Programmation orientée objet

Arnaud Labourel

Le penser objet

Analyse d'un problème en programmation objet

Penser objet : décomposer le programme en objets

- Quels sont les objets nécessaires à la résolution du problème ?

  $\Rightarrow$ décomposition du problème en objets

- À quels modèles des objets correspondent-il ?

  $\Rightarrow$ Quelles sont les classes ?

- Quels sont les fonctionnalités/opérations dont on doit/veut pouvoir disposer sur ces objets ?

  $\Rightarrow$ Quelles sont les méthodes des classes ?

- Quelle est la structure des données de l'objet ?

  $\Rightarrow$ Quelles sont les attributs des classes ?

Exemple de problème

- un catalogue regroupe des articles, il permet de trouver un article à partir de sa référence.
- un article est caractérisé par un prix et une référence que l'on peut obtenir. On veux aussi pouvoir déterminer si un article est plus cher qu'un autre
- une commande est créée pour un client et un catalogue donnés, on peut ajouter des articles à une commande, accéder à la liste des articles commandés ainsi que prix total des articles et le montant des frais de port de la commande.
- un client peut créer une commande pour un catalogue et commander dans cette commande des articles à partir de leur référence.

Classes du problèmes

- un **catalogue** regroupe des **articles**, il permet de trouver un article à partir de sa **référence**.
- un **article** est caractérisé par un prix et une référence que l'on peut obtenir. On veux aussi pouvoir déterminer si un article est plus cher qu'un autre
- une **commande** est créée pour un **client** et un **catalogue** donnés, on peut ajouter des **articles** à une commande, accéder à la liste des articles commandés ainsi que prix total des articles et le montant des frais de port de la commande.
- un **client** peut créer une **commande** pour un catalogue et commander dans cette commande des **articles** à partir de leur référence.

Méthodes

- un catalogue regroupe des articles, il permet de **trouver** un article à partir de sa référence.
- un article est caractérisé par un prix et une référence que l'on **peut obtenir**. On veut aussi pouvoir **déterminer** si un article est plus cher qu'un autre.
- une commande est créée pour un client et un catalogue donnés, on peut **ajouter** des articles à une commande, **accéder** à la liste des articles commandés ainsi que le prix total des articles et le montant des frais de port de la commande.
- un client peut **créer** une commande pour un catalogue et **commander** dans cette commande des articles à partir de leur référence.

Méthodes de la classe `Catalog`

Description d'un catalogue

un catalogue regroupe des articles, il permet de **trouver** un article à partir de sa référence.

Méthodes :

- `get_item(self, reference)`

Méthodes de la classe `Item`

Description d'un article

un article est caractérisé par un prix et une référence que l'on **peut obtenir**.
On veut aussi pouvoir **déterminer** si un article est plus cher qu'un autre.

Méthodes :

- `get_price(self)`
- `get_reference(self)`
- `is_more_expensive_than(self, other_item)`

Constructeurs et méthodes de la classe `Order`

Description d'une commande

une commande est **créée** pour un client et un catalogue donnés, on peut **ajouter** des articles à une commande, **accéder** à la liste des articles commandés ainsi que le prix total des articles et le montant des frais de port de la commande.

Méthodes et constructeurs :

- `__init__(self, client, catalog)` (constructeur)
- `add_item(self, item)`
- `get_all_items(self)`
- `get_total_price_of_items(self)`
- `get_shipping_cost(self)`
- `get_client(self)`
- `get_catalog(self)`

Constructeurs et méthodes de la classe `Client`

Description d'un client

un client peut **créer** une commande pour un catalogue et **commander** dans cette commande des articles à partir de leur référence.

Méthodes :

- `create_order(self, catalog)`
- `order_item(self, order, reference)`

Vocabulaire programmation objet

Objet

Un **objet** :

- peut être **construit**
- est **structuré** : il est constitué d'un ensemble d'**attributs** (données de l'objet)
- possède un **état** : la valeur de ses attributs
- possède une **interface** : les opérations applicables sur l'objet appelées **méthodes**

Syntaxe des objets

Construire un objet

```
objet = Classe(arguments);
```

Accéder à un attribut d'un objet

```
objet.attribut = ...
```

Appeler une méthode d'un objet

```
objet.méthode(arguments)
```

Classe

Une **classe** (d'objet) définit des :

- **constructeurs** : des façons de construire/instancier les objets (**instances** de la classe)
- **attributs** (champs, propriétés ou données membres) : la structure des objets de la classe
- **méthodes** : le comportement des objets de la classe

Exemple de syntaxe de définition d'une classe

```
class NameOfTheClass:
  # Constructor
  def __init__(self, argument1, argument2):
    self.attribute1 = argument1
    self.attribute2 = argument2

  # Method
  def get_attribute2(self):
    return self.attribute2
```

Héritage

Une classe peut hériter d'une autre classe. Dans ce cas, cette classe appelée classe *fille* (*subclass*) hérite des méthodes de la classe dite *mère* (*superclass*).

Pour créer une classe fille, on utilise l'instruction `class`, suivie comme d'habitude du nom que l'on veut attribuer à la nouvelle classe, et on place entre parenthèses le nom de la classe mère.

```
class ClasseFille(ClasseMère):
  pass
```

Fonction super

La fonction `super()` permet d'accéder aux méthodes de la classe mère. Par exemple dans le code ci-dessous, cela nous permet d'accéder à la méthode `__init__` de `Point` dans `ColoredPoint` :

```
class Point:
  def __init__(self, x, y):
    self.x = x
    self.y = y

class ColoredPoint(Point):
  def __init__(self, x, y, color):
    super().__init__(x, y)
    self.color = color
```

Pièges de l'héritage

Bonnes utilisations de l'héritage

- Créer une nouvelle classe avec des fonctionnalités supplémentaires.
- Éviter la duplication de code entre deux classes en créant une classe mère contenant le code partagé.

Mauvaises utilisations de l'héritage

- Créer une classe fille ne correspondant pas à la classe mère : par exemple, avoir une classe voiture héritant d'une classe moteur (une voiture n'est pas un moteur même si on peut démarrer les deux).
- Créer une classe fille ne se comportant pas comme la classe mère : par exemple, avoir une classe carré héritant d'une classe rectangle (une méthode changeant la longueur n'aura pas le même comportement pour un carré et un rectangle).