Chain of Responsibilty

Gruppe: PATT_ 25

Indhold

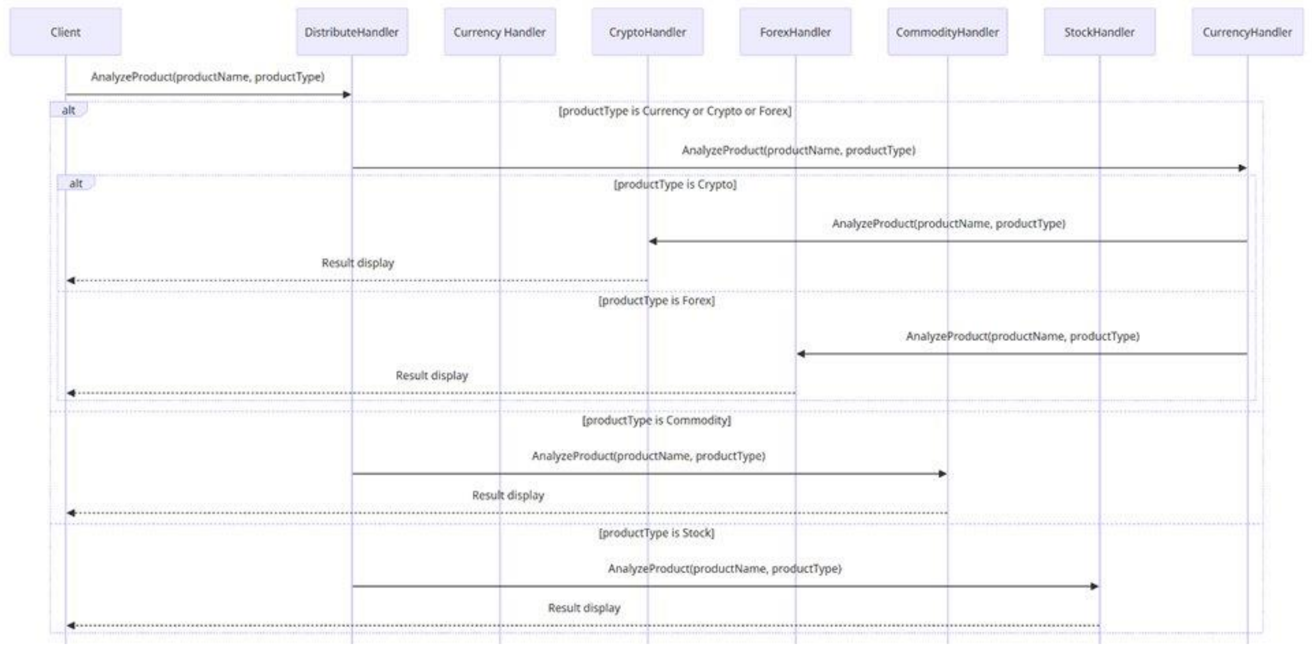
Beskrivelse af pattern

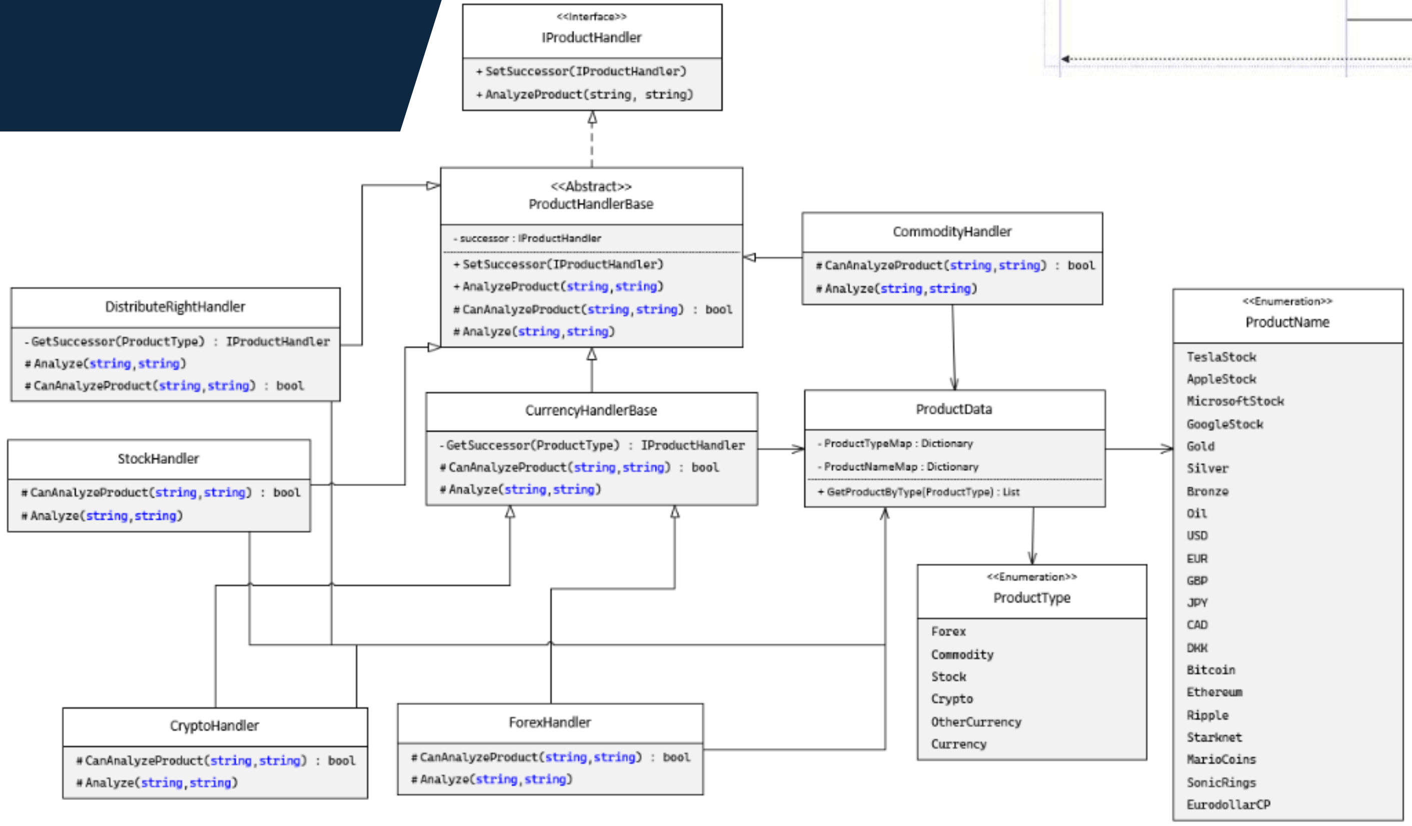
