TASARIM KALIPLARI

metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir."Singleton Pattern" (Tekil Tasarım Deseni) kavramını açıklayan bir diyagramdır. Bu desen, bir sınıfın yalnızca bir örneğinin (instance) oluşturulmasını ve bu örneğe global bir erişim noktası sağlanmasını amaçlar.

**Private Constructor (Özel Kurucu):** Sınıfın dışından yeni örnekler oluşturulmasını engeller.

**Static Instance (Statik Örnek):** Sınıfın tek örneğini tutar.

**Public Static Method (Genellikle getInstance()):** Sınıfın tek örneğine erişim sağlar; eğer örnek henüz oluşturulmamışsa, oluşturur.

Kaç tip tasarım kalıbı var?"

1. **Creational - Oluşturma**  
   ➤ Yeni **nesnelerin nasıl oluşturulacağını** tanımlar.  
   ➤ Örnek: Singleton, Factory Method, Builder, Abstract Factory, Prototype
2. **Structural - Yapısal**  
   ➤ Farklı nesnelerin **birbirleriyle nasıl ilişkilendirileceğini** ve **daha büyük yapılar oluşturacağını** açıklar.  
   ➤ Örnek: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
3. **Behavioral - Davranışsal**  
   ➤ Nesneler **birbiriyle nasıl iletişim kurar**, **davranışlarını nasıl organize eder**, bunu tanımlar.  
   ➤ Örnek: Observer, Strategy, Command, Iterator, State, Mediator, Visitor, Chain of Responsibility

**1. Creational (Oluşturucu) Tasarım Kalıpları**

Nesne oluşturmayı soyutlayan kalıplardır.

* **Tanım**: Nesne oluşturma süreciyle ilgilidir. Amaç, bir nesne yaratılırken uygulamanın o nesnenin hangi sınıftan geldiğine bağımlı olmasını **engellemektir**.
* **Faydası**: İstemcinin (client) doğrudan nesne oluşturmak yerine, bu işin kalıplar aracılığıyla yapılmasını sağlar; böylece **bağımlılık azalır**, **esneklik artar**.
* **Örnekler**:
  + Singleton
  + Builder
  + Prototype
  + Abstract Factory
  + Factory Method

**🔹 2. Behavioral (Davranışsal) Tasarım Kalıpları**

Nesneler ve sınıflar arasındaki **iletişimi** düzenler.

* **Tanım**: Sınıf ve nesnelerin nasıl **etkileşime girdiğini** ve **sorumlulukların nasıl dağıtıldığını** belirler.
* **Faydası**: Karmaşık davranışları merkezi olmayan şekilde yönetir, **esnek ve yeniden kullanılabilir çözümler** sunar.
* **Örnekler**:
  + Template Method
  + Interpreter
  + Visitor
  + Mediator
  + Iterator
  + Command
  + Memento
  + Observer
  + Chain of Responsibility
  + State
  + Strategy *(aslında bu desen bazı kaynaklarda davranışsal olarak da kabul edilir, ancak yukarıdaki listede Structural kısmına yazılmış)*

**🔹 3. Structural (Yapısal) Tasarım Kalıpları**

Nesnelerin bir araya gelerek **daha büyük yapılar** oluşturmasını sağlar.

* **Tanım**: Sınıf ve nesneleri birleştirerek daha **karmaşık ama yönetilebilir yapılar** oluşturur.
* **Faydası**: Alt sistemlerin veya bileşenlerin **birbirine nasıl bağlanacağını** belirleyerek esneklik sağlar.
* **Örnekler**:
  + Decorator
  + Flyweight
  + Composite
  + Facade
  + Adapter
  + Proxy
  + Bridge

notların kapsamı dışında kalan ek desenler olabilir.

**Head First Design Patterns**

Değişen ne varsa onu al ve kapsülle, böylece kodunun geri kalan kısmı etkilenmesin

• Kod değişikliği az, esneklik fazla

HAS-A can be better than IS-A

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**🔷 Abstract Factory Nedir?**

**Abstract Factory**, **Creational (Oluşturucu)** tasarım kalıplarından biridir.

**Abstract Factory**, **ilgili nesne ailelerini** oluşturan bir **arayüz (interface)** sunar, ama bu nesnelerin **tam sınıflarını belirtmeden** oluşturulmasını sağlar.

Yani, **birbirine bağımlı nesneleri birlikte oluşturmak** istediğinde kullanılır.

