Strategy Pattern

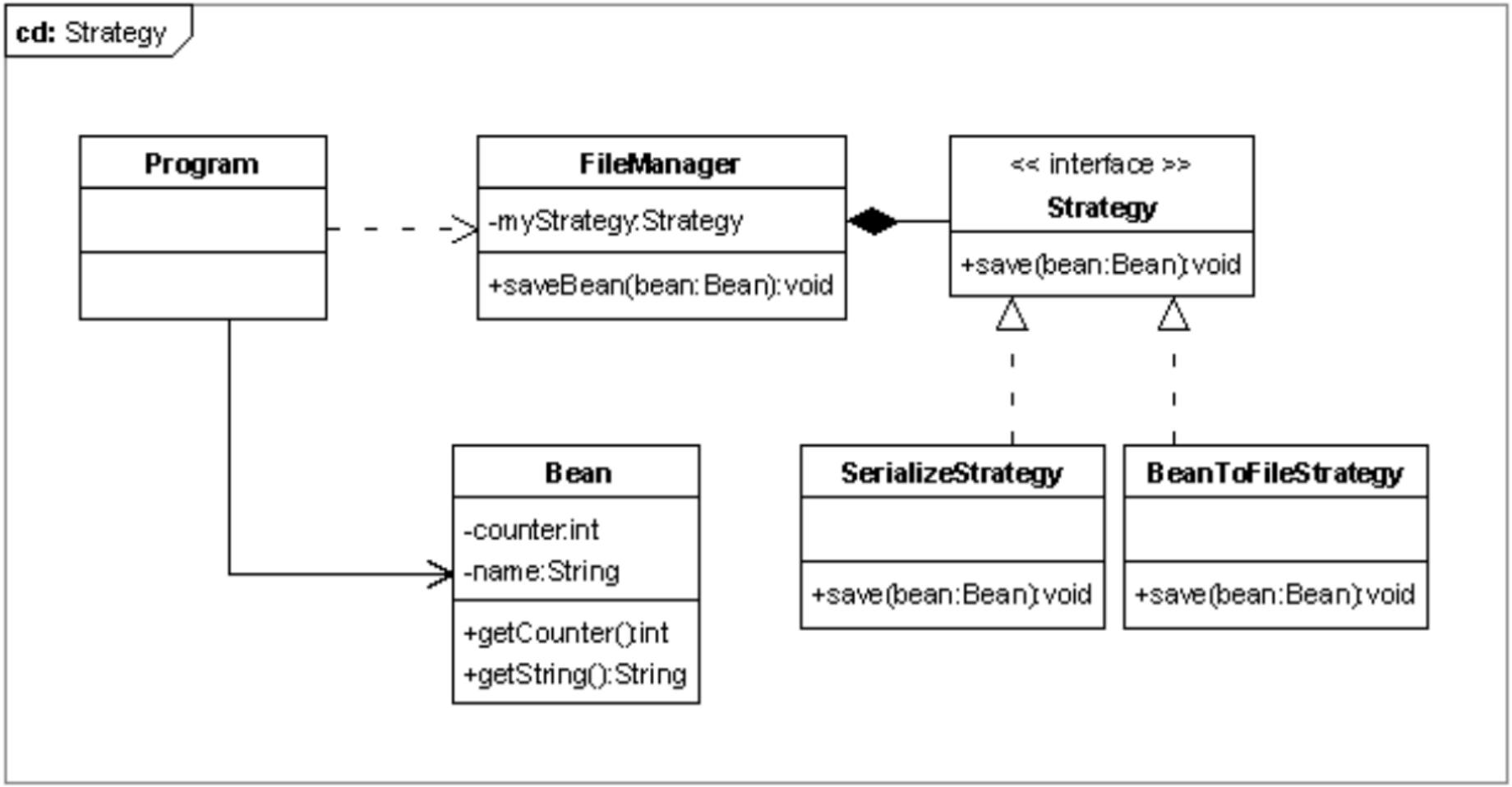
Dr. Öğr. Üyesi Fatih ÖZYURT

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Strateji (Strategy Pattern)

- Bir işlemi yerine getirmek için birden fazla yöntem (algoritma) mevcut olabilir.
- **Yerine** göre bir yöntem seçip, uygulamak için strateji tasarım şablonu kullanılır. Her yöntem (algoritma) bir sınıf içinde **implemente** edilir.
- Strategy tasarım deseni, bir algoritma ailesi tanımlamamızı, her birini ayrı bir sınıfa koymamızı ve nesnelerinin birbiriyle değiştirilebilir hale getirmenizi sağlayan **davranışsal** bir tasarım modelidir.