HVAD ER CHAIN OF RESPONSIBILITY?

FORMÅL MED DESIGNMØNSTERET

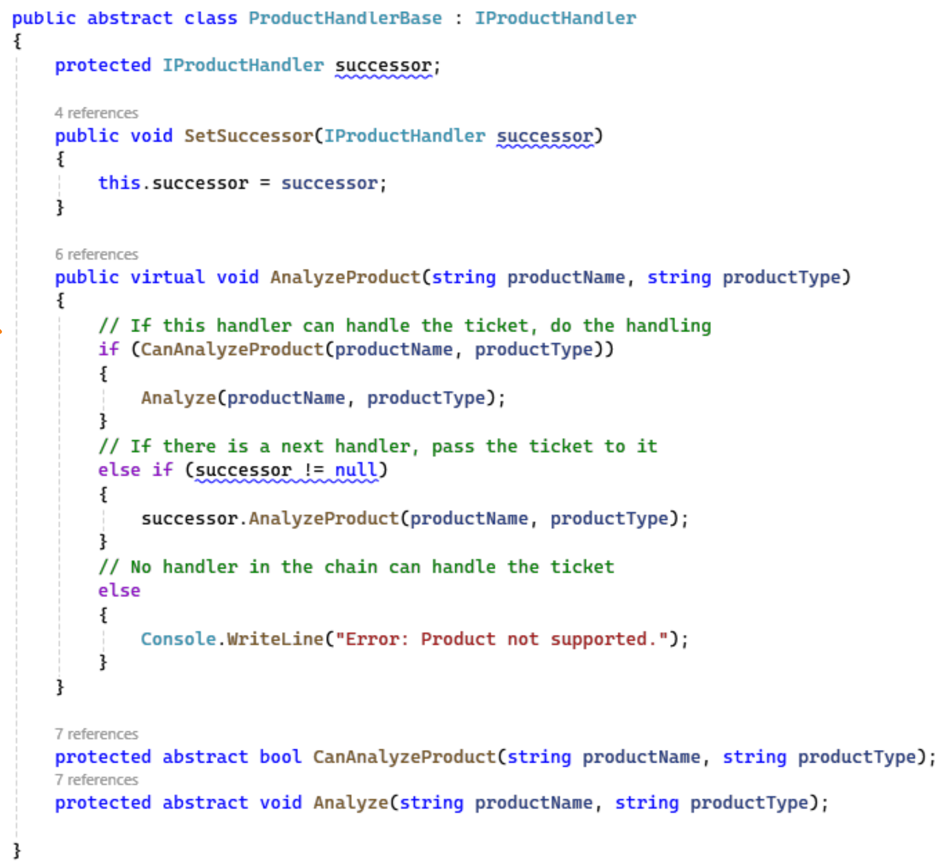
OVERBLIK OVER COR I FORHOLD TIL KURSETS ANDRE DESIGNMØNSTRE

