Nesneye Yönelimli Programlama – BLM 205

Hafta 10: Tasarım Kalıpları I

Fenerbahçe Üniversitesi

Öğretim Elemanları

Öğretim Üyesi: Dr. Vecdi Emre Levent

Ofis: 311

Email: emre.levent@fbu.edu.tr

Asistan: Arş. Gör. Uğur Özbalkan

Ofis: 307

Email: ugur.ozbalkan@fbu.edu.tr

Asistan: Arş. Gör. Ecenur Alioğulları

Ofis: 307

Email: ecenur.aliogullari@fbu.edu.tr

Ders Planı

- Tasarım Kalıpları
  - Fabrika
  - Adapter

Tasarım Kalıpları

Sadece

- Soyutlama
- Kalıtım
- Çok biçimlilik

gibi kavramlar sizi iyi bir nesneye yönelimli uygulama yazılımcısı yapamaz.

Tasarım Kalıpları

Bu kavramları kullanarak; esnek, bakımı kolay, yeniden kullanılabilir tasarımlar yapabilmek çok önemlidir.

Tasarım Kalıpları

Tasarım kalıpları yazılım tasarımında sıklıkla karşılaşılan sorunlar için taslak çözümlerdir.

Bu taslak çözümler, probleme göre düzenlenebilmektedir.

Tasarım Kalıpları

Tasarım kalıpları, kodunuza yapıştırılabilecek bir kod parçacığı veya kütüphane değildir.

Bir problemi çözmek için genel bir konsepti ifade etmektedir.

Probleminize uygun kalıbı seçerek, ilgili kalıbın gereksinimlerini yerine getirerek çözüm sağlanabilir.

Tasarım Kalıpları

Tasarım kalıbı yaklaşımını kullanmanın gerekliliği, sıklıkla karşılaşılan problemlere denenmiş ve doğrulanmış çözüm kalıpları sunmasıdır.

Birden çok kişi ile takım halinde çalışılan projelerde, problemin çözümü için yapılacak önerinin tasarım kalıplarından biri olması ve herkes tarafından anlaşılabilir olması; çok kısa sürede çözüme gitmeyi sağlayacaktır.

Örn, "singleton tasarım kalıbını kullanalım" dendiğinde, takımın tamamı kastedilen çözümün nasıl olduğunu anlayacaktır.

Tasarım Kalıpları

Tasarım kalıpları, karmaşıklıkları, detay seviyeleri, uygulanabilirlikleri açısından farklılık göstermektedirler.

Tasarım kalıpları 3 farklı kategoride incelenebilirler.

- Yaratılışsal (Creational): Esneklik ve yeniden kullanılabilirlik
- Yapısal (Structural): Obje ve sınıfların, büyük yapılara entegrasyonu
- Davranışsal (Behavioral): Objeler arasında sorumluluk ataması ve efektif iletişim

Tasarım Kalıpları

Yaratılışsal (Creational): Esneklik ve yeniden kullanılabilirlik

- Fabrika
- Soyut Fabrika
- Builder
- Prototype
- Singleton

Tasarım Kalıpları

Yapısal (Structural): Obje ve sınıfların, büyük yapılara entegrasyonu

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Tasarım Kalıpları

Behavioral (Davranışsal): Objeler arasında sorumluluk ataması ve efektif iletişim

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Observer
- State
- Strategy
- Template Method
- Visitor

Tasarım Kalıpları

Tasarım kalıplarını kullanmanın en iyi yolu, tüm tasarım kalıplarını farklı senaryolar ile öğrenip, yapılacak yeni tasarımlarda nasıl uygularımı düşünmektedir.

Daha önceki yazılımcıların tecrübelerinin yeniden kullanımını sağlamaktadır.

