AARHUS UNIVERSITET

Software Engineering and Architecture

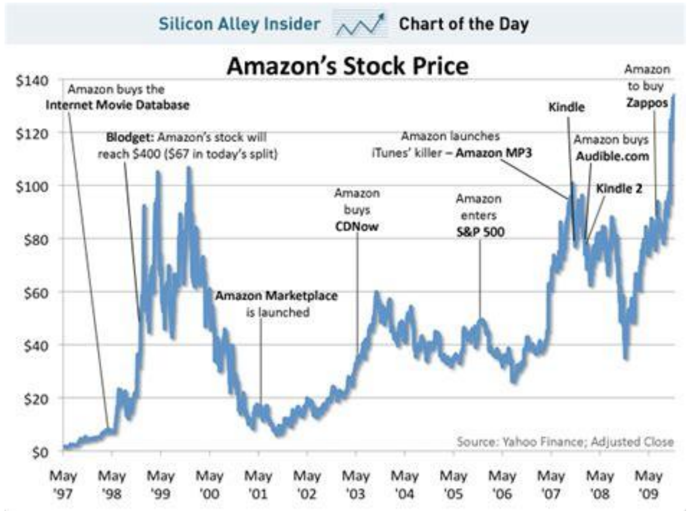
Pattern Catalog: Proxy

AARHUS UNIVERSITET

Case

- We need to show a graph of stock prices for a company.
- We use a **paid** web service to fetch on-line stock prices
  - **25cent pr request**
  - Interface:

```
interface StockPrice {
  double getQuote();
  int getServiceFee();
}
```



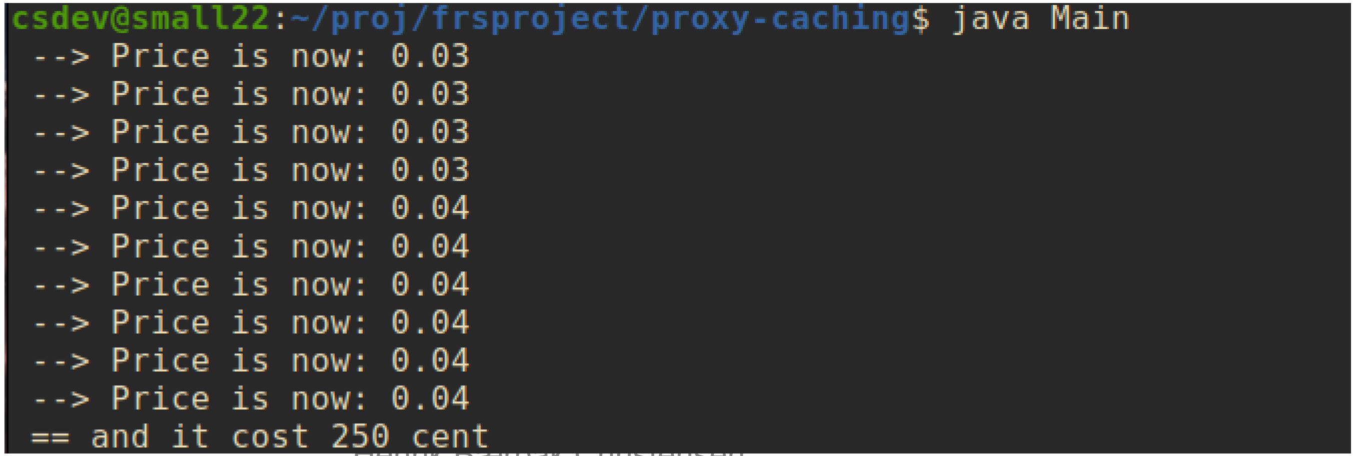
AARHUS UNIVERSITET

So, we can make a “graph”

- No, I did not want to write graph software, so I did a loop

```java
public class Main {
  public static void main(String[] args) throws InterruptedException {
    // Create connection to Webserver with stockprices
    StockPrice stockPrice = new RealStockPrice("SAS");
    // Get stock price every 2 seconds
    for (int i = 0; i < 10; i++) {
      System.out.println(" --> Price is now: " + stockPrice.getQuote());
      Thread.sleep(2000);
    }
    System.out.println(" == and it cost " + stockPrice.getServiceFee() +" cent");
  }
}
```