**🔧 Ne İşe Yarar?**

* Farklı ürün aileleri (örneğin: Windows UI vs MacOS UI) arasında **bağımlılık olmadan geçiş yapmanı sağlar.**
* Kodun, oluşturduğu nesnelerin **somut türlerine bağlı kalmadan çalışmasını sağlar.**
* Yeni ürün ailelerini sisteme **minimum kod değişikliğiyle** ekleyebilirsin.

**📌 Ne Zaman Kullanılır?**

* Birbirine **bağımlı** nesne setleri birlikte kullanılacaksa.
* Uygulamanın çalıştığı ortam/platform değişebiliyorsa (örnek: Windows vs Mac vs Linux).
* Ürünlerin **türünü belirtmeden** üretmek istiyorsan.

**🧠 Özet:**

| **Özellik** | **Açıklama** |
| --- | --- |
| Kalıp Adı | Abstract Factory |
| Tür | Creational (Oluşturma) |
| Amaç | İlgili nesneleri, türlerini belirtmeden üretmek |
| Faydası | Bağımlılığı azaltır, kolay genişleme sağlar |
| Kullandığı Yapılar | Arayüzler, soyut sınıflar, alt sınıflar |
| Kullanım Durumu | GUI sistemleri, veritabanı bağlantıları, platforma özel üretimler |

Örnek :

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir. metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**✅ PROS – Avantajlar**

| **Avantaj** | **Açıklama** |
| --- | --- |
| **Isolation of concrete classes** | Somut sınıflar kullanıcıdan saklanır. Kullanıcı sadece interface kullanır. Bu da bağımlılığı azaltır. |
| **Exchanging Product Families easily** | Örneğin, birden fazla platforma (Windows/Mac/Linux) destek vermek kolaydır. Sadece uygun factory kullanılır. |
| **Promoting consistency among products** | Aynı “aileden” ürünler birlikte kullanılır. (örneğin hepsi Windows bileşeni) Uyumsuz bileşenler bir araya gelmez. |

**❌ CONS – Dezavantaj**

| **Dezavantaj** | **Açıklama** |
| --- | --- |
| **Difficult to support new kinds of products** | Yeni bir ürün (örneğin yeni bir buton türü) eklemek zordur çünkü tüm factory sınıflarına yeni metot eklemen gerekir. Bu da mevcut kodun değişmesine neden olur (Open/Closed prensibine aykırı olabilir). |

**🏭 Factory Method Nedir?**

**Factory Method**, bir **Creational (Oluşturucu)** tasarım desenidir ve amacı, nesne yaratma işlemini alt sınıflara bırakarak **kodda gevşek bağlılık (loose coupling)** sağlamaktır.

**Factory Method**, nesne oluşturma işini doğrudan new anahtar kelimesiyle yapmaz. Bunun yerine, bir **factory method** aracılığıyla hangi sınıfın örnekleneceğine **alt sınıf karar verir**.

**🎯 Amaç**

* **Nesne oluşturmayı soyutlamak**.
* Kodun, **hangi sınıfın** örneklendiğini **bilmeden** çalışmasını sağlamak.
* Yeni sınıflar eklenmesini kolaylaştırmak (esnek ve genişletilebilir yapı).

Bir üst sınıfın birden fazla alt sınıfı vardır ve girdiye bağlı olarak alt sınıflardan birini döndürmek için.

**✅ Avantajları**

| **Avantaj** | **Açıklama** |
| --- | --- |
| Kod, hangi nesnenin oluşturulduğunu bilmez | Yani Truck mı Ship mi bilmiyor |
| Yeni ürün eklemek kolaydır | Yeni sınıf + yeni alt Factory yazarsın |
| Bağımlılığı azaltır | Kod soyutlama üzerinden çalışır |

**❌ Dezavantajları**

| **Dezavantaj** | **Açıklama** |
| --- | --- |
| Sınıf sayısını artırır | Her ürün için ayrı bir sınıf ve factory gerekir |
| Basit uygulamalarda karmaşık kaçabilir | Gereksiz soyutlama olabilir |

**📌 Factory Method vs Abstract Factory**

| **Özellik** | **Factory Method** | **Abstract Factory** |
| --- | --- | --- |
| Amaç | Tek bir ürün nesnesi oluşturmak | Ürün ailesi oluşturmak |
| Ürün Sayısı | Genellikle 1 | Genellikle birçok ürün birlikte |
| Örnek | Truck, Ship | WindowsButton, WindowsCheckbox |

metin, diyagram, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**👷 "The Creator is a class..."**

"The Creator is a class that contains the implementations for all of the methods to manipulate products, except for the factory method."

