut on faire autrement ?



Exemples de prog. fonctionnelle

- Pas forcément besoin de classes, une stratégie peut être donnée par des paramètres fonctionnels

```
using namespace std::complex_literals;
using vec2d = std::complex<double>;

template<typename Trajectoire, typename Pointilles>
void tracer(Svgfile& svgout, vec2d depart, double distance,
            Trajectoire trajectoire, Pointilles pointilles)
{
    vec2d pos = depart;
    double s = 0.0; /// Abscisse curviligne

    while ( abs(s) < distance )
    {
        vec2d step  = trajectoire(s);
        double larg = pointilles(s);

        vec2d nextpos = pos + step;
        s += abs(step);

        svgout.addLine( pos.real(),     pos.imag(),
                        nextpos.real(), nextpos.imag(), larg );
        pos = nextpos;
    }
}
```

Appels aux fonctions reçues en paramètres

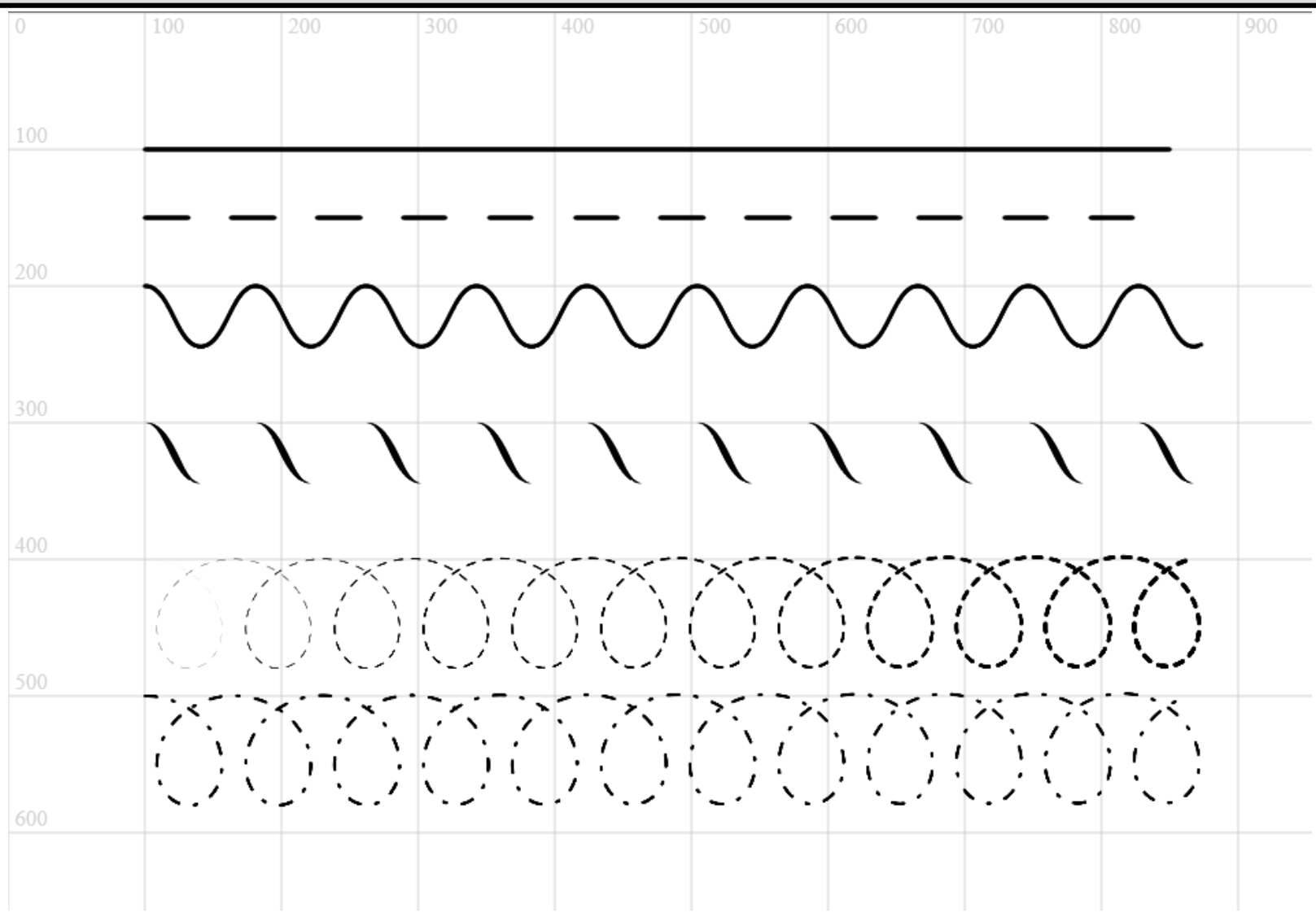
Exemples de prog. fonctionnelle

- Les **lambdas** sont des objets-fonctions anonymes qui peuvent être utilisés comme des "littéraux fonctionnels"

```
int main()
{
    Svgfile svgout;
    svgout.addGrid();

    tracer(svgout, 100.+100.i, 750,
           [](double s) { return 1.0; },
           [](double s) { return 3.0; }  );

    tracer(svgout, 100.+150.i, 750,
           [](double s) { return 1.0; },
           [](double s) { return sin(0.1*s)<0 ? 0 : 3; }  );

    tracer(svgout, 100.+200.i, 1200,
           [](double s) { return 2.0i*sin(0.05*s) + 1.0; },
           [](double s) { return 3; }  );

    tracer(svgout, 100.+300.i, 1200,
           [](double s) { return 2.0i*sin(0.05*s) + 1.0; },
           [](double s) { return 5*sin(0.05*s); }  );

    tracer(svgout, 100.+400.i, 3000,
           [](double s) { return 0.2*exp(0.025i*s) + 0.1; },
           [](double s) { return sin(0.5*s)<0 ? 0 : s/1000; }  );

    const std::array<int, 15> points{2,2,2,2,0,0,0,0,0,2,0,0,0,0,0};
    tracer(svgout, 100.+500.i, 3000,
           [](double s) { return 0.2*exp(0.025i*s) + 0.1; },
           [&points](double s) {
               return points[points.size()*(.03*s-floor(.03*s))]; }  );
```

Exemples de prog. fonctionnelle

- Ce qui permet une souplesse maximale dans l'utilisation d'une fonction paramétrable en stratégies



Exemples de prog. fonctionnelle

- L'évolution du langage intègre les « patterns »
  et les patterns finissent par disparaître dans le langage

Effective GoF Patterns - Tobias Darm (video)

Evolution

- Assembler → if/else, do-while, ...
- C → class, polymorphism, ...
- C++ → Strategy, Command, ...
- C++11 → ...