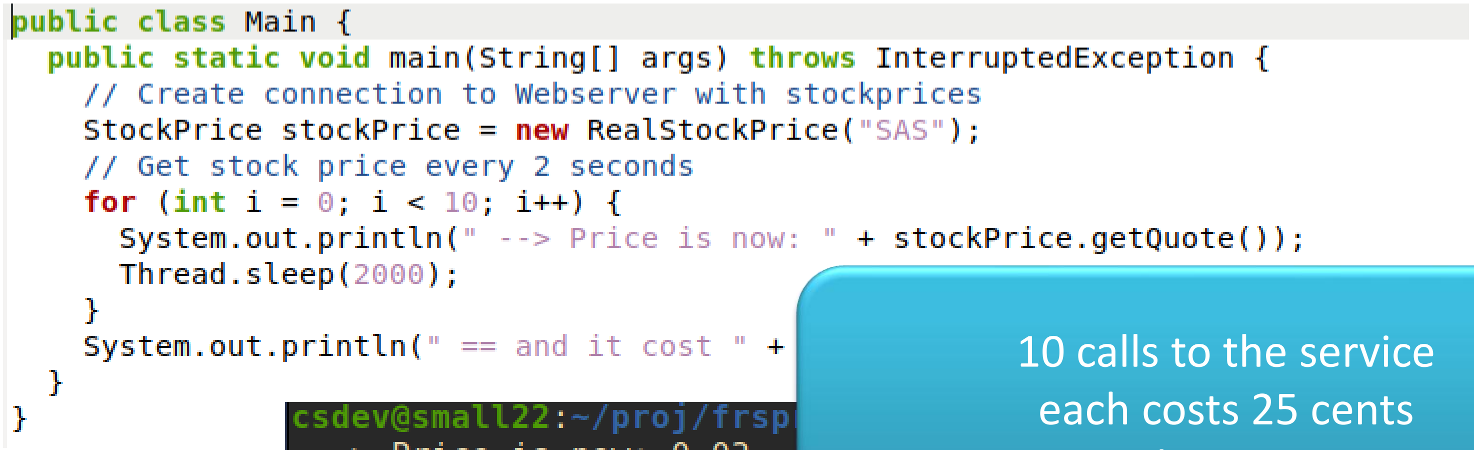
Impl., that connects to service and get the price…



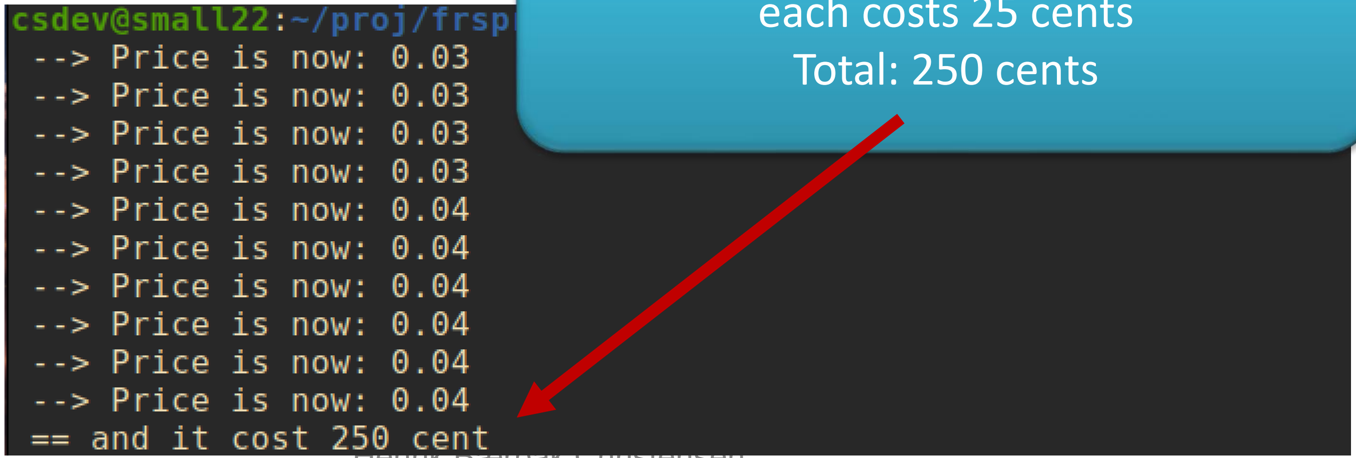
```
csdev@small22:~/proj/frsproject/proxy-caching$ java Main
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 == and it cost 250 cent
```

So, we can make a “graph”

- No, I did not want to write graph software, so I did a loop:



```
public class Main {
  public static void main(String[] args) throws InterruptedException {
    // Create connection to Webserver with stockprices
    StockPrice stockPrice = new RealStockPrice("SAS");
    // Get stock price every 2 seconds
    for (int i = 0; i < 10; i++) {
      System.out.println(" --> Price is now: " + stockPrice.getQuote());
      Thread.sleep(2000);
    }
    System.out.println(" == and it cost " +
  }
}
```



```
csdev@small22:~/proj/frsp
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 == and it cost 250 cent
```

