**Imagine a Real-World Scenario:**

Imagine you're making different types of hot beverages, like tea and coffee. While the overall process of making these beverages is somewhat similar (boil water, brew, add condiments), there are specific steps unique to each type. You want a consistent way to make these beverages while allowing flexibility in the unique steps.

**Template Class (BeverageTemplate):**

* This is like a set of instructions for making beverages.
* It defines a fixed sequence of steps to follow: boil water, brew, pour in a cup, and add condiments.
* It also includes common methods that can be shared across different beverage types.

**Concrete Classes (Tea and Coffee):**

* These are like the actual beverage types you want to make.
* They inherit from the BeverageTemplate class, which gives them the fixed sequence of steps to follow.
* They implement the specific brewing and condiment steps unique to their type.

**Client Code (Main Method):**

* This is where you bring everything together.
* You create instances of the concrete beverage types.
* You call the MakeBeverage method on these instances to follow the defined sequence of steps.

**In Simple English:**

The Template Method pattern is like having a recipe for making different types of hot beverages. Imagine you want to make tea and coffee, both with similar steps but with some unique steps for each type. The Template Method pattern allows you to define a standard way of making beverages with a sequence of common steps, like boiling water and pouring into a cup. This sequence is provided by the template. The specific steps, like brewing and adding condiments, are left for each beverage type to define. This pattern helps you maintain consistency in the process of making beverages while allowing you to customize the unique steps for each type. Just like you can follow a recipe to make tea or coffee, in your code, you can use the template method to make different beverages following the same general process.

**Step 1: Define the Abstract Template (BeverageTemplate)**

public abstract class BeverageTemplate

{

public void MakeBeverage()

{

BoilWater();

Brew();

PourInCup();

AddCondiments();

}

**// Define template methods and hooks**

protected abstract void Brew();

protected abstract void AddCondiments();

**// Other common methods that can be shared across subclasses**

protected void BoilWater()

{

Console.WriteLine("Boiling water");

}

protected void PourInCup()

{

Console.WriteLine("Pouring into cup");

}

}

The BeverageTemplate class defines a template method MakeBeverage that orchestrates the steps of making a beverage.

It includes common methods (BoilWater and PourInCup) that can be shared among different beverages.

It declares abstract methods Brew and AddCondiments, which represent the steps that subclasses need to implement.

**Step 2: Implement Concrete Templates (Tea and Coffee)**

public class Tea : BeverageTemplate

{

protected override void Brew()

{

Console.WriteLine("Steeping the tea");

}

protected override void AddCondiments()

{

Console.WriteLine("Adding sugar");

}

}

public class Coffee : BeverageTemplate

{

protected override void Brew()

{

Console.WriteLine("Dripping coffee through filter");

}

protected override void AddCondiments()

{

Console.WriteLine("Adding sugar and milk");

}

}

The Tea and Coffee classes inherit from BeverageTemplate.

They provide implementations for the abstract methods Brew and AddCondiments.

**Step 3: Client Code (Main Method)**

static void Main(string[] args)

{

BeverageTemplate tea = new Tea();

tea.MakeBeverage();

BeverageTemplate coffee = new Coffee();

coffee.MakeBeverage();

}

In the Main method, you create instances of Tea and Coffee.

You call the MakeBeverage method on these instances to make the beverages.

**Explanation**:

**Template Method Pattern**: The Template Method pattern defines the structure of an algorithm but lets subclasses provide their own implementations for some steps of that algorithm.

**Step 1**: The BeverageTemplate class acts as a template for making beverages. It provides a fixed structure with template method MakeBeverage, which defines a sequence of steps. Some of these steps are left as abstract methods to be implemented by concrete subclasses.

**Step 2**: The Tea and Coffee classes are concrete implementations of the template. They provide specific implementations for the abstract methods, defining how tea and coffee are brewed and what condiments are added.

**Step 3**: In the Main method, you create instances of Tea and Coffee, representing specific beverage types. When you call MakeBeverage on these instances, the template method orchestrates the steps, including the ones provided by the subclasses.

The output of the program demonstrates how the Template Method pattern works by providing a common structure for making beverages while allowing individual beverage types (tea and coffee) to define their own brewing and condiment addition steps. This separation of concerns and use of a template method helps maintain consistency and avoids duplication in the code.