Software Design Pattern: Template Method

Anton Seitz
INF22B
DHBW
Stuttgart, Germany
email-adress-to-add

Jannis Gehring
INF22B
DHBW
Stuttgart, Germany
inf22115@lehre.dhbw-stuttgart.de

***Abstract*—Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.**

Contents

I) Motivation ... 1
II) The Template Method Design Pattern ... 1
II.A) Different types of abstract steps ... 1
II.B) Class diagram ... 1
III) Example ... 2
III.A) General project information ... 2
III.B) Usage ... 2
III.C) The template method ... 3
III.D) Class diagram ... 3
III.E) Concepts of Template Method Design Pattern (TMDP) in the example ... 3
IV) Real-World examples ... 3
V) Pro/Con of Template Method ... 3
VI) Differentiation to other patterns ... 3
VII) Outlook ... 3
References ... 3

I. Motivation

II. The Template Method Design Pattern

With the Template Method Design Pattern (TMDP), the algorithm is divided into several abstract steps, which are called in the *Template Method (TM)*. This method implements the solution to the problem on a high level, allowing the different, specific step implementations to deal with the differing contexts. The steps as well as the TMDP are aggregated in an abstract class.

When the *TM* is to be applied to a new context, a subclass of the abstract class is created. Then, at minimum all required functions declared in the abstract class have to be implemented, before the TM can be performed for the new class.

A. Different types of abstract steps

In general, there are three types of abstract steps:

1) **Required abstract steps**
These steps are defined as abstract methods in the abstract class. Therefore, they have to be implemented by every concrete class, otherwise the TM cannot be performed.
2) **Optional abstract steps**
For these steps, a default implementation exists in the abstract class. Therefore, they do not have to be implemented by the concrete class but can be if the context requires so. This can be useful when steps are *exactly* the same for multiple contexts.
3) **Hook methods**
For these steps, no implementation is given in the abstract class, but they are not abstract classes either. Instead, they can be implemented in concrete classes for them to *hook into* the template method. One example would be the allowance of logging in between of different steps of the template method.

B. Class diagram

The following class diagram displays the different classes and the different types of abstract steps. Whilst the functions `required_step1` and `required_step2` are implemented by every concrete classes, `optional_step` and `hook_method` are only implemented by some of them.

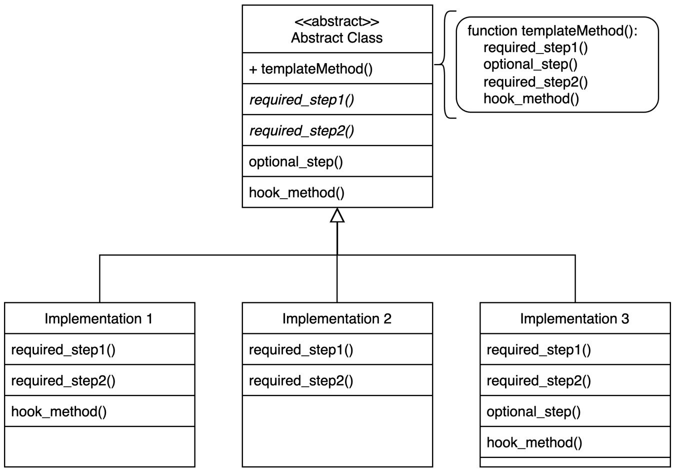

Fig. 1: Class diagram showcasing the different concepts of Template Method Design Pattern (TMDP)

III. Example

A. General project information

For showcasing TMDP, a context had to be chosen which fits the situation of similar algorithms with differing concrete implementations. The implemented application, in short, fetches, processes and displays data of differing source and contexts. The different contexts are presented subsequently:

1) **`CryptoVisualize` - Price of Bitcoin**
   The price of the wide-spread crypto currency Bitcoin is retrieved for every day since the first of July 2024. The data is then being transformed and visualized as a plotly line-chart.
2) **`DogVisualize` - Picture of Dog**
   A random picture of a dog is being fetched and displayed.
3) **`AutobahnVisualize` - German Highway lorries**
   Informations regarding different lorries of the notorious german highways ('Autobahn') are fetched. Then, the lorries are displayed on a plotly map-chart. They are color-coded according to the highway they belong to, which allows the retracing of individual highways.

B. Usage

The kind of visualization displayed can be configured by a command line argument when running this command:

```
python3 src/main.py -c [specifier]
```

These are the specifiers for the different classes:

context	specifier
CryptoVisualize	crypto
DogVisualize	dog
AutobahnVisualize	autobahn

Example outputs with their corresponding commands are presented subsequently.

