# HET PROXY PATTERN

Veerle Ongenae, Wijnand Schepens

## Het Proxy Pattern



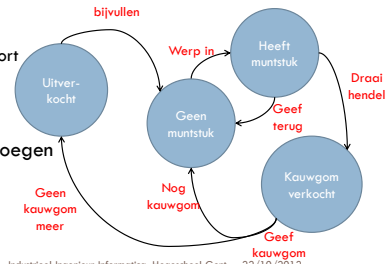
- Proxy's
  - Controleren en regelen
  - Toegang tot objecten
  - Optreden als vervanger
  - Vertegenwoordigen objecten
  - Methode aanroepen over het internet sturen



## Kauwgomautomaat



- Monitoren?
  - Voorraad
  - Statusrapport
- getAantal
- getStatus
- Locatie toevoegen






## Monitorcode



```
public class KauwgomMachine {
    …
    String locatie;
    public KauwgomMachine(String locatie, int aantal) {
        …
        this.locatie = locatie;
    }
    public String getLocatie() {
        return locatie;
    }
    …
}
```

```
public class MonitorKauwgomMachine {
    KauwgomMachine kauwgomMachine;
    public MonitorKauwgomMachine(KauwgomMachine machine) {
        this.kauwgomMachine = machine;
    }
    public void report(){
        System.out.println("…" + kauwgomMachine.getLocatie());
        … kauwgomMachine.getAantal() …
        … kauwgomMachine.getStatus() …
    }
}
```



## Remote monitoren



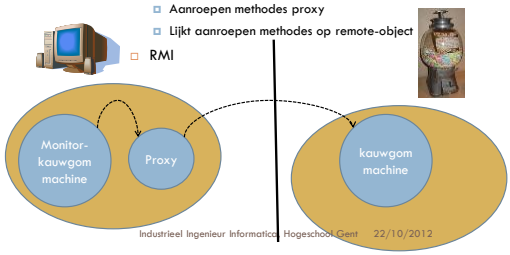
- Nu: kauwgommachine en monitor in zelfde JVM
- Gewenst
  - Monitoren vanop afstand
- Oplossing
  - Remote Proxy
    - Vervangt reëel object
    - Gedrag zoals KauwgomMachine-object
    - Communiceert over het netwerk met het echte KauwgomMachine-object
    - Monitor heeft referentie naar proxy
    - KauwgomMachine
      - Service: aanvragen via netwerk accepteren



## Remote Proxy



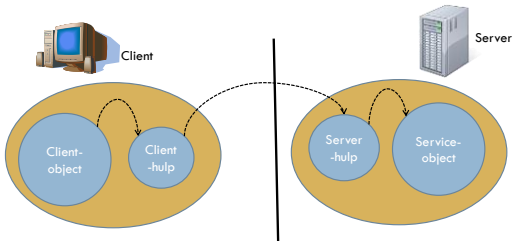
- Lokale vertegenwoordiger van een remote object
- Remote: andere JVM
- Client
  - Aanroepen methodes proxy
  - Lijkt aanroepen methodes op remote-object
- RMI



