Module 1: Factory Method Pattern – Simple Version

Design Patterns with Real-World Thinking

Instructor: Ayush Singh

Academic Term: 2025–2026

Objective

To understand how the Factory Method pattern can help simplify long if-else logic when selecting behavior based on input — using relatable, real-world examples and simple Python code.

Problem Introduction

Imagine a school app that lets you download your report card. Users can choose:

- PDF format
- CSV format
- JSON format

Here is how it starts:

```
def export_report(format, data):
    if format == "pdf":
        print("Exporting as PDF")
    elif format == "csv":
        print("Exporting as CSV")
    elif format == "json":
        print("Exporting as JSON")
    else:
        print("Unknown format")
```

This works... but what if tomorrow someone asks for TXT, or Excel, or DOCX? You have to keep editing this function.

Why This is Bad

- Code gets longer and messier
- Hard to test each part separately
- Breaks the Open-Closed Principle: you're modifying old code constantly

Real-Life Analogy: Ice Cream Stall

You run a stall. Someone asks:

- "Can I get Chocolate?" → You mix chocolate
- "Can I get Mango?" → You mix mango
- "Can I get Strawberry?" → You add another if-else

What if you had machines — one for each flavor — and just pressed a button? That **button system** is the Factory. The **machine** is the class.

Refactoring Using Factory Method

We'll now break this into clear steps:

Step 1: Create Base Class

```
class ReportExporter:
    def export(self, data):
        raise NotImplementedError()
```

Step 2: Create Classes for Each Format

```
class PDFExporter(ReportExporter):
    def export(self, data):
        print("Exporting report as PDF")

class CSVExporter(ReportExporter):
    def export(self, data):
        print("Exporting report as CSV")

class JSONExporter(ReportExporter):
    def export(self, data):
        print("Exporting report as JSON")
```

Step 3: Create the Factory

```
class ExporterFactory:
    @staticmethod
    def get_exporter(format):
        if format == "pdf":
            return PDFExporter()
        elif format == "csv":
            return CSVExporter()
        elif format == "json":
            return JSONExporter()
```

```
        else:
            raise ValueError("Unsupported format")
```

Step 4: Use It

```
exporter = ExporterFactory.get_exporter("pdf")
exporter.export({"name": "Ayush", "marks": 95})
```

What Improved?

- No more long if-else chains
- Each format is now a small, easy-to-test class
- You can add new exporters without touching the old ones

Adding a New Format is Easy

Just create a `TXTExporter` class and register it in the factory. No other code needs to be changed!

Other Real-Life Examples of Factory

- Swiggy chooses the right restaurant class for each order
- Canva exports images in PNG, PDF, JPG — using exporter classes
- WhatsApp handles images, videos, docs differently — factory pattern!

Key Takeaways

1. Don't use giant if-else ladders for types or formats
2. Use Factory Method to organize object creation
3. It makes your code cleaner, safer, and future-proof

Prepared by: Ayush Singh
SecondBrain Labs, 2025