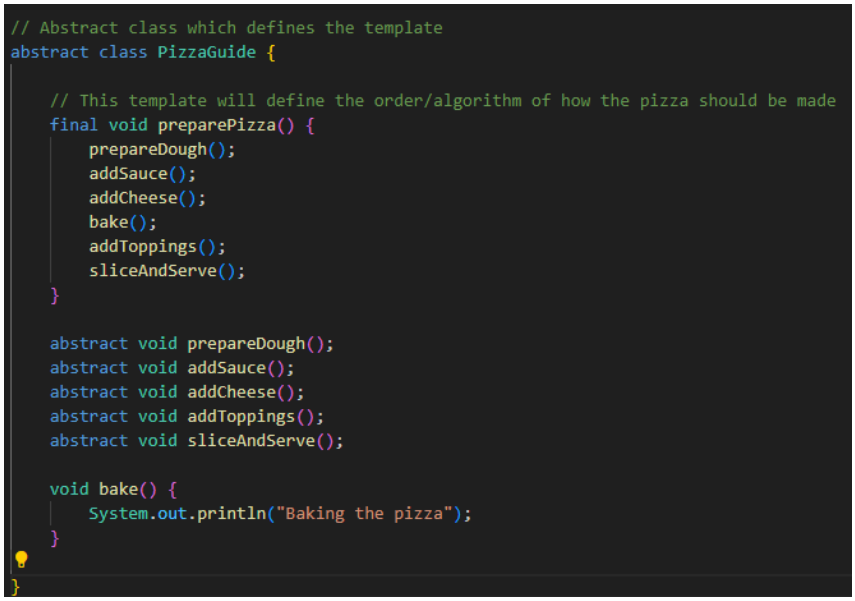
The template pattern is like having a step by step template to build things. It helps you do the same steps in the right order every single time. It's just like building a house with a blueprint.



```java
// Abstract class which defines the template
abstract class PizzaGuide {

    // This template will define the order/algorithm of how the pizza should be made
    final void preparePizza() {
        prepareDough();
        addSauce();
        addCheese();
        bake();
        addToppings();
        sliceAndServe();
    }

    abstract void prepareDough();
    abstract void addSauce();
    abstract void addCheese();
    abstract void addToppings();
    abstract void sliceAndServe();

    void bake() {
        System.out.println("Baking the pizza");
    }
}
```

The abstract class name is PizzaGuide. This is the template class which will define the algorithm for making a pizza.
The preparePizza method is the template method that specifies the steps for making a pizza.

```
prepareDough(); addSauce(); addCheese(); bake(); addToppings();
sliceAndServe();
```

These are the abstract methods which will be implemented by the subclasses.
The bake() method at the end is a default implementation for baking the pizza.

```java
class VeggiePizza extends PizzaGuide {
    @Override
    void prepareDough() {
        System.out.println("Preparing thin crust dough.");
    }

    @Override
    void addSauce() {
        System.out.println("Adding tomato sauce.");
    }

    @Override
    void addCheese() {
        System.out.println("Adding Mozzarella cheese.");
    }

    @Override
    void addToppings() {
        System.out.println("Adding Green Peppers and pineapple.");
    }

    @Override
    void sliceAndServe() {
        System.out.println("Slicing and serving the Veggie Pizza");
    }
}
```

This is the Concrete subclass VeggiePizza. The main purpose of this class is to represent a specific type of Pizza.
The methods have their own implementation on how to create the pizza.
This is also the subclass of PizzaGuide.

```java
class MeatPizza extends PizzaGuide {
    @Override
    void prepareDough() {
        System.out.println("Preparing thick crust dough.");
    }

    @Override
    void addSauce() {
        System.out.println("Adding tomato sauce.");
    }

    @Override
    void addCheese() {
        System.out.println("Adding Mozzarella cheese.");
    }

    @Override
    void addToppings() {
        System.out.println("Adding Chicken and bacon crumble.");
    }

    @Override
    void sliceAndServe() {
        System.out.println("Slicing and serving the Meat Pizza");
    }
}
```

This is the MeatPizza class which is also a Concrete subclass of the PizzaGuide.
This has the same methods but with different implementations.

The template pattern has a lot of benefits because it enforces a consistent algorithmic structure for making the pizza in the PizzaGuide. The template method preparePizza defines steps in a specific order and the concrete subclasses provide implementations for individual steps. Without the pattern each subclass might implement the pizza making process independently, leading to inconsistent structures across different pizza types, which means that they would implement the pizza making process on their own without following a sequence of steps. The inconsistent structure could mean that one class might choose to add the toppings first and the other class choose to bake the pizza first.
The other good thing about this pattern is that it promotes code reusability. This means that by defining a common template in the subclass, the shared steps will be implemented and reused by all the subclasses. This will make the code more maintainable and reduce code redundancy. Without the pattern, similar steps may be duplicated across multiple subclasses, complicating maintenance. If a step in the pizza making process needs to be updated, it must be changed in several places.