Design Patterns

Proxy Pattern*

ebru@hacettepe.edu.tr
ebruakcapinarsezer@gmail.com
http://yunus.hacettepe.edu.tr/~ebru/
@ebru176

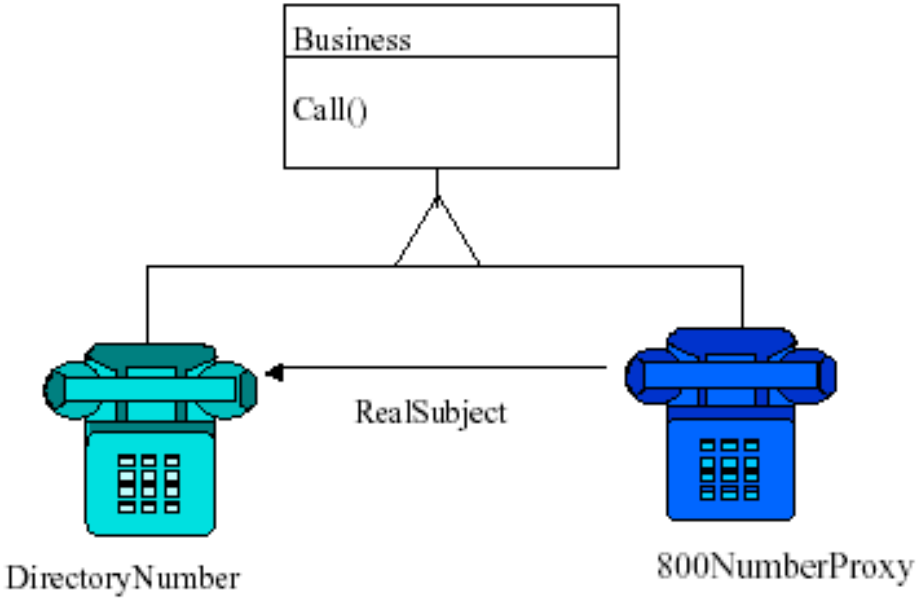
Kasım 2017

*modified from http://*courses.washington.edu/cssap442*

- The *Proxy* Design Pattern
  - *A structural* design pattern
  - Intent
    - Provide a surrogate or placeholder for another object to control access to it

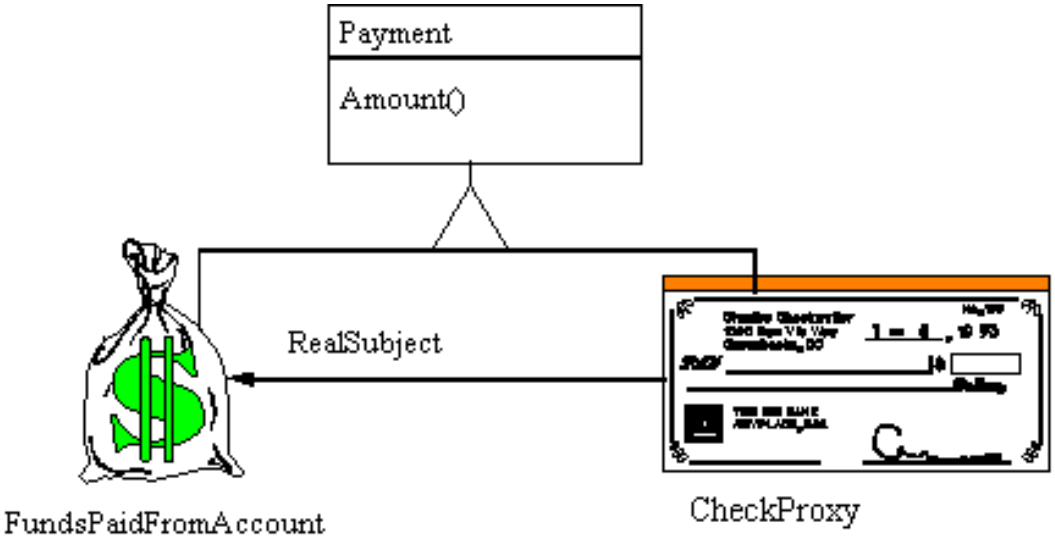
- The *Proxy* Design Pattern
  - Problem
    - Situation I: Wish to control access to an object because:
      - Housekeeping activities are required
      - Protection levels exist
      - Objects sit in different address spaces
      - Actual object too expensive to build immediately
    - Situation II: Wish to provide transparent management of services
      - The client "thinks" it is using the actual service when rather it is using a proxy for the actual service

