Questions & Answers
The Abstract Factory

Hans Philippi

October 9, 2019

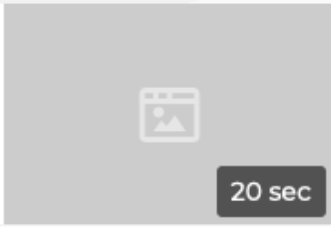
Outline

- Kahoot last week
- From Strategy to Factory
- Quiz
- Questions?

Kahoot last week

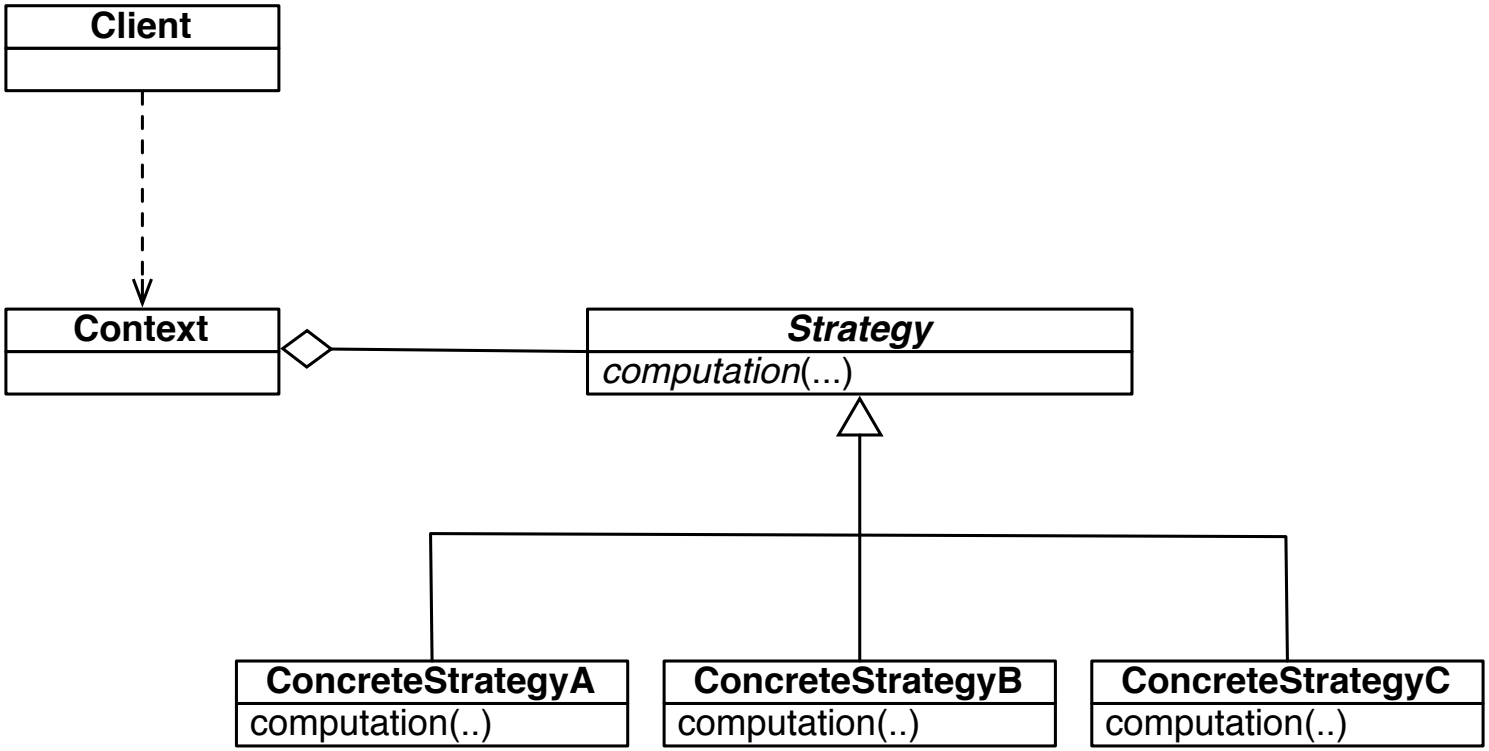
4 - Quiz

A Java program connects to a DBMS using JDBC/SQLite. We will switch the actual DBMS later on.