Udvikling af design

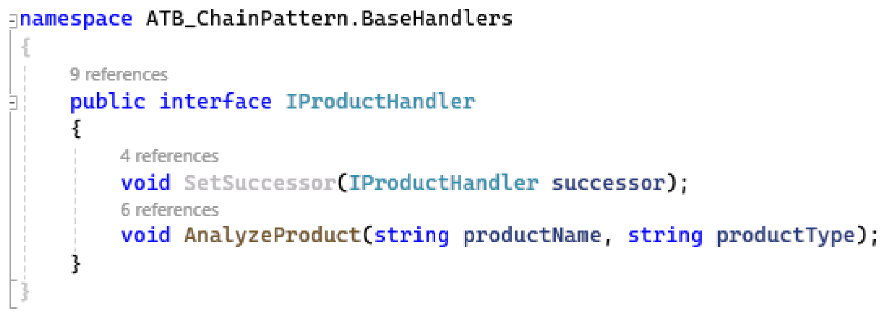

Client
DistributeHandler
Currency Handler
CryptoHandler
ForexHandler
CommodityHandler
StockHandler
CurrencyHandler
AnalyzeProduct(productName, productType)
alt
[productType is Currency or Crypto or Forex]
AnalyzeProduct(productName, productType)
alt
[productType is Crypto]
AnalyzeProduct(productName, productType)
Result display
[productType is Forex]
AnalyzeProduct(productName, productType)
Result display
[productType is Commodity]
AnalyzeProduct(productName, productType)
Result display
[productType is Stock]
AnalyzeProduct(productName, productType)
Result display


<<Interface>>
IProductHandler
+ SetSuccessor(IProductHandler)
+ AnalyzeProduct(string, string)
<<Abstract>>
ProductHandlerBase
- successor : IProductHandler
+ SetSuccessor(IProductHandler)
+ AnalyzeProduct(string, string)
# CanAnalyzeProduct(string, string) : bool
# Analyze(string, string)
CommodityHandler
# CanAnalyzeProduct(string, string) : bool
# Analyze(string, string)
DistributeRightHandler
- GetSuccessor(ProductType) : IProductHandler
# Analyze(string, string)
# CanAnalyzeProduct(string, string) : bool
StockHandler
# CanAnalyzeProduct(string, string) : bool
# Analyze(string, string)
CurrencyHandlerBase
- GetSuccessor(ProductType) : IProductHandler
# CanAnalyzeProduct(string, string) : bool
# Analyze(string, string)
ProductData
- ProductTypeMap : Dictionary
- ProductNameMap : Dictionary
+ GetProductByType(ProductType) : List
<<Enumeration>>
ProductName
TeslaStock
AppleStock
MicrosoftStock
GoogleStock
Gold
Silver
Bronze
Oil
USD
EUR
GBP
JPY
CAD
DKK
Bitcoin
Ethereum
Ripple
Starknet
MarioCoins
SonicRings
EurodollarCP
<<Enumeration>>
ProductType
Forex
Commodity
Stock
Crypto
OtherCurrency
Currency
CryptoHandler
# CanAnalyzeProduct(string, string) : bool
# Analyze(string, string)
ForexHandler
# CanAnalyzeProduct(string, string) : bool
# Analyze(string, string)