- The *Proxy* Design Pattern
  - Non-software example (1)
    - Toll-free numbers



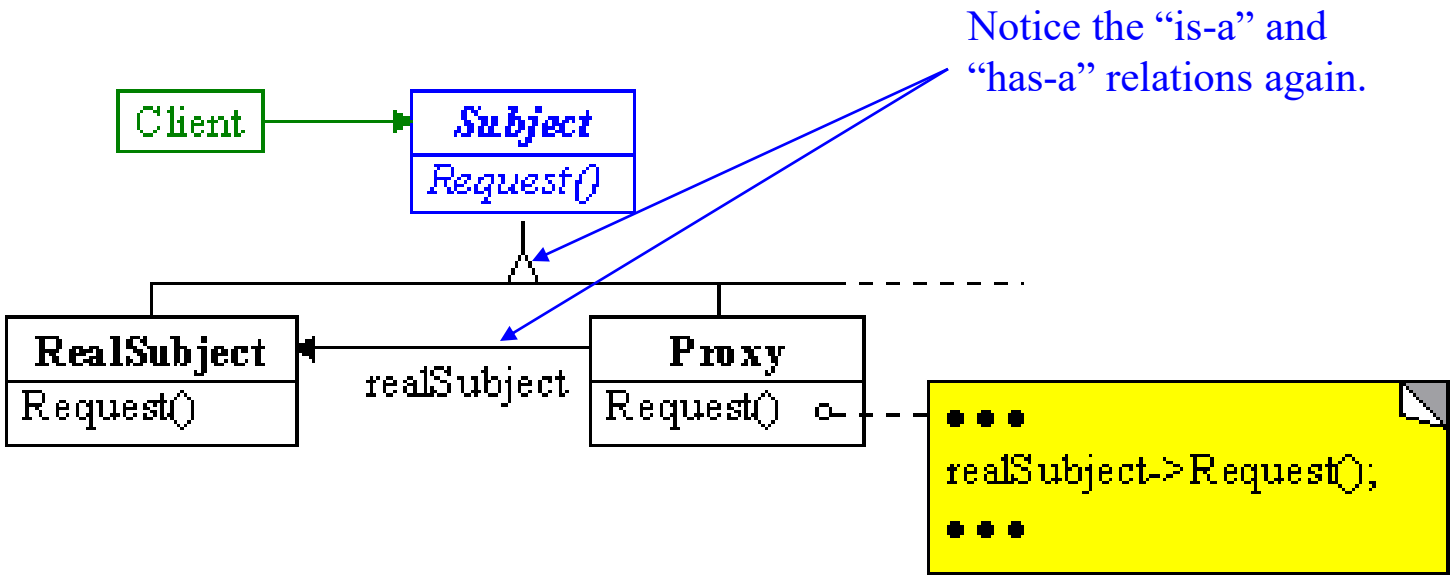
- Client dials the "800" area code just like they would for a real area code
- Proxy records billing information and connects to actual area code