## RMI

7

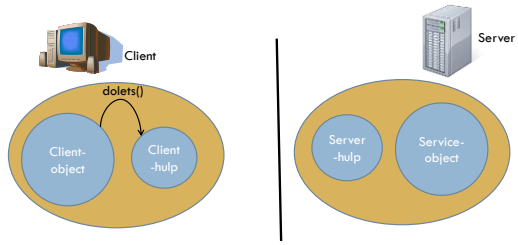
- Remote Method Invocation






## Methode-oproep op remote object

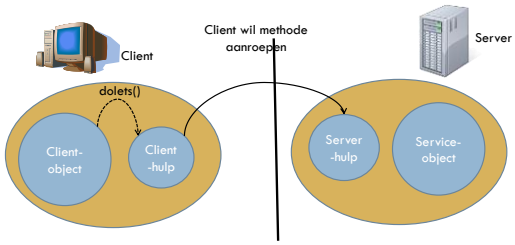
8






## Methode-oproep op remote object

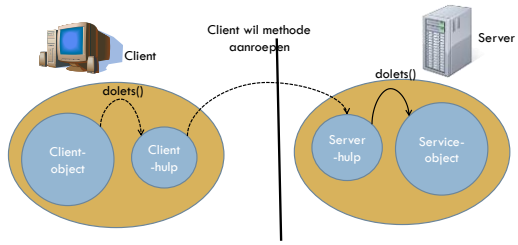
9






## Methode-oproep op remote object

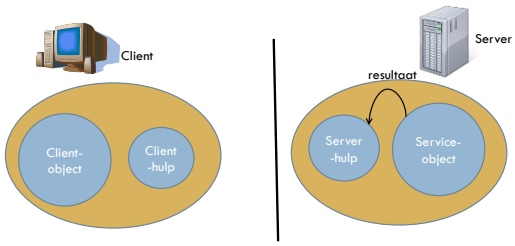
10






## Methode-oproep op remote object

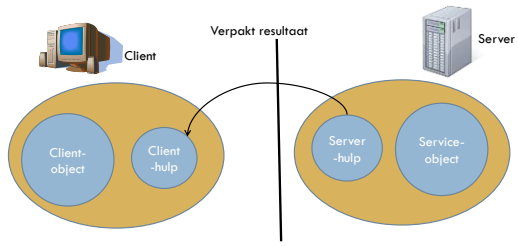
11





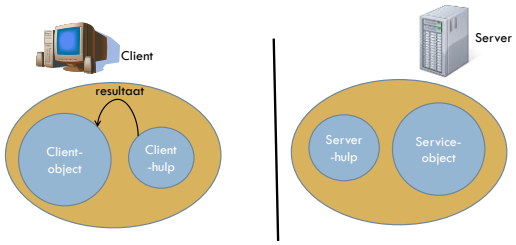

## Methode-oproep op remote object

12



## Methode-oproep op remote object

13

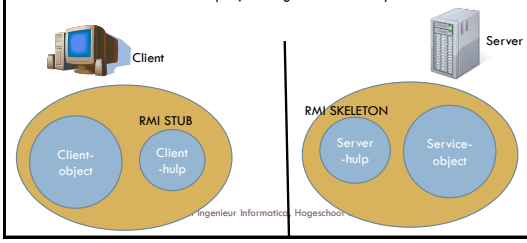





## RMI

14

- Bouwt clienthulp- en serverhulpobjecten
- Geen netwerk- of IO-code schrijven
- Infrastructuur om remote-objecten te zoeken
- Zelfde methode-aanroepen, maar gooien wel excepties

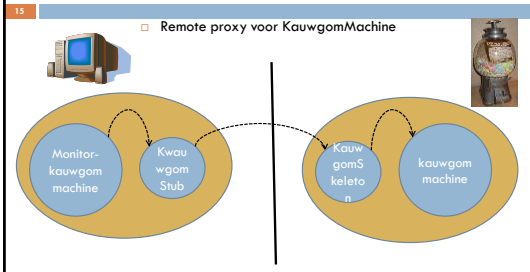





## Terug naar voorbeeld

15

- Remote proxy voor KauwgomMachine

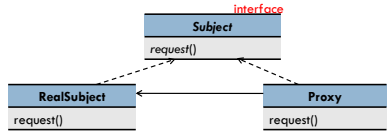





## Proxy Pattern Definitie

16

- Het Proxy Pattern zorgt voor een surrogaat of plaatsvervanger voor een ander object om de toegang hiertoe te controleren.






## Verschillende soorten proxy's

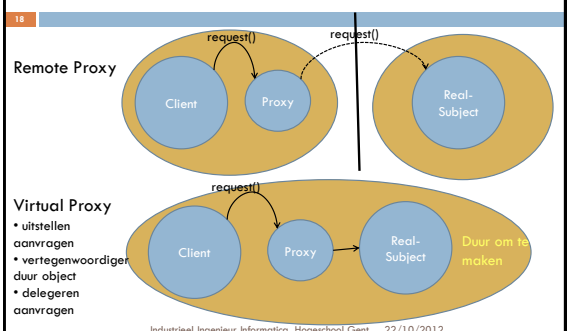
17

- Remote proxy
  - Toegang tot remote object
- Virtuele proxy
  - Toegang tot bron, duur om maken
- Protectie proxy
  - Toegang tot bron op basis van toegangsrechten



## Virtuele Proxy

18



## Voorbeeld CD-cover

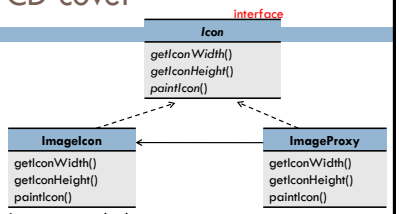
19

- GUI
  - Menu met CD-titels
  - Selectie CD → cover ophalen via netwerk
    - Vraagt wat tijd
    - Boodschap tonen
    - Applicatie mag niet hangen
- Virtuele proxy
  - Plaats afbeelding innemen
  - Laden op achtergrond
  - Eénmaal geladen delegeren tonen naar afbeelding