10 calls to the service
each costs 25 cents
Total: 250 cents

But…

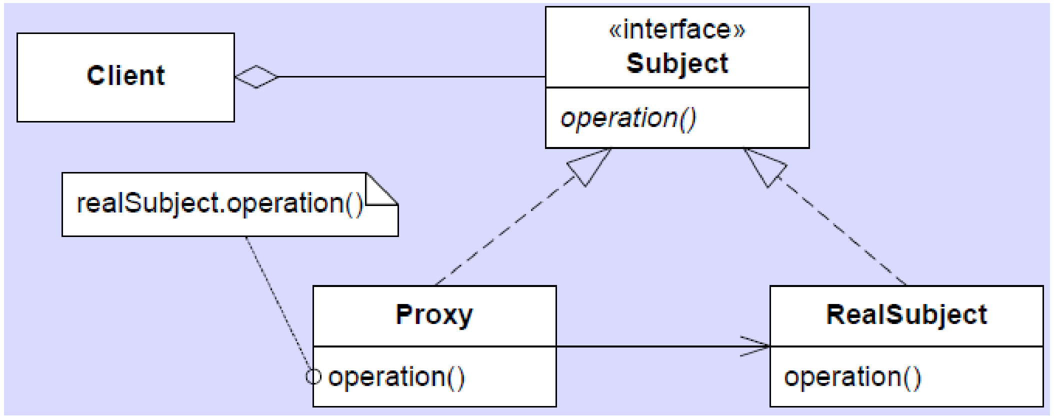
- It is costly to fetch almost the same price over and over again. Why do we not *cache* it for some period of time?
  - **Caching**: *Store a local copy of an "expensive-to-get" value, and use that for a specified period of time, to avoid expensive requests…*
- How does 3-1-2 look?
  - 3: Encapsulate variation: *We need to vary the request call – either make it or replace by local copy*
  - 1: Program to interface: *Insist client only accesses the stock request through an interface*
  - 2: Favor composition: *Compose behavior by putting an "intermediate", the proxy, that will do a real fetch after some time limit.*

The Process

- 3 + 1: Already done:

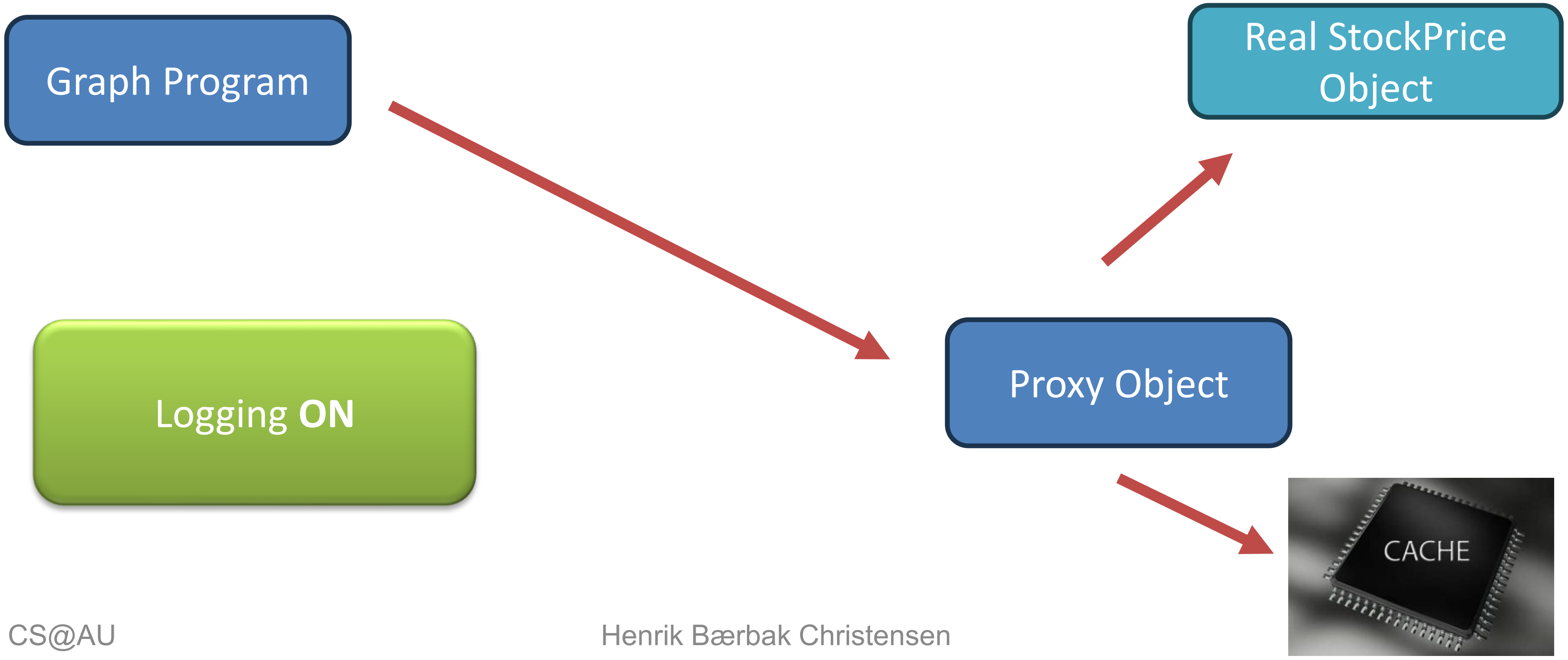
```
interface StockPrice {
  double getQuote();
  int getServiceFee();
}
```

