Factory Design Pattern (Java) - Cheat Sheet

What is the Factory Design Pattern?

A creational pattern that provides a factory method to create objects without specifying their exact concrete classes. The client asks the factory for an object; the factory decides which subclass to instantiate.

Why it's Useful & Effective

- Decouples object creation from usage
- Supports Open/Closed Principle (add new types easily)
- Centralizes creation logic -> easier maintenance
- Allows runtime decision of object type
- Simplifies testing and extension

Key Parts

1. Interface / abstract type (e.g., Transport)
2. Concrete classes (Truck, Ship, Drone)
3. Factory class (TransportFactory)
4. Client (Main) that uses the factory

Quick Example

```
public interface Transport { void deliver(); }

public class Truck implements Transport {
    public void deliver() { System.out.println("Delivering by land"); }
}

public class TransportFactory {
    public Transport createTransport(String type) {
        if ("TRUCK".equals(type)) return new Truck();
        throw new IllegalArgumentException("Unknown type");
    }
}

public class Main {
    public static void main(String[] args) {
        TransportFactory factory = new TransportFactory();
        Transport t = factory.createTransport("TRUCK");
        t.deliver(); // Output: Delivering by land
    }
}
```

Diagram (conceptual)

Client -> asks Factory -> factory returns concrete Product (Truck, Ship, etc.) -> client uses product

Real-world Use Cases

- Notification systems (Email, SMS, Push)
- Logger libraries (file, console, database)
- GUI frameworks (WindowsButton, MacButton)
- Transport systems (Truck, Ship, Drone)

Big Idea

Client code does NOT depend on concrete classes; it works with interfaces.
Adding new types requires minimal changes - keeps code clean and flexible.