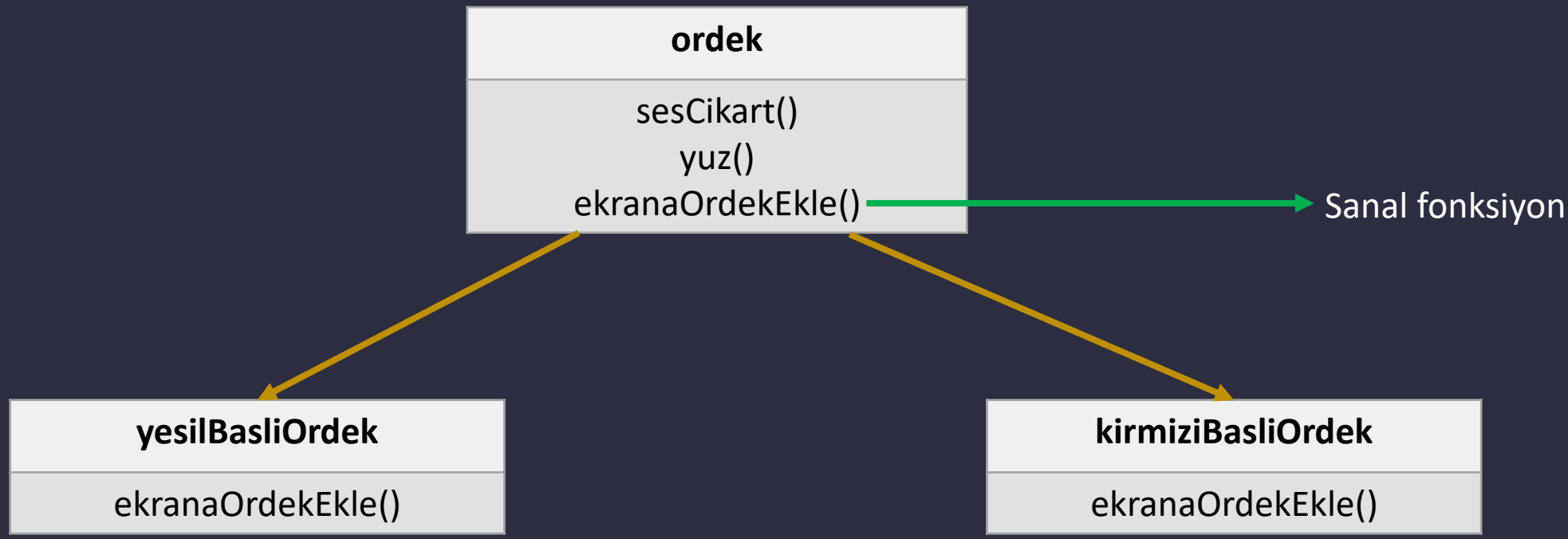
Tasarım Kalıpları

Tasarım kalıbı örneği

Ördeklerin yüzdüğü bir simulasyon programı yazılacaktır.