- 2: Composition: The intermediate / 'object-in-front'



Again, Visually

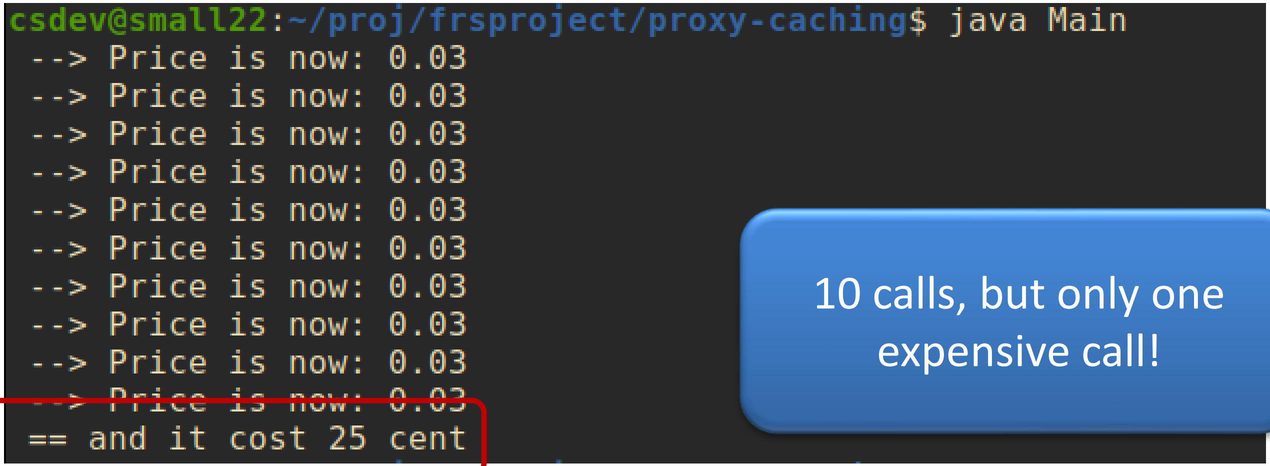
- We use an intermediate object



AARHUS UNIVERSITET

The New Design

```java
public class Main {
  public static void main(String[] args) throws InterruptedException {
    // Create connection to Webserver with stockprices
    StockPrice stockPrice = new ProxyStockPrice(new RealStockPrice("SAS"));
    // Get stock price every 2 seconds
    for (int i = 0; i < 10; i++) {
      System.out.println(" --> Price is now: " + stockPrice.getQuote());
      Thread.sleep(2000);
    }
    System.out.println(" == and it cost " + stockPrice.getServiceFee() +" cent");
  }
}
```

