**Use Case:** A vehicle rental service where you can rent different types of vehicles (Car, Bike). The Factory Method Pattern allows for easy addition of new vehicle

**Working:**

**1. Interfaces & Classes**:

Transport Interface: Think of this as a blueprint for any type of transport. It has one method, hire which defines what it means to hire a transport vehicle, whether it’s a Sedan or a Motorcycle.

Sedan and Motorcycle Classes: These are the specific types of vehicles that implement the Transport blueprint. Each class has its own version of the hire method. When you hire a Sedan, it says, "Hiring a Sedan," and when you hire a Motorcycle, it says, "Hiring a Motorcycle."

**2. Abstract Factory & Concrete Factories:**

TransportFactory Abstract Class: This is like a general factory that doesn’t make anything by itself but defines how to create transport. It has an abstract method, createTransport

which subclasses must implement to specify what type of transport to create.

SedanFactory and MotorcycleFactory: These are the specific factories that know how to make either a Sedan or a Motorcycle. Each factory overrides the createTransport method to produce a new instance of its respective vehicle.

**3. Main Class:**

Main: This is where everything comes together. It demonstrates how to use the factories and the transport objects:

First, it creates a SedanFactory and a MotorcycleFactory.

Then, it uses each factory to create a transport object (either a Sedan or a Motorcycle).

Finally, it calls the method on each transport, which prints out the appropriate message for the chosen vehicle.

**Benefits of Using the Factory Method Pattern**

**Decoupling Object Creation**: The main part of the code doesn’t need to know the details of how each transport is created. It only interacts with the Transport interface, making the code cleaner and easier to manage.

**Flexibility:**  Ifyou want to add a new transport type like a Truck you can easily do this by creating a new factory class. You won’t need to change the existing code, which helps prevent bugs.

**Loose Coupling:** The code in the Main class doesn’t directly depend on the specific transport classes. This makes it easier to change or extend the code later without breaking anything.

**Conclusion**

Overall, this code is a great example of the Factory Method Design Pattern in action. It shows how to create different types of objects based on a common interface and a general factory class, making the system more flexible and easier to maintain. This approach is especially useful as your project grows and you need to add new features.