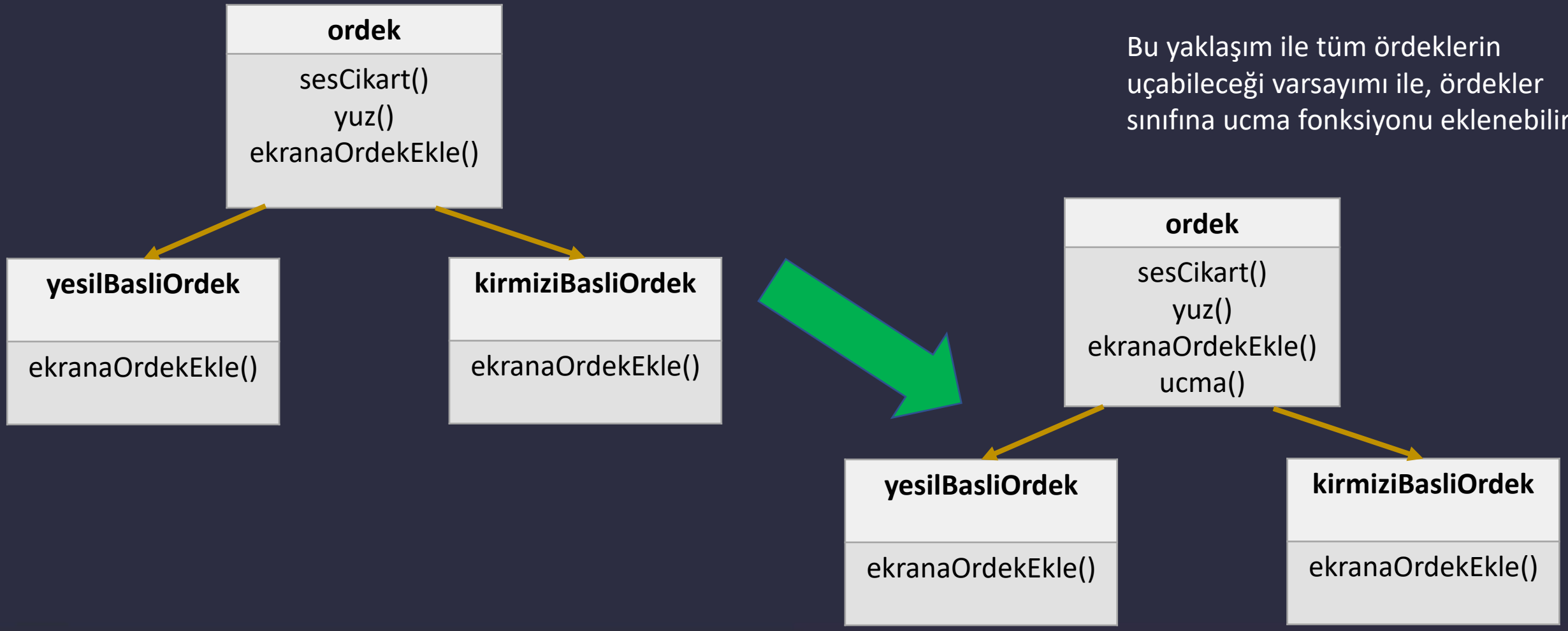
Tasarım Kalıpları

İlk durum



Tasarım Kalıpları

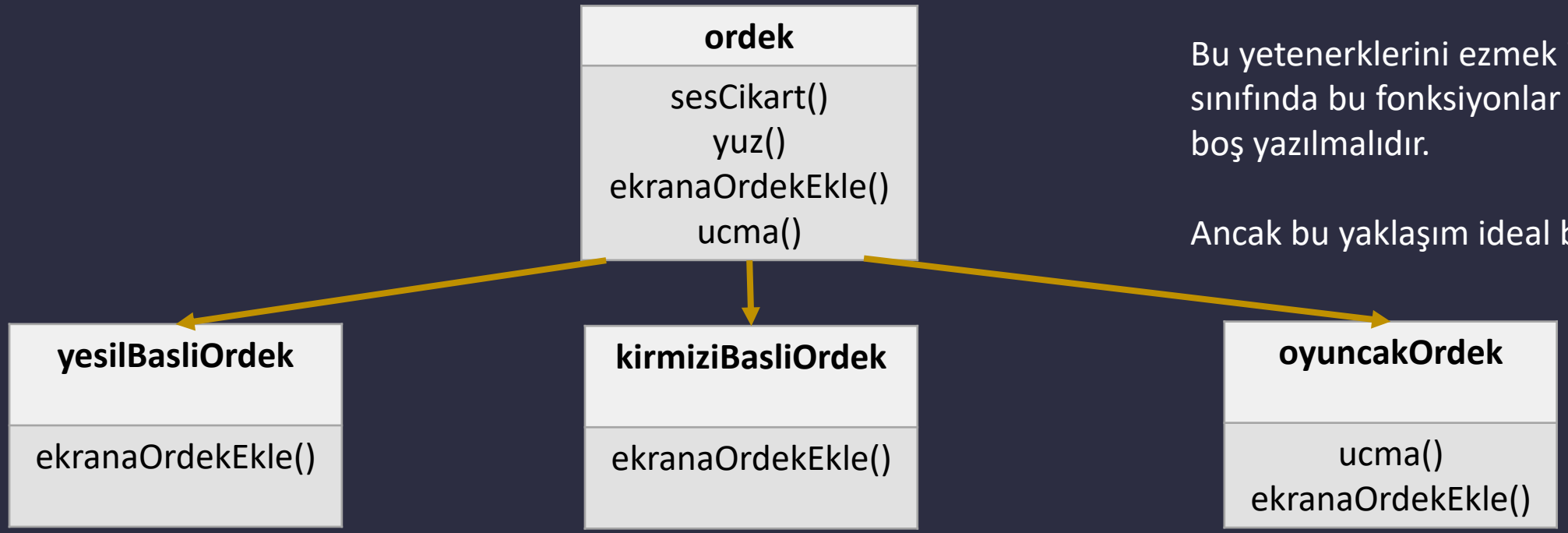
Ördeklerin uçması isteği geldiğinde;



Bu yaklaşım ile tüm ördeklerin uçabileceği varsayımı ile, ördekler sınıfına ucma fonksiyonu eklenebilir

Tasarım Kalıpları

Yeni bir ördek sınıfı geldiğinde;



Oyuncak ordekler uçamaz ve ses çıkaramaz.

Ancak ördekler sınıfından türediği için sesCikart ve ucma fonksiyonlarına sahip.

Bu yetenerklerini ezmek için, oyuncakOrdek sınıfında bu fonksiyonlar tekrar tanımlanıp içi boş yazılmalıdır.