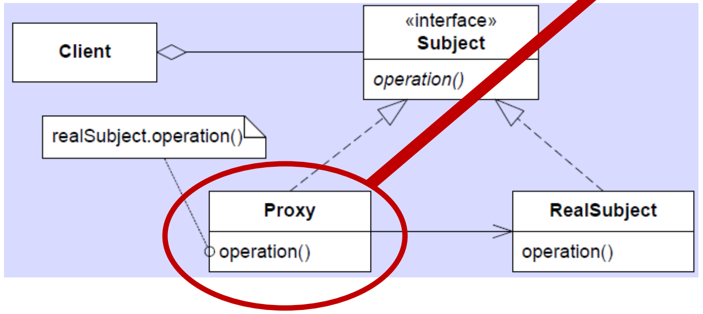


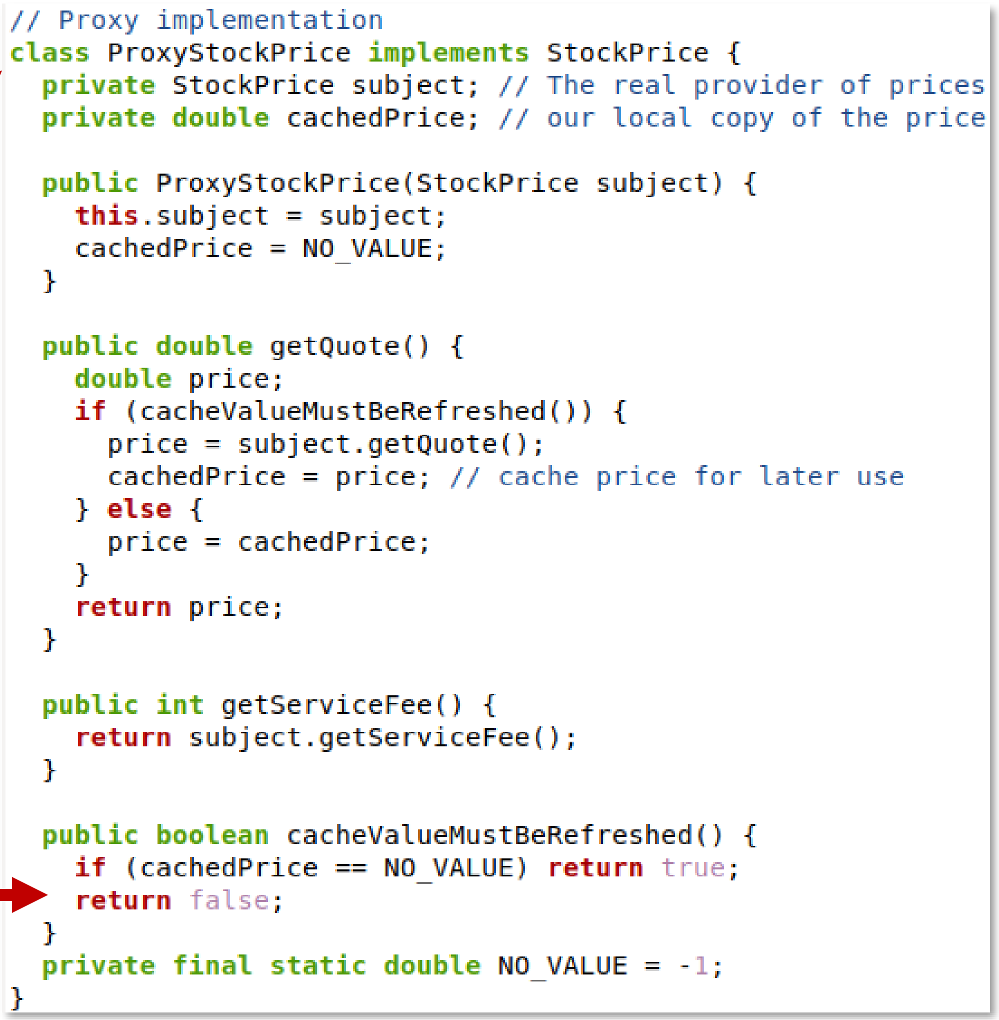
```
csdev@small22:~/proj/frsproject/proxy-caching$ java Main
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 == and it cost 25 cent
```

10 calls, but only one expensive call!

Proxy Code

- Our Proxy Code



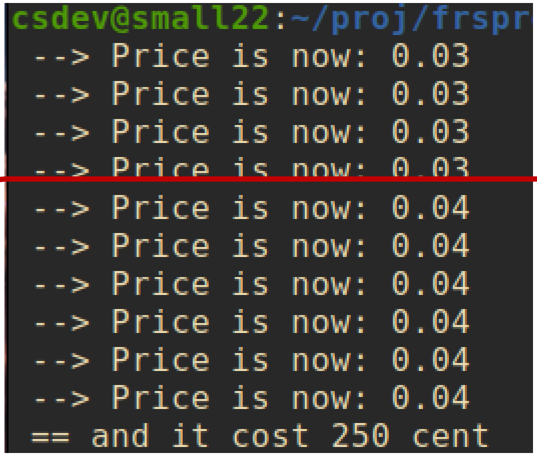


```
// Proxy implementation
class ProxyStockPrice implements StockPrice {
  private StockPrice subject; // The real provider of prices
  private double cachedPrice; // our local copy of the price

  public ProxyStockPrice(StockPrice subject) {
    this.subject = subject;
    cachedPrice = NO_VALUE;
  }

  public double getQuote() {
    double price;
    if (cacheValueMustBeRefreshed()) {
      price = subject.getQuote();
      cachedPrice = price; // cache price for later use
    } else {
      price = cachedPrice;
    }
    return price;
  }

  public int getServiceFee() {
    return subject.getServiceFee();
  }

  public boolean cacheValueMustBeRefreshed() {
    if (cachedPrice == NO_VALUE) return true;
    return false;
  }
  private final static double NO_VALUE = -1;
}
```