- The *Proxy* Design Pattern
  - Non-software example (2)
    - Checks

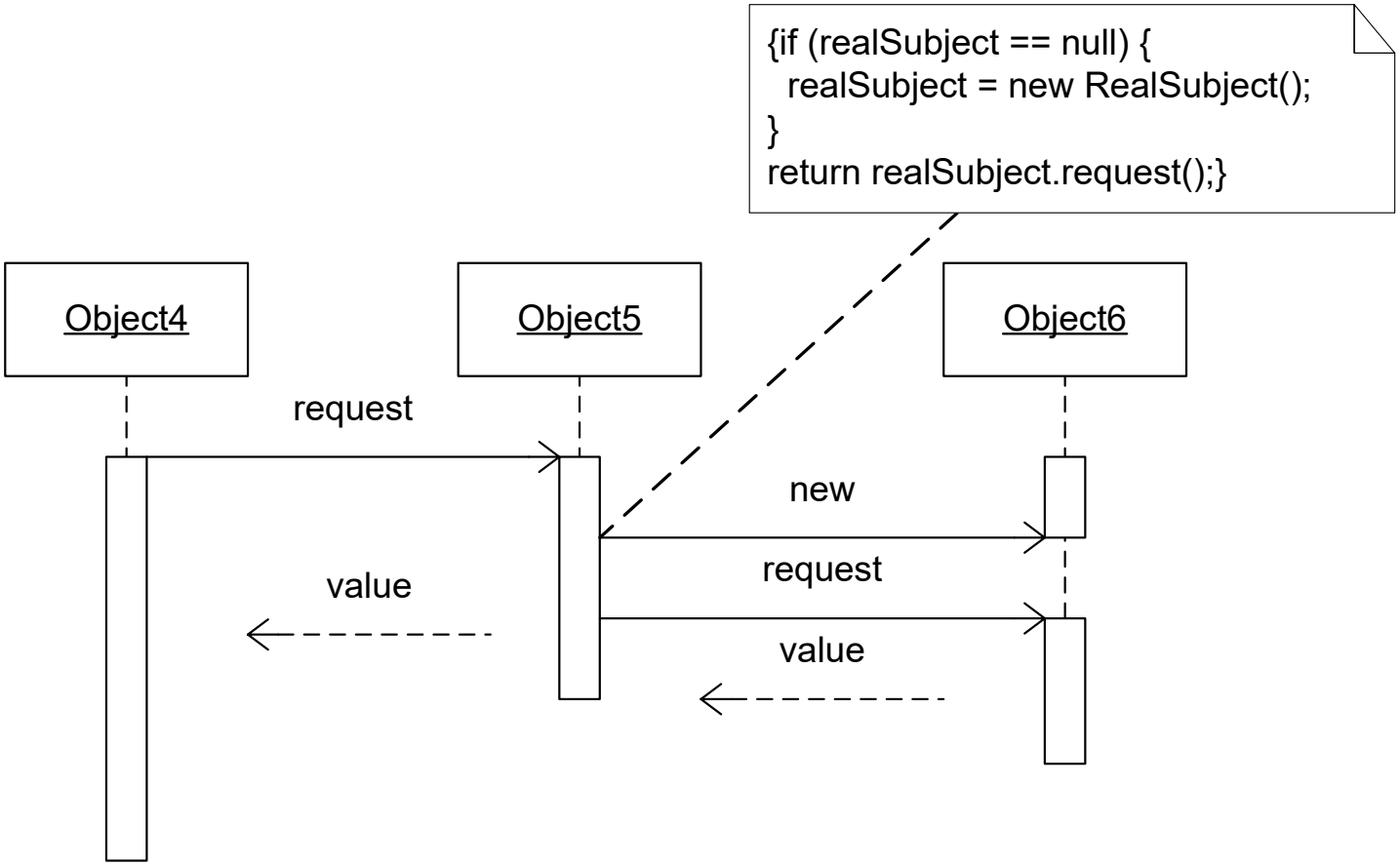


  - Check is a proxy for the actual funds in the bank

- The *Proxy* Design Pattern
  - Abstract Description
    - Object Proxy Pattern



- The *Proxy* Design Pattern
  - Abstract Description
    - Sequence Diagram (typical proxy application)



Proxy Sample Code

```
import java.util.*;
interface Image {
    public void displayImage();
}

class RealImage implements Image
{
    private String filename;
    public RealImage(String filename) {
        this.filename = filename;
        loadImageFromDisk();
    }
    private void loadImageFromDisk() {
        System.out.println("Loading "+filename);
        // Potentially expensive operation // ...
    }
    public void displayImage() {
        System.out.println("Displaying "+filename); }
}
```

```
class ProxyImage implements Image {
    private String filename;
    private Image image;

    public ProxyImage(String filename) {
        this.filename = filename;
    }
    public void displayImage() {
        if (image == null) {
        image = new
                RealImage(filename);
        // load only on demand
        }
        image.displayImage();
    }
}
```