Ancak bu yaklaşım ideal bir çözüm değildir.

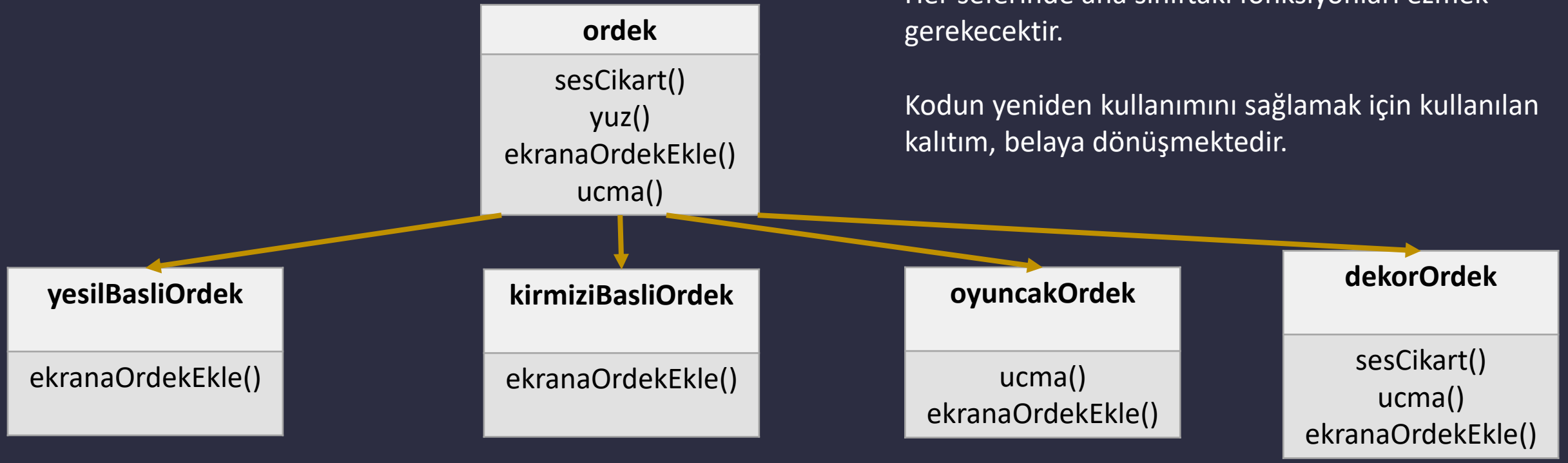
Tasarım Kalıpları

Yeni bir ördek sınıfı geldiğinde;

Çünkü her zaman farklı bir sınıf gelip, ana sınıftaki özelliklerden bazılarına sahip olmayabilir.

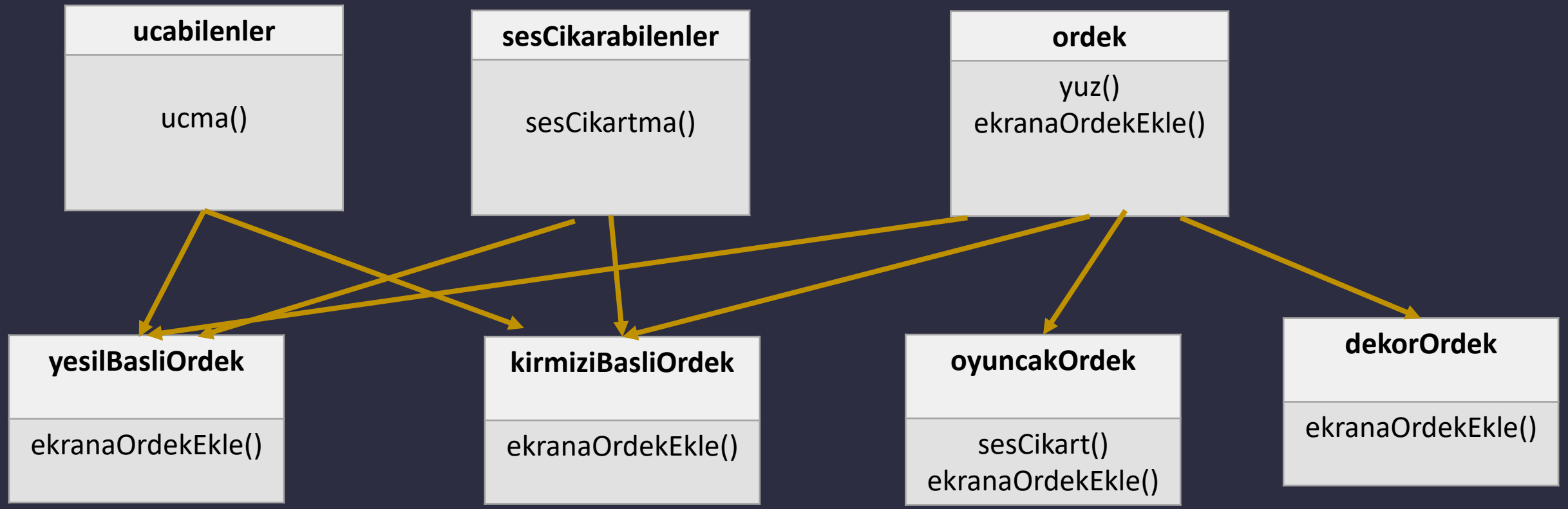
Her seferinde ana sınıftaki fonksiyonları ezmek gerekecektir.