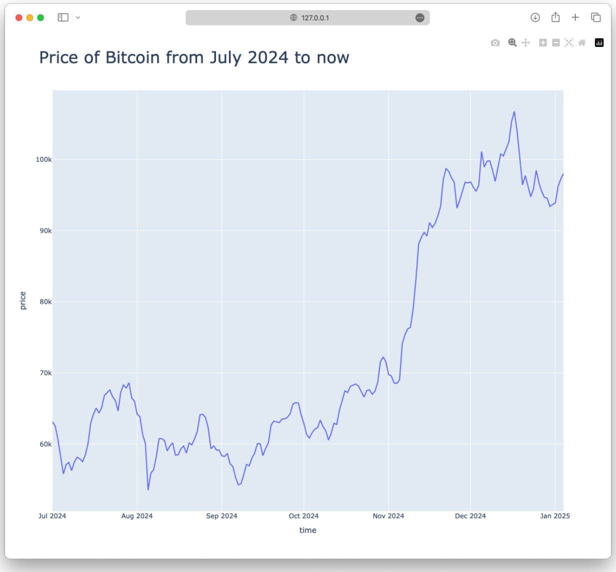

Fig. 2: Command: `python3 src/main.py -c crypto`

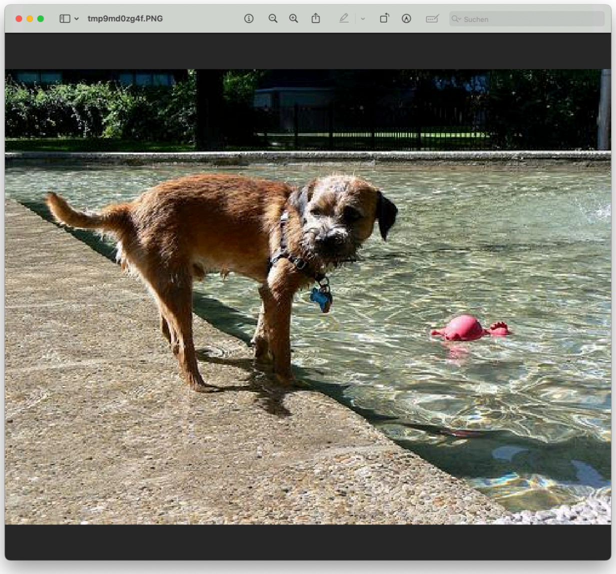

Fig. 3: Command: `python3 src/main.py -c dog`

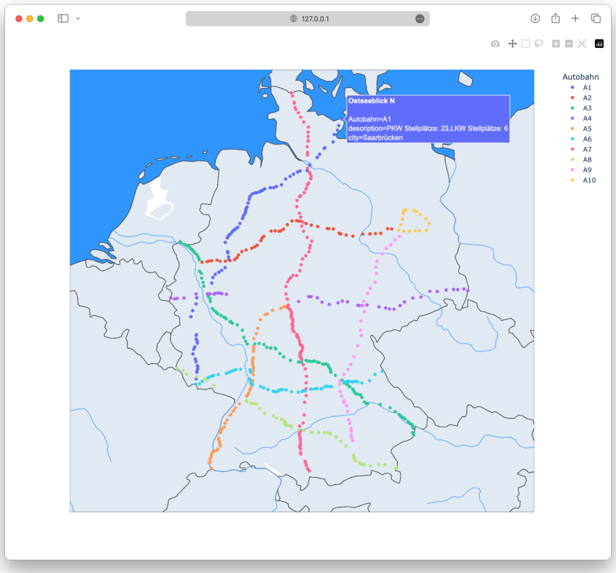

Fig. 4: Command: `python3 src/main.py -c autobahn`

C. *The template method*

In python, a class is declared as abstract by inheriting from the class of the module.
The structure of the TM allows the explanation of the different types of abstract steps in the example:

```
# Template Method
def show_me_stuff(self) -> None:
    api_url = self.get_api_url()
    content = self.api_requests(api_url)
    data = self.process_content(content)
    self.print_report(data)
    self.visualize_data(data)
    return
```

Listing 2: The example TM.

and are different for every class and are therefore implemented as abstract methods. Of course, a really basic implementation of the methods could be implemented in general, retrieving data from a sample API and visualizing it. The abstract methods are better, because they suggest to the developer that without a concrete implementation of those classes, there is no sense in using the TM. In python, this can be implemented by a decorator and an empty body:

```
@abstractmethod
def get_api_url(self) -> str:
    pass
```

and have default implementations. For , it is just performing one api request and returning the content. For , it is returning the input.

D. *Class diagram*

E. *Concepts of Template Method Design Pattern (TMDP) in the example*

IV. Real-World examples

V. Pro/Con of Template Method

VI. Differentiation to other patterns

VII. Outlook

References