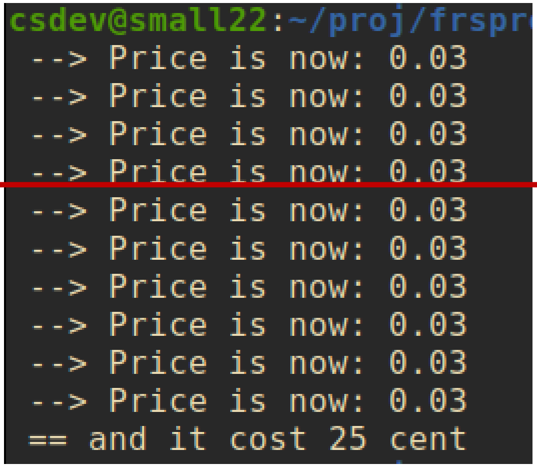
- Note: This is an *over-eager caching*!
  - Normally cache should be 'cold' after N seconds

(Caching Liability)

- A liability of caching (**not of Proxy pattern in general)** is that a local copy of course may be out-of-synch with the real value…



```
csdev@small22:~/proj/frspr
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 --> Price is now: 0.04
 == and it cost 250 cent
```

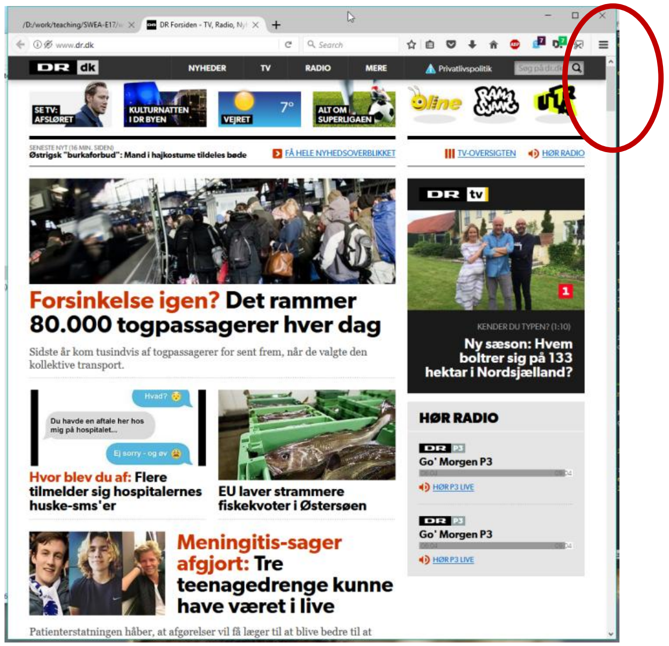


```
csdev@small22:~/proj/frspr
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 --> Price is now: 0.03
 == and it cost 25 cent
```

- *For some time, a wrong value is shown…*
  - *The real price has gone up, but we still use the cached value…*

Case used in FRS

- A web page contains a lot of images, many of which will never be displayed as they are ‘at the end of the scroll’ where the average user do not see at all. Can we *avoid downloading* them?
  - Same for a long Word document etc.
    - Loading from disk also takes time…
- But we still need image size in order to lay out the page properly,
  - So *some info needs to be fetched*



SENESTE NYT (16 MIN. SIDEN)
Østrigsk "burkaforbud": Mand i hajkostume tildeles bøde

Forsinkelse igen? Det rammer 80.000 togpassagerer hver dag

Sidste år kom tusindvis af togpassagerer for sent frem, når de valgte den kollektive transport.

Ny sæson: Hvem boltrer sig på 133 hektar i Nordsjælland?

Hvor blev du af: Flere tilmelder sig hospitalernes huske-sms'er

EU laver strammere fiskekvoter i Østersøen

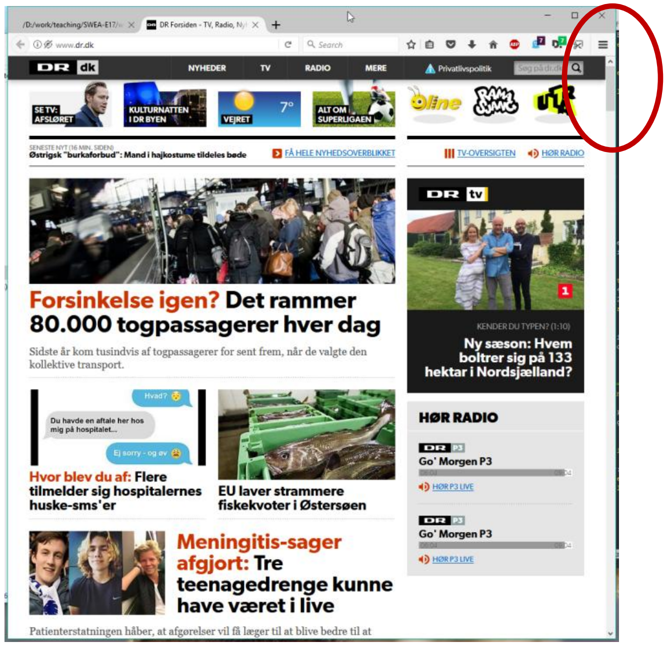
Meningitis-sager afgjort: Tre teenagedrenge kunne have været i live