Proxy Sample Code

```
class ProxyExample {
    public static void main(String[] args) {
        Image image1 = new  ProxyImage("HiRes_10MB_Photo1");
        Image image2 = new proxyImage("HiRes_10MB_Photo2");
        Image image3 = new ProxyImage("HiRes_10MB_Photo3");

        image1.displayImage(); // loading necessary
         image2.displayImage(); // loading necessary
        image1.displayImage(); // no loading necessary; already done
         // the third image will never be loaded - time saved!
    }
 }
```

Output :

```
Loading HiRes_10MB_Photo1
Displaying HiRes_10MB_Photo1
Loading HiRes_10MB_Photo2
Displaying HiRes_10MB_Photo2
Displaying HiRes_10MB_Photo1
```

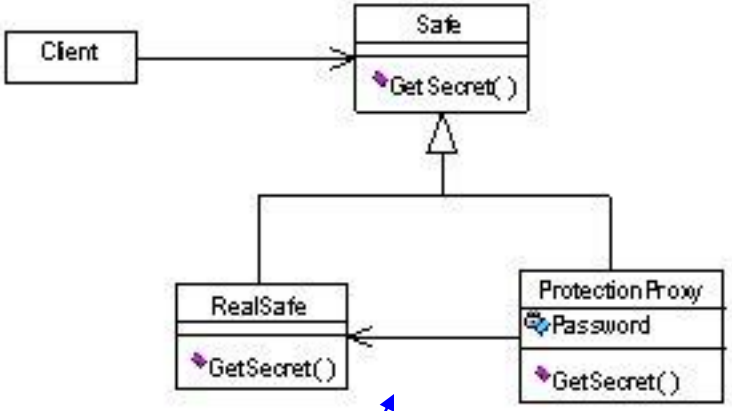
- The *Proxy* Design Pattern
  - Consequences
    - Isolates `RealSubject` from client
      - Forces controllable indirection
    - Costs indirection
      - True for all delegation models
    - May require duplicate information in `Proxy` and `RealSubject`
      - Must support same "properties"
      - These must be supported even before instantiation of `RealSubject`

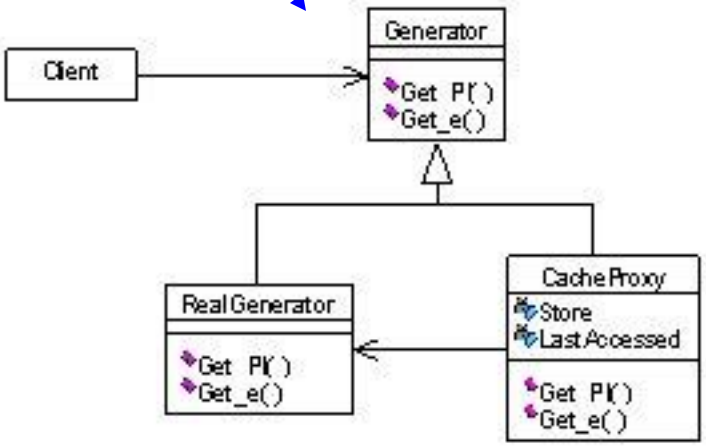
- The *Proxy* Design Pattern
  - Implementation Issues
    - Three basic types of proxies

      1. Remote Proxy
         - » Proxy used to "hide" the location of RealSubject
         - » Example: RMI implementation

      2. Virtual Proxy
         - » Performs optimizations
         - » Example: Image rendering in HTML

- The *Proxy* Design Pattern
  Implementation Issues
  - Three basic types of proxies
    3. Housekeeping Proxy
       - » Performs additional maintenance duties
       - » Example: Cache Proxy





       - » Example: Protection Proxy

- The *Proxy* Design Pattern Common Variations
  - Firewall Proxy (a *protection* proxy)
    - Protects targets from bad clients (or vice versa)
  - Synchronization Proxy (a *housekeeping* proxy)
    - Provides multiple accesses to a target object
  - Smart Reference Proxy (a *housekeeping* proxy)
    - Provides additional actions whenever a target object is referenced
  - Copy-on-Write Proxy (a *virtual* proxy)
    - Defers cloning an object until required