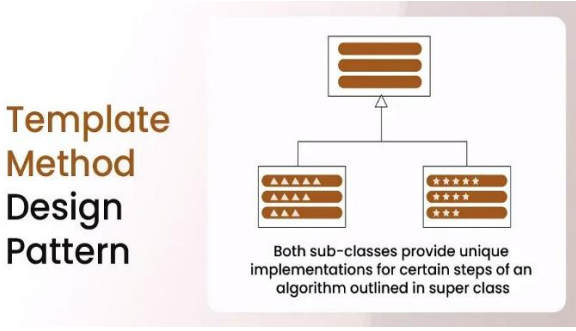
Template Method Pattern

Defⁿ: The *Template Method Design Pattern* is a *behavioral* design pattern that provides a *blueprint* for *organizing code*, making it *flexible* and *easy* to *extend*. With this pattern, you define the core steps of an *algorithm* in a method but *allow subclasses* to *override* specific steps *without* changing the *overall structure*.



🔑 **Key Idea:**

- The parent class provides a *template method* that outlines the algorithm *step-by-step*.
- Certain steps are implemented in the *base class*, while others are left as *abstract* or *hook methods* to be overridden by *subclasses*.
- This ensures *code reuse* and enforces a *standard process*, while still allowing *flexibility* in *specific steps*.

In simple terms:

- You *fix* the *overall steps* of an algorithm in a base class.
- Some steps are *common* for all, and some steps are *left empty* so *subclasses* can fill them with *their own logic*.
- This way, the *process stays* the *same*, but the *details* can *change*.

The *Template Pattern* is all about keeping the *order* of *steps fixed*.

- The base class decides *what comes first, second, third…* (the sequence).
- The subclasses just decide *how each step is done*.

So, the *process*/*order* never changes, only the *details* inside some steps *can change*.

***Ex:** Machine Learning Pipeline*

Imagine you are a *data scientist* who needs to experiment with different *machine learning models* such as *Neural Networks*, *Decision Trees*, and later possibly *Random Forests*, *SVMs*, or others. Regardless of the model type, you must always follow a *standardized pipeline* to train and evaluate your models.
The pipeline generally consists of the following steps:

1. *Load Data*
2. *Pre-process Data*
3. *Train Model*
4. *Evaluate Model*
5. *Save Model*

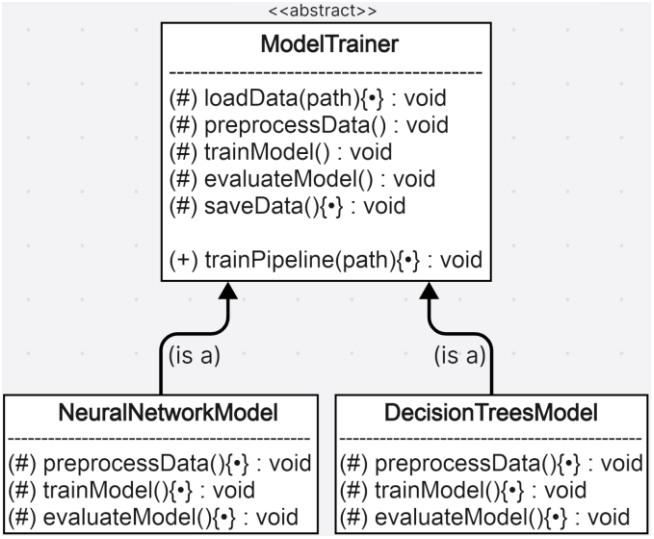
Using the *Template Method Pattern*, we can define a *base abstract class* (or *interface*) that enforces this pipeline. The base class provides the *template method* that *executes* these *steps* in the *correct order*. Some steps (e.g., *Load Data, Save Model*) can be *implemented directly* in the *base class* since they may remain common, while others (e.g., *Train Model, Evaluate Model*) are *left abstract* so that *subclasses* (specific *model implementations*) provide their *own logic*.
This approach ensures:

- Consistency: Every model follows the *same sequence* of *operations*.
- Reusability: Common tasks (like *loading*/*saving data*) are implemented *once*.
- Extensibility: Adding a *new model* (e.g., SVM) requires only *implementing* the varying steps *without changing* the overall pipeline.

👉 *In essence:* The Template Method Pattern allows us to *fix* the *order* of the machine learning workflow while still permitting *flexibility* in how individual steps are executed for different models.

@Sibasundarj8

***UML of ML pipeline*:**



Code Link: *https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/20___Template%20Method%20Pattern%20code*

***Standard UML*:**