Implementering

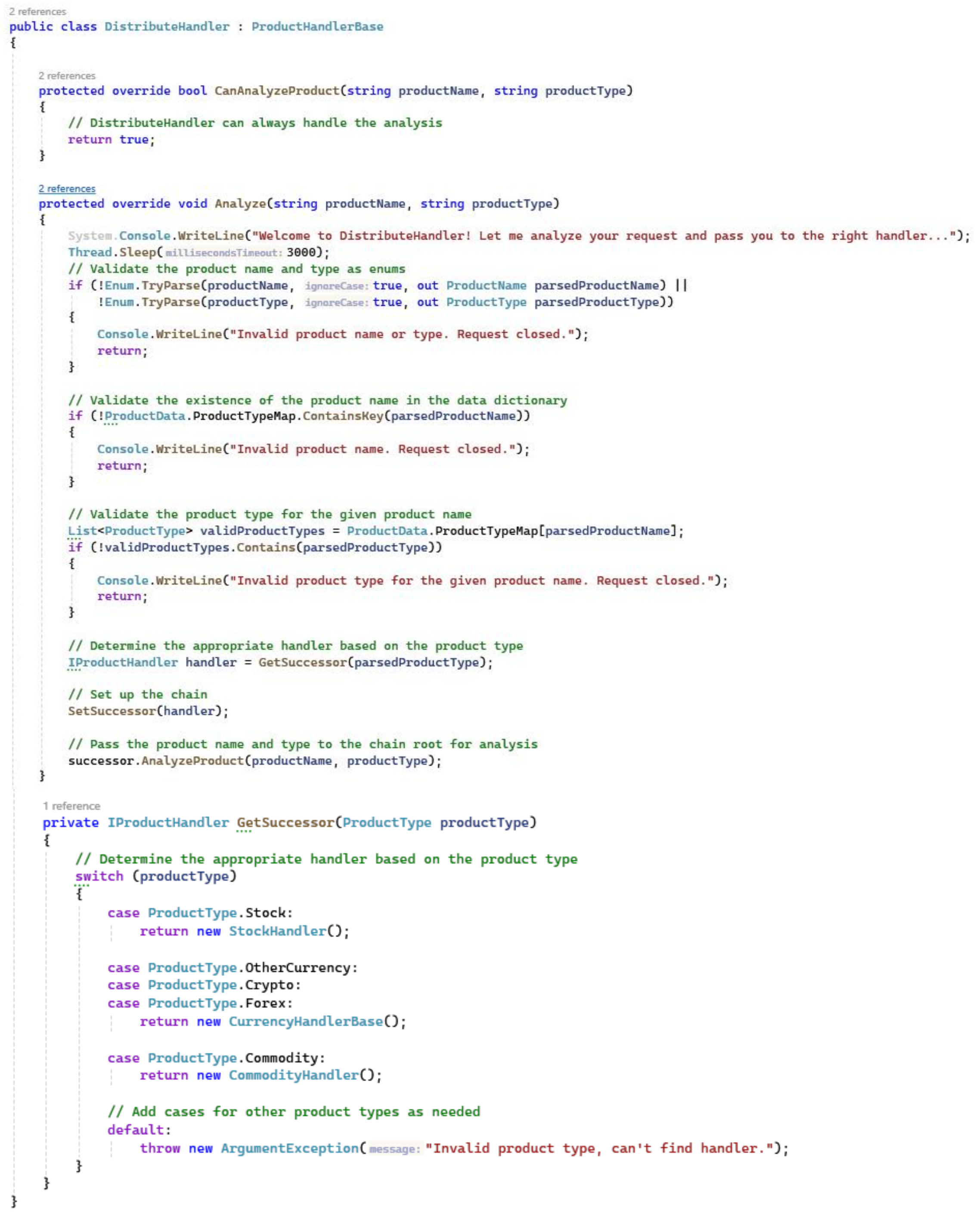


```
public abstract class ProductHandlerBase : IProductHandler
{
    protected IProductHandler successor;

    4 references
    public void SetSuccessor(IProductHandler successor)
    {
        this.successor = successor;
    }

    6 references
    public virtual void AnalyzeProduct(string productName, string productType)
    {
        // If this handler can handle the ticket, do the handling
        if (CanAnalyzeProduct(productName, productType))
        {
            Analyze(productName, productType);
        }
        // If there is a next handler, pass the ticket to it
        else if (successor != null)
        {
            successor.AnalyzeProduct(productName, productType);
        }
        // No handler in the chain can handle the ticket
        else
        {
            Console.WriteLine("Error: Product not supported.");
        }
    }

    7 references
    protected abstract bool CanAnalyzeProduct(string productName, string productType);
    7 references
    protected abstract void Analyze(string productName, string productType);

}
```



