


Python でファクトリメソッド？

ナイトウ 

Press Space for next page →

このスライドはこちらで

<https://kosuke222naito.github.io/20240803-It/>





- [!\[\]\(6302aad5aed157b291fddf37b4870784_img.jpg\) 旧 Twitter !\[\]\(a9ca2c237943a6d0a9f22252f295b6f3_img.jpg\)](#)
- [!\[\]\(9a01a64e0b4ff865df7d32ee7991fe8b_img.jpg\) GitHub !\[\]\(6aefe9a3d997eb8b55c40ecd5fa7053f_img.jpg\)](#)
- [!\[\]\(baa8f8ba8c970db55300f5bb45bb3460_img.jpg\) Zenn !\[\]\(a6e28495607b2299466d3d5d3193848c_img.jpg\)](#)

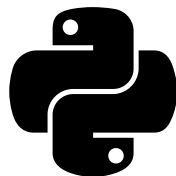


Table of Contents

1. [Python でファクトリメソッド?](#)
2. [Table of Contents](#)
3. [自己紹介](#)
4. [ファクトリメソッド](#)
5. [Python での例](#)
6. [ファクトリメソッドを採用した経緯](#)
7. [実際の使用例\(チャットbot\)](#)
8. [ファクトリメソッドを採用してみての感想](#)
9. [まとめ](#)



- [🐦 旧 Twitter](#) ⁺
- [🐙 GitHub](#) ⁺
- [📖 Zenn](#) ⁺

I'm ナイトウ

- Python 🐍 で Web バックエンド
- Web フロントもやりたい
- Vue ♡ が好き
- since: 2021 (4年目)
- 駆け出しエンジニア? (非 CS 専攻)
- 柏 (東葛dev 運営?)
- 自作キーボード
- 新本格ミステリ
- エヴァ、シュタゲ

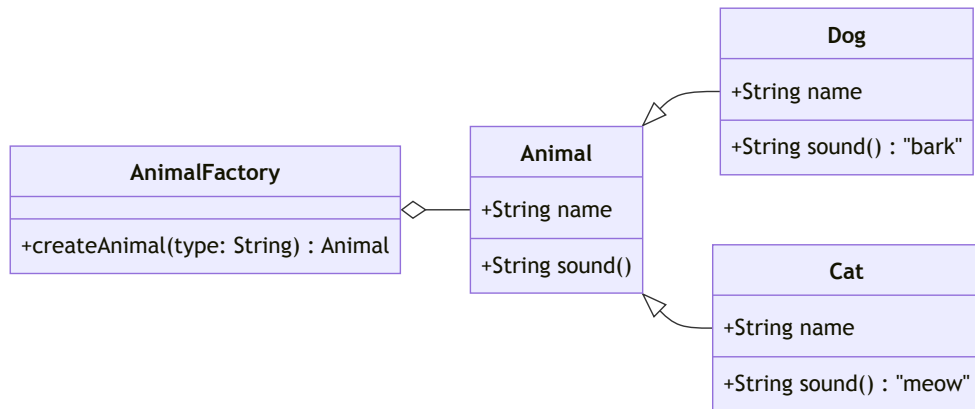
ファクトリメソッド

ファクトリメソッド #とは

他のクラスのコンストラクタをサブクラスで上書き可能な自分のメソッドに置き換えることで、アプリケーションに特化したオブジェクトの生成をサブクラスに追い出し、クラスの再利用性を高めることを目的とする

[🌐 Factory Method パターン - Wikipedia](#)

クラス図



ファクトリメソッドのいいところ

1. 再利用性の向上
2. 変更の容易さ
3. 可読性の向上



Python での例

```
1  from abc import ABC, abstractmethod
2
3  class Animal(ABC):
4      @abstractmethod
5      def __init__(self, name):
6          self.name = name
7
8      @abstractmethod
9      def speak(self):
10         pass
11
12     class Dog(Animal):
13         def __init__(self, name):
14             super().__init__(name)
15
16         def speak(self):
17             return "Woof!"
18
19     class Cat(Animal):
20         def __init__(self, name):
21             super().__init__(name)
22
23         def speak(self):
24             return "Meow!"
```

```
26  class AnimalFactory:
27      @staticmethod
28      def create(animal_type, name):
29          if animal_type == "dog":
30              return Dog(name)
31          elif animal_type == "cat":
32              return Cat(name)
33          else:
34              raise ValueError("Unknown type")
35
36
37  dog = AnimalFactory.create("dog", "Buddy")
38  cat = AnimalFactory.create("cat", "Whiskers")
39
40  print(dog.speak()) # "Woof!"
41  print(cat.speak()) # "Meow!"
```




ファクトリメソッドを採用した経緯

- 現在チャットbotを利用したアプリを作成中(PoC)
- 外部のAPIサービスを利用して実現する必要がある
- しかしサービス選定の時間はない

そこで利用するサービスに依存せず
コードの変更を簡単にしたい



実際の使用例(チャットbot)

