Langage C++

- CIR2 -

TP6: Composition

Nils Beaussé (nils.beausse@isen-ouest.yncrea.fr)

Partie 1 : Classe Point :

Implémentez une classe **Point** comprenant :

1. Deux attributs privés : x (abscisses), y (ordonnées)
2. Un constructeur comprenant deux arguments
3. Un constructeur de copie
4. Les accesseurs et les manipulateurs

Contrainte : L'utilisation d'un constructeur par défaut est interdite dans la classe Point

Partie 2: Classe Poney :

Dans cet exercice on cherche à implémenter une classe poney pour déclarer plusieurs poneys qui pourront vagabonder librement dans un champ en 2D (chaque poney a donc une position X/Y).
Les poneys n'aiment pas la redondance ! Pour leur faire plaisir, on va tenter de factoriser au maximum le code grâce aux méthodes vues dans le cours, notamment l'appel des constructeurs des classes encapsulées dans la classe Poney.

Pour construire cette classe poney, vous implémenterez :

1. Ces deux attributs privés :

Nom	Type	Description
identifier	char*	Nom du Poney
position	Point	Position du Poney (en X/Y)

2. Un constructeur par défaut (valeurs par défaut : (0,0) pour la position, "" pour l'identifiant). **Contrainte** : L'utilisation de Point::setX() ou Point::setY() est interdite
3. Un constructeur de deux arguments. **Contrainte** : L'utilisation de Point::setX() ou Point::setY() est interdite
4. Un destructeur. **Contrainte** : L'utilisation de Point::setX() ou Point::setY() est interdite
5. Un constructeur de copie. **Contrainte** : L'utilisation de Point::setX() ou Point::setY() est interdite
6. Les accesseurs et les manipulateurs
7. Une méthode float moveTo(p) qui déplace le poney au point p et renvoie la distance parcourue (choisir le bon type d'argument)
8. Une méthode float moveTo(p2) qui déplace le poney à la position d'un second poney p2 et renvoie la distance parcourue pour que les deux poneys broutent ensemble (choisir le bon type d'argument).
9. Une méthode float moveTo(x,y) qui déplace le poney aux coordonnées (x,y) et renvoie la distance parcourue (choisir le bon type d'argument)
10. Une méthode void print() qui affiche les caractéristiques du point sous la forme : [identifier] (x,y).
    Exemple: [PINKY PIE LA PONEYTE] position : (34,10)
11. Une méthode void reset() qui re-initialise la position à (0,0)

Partie 3: Tests :

1. Soit la fonction de test suivante :

```
void PoneyIdTest() {
   Point position(0,10);
   char identifier[] = "Fluttershy la poneyte";
   Poney poney(identifier, position);
   poney.print(); // affichage 1
   char * retrieveId = poney.getIdentifier();
   retrieveId[0] = 'B';
   poney.print(); // affichage 2
}
```

Si votre code compile, vous obtiendrez l'affichage 1 :

```
[Fluttershy la poneyte] Position : (0,10)
```

Mais l'affichage 2 produira :

```
[Bluttershy la poneyte] Position : (0,10)
```

Avez-vous une explication ?

Changez le prototype de getIdentifier() en :

```
const char* getIdentifier() const;
```

Ceci résout-il votre problème ? Pourquoi ?
Que signifient les deux const ? Et si j'écris const char* const getIdentifier() const ? Que signifie le troisième const ajoutée ?

2. Soit la fonction de test suivante :

```
void poneyIdTest2() {
   Point position(0,10);
   char identifier[] = "Rainbow dash";
   Poney poney(identifier, position);
   poney.print(); // affichage 1
   identifier[0] = 'X';
   poney.print(); // affichage 2 }
```

Vérifier si l'affichage est bien :

```
[Rainbow dash] Position : (0,10)
```

3. Essayer de prévoir l'affichage de la fonction de test suivante et exécuter-là par la suite pour vérifier vos réponses :

```
void poneyCopyTest(){
   Poney p1("Rarity", Point(0,0));
   Poney p2 = p1;
   p1.setIdentifier("Twilight");
   p1.print();
   p2.print();
}
```

4. Essayer de prévoir l'affichage de fonction de test suivante et exécutez-là par la suite pour vérifier vos réponses :

```
void poneyMoveTests(){
   Poney p1("Twilight", Point(0,0));
```

```
    Poney p2("Spike", Point(0,10));

    Point p = p1.getPosition();
    float distance = 0;

    distance = p1.moveTo(p2);
    std::cout << "distance : " << distance << std::endl; // affichage 1
    p1.print(); // affichage 2

    p.setY(50);
    std::cout << p.getY();
    p1.print(); // affichage 3
    distance = p1.moveTo(p);

    std::cout << "distance : " << distance << std::endl; // affichage 4
    p1.print(); // affichage 5
    distance = p1.moveTo(0, 80);

    std::cout << "distance : " << distance << std::endl; // affichage 6
    p1.print(); // affichage 7
    p2.print(); // affichage 8
}
```

Partie 4 : Dépendance :

Les poneys sont drogués au FOIN ! Et ils sont très possessif envers leur botte de foin. Chaque poney ne peut avoir qu'une botte de foin et marque sa botte pour que les autres ne la lui vole pas.

Implémentez une nouvelle classe « Foin » qui :

1. Contiendra un nombre de brin de foin, qui ne pourra pas descendre en dessous de 0.
2. Contiendra des accesseurs et mutateurs pour ces brins de foins (« manger » et « remplir » pour les mutateurs et getFoin pour les accesseurs). Ils devront être sécurisés pour éviter d'avoir un nombre négatif de brin de foin ou pour remplir au-delà d'une certaine limite !
3. Permettra d'avoir un indicateur permettant d'atteindre la classe poney qui possède cette botte de foin.

Modifiez la classe Poney pour contenir la classe botte de foin (car le poney porte souvent la botte sur lui).

Quel est l’éventuel problème vu en cours ?

Comment le résoudre ?

Implémentez la solution et testez là !