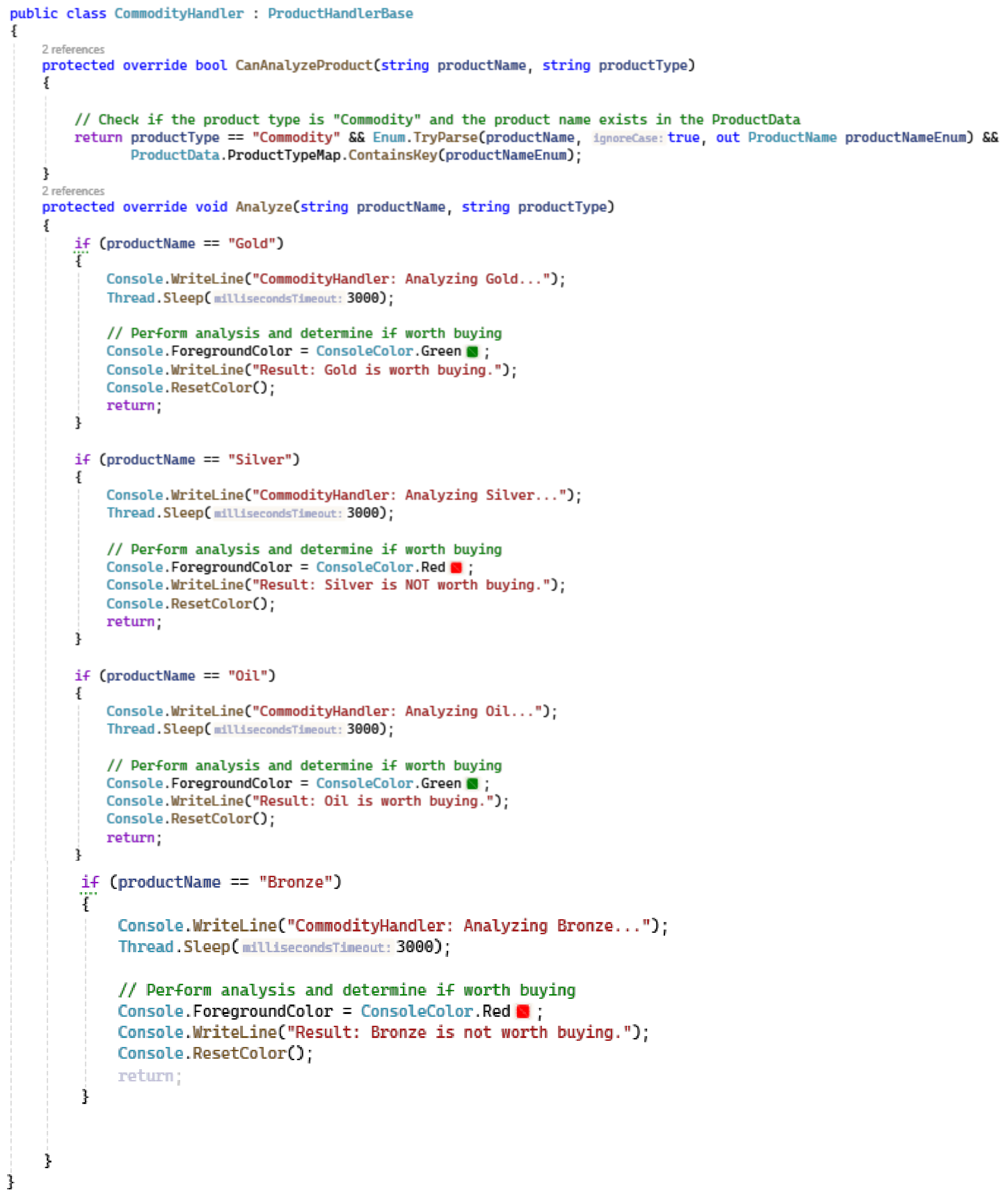
```
namespace ATB_ChainPattern.BaseHandlers
{
    9 references
    public interface IProductHandler
    {
        4 references
        void SetSuccessor(IProductHandler successor);
        6 references
        void AnalyzeProduct(string productName, string productType);
    }
}
```