Strateji (Strategy Pattern)



Strateji (Strategy Pattern)

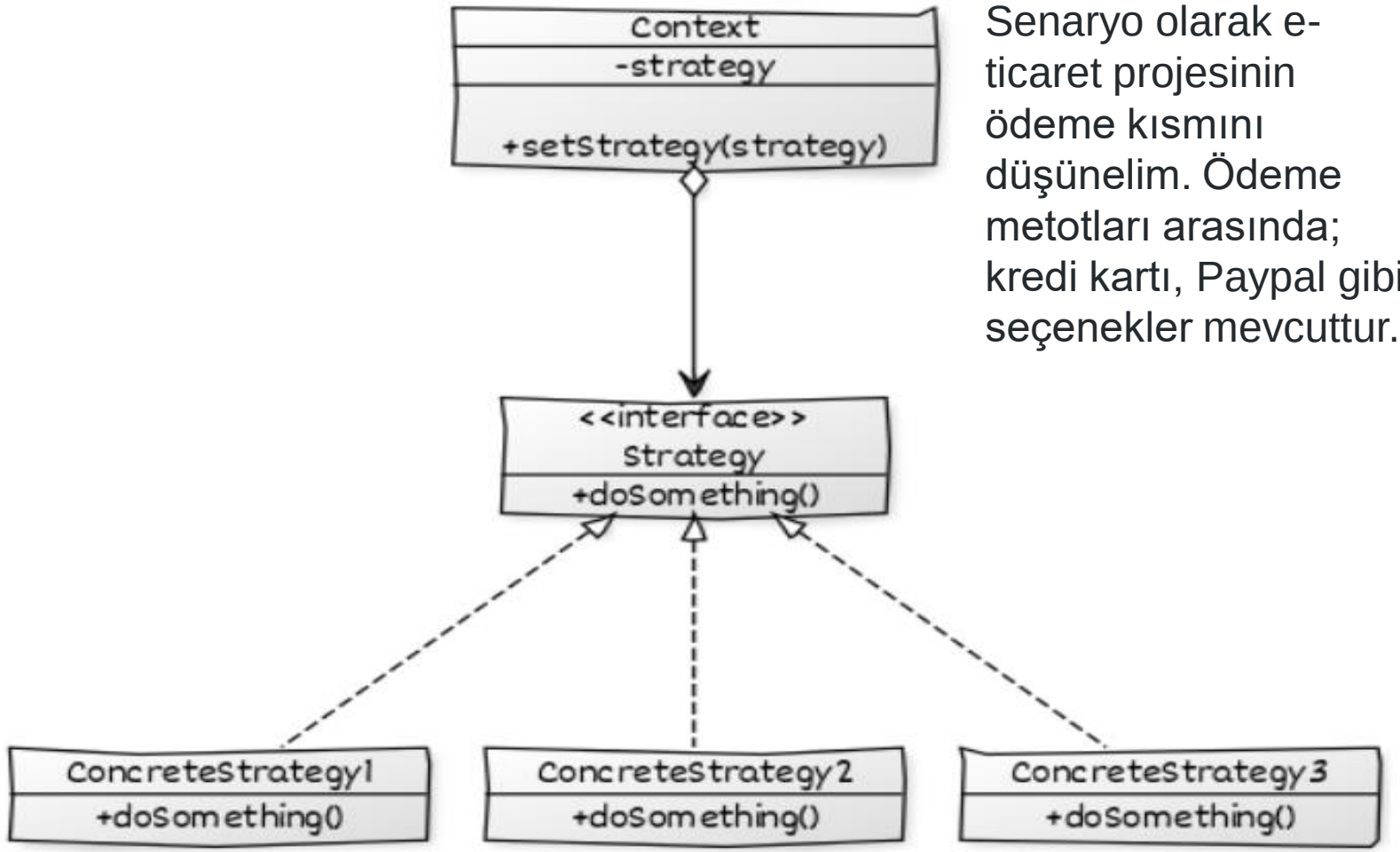
- Bean ismindeki bir sınıftan olan nesneleri iki değişik yöntemle bir dosyada tutmak istiyoruz.
- Bean basit bir Java sınıfıdır ve counter ve name isimlerinde iki sınıf değişkenine sahiptir. get() metotları üzerinden sınıf değişkenlerine ulaşılır.