Kodun yeniden kullanımını sağlamak için kullanılan kalıtım, belaya dönüşmektedir.



Tasarım Kalıpları

Yeni bir ördek sınıfı geldiğinde;



Tasarım Kalıpları

Fabrika (Factory)

En sık kullanılan tasarım kalıplarındandır.

Kullanıcı (Programcı)'ya öyle bir arayüz vermektedir ki, oluşturulacak bir nesnenin türüne, o arayüzü içeren sınıfın, alt sınıfları karar verebilmektedir.

Bir sınıfta, hangi alt sınıfın yaratılması gerektiği baştan belli olmayan durumlarda tercih edilebilir.

Tasarım Kalıpları

Fabrika (Factory)

Örnek:

Bir kargo uygulaması bulunmakta ve uygulamada taşımacılık için sadece kamyon kullanılmaktadır. Kamyonların işlemleri kamyon isimli sınıfta tutulmakta ve kodun büyük çoğunluğu bu sınıfın içerisindedir.

Tasarım Kalıpları

Fabrika (Factory)

Örnek:

Uygulama popülerleştikçe yeni taşımacılık üniteleri ekleme gereksinimi doğdu ve gemi taşımacılığı uygulamaya eklenmek isteniyor.

Tasarım Kalıpları

Fabrika (Factory)

Örnek:

Böyle bir durumda gemi isminde bir sınıf eklemek, tüm kod mimarisini değiştirecektir. Üstelik yeni bir sınıf eklenmesi istendiğinde (örn. uçak) bu değişiklikler, tekrar tekrar yapılması gerekecektir.

Sonuç olarak, kodun içerisinde bir biri içerisine geçmiş koşul kontrolleri ile (if) yapılmış oldukça karmaşık bir kod ortaya çıkacaktır.

Tasarım Kalıpları

Fabrika (Factory)

Örnek 2:

Bir dilden diğer bir dile çeviri yapan bir uygulama bulunmaktadır. Şu anda 10 dile kadar çeviri yapmaktadır. Uygulama popülerleşti ve 5 yeni dile ekleme ihtiyacı oluştu.

Bu durumda, kodun büyük çoğunluğu mevcut kodlar ile iç içe olduğu için yeni bir dil eklemek için büyük bir efor gerekecektir.

Tasarım Kalıpları

Örnek Kod Parçacığı

```
class fransizca:
    sozluk = {"car": "voiture", "bike": "bicyclette", "cycle":"cyclette"}

    def cevir(self, msg):
        return self.sozluk.get(msg, msg)

class ispanyolca:
    sozluk = {"car": "coche", "bike": "bicicleta", "cycle":"ciclo"}

    def cevir(self, msg):
        return self.sozluk.get(msg, msg)

class ingilizce:
    def cevir(self, msg):
        return msg

fr = fransizca()
en = ingilizce()
sp = ispanyolca()

mesaj = ["car", "bike", "cycle"]

for msg in mesaj:
    print(en.cevir(msg))
    print(fr.cevir(msg))
    print(sp.cevir(msg))
    print()
```

Çıktı

```
car
voiture
coche

bike
bicyclette
bicicleta

cycle
cyclette
ciclo
```

Yeni bir sınıf eklendiğinde, tek tek objeler oluşturulmalıdır, mevcut kod'a eklemeler yapılacaktır.