Implementering



```
2 references
public class DistributeHandler : ProductHandlerBase
{
    2 references
    protected override bool CanAnalyzeProduct(string productName, string productType)
    {
        // DistributeHandler can always handle the analysis
        return true;
    }

    2 references
    protected override void Analyze(string productName, string productType)
    {
        System.Console.WriteLine("Welcome to DistributeHandler! Let me analyze your request and pass you to the right handler...");
        Thread.Sleep(millisecondsTimeout: 3000);
        // Validate the product name and type as enums
        if (!Enum.TryParse(productName, ignoreCase: true, out ProductName parsedProductName) ||
            !Enum.TryParse(productType, ignoreCase: true, out ProductType parsedProductType))
        {
            Console.WriteLine("Invalid product name or type. Request closed.");
            return;
        }

        // Validate the existence of the product name in the data dictionary
        if (!ProductData.ProductTypeMap.ContainsKey(parsedProductName))
        {
            Console.WriteLine("Invalid product name. Request closed.");
            return;
        }

        // Validate the product type for the given product name
        List<ProductType> validProductTypes = ProductData.ProductTypeMap[parsedProductName];
        if (!validProductTypes.Contains(parsedProductType))
        {
            Console.WriteLine("Invalid product type for the given product name. Request closed.");
            return;
        }

        // Determine the appropriate handler based on the product type
        IProductHandler handler = GetSuccessor(parsedProductType);

        // Set up the chain
        SetSuccessor(handler);

        // Pass the product name and type to the chain root for analysis
        successor.AnalyzeProduct(productName, productType);
    }

    1 reference
    private IProductHandler GetSuccessor(ProductType productType)
    {
        // Determine the appropriate handler based on the product type
        switch (productType)
        {
            case ProductType.Stock:
                return new StockHandler();

            case ProductType.OtherCurrency:
            case ProductType.Crypto:
            case ProductType.Forex:
                return new CurrencyHandlerBase();

            case ProductType.Commodity:
                return new CommodityHandler();

            // Add cases for other product types as needed
            default:
                throw new ArgumentException(message: "Invalid product type, can't find handler.");
        }
    }
}
```

Implementering



```
public class CommodityHandler : ProductHandlerBase
{
    2 references
    protected override bool CanAnalyzeProduct(string productName, string productType)
    {

        // Check if the product type is "Commodity" and the product name exists in the ProductData
        return productType == "Commodity" && Enum.TryParse(productName, ignoreCase: true, out ProductName productNameEnum) &&
               ProductData.ProductTypeMap.ContainsKey(productNameEnum);
    }
    2 references
    protected override void Analyze(string productName, string productType)
    {
        if (productName == "Gold")
        {
            Console.WriteLine("CommodityHandler: Analyzing Gold...");
            Thread.Sleep(millisecondsTimeout: 3000);

            // Perform analysis and determine if worth buying
            Console.ForegroundColor = ConsoleColor.Green 🟩 ;
            Console.WriteLine("Result: Gold is worth buying.");
            Console.ResetColor();
            return;
        }

        if (productName == "Silver")
        {
            Console.WriteLine("CommodityHandler: Analyzing Silver...");
            Thread.Sleep(millisecondsTimeout: 3000);

            // Perform analysis and determine if worth buying
            Console.ForegroundColor = ConsoleColor.Red 🟥 ;
            Console.WriteLine("Result: Silver is NOT worth buying.");
            Console.ResetColor();
            return;
        }

        if (productName == "Oil")
        {
            Console.WriteLine("CommodityHandler: Analyzing Oil...");
            Thread.Sleep(millisecondsTimeout: 3000);

            // Perform analysis and determine if worth buying
            Console.ForegroundColor = ConsoleColor.Green 🟩 ;
            Console.WriteLine("Result: Oil is worth buying.");
            Console.ResetColor();
            return;
        }
        if (productName == "Bronze")
        {
            Console.WriteLine("CommodityHandler: Analyzing Bronze...");
            Thread.Sleep(millisecondsTimeout: 3000);

            // Perform analysis and determine if worth buying
            Console.ForegroundColor = ConsoleColor.Red 🟥 ;
            Console.WriteLine("Result: Bronze is not worth buying.");
            Console.ResetColor();
            return;
        }


    }
}
```

Konklusion

COR: HIERARKISK HÅNDTERING AF FORESPØRGSLER

C# IMPLEMENTERING: FLEKSIBEL STRUKTUR FOR PRODUKTANALYSE

DISTRIBUTEHANDLER OG COMMODITYHANDLER: KERNEELEMENTER

SKALERING: UDVIDELSESMULIGHEDER OG KOMPLEKSE SYSTEMER