Smells like Facade	✓
Smells like Adapter	✗
Smells like neither of them	✓
Smells like both of them	✗

Strategy revisited



Separate the selection of algorithm from the implementation of the algorithm. This allows for the selection to be made in context.

E-commerce example revisited

- What was the advantage of the Strategy Pattern?
- Recall the original code of SalesOrder ...
- ... and notice the lack of *cohesion*

```
switch (country)
    case NL:
        priceNL =
            Finance.DollarEuroConvertor(price);
        localPrice = 1.21f * priceNL;
    ...
    case IRL:
    ...
        if (article.IRLTaxClass = ‘‘standard’’)
            localPrice = 1.23f * priceIRL;
        else if (article.IRLTaxClass = ...
```

Strategy revisited

SalesOrder has the following responsibilities:

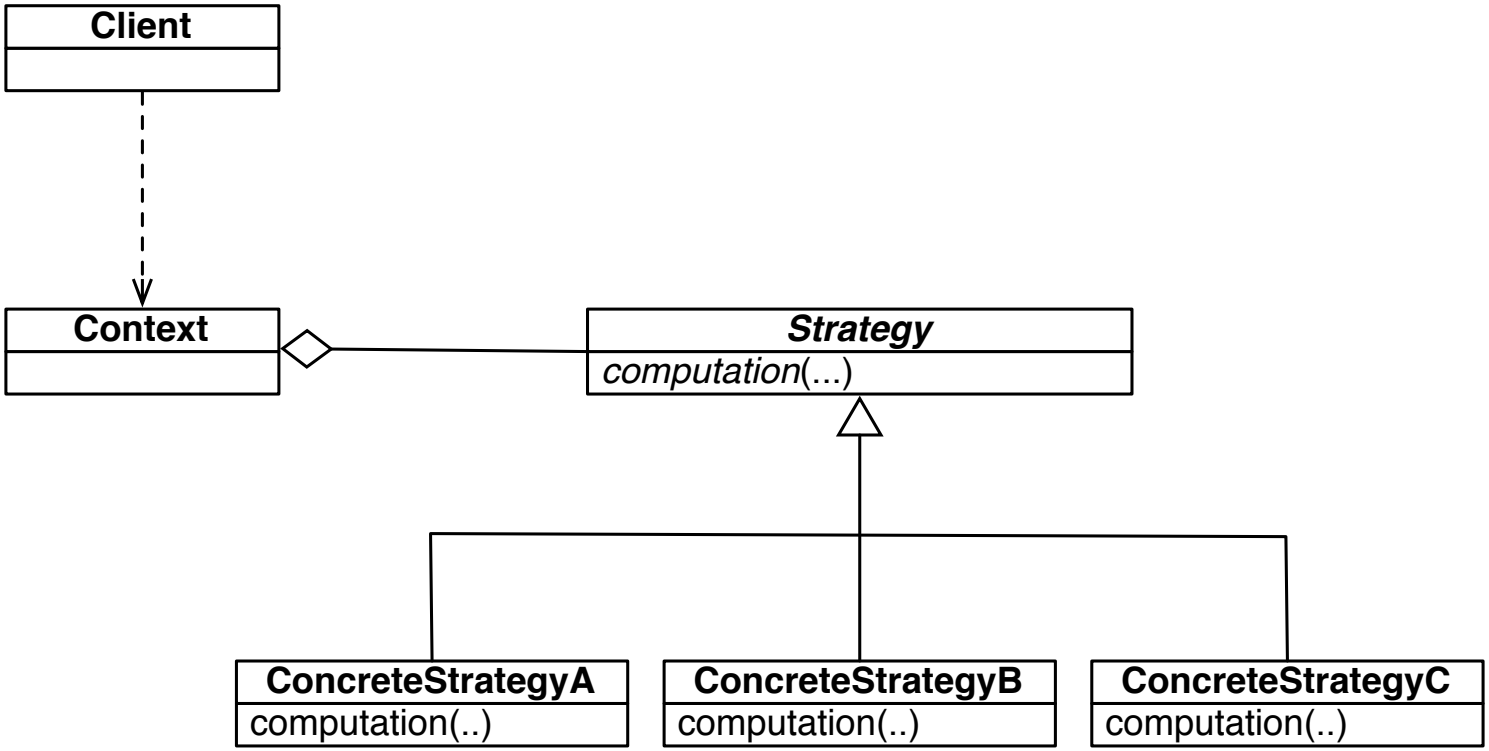
- Deal with the global control of a sales transaction
- Deal with the variation in currencies
- Deal with the variation in taxes
- Deal with the variation in shipping procedures
- Deal with the variation in ...