Tasarım Kalıpları

Örnek Kod Parçacığı

```
class fransizca:
    sozluk = {"car": "voiture", "bike":
"bicyclette", "cycle":"cyclette"}

    def cevir(self, msg):
        return self.sozluk.get(msg, msg)

class ispanyolca:
    sozluk = {"car": "coche", "bike":
"bicicleta", "cycle":"ciclo"}

    def cevir(self, msg):
        return self.sozluk.get(msg, msg)

class ingilizce:
    def cevir(self, msg):
        return msg


def Factory(language ="English"):
    diller = {
        "French": fransizca,
        "English": ingilizce,
        "Spanish": ispanyolca,
    }

    return diller[language]()
```

Örnek Kod Parçacığı

```
fr = Factory("French")
en = Factory("English")
sp = Factory("Spanish")

mesaj = ["car", "bike", "cycle"]

for msg in mesaj:
    print(en.cevir(msg))
    print(fr.cevir(msg))
    print(sp.cevir(msg))
    print()
```

Çıktı

```
car
voiture
coche

bike
bicyclette
bicicleta

cycle
cyclette
ciclo
```

Programcı (Client) kodunda sadece fabrika metodu ile konuşulur. Sınıf detayları ile programcı uğraşmaz.

Tasarım Kalıpları

Fabrika Sınıfı

Tasarım Kalıpları

Örnek Kod Parçacığı

```
class sekiller():
    def alan():
        pass

    def cevre():
        pass

class dikdortgen(sekiller):
    def __init__(self, boy, en):
        self.boy = boy
        self.en = en

    def alan(self):
        return self.en * self.boy

    def cevre(self):
        return 2 * (self.en + self.boy)

class kare(sekiller):
    def __init__(self, width):
        self.en = en

    def alan(self):
        return self.en ** 2

    def cevre(self):
        return 4 * self.en
```

Örnek Kod Parçacığı

```
class cember(sekiller):
    def __init__(self, cap):
        self.cap = cap

    def alan(self):
        return 3.14 * self.cap * self.cap

    def cevre(self):
        return 2 * 3.14 * self.cap


def fabrika(isim):
    if isim == 'cember':
        radius = input("Cap girin: ")
        return cember(float(radius))

    elif isim == 'dikdortgen':
        height = input("Boy girin: ")
        width = input("En girin: ")
        return dikdortgen(int(height),
int(width))

    elif isim == 'kare':
        width = input("En girin: ")
        return kare(int(width))
```

Örnek Kod Parçacığı

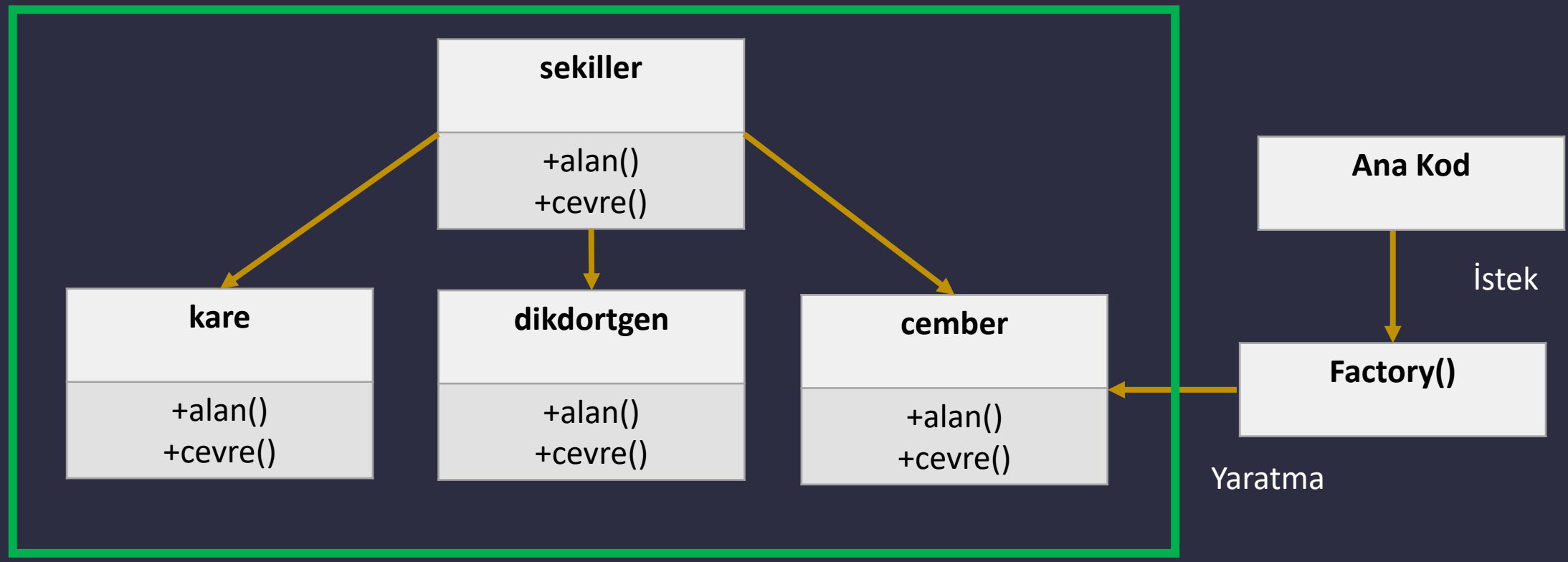
```
sekilAdi = "dikdortgen"
donenSekil = fabrika(sekilAdi)

print(f"Yaratilan obje
{type(donenSekil)}")
print(f"Alan: {donenSekil.alan()}")
print(f"Cevre: {donenSekil.cevre()}")
```