Patienterstatningen håber, at afgørelser vil få læger til at blive bedre til at

AARHUS UNIVERSITET

Proxy

- *Download on demand* – i.e. only download the “heavy stuff” = the raster image, when the image become visible
- The point:
  - Load the “small data” immediately
    - Like image width and image height
  - Load the “large data” on demand
    - Like the graphics: PNG or JPEG data
- *Two step loading*
  - *i.load() (1); i.show() (2)*



Østrigsk "burkaforbud": Mand i hajkostume tildeles bøde

Forsinkelse igen? Det rammer 80.000 togpassagerer hver dag

Sidste år kom tusindvis af togpassagerer for sent frem, når de valgte den kollektive transport.

KENDER DU TYPEN? (1:10)
Ny sæson: Hvem boltrer sig på 133 hektar i Nordsjælland?

Hvor blev du af: Flere tilmelder sig hospitalernes huske-sms'er

EU laver strammere fiskekvoter i Østersøen

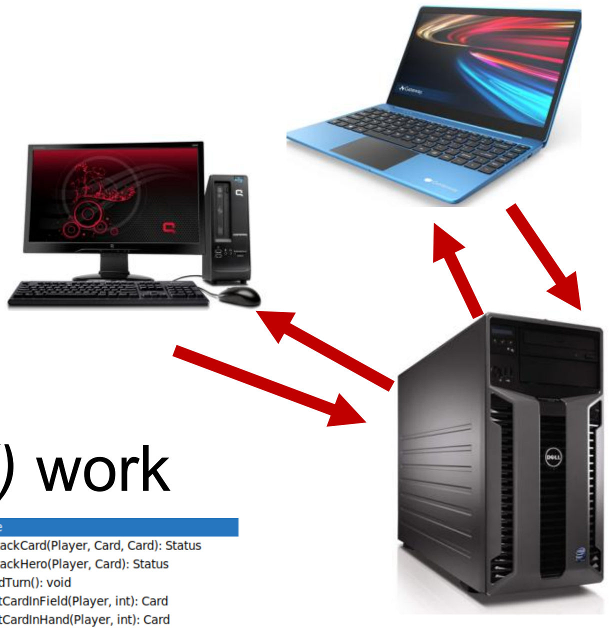
Meningitis-sager afgjort: Tre teenagedrenge kunne have været i live

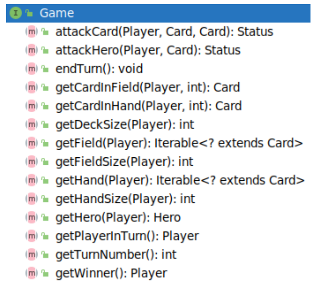
Patienterstatningen håber, at afgørelser vil få læger til at blive bedre til at

AARHUS UNIVERSITET

Another Case

- Proxy Pattern is central in *Distributed Computing*
- HotStone –
  - Bente and Arne plays a game
    - Bente on Bente’s computer
    - Arne on Arne’s computer
  - The game’s real state is on *hotstone.littleworld.dk*
- Then, how does *getCardInField()* work on Bente’s Computer ?
  - The card state is *not* on Bente’s computer !!!

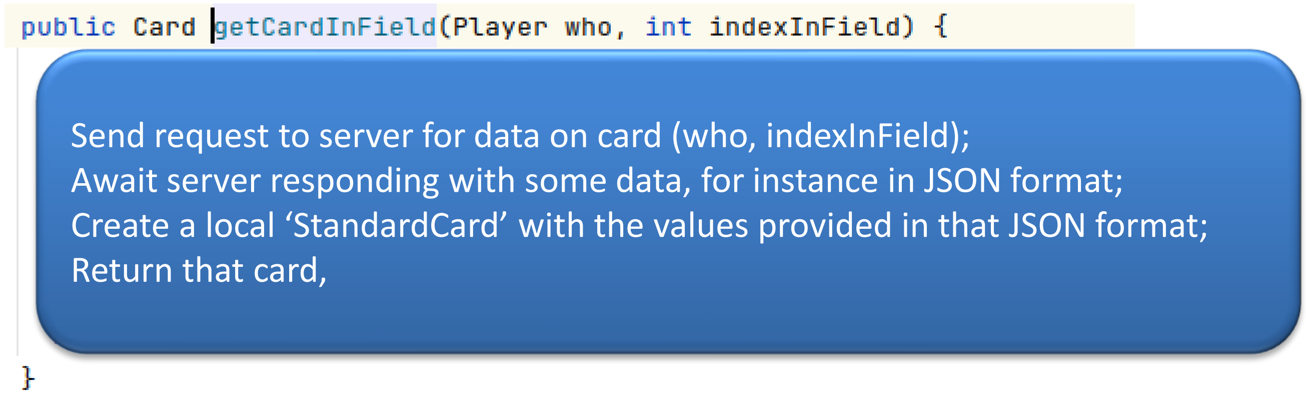




```
Game
  attackCard(Player, Card, Card): Status
  attackHero(Player, Card): Status
  endTurn(): void
  getCardInField(Player, int): Card
  getCardInHand(Player, int): Card
  getDeckSize(Player): int
  getField(Player): Iterable<? extends Card>
  getFieldSize(Player): int
  getHand(Player): Iterable<? extends Card>
  getHandSize(Player): int
  getHero(Player): Hero
  getPlayerInTurn(): Player
  getTurnNumber(): int
  getWinner(): Player
```