... where *Deal with* means: be aware of the naughty details

What was the advantage of the Strategy Pattern?

- ... and notice the lack of *cohesion*
- Recall: *"Find what varies, and encapsulate it"*
- Recall: *"Program to an interface, not to an interpretation"*

Strategy revisited



Separate the selection of algorithm from the implementation of the algorithm. This allows for the selection to be made in context.

Strategy patterns applied

```
abstract class Tax {
    public abstract float Calculate
        (float:  amount)
    ...
abstract class Currency {
    public abstract float Convert
        (float:  amount)
    ...
public class TaxNL : Tax {
    public override float Calculate
        (float:  amount)
    { return 1.21f * amount; }
    ...
```

Strategy revisited

```
Tax myTax;
Currency myCurrency;
...
case NL:
    myTax = new TaxNL();
    myCurrency = new CurrencyEuro();
...
case UK:
    myTax = new TaxUK();
    myCurrency = new CurrencyPoundSterling();
...
localPrice = myCurrency.Convert(price);
finalPrice = myTax.Calculate(localPrice);
```

Strategy: gains

SalesOrder has the following responsibilities:

- Deal with the global control of a sales transaction
- ~~Deal with the variation in currencies~~
- ~~Deal with the variation in taxes~~
- ~~Deal with the variation in shipping procedures~~
- ~~Deal with the variation in ...~~
- ...

Strategy: gains & doubts

SalesOrder has the following responsibilities:

- Deal with the global control of a sales transaction
- ~~Deal with the variation in currencies~~
- ~~Deal with the variation in taxes~~
- ~~Deal with the variation in shipping procedures~~
- ~~Deal with the variation in ...~~
- ... *but SalesOrder still has to know that Ireland has the euro!*

Audience:
shouldn't we somehow group the issues for each country?

Pattern: Factories

- At some point we will need to create concrete instances of the abstract classes representing the different strategies (Tax, Currency, ...)
- To create concrete instances, we have to know about concrete subclasses of the abstract class
- In concreto: *... SalesOrder still has to know that Ireland has the euro!*
- To avoid breaking abstraction, we want to separate the *creation* of objects from their *use*

Creating the factories

```
public abstract class SalesFactory
{
    public abstract Tax GetTax();
    public abstract Currency GetCurrency();
    ...
}
```

Creating the factories

```
public class IRL_Factory :  SalesFactory
{
    public override Tax GetTax()
    {
        return new IRLTax();
        // refers to concrete Tax subclass
    }
    public override Tax GetCurrency()
    {
        return new CurrencyEuro();
        // refers to concrete Currency subclass
    }
```

What did we gain?

- In the original Strategy version, SalesOrder still had to be aware of annoying details like *Ireland has the euro*
- In a factory approach, we will see this choice in the code of SalesOrder:

```
switch (country)
    case IRL:
        mySalesFactory = new IRL_Factory();
...
    case NL:
        mySalesFactory = new NL_Factory();
...
// in the main code of SalesOrder:
myCurr = mySalesFactory.GetCurrency();
myTax = mySalesFactory.GetTax();
```

What did we gain?

But SalesOrder still has to make the choice with respect to the country?

- Yes, but that is OK!
- Recall the GRASP principle Information Expert:
- SalesOrder knows about the country ...
- ... so SalesOrder chooses the country ...
- ... but is unaware of any details involving that choice
- Cohesion!