Factory Design Patten

طبقنا هنا مثال البيتزا، حيث إنه ما كان بس بيحاكي اختيار نوع البيتزا زي ما عملنا في نمط ديكوريتور، وإنما حاكى مثال فيه خطوات متكاملة: اختيار نوع البيتزا، إنشاؤها، ثم تنفيذ باقي العمليات مثل الخبز والتقطيع والتغليف.

كان عنا تحدي: كيف نختار نوع البيتزا المناسب حسب اختيار المستخدم، ثم ننشئه ونتابع عليه باقي العمليات.
لو بدنا ننشئ النوع المناسب في الـ `Main` أو عند الـ `Client` ، راح نحتاج نكرر كلمة `new` كثير، وهالشي بيؤدي إلى:

- تكرار في الكود - ممكن نكرر نفس سطر الانشاء في اجزاء كثيرة في المشروع، هذا يؤدي ->
  - صعوبة في الصيانة (maintainability).
  - العميل (Client) بيصير شايف تفاصيل المفروض ما يطلع عليها. (تفاصيل الانشاء مثل المتغيرات..)
- بنقلل المرونة.
- والـ `new` نفسه معناه coupling واعتمادية عالية على الـ concrete classes.

🧠 **يعني باختصار:**

> الإنشاء هو الجزء المتغيّر، والباقي (الخبز والتقطيع...) ثابت ← لازم نفصل المتغير (الإنشاء) عن الباقي.

وعشان هيك قمنا بنقل عملية الانشاء إلى كائن مستقل (المصنع).
تذكر مبدأ :Encapsulate what varies
وهذا كان باختصار ال **Simple Factory**

> ولاحظ انه لا يُعتبر نمطًا رسميًا ضمن GoF،ولكن الـ Simple Factory يُستخدم كخطوة أولى نحو تحسين إنشاء الكائنات."

لكن هاد الحل كان محدود شوي، لأنه بيدعم فقط مصنع واحد. لما احتجنا أكثر من مصنع، طورنا الفكرة لـ **Factory Method**.

في Factory Method:

- خلّينا الـ Factory نفسه مجرد (abstract).
- وأنشأنا منه مصانع مختلفة حسب الحاجة.
- وكل مصنع خصصنا فيه عملية الإنشاء حسب نوع البيتزا والمنطقة.

✅ هيك صرنا نتعامل مع abstractions فقط، والـ concrete classes ما بتعرف عن بعض ← **Decoupling**.

والهدف كله من هاد النمط:

> **Encapsulate instantiation**
>
> نحقق مبدأ **Dependency Inversion**
> يعني:
> لا يجوز يكون الـ high-level component معتمد على الـ low-level.
> ++ ما لازم يكون عندي أي reference لـ concrete class، ولازم أبتعد عن استخدام `new` مباشرًا.
> إذا استخدمته، لازم يكون عن طريق factory.
> +++كل إشي بيشتق من abstraction، ما في override مباشر، فقط تخصيص.

...

لكن لما كل مصنع صار ينتج أصناف مختلفة (Cheese, Greek, Pepperoni...)، صرنا نستخدم if-else على `type` داخل المصنع نفسه.
وهذا بيرجعنا لنقطة:

❌ كسر مبدأ **Open/Closed**
(إذا بدي أضيف نوع جديد، لازم أعدل على الكود بدل ما أمدده فقط).

يعني صرنا نحتاج **تقريبًا طبقة أخرى من Factory Pattern !!**

لكن لو بدنا نعمل مصنع منفصل لكل صنف (يعني نوع بيتزا معين)، رح يصير عندي عدد كبير من المصانع وتفرعات كثيرة من الكلاسات.

🔺 **وهنا بتظهر مشكلة الـ Class Explosion**
كل صنف بده مصنع خاص، وكل مصنع بده class، وهذا بيكبر عدد الملفات والكلاسات بشكل غير عملي، وبيزيد التعقيد بدل ما يقلله.

فالحل الأفضل هو: نخلي كل مصنع (مثلاً NYPizzaStore أو ChicagoPizzaStore) يحتوي داخله methods مسؤولة عن إنشاء الأصناف اللي بنتجها، وبهالطريقة بنحافظ على التوسعة بدون ما نفجر المشروع بالكلاسات 😄 .

و هذا الحل هو Abstract Factory

> eng. azza eid