Strateji (Strategy Pattern)

- Strateji tasarım şablonunu uygulayabilmemiz için Strategy isminde bir interface sınıfı tanımlamamız gerekiyor.
- Bu interface sınıfı bünyesinde altsınıflar tarafından implemente edilmesi gereken metot ya da metotlar bulunur.
- Örneğimizde basit bir Java nesnesini bir dosyaya aktarmak üzere save(Bean bean) isminde bir metot tanımlıyoruz. Uygulamak istediğimiz yönteme göre Strategy interface sınıfını implemente eden bir algoritma seçerek, işlemi gerçekleştireceğiz.
- Strategy interface sınıfı aşağıdaki yapıya sahiptir:

```
public interface Strategy {
    void save(Bean bean);
}
```

Strateji (Strategy Pattern)



Senaryo olarak e-ticaret projesinin ödeme kısmını düşünelim. Ödeme metotları arasında; kredi kartı, Paypal gibi seçenekler mevcuttur.

Strateji (Strategy Pattern)

```
interface IPayment
{
    void Pay(int amount);
}

class PaypalPayment : IPayment
{
    private string _email;
    private string _password;

    public PaypalPayment(string email, string password)
    {
        _email = email;
        _password = password;
    }

    public void Pay(int amount)
    {
        // Kredi kartı ödeme işlemlerinin gerçekleştiği yer.
        Console.WriteLine("Paid by Paypal.");
    }
}
```

// UML diyagramındaki Strategy arayüzüne denk gelmektedir.
// Benzer işlemleri yapan sınıflar bu arayüzden türeyecektir.

// UML diyagramındaki ConcreteStrategy sınıfına denk gelmektedir.
// UML diyagramındaki Strategy arayüzüne denk gelen arayüzü uyguluyor.

Strateji (Strategy Pattern)

```
class CreditCartPayment : IPayment
{
    private string _name;
    private string _cardNumber;
    private string _cvv;
    private string _dateOfExpiry;

    public CreditCartPayment(string name, string ccNum, string cvv, string
expiryDate)

    {
        _name = name;
        _cardNumber = ccNum;
        _cvv = cvv;
        _dateOfExpiry = expiryDate;
    }

    public void Pay(int amount)
    {
        // Kredi kartı ödeme işlemlerinin gerçekleştiği yer.
        Console.WriteLine("Paid by credit cart.");
    }
}
```

// UML diyagramındaki ConcreteStrategy sınıfına denk gelmektedir.
// UML diyagramındaki Strategy arayüzüne denk gelen arayüzü uyguluyor

Strateji (Strategy Pattern)

```
// Sepette bulunan ürünleri temsil eden sınıf.
// Ürün kodu ve fiyat bilgisi temel olarak yeterlidir.
class Item
{
    private string _upcCode;
    private int _price;

    public Item(string upcCode, int price)
    {
        _upcCode = upcCode;
        _price = price;
    }

    public int GetPrice()
    {
        return _price;
    }
}
```

Strateji (Strategy Pattern)

```
// UML diyagramındaki Context sınıfına denk gelmektedir.
// UML diyagramındaki Strategy(IPayment) arayüzüne ait referansı tutmaktadır.
class ShoppingCart
{
    private List<Item> _items;
    private IPayment _payment;

    public ShoppingCart()
    {
        _items = new List<Item>();
    }

    public void SetPaymentMethod(IPayment payment)
    {
        _payment = payment;
    }

    public void AddItem(Item item)
    {
        _items.Add(item);
    }

    public int CalculateTotal()
    {
        int sum = 0;
        foreach (Item item in _items)
        {
            sum += item.GetPrice();
        }
        return sum;
    }

    public void Pay()
    {
        int amount = CalculateTotal();
        _payment.Pay(amount);
    }
}
```

Strategy tasarım şablonu ne zaman kullanılır?

- Bir işlemi birden fazla yöntem (algoritma) ile implemente etmek için streteji tasarım şablonum kullanılır. Sistem gereksimleri doğrultusunda en uygun yöntem seçilerek, işlemin gerçekleştirilmesinde kullanılır.
- Bir işlemin birden fazla yapılması, çalışma zamanında algoritma değişikliği gibi durumlarda kullanılır, karışık if-else durumundan da kurtarır.
- Soyutlamadan dolayı da **Loosely Coupled** bir uygulama halini alır.

Referanslar

1. Özcan acar Design pattern kitabı

2. https://javabeginnerstutorial.com/

3. Yusuf Yılmaz Ders notları

Sorularınız