✅ Factory Method Pattern in Java

🔹 Real-Life Analogy

Imagine you run a logistics company. You ship by **Truck** or **Ship**, depending on the region. The creation logic for these vehicles differs—but the client shouldn't care *how* they're created, just *get the right vehicle*.

🔸 1. Simple Factory Pattern (not a GoF pattern)

A simple class with a static method to create objects based on input.

➤ Code:

```
// Product
interface Transport {
    void deliver();
}

// Concrete Products
class Truck implements Transport {
    public void deliver() {
        System.out.println("Deliver by land in a box");
    }
}

class Ship implements Transport {
    public void deliver() {
        System.out.println("Deliver by sea in a container");
    }
}

// Simple Factory
class TransportFactory {
    public static Transport createTransport(String type) {
        if (type.equalsIgnoreCase("truck")) {
            return new Truck();
        } else if (type.equalsIgnoreCase("ship")) {
            return new Ship();
        }
        throw new IllegalArgumentException("Unknown transport type");
    }
}

// Client
public class SimpleFactoryDemo {
    public static void main(String[] args) {
        Transport t1 = TransportFactory.createTransport("truck");
        t1.deliver();

        Transport t2 = TransportFactory.createTransport("ship");
        t2.deliver();
    }
}
```

🔎 Issues:

- **Tightly coupled**: New `if-else`/`switch` in factory for each type.
- Violates **Open-Closed Principle**: Need to modify factory for every new transport.

🔶 2. Factory Method Pattern (GoF Design Pattern)

- Define a factory interface.
- Subclasses **override the factory method** to create objects.

➢ Code:

```
// Product
interface Transport {
    void deliver();
}

// Concrete Products
class Truck implements Transport {
    public void deliver() {
        System.out.println("Deliver by land in a box");
    }
}

class Ship implements Transport {
    public void deliver() {
        System.out.println("Deliver by sea in a container");
    }
}

// Creator (abstract)
abstract class Logistics {
    public void planDelivery() {
        // Factory Method
        Transport transport = createTransport();
        transport.deliver();
    }

    // Factory Method (to be overridden)
    protected abstract Transport createTransport();
}

// Concrete Creators
class RoadLogistics extends Logistics {
    protected Transport createTransport() {
        return new Truck();
    }
}

class SeaLogistics extends Logistics {
    protected Transport createTransport() {
        return new Ship();
    }
}


// Client
```

```
public class FactoryMethodDemo {
    public static void main(String[] args) {
        Logistics logistics1 = new RoadLogistics();
        logistics1.planDelivery();  // Uses Truck

        Logistics logistics2 = new SeaLogistics();
        logistics2.planDelivery();  // Uses Ship
    }
}
```

🔁 Simple Factory vs Factory Method – Key Differences

Aspect	Simple Factory	Factory Method
Pattern Type	Not GoF pattern	GoF Creational Pattern
Object creation	Done in one static factory class	Deferred to subclasses
Extensibility	❌ Hard to extend without modifying	✅ Open for extension (OCP compliant)
Use of Inheritance	No	Yes (polymorphism)
Client's Role	Chooses type (via `if` or `switch`)	Uses subclass, unaware of object type
When to Use	Few product types, simple logic	Many types, need for scalability

---

✅ Summary

- **Use Simple Factory** for:
    - Few product types.
    - You need a quick centralized way to instantiate.
- **Use Factory Method** for:
    - Multiple product types.
    - You want to **follow OOP principles** like Open/Closed.
    - You anticipate changes or extensions.