インターフェース(抽象基底クラス)の定義

```
1  from abc import ABC, abstractmethod
2  from enum import Enum
3
4  class ChatBot(ABC):
5      @abstractmethod
6      def generate_response(self, messages) → str:
7          pass
8
9
10 class ChatBotServices(Enum):
11     OPENAI = "openai"
```

実際の使用例(チャットbot)

具象クラスの作成(OpenAI API)

```
1  from openai import OpenAI
2
3  class OpenAIChatBot(ChatBot):
4      def __init__(self):
5          self.chatgpt_client = OpenAI(api_key=OPEN_AI_API_KEY)
6
7      def generate_response(self, messages) → str:
8          completion = self.chatgpt_client.chat.completions.create(
9              messages=messages,
10             model="gpt-4o",
11         )
12         content = completion.choices[0].message.content
13
14         return content
```

実際の使用例(チャットbot)

ファクトリ

```
1  class ChatBotFactory:
2      _class: dict[str, type[ChatBot]] = {}
3
4      @classmethod
5      def register(cls, chat_bot_service: ChatBotServices):
6          def wrapper(cls_):
7              cls._class[chat_bot_service.value] = cls_
8              return cls._class[chat_bot_service.value]
9
10         return wrapper
11
12     @classmethod
13     def get_chat_bot_class(cls, chat_bot_service: ChatBotServices):
14         return cls._class[chat_bot_service.value]
15
16
17  ChatBotFactory.register(ChatBotServices.OPENAI)(OpenAIChatBot)
```

<https://zenn.dev/miyaji26/articles/fe4a50319ed799>

実際の使用例(チャットbot)

デコレータとして

```
from openai import OpenAI

class OpenAIChatBot(ChatBot):
    def __init__(self):
        self.chatgpt_client = OpenAI(api_key=OPEN_AI_API_KEY)

    def generate_response(self, messages) → str:
        completion = self.chatgpt_client.chat.completions.create(
            messages=messages,
            model="gpt-4o",
        )
        content = completion.choices[0].message.content

        return content
```

実際の使用例(チャットbot)

ファクトリでインスタンスを作成

```
1 ChatBotClass = ChatBotFactory.get_chat_bot_class(ChatBotServices.OPENAI)
2 chat_bot = ChatBotClass()
```

これがやりたかった

```
1 ChatBotFactory.register(ChatBotServices.OPENAI)(OpenAIChatBot)
2 ChatBotFactory.register(ChatBotServices.CLAUDE)(ClaudeChatBot)
3 ChatBotFactory.register(ChatBotServices.GOOGLE_GEMINI)(GoogleGeminiChatBot)
4 ChatBotFactory.register(ChatBotServices.MICROSOFT_BING)(MicrosoftBingChatBot)
```

柔軟に使うサービスを選択できる

```
1 ChatBotClass = ChatBotFactory.get_chat_bot_class(ChatBotServices.OPENAI)
2 chat_bot = ChatBotClass()
3 chat_bot_response = chat_bot.generate_response(messages)
```


ファクトリメソッドのあんまりよくないところ

使わない場合に比べてコードが複雑(難解)になる



補うために

- ドキュメントを作成
- ペアプロ

ドキュメントレビューでメンバーを巻き込む



```
1  # chat-app
2
3  ## 基本設計方針
4
5  利用する外部APIがまだ決まっていないのでファクトリメソッドに
6  抽象クラス(インタフェース)を各サービスの外部APIを用いて実装
7  呼び出し元でどのサービスを利用するかを決める。
8
9  ## ディレクトリ構成
10
11  `tree.txt` を参照
12
13  ### `services` ディレクトリ
14
15  ビジネスロジックなどを担う
16
17  - `chat`
18
19      - AI チャットサービス(Open AIなど)を利用したチャット生
20      - ChatGPT が苦手な最新情報(天気情報など)の取得を別の外
21
22  - `geocoding`
23
24      - ジオコーディング処理を担う
25      - あくまでも関数としてのふるまいまでにとどめる
```



ファクトリメソッドを採用してみたの感想

- 🙄 過剰だったかもしれない
- 📖 ドキュメントの作成を丁寧に行う必要があった
- 🙋 この作りにしたおかげで他メンバーへの引き継ぎスムーズにできた
- 🤔 コンストラクタで異なる引数を受け取る場合
- 🤔 Python の理解も足りてない
- 🤖 果たしてこれは本当にファクトリメソッドだったのか

まとめ

- ファクトリメソッド: オブジェクトの生成をサブクラスに追い出し、クラスの再利用性を高める
- ファクトリメソッドを使うべき状況
 - オブジェクトの生成方法が複雑な場合
 - 生成するオブジェクトの種類が増減する可能性がある場合
 - 生成ロジックをカプセル化したい場合