**Açıklama:**

* “Creator” sınıfı, ürünle ilgili işlemleri tanımlar (örneğin: sipariş, taşıma, fatura yazma gibi).
* Ancak **ürünü oluşturma kısmı** soyuttur (yani factoryMethod() tanımlanmaz, sadece bildirilir).
* Bu yöntem (factory method), alt sınıflarda tanımlanır.

**🧩 "All products must implement the same interface..."**

Tüm ürün sınıfları aynı arabirimi (interface) ya da soyut sınıfı kullanır. Böylece sistem, ürünlerin **tam olarak hangi sınıf olduğunu bilmeden** onlarla çalışabilir.

**Örnek:**

csharp

Kodu kopyala

interface ITransport

{

void Deliver();

}

**🧭 Şematik Yapı (Açıklamaları)**

| **Parça** | **Açıklama** |
| --- | --- |
| **Product** | Ortak arabirim / soyut sınıf (örn. ITransport) |
| **Creator** | Soyut sınıf – factoryMethod() burada tanımlanır (abstract olarak) |
| **ConcreteProduct** | Ürünün somut hali (örn. Truck, Ship) |
| **ConcreteCreator** | factoryMethod()'u override eden sınıf (örn. RoadLogistics) |

**📌 "The abstract factoryMethod()..."**

factoryMethod() soyut bir metottur. Bunu, **alt sınıflar (Concrete Creators)** implement eder.

**🧱 "The Concrete Creator is responsible..."**

Concrete Creator:

* **Hangi ürünün üretileceğini bilir.**
* factoryMethod() metodunu override ederek, bir **ConcreteProduct** üretir.

**Yani:** Bu sınıf ürünün nasıl üretileceğini bilen **tek yerdir**. Ana sistem (Creator) bu bilgiyi bilmez.

**🔁 Özetle**

Bu notta anlatılanlar:

| **Kavram** | **Açıklama** |
| --- | --- |
| **Amaç** | Nesne oluşturmayı soyutlayarak sistemin bağımlılığını azaltmak |
| **Factory Method()** | Creator sınıfında soyut, alt sınıflarda somut olarak tanımlanır |
| **Ürünler** | Ortak interface’e sahiptir (polimorfizm sağlanır) |
| **Concrete Creator** | Hangi ürünün nasıl oluşturulacağını bilen tek yerdir |
| **Avantaj** | Yeni ürün türleri eklemek kolay, kod esnek ve genişletilebilir hale gelir |

* **Factory Method**, bazen "**Sanal Yapıcı (Virtual Constructor)**" olarak adlandırılır.
* Çünkü new kullanmadan nesne yaratmayı sağlar ama yine de **dinamik olarak** doğru türde nesne üretir.

Factory Method deseninin, **kütüphanelerin çalışma mantığı** ile ilişkili olduğu söyleniyor.

**Açıklama:**

* Bir kütüphane (örneğin bir GUI ya da veritabanı kütüphanesi), **soyut sınıflar** tanımlar.
* Ama **hangi somut sınıfın** örnekleneceğini **uygulama** (client) belirler.
* Kütüphane ne zaman nesne yaratılması gerektiğini bilir ama neyin yaratılacağını bilmez. Bu ayrım önemlidir.

**talep üzerine** doğru "nesneyi" sağlar

public abstract class Creator

{

public abstract IProduct CreateProduct(); // Tek bir factory method

}

* **tek bir fabrika metodu** vardır: CreateProduct() gibi.
* Bu metod **alt sınıflar tarafından override edilir**,

public interface IGUIFactory

{

IButton CreateButton();

ICheckbox CreateCheckbox();

}

* Bu desen bir **factory nesnesi** üretir.
* Ve bu nesne, **birbirine ait birden fazla ürünü yaratmak için** birden fazla factory metoduna sahiptir.

**🔁 Fark Özet Tablosu**

| **Özellik** | **Factory Method** | **Abstract Factory** |
| --- | --- | --- |
| Yapı | Tek bir metod | Birden fazla metod |
| Ürün Sayısı | Genelde 1 | Birden fazla, ürün ailesi |
| Kullanım | Alt sınıf tek ürün yaratır | Nesne birden fazla ürün yaratır |
| Amaç | Yaratmayı alt sınıfa bırakmak | Ürün ailesini soyutlamak ve bir arada üretmek |