Çıktı

```
Boy girin: 4
En girin: 5
Yaratilan obje <class '__main__.dikdortgen'>
Alan: 20
Cevre: 18
```

Tasarım Kalıpları

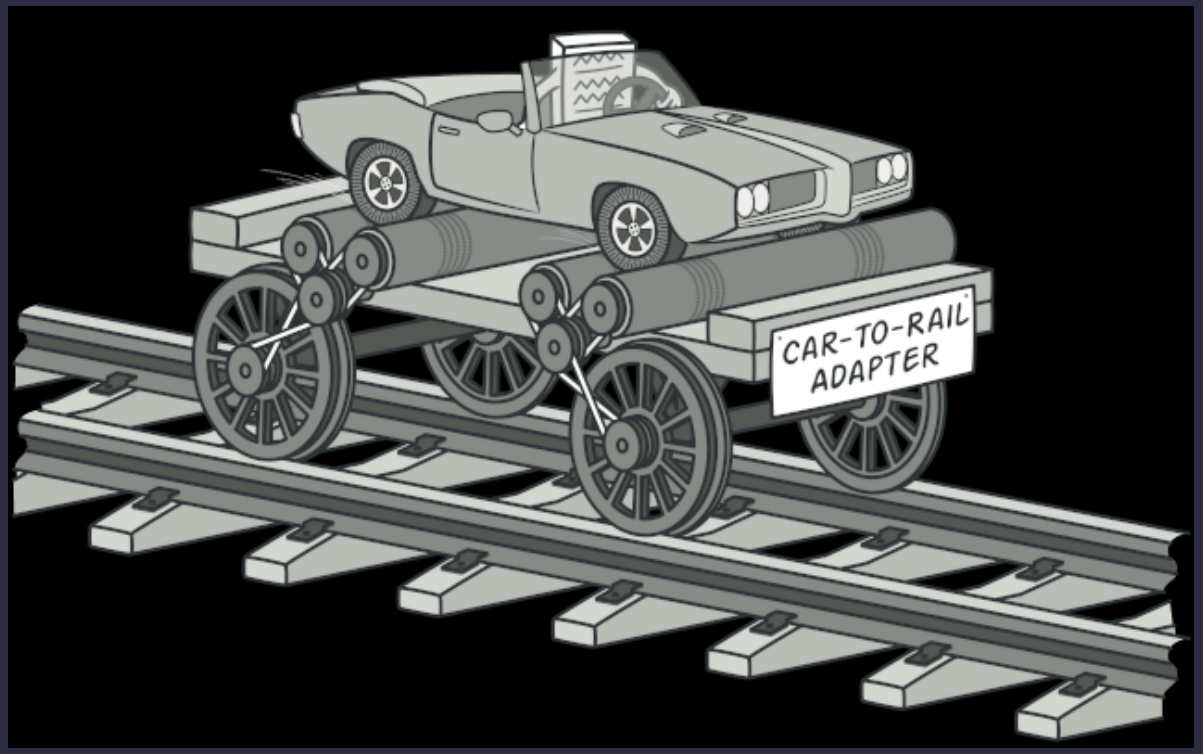
Fabrika Sınıfı



Tasarım Kalıpları

Adapter

Adaptör tasarım kalıbı, uyumsuz arayüzlerin dönüşümünü sağlayan bir tasarım kalıbıdır.



