Question **3**

Correct

Note de 3,00 sur 3,00

Soit la classe Piece ci-dessous. On aimerait pouvoir créer de nouveaux objets à partir d'une pièce déjà existante.

La copie d'un pièce doit respecter ces règles:

- Un objet copié doit avoir son identifiant (id_) incrémenté de 1 par rapport à l'original.
- Un objet copié doit avoir les même fabriquants que l'objet de base. Cependant ajouter un fabriquant après la copie ne doit impacter que l'objet concerné.
- Tous les autres attributs doivent être copiés tels qu'ils sont.

Vous faites vos définitions à l'extérieur de la classe et vous ne pouvez pas changer les signatures des méthodes ou en ajouter de nouvelles par rapport aux déclarations dans la classe ci-dessous.

```
class Piece
{
public:
        Piece(const string& nom, const unsigned int
id) :
                nom_(nom), id_(id)
        {
                maxFab_    = 10;
                nombreFab_ = 0;

                fabriquants_ = new string[maxFab_];
        }

        Piece(const Piece& p);

        ~Piece()
        {
                delete[] fabriquants_;
        }

        void ajouterFab(const string& fab);

        string getNom() { return nom_; }
        unsigned getId() { return id_; }
        unsigned getPrix() { return prix_; }
        unsigned getNombreFab() { return nombreFab_; }
        unsigned getNombreFabMax() { return maxFab_; }
        string* getFabriquants() { return
fabriquants_; }

private:
        string   nom_;
        unsigned id_;
        double   prix_;

        unsigned maxFab_;
        unsigned nombreFab_;

        string* fabriquants_;
};
```

Par exemple:

Test	Résultat

`Piece p6("roue", 0);`
`Piece p7(p6);`
 	
`cout << (p7.getId() == p6.getId() + 1);`
`cout << (p7.getPrix() == p6.getPrix());`
`cout << (p7.getNom() == p6.getNom());`
`cout << (p7.getNombreFab() == p6.getNombreFab());`
`cout << (p7.getNombreFabMax() == p6.getNombreFabMax());`
	11111

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
//TODO: Permettre la construction par copie comme décrit dans l'énoncé.
Piece::Piece(const Piece& p) :
nom_(p.nom_), id_(p.id_ + 1), prix_(p.prix_),
nombreFab_(p.nombreFab_), maxFab_(p.maxFab_)
{
    fabriquants_ = new string[maxFab_];
```

```
    for(auto i : range(nombreFab_)){
        fabriquants_[i] = p.fabriquants_[i];
    }
}
```

Tous les tests ont été réussis ! ✔

Solution de l'auteur de la question (Cpp):

```
Piece::Piece(const Piece& p) : nom_(p.nom_), id_(p.id_ + 1), prix_(p.prix_),
                        maxFab_(p.maxFab_), nombreFab_(p.nombreFab_)
{
    fabriquants_ = new std::string[maxFab_];
    for(auto i : range(nombreFab_))
        fabriquants_[i] = p.fabriquants_[i];
}

```

Correct

Note pour cet envoi : 3,00/3,00.

Question **4**

Correct

Note de 3,00 sur 3,00

- Modifiez le code déjà écrit afin de rendre la classe Point générique (ses attributs seront d'un type T).
- Vous devez aussi mettre la définition du constructeur à l'extérieur de la classe plutôt que dans la classe comme il l'est actuellement (cette partie de la question n'est pas testée par les tests CodeRunner).
- Ne changez pas les noms des attributs et laissez-les public.

Nous voulons qu'un code comme le suivant ait deux points dont les types des attributs sont différents (on compile en C++17):

```
Point pointInt(3, 4);
Point pointDouble(3.5, 4.5);
```

Réponse : (régime de pénalités : 0 %)

Réinitialiser la réponse

```
template <typename T>
class Point
{
public:
    Point(T i, T j);

    T i_;
    T j_;
};

template <typename T>
Point<T>::Point(T i, T j): i_(i), j_(j) {}
```

	Test	Résultat attendu	Résultat obtenu	
✔	`Point p(3, 4);` `cout << is_same_v<decltype(p.i_), int> << is_same_v<decltype(p.j_), int>;`	11	11	✔
✔	`Point p(3, 4);` `cout << (p.i_ == 3) << (p.j_ == 4);`	11	11	✔
✔	`Point p(3.5, 4.5);` `cout << is_same_v<decltype(p.i_), double> << is_same_v<decltype(p.j_), double>;`	11	11	✔
✔	`Point p(3.5, 4.5);` `cout << (p.i_ == 3.5) << (p.j_ == 4.5);`	11	11	✔

Tous les tests ont été réussis ! ✔

Solution de l'auteur de la question (Cpp):

```
template <typename T>
class Point
{
public:
    Point(T i, T j);

    T i_;
    T j_;
};

template <typename T>
Point<T>::Point(T i, T j): i_(i), j_(j) {}
```

Correct

Note pour cet envoi : 3,00/3,00.