Proxy to the Rescue

- Proxy is a surrogate/placeholder for another object



```
public Card getCardInField(Player who, int indexInField) {
    Send request to server for data on card (who, indexInField);
    Await server responding with some data, for instance in JSON format;
    Create a local 'StandardCard' with the values provided in that JSON format;
    Return that card,
}
```

- Note – no (permanent) data on the client, all accessor and mutators become “call server, get answer”

AARHUS UNIVERSITET

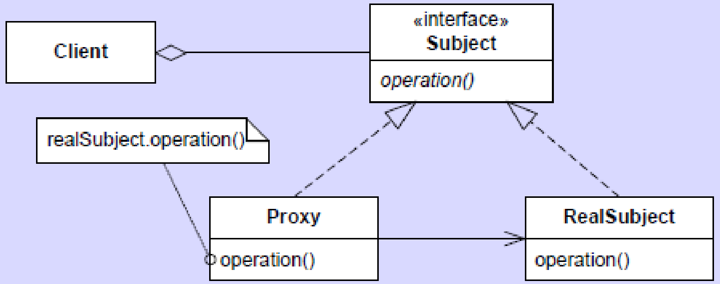
[25.1] Design Pattern: Proxy

Intent Provide a surrogate or placeholder for another object to control access to it.

Problem An object is highly resource demanding and will negatively affect the client's resource requirements even if the object is not used at all; or we need different types of housekeeping when clients access the object, like logging, access control, or pay-by-access.

Solution Define a placeholder object, the Proxy, that acts on behalf of the real object. The proxy can defer loading the real object, control access to it, or in other ways lower resource demands or implement housekeeping tasks.

Structure:



Roles A **Client** only interacts via a **Subject** interface. The **RealSubject** is the true object implementing resource-demanding operations (bandwidth, computation, memory usage, etc.) or operations that need access control (security, pay-by-access, logging, etc.). A **Proxy** implements the **Subject** interface and provides the relevant access control by holding a reference to the real subject and delegating operations to it.

Cost - Benefit It *strengthens reuse* as the housekeeping tasks are separated from the real subject operations. Thus the subject does not need to implement the housekeeping itself; and the proxy can act as proxy for several different types of real subjects.

AARHUS UNIVERSITET

Consequences

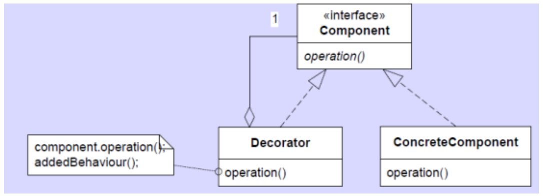
- Benefits
  - Supports all types of access control: deferred access, pay-by-access, access logging (audit trail), access control, …
  - Decouples domain behavior and access control behavior (**Different roles! Cohesion!**)
  - **Is the core technology in remote method invocation**
    - Java RMI
    - .Net Remoting

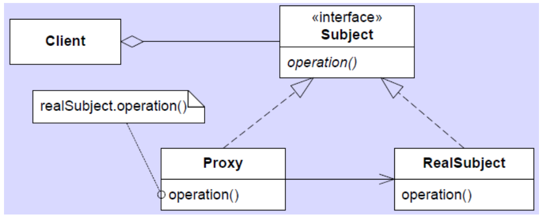
And in SWEA Broker…

AARHUS UNIVERSITET

Decorator Versus Proxy

- The look alike!
  - Indeed, the UML looks very similar





- *But they look at different things*
  - *Decorator looks at the object itself (looking* ***inside****)*
    - *Adding* behaviour to the object itself
  - *Proxy looks at the user of the object (looking* ***outside****)*
    - Monitoring/controlling who is *using* the object