Tasarım Kalıpları

Örnek Kod Parçacığı

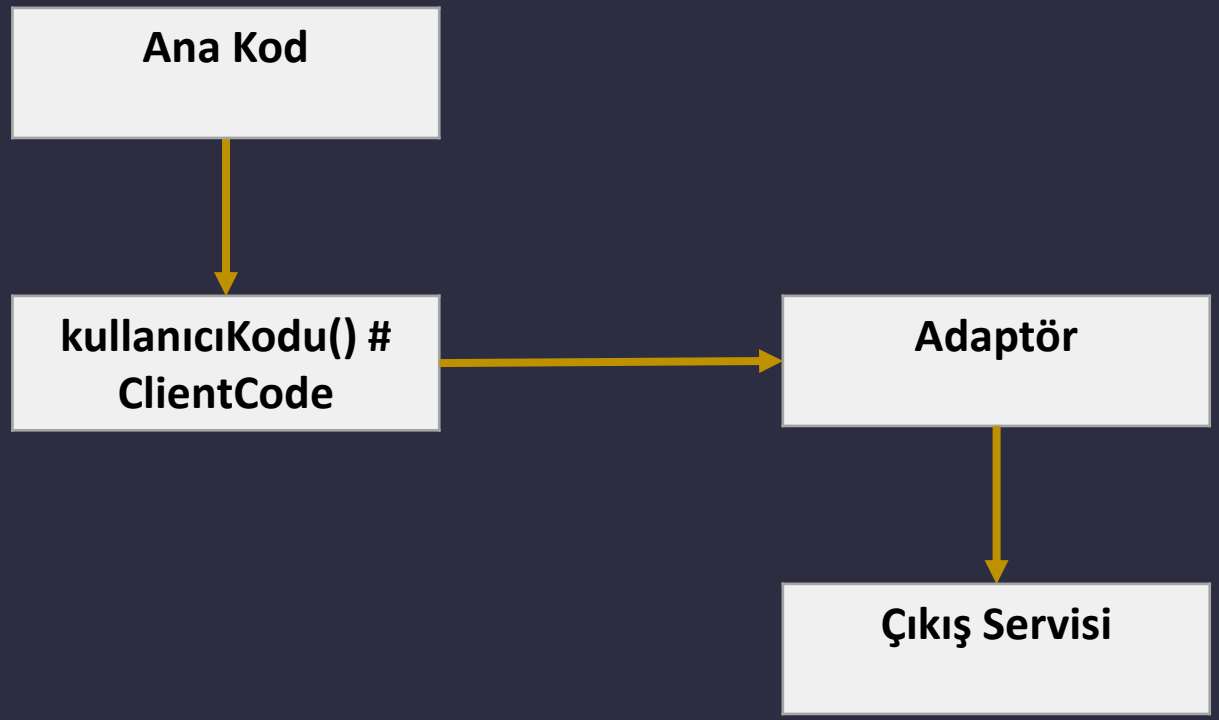
```
class hedef:
    def cikisVer(self):
        return "Merhaba"

class adapteOlacak:
    def girisAl(self):
        return "abahreM"

class adaptor(hedef, adapteOlacak):
    def cikisVer(self):
        return f"Adaptor: {self.girisAl()[::-1]}"

def kullaniciKodu(arg):
    print(arg.cikisVer(), end="")


hedef1 = hedef()
kullaniciKodu(hedef1)
print("\n")

adapteOlacak1 = adapteOlacak()
print(f"Adapte Olacak: {adapteOlacak1.girisAl()}",
end="\n\n")

adaptor1 = adaptor()
kullaniciKodu(adaptor1)
```

Çıktı

```
Merhaba

Adapte Olacak: abahreM

Adaptor: Merhaba
```

Adaptor sınıfı, giriş'i beklenen çıkış'a döndürmektedir.

Adaptör Sınıfı



Tasarım Kalıpları

Örnek Kod Parçacığı

```
class adapteOlacak:
    def __init__(self, sayi):
        self.sayi = sayi

    def girisAl(self):
        return self.sayi

class adaptor(adapteOlacak):
    def cikisVer(self):
        return self.sayi * 1.609344

def kullaniciKodu(arg):
    print(arg.cikisVer())


adaptor1 = adaptor(3.5)
kullaniciKodu(adaptor1)
```

Çıktı

5.632704

Mil girişi alıp KM'a dönüştüren adaptör sınıfı