## Ontwerp CD-cover

20




- ImageProxy
  - Maakt ImageIcon en start laden
  - Tijdens het laden boodschap tonen
  - Afbeelding geladen: alle methodes delegeren naar ImageIcon
- Nieuwe afbeelding → nieuwe proxy



## ImageProxy - code

21

```
class ImageProxy implements Icon {
    ImageIcon imageIcon; URL imageURL;
    Thread haalThread; boolean ophalen = false;
    public ImageProxy (URL url) { imageURL = url; }
    public int getIconWidth() {
        if (imageIcon != null) { return imageIcon.getIconWidth();
        } else { return 800; }
    }
    public int getIconHeight() {
        if (imageIcon != null) { return imageIcon.getIconHeight();
        } else { return 600; }
    }
    public void paintIcon (…) {…}
}
```



## ImageProxy – vervolg code

22

```
class ImageProxy implements Icon {
    ImageIcon imageIcon; URL imageURL; Thread haalThread; boolean ophalen = false;
    public ImageProxy (URL url) {…}  public int getIconWidth() {…}  public int getIconHeight() {…}
    public void paintIcon (final Component c, Graphics g, int x, int y) {
        if (imageIcon != null) { imageIcon.paintIcon(c, g, x, y);
        } else { g.drawString(" cd-cover wordt geladen, even geduld …", g, x+300, y+190);
            if (!ophalen) {
                ophalen = true;
                haalThread = new Thread (new Runnable() {
                    public void run () {
                        try {
                            imageIcon = new ImageIcon(imageURL, "cd-cover");
                            c.repaint();
                        } catch (Exception e) {…}  } });
                haalThread.start();
    } } } }
```



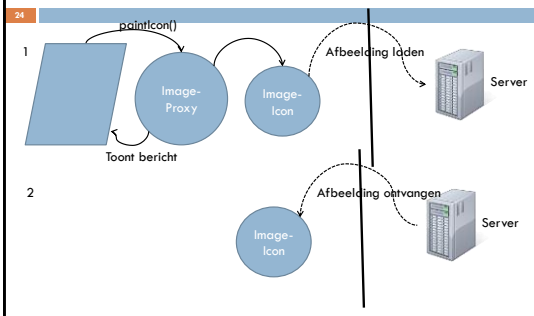
## ImageProxy verbeteren??

23

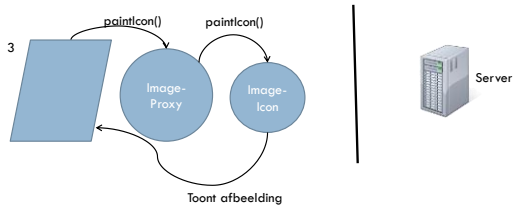


## Visuele samenvatting

24



## Visuele samenvatting






## Opmerkingen



- Proxy ↔ Decorator
  - Gelijkend
    - Object inpakken; doorgeven
  - Verschillende doelstellingen
    - Extra gedrag toevoegen ↔ Toegang regelen
    - Object meekrijgen ↔ object aanmaken
- Afdwingen gebruik Proxy i.p.v. RealSubject
  - Via factory
- Eerder opgehaalde afbeeldingen bewaren?
  - Caching proxy



## Verschillende soorten proxy's



- Remote proxy
  - Toegang tot remote object
- Virtuele proxy
  - Toegang tot bron, duur om maken
- Protectie proxy
  - Toegang tot bron op basis van toegangsrechten



## Protectie proxy met Java API



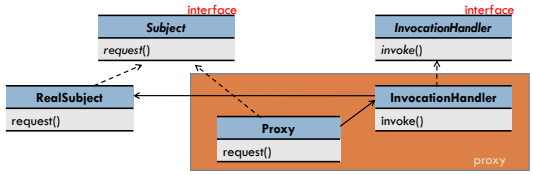
- Java
  - Ondersteuning proxy's
  - java.lang.reflect
- Aanmaken proxyklasse
  - Implementeert een aantal interfaces
  - Stuurt methodeoproepen door naar opgegeven klasse
  - Wordt aangemaakt at runtime (dynamic proxy)



## Proxy in Java API



- Proxy gegenereerd door Java
- InvocationHandler
  - Regelt toegang
  - Reageren op methodeaanroepen naar proxy
    - Doet werk voor proxy






## Voorbeeld: datingservice



```
public interface PersonBean {
    String getName();
    String getGender();
    String getInterests();
    int getHotOrNotRating();

    void setName(String name);
    void setGender(String gender);
    void setInterests(String interests);
    void setHotOrNotRating (int rating);
}
```

```
public interface PersonBeanImpl  implements PersonBean {
    String name, gender, interests; int rating, ratingCount = 0;
    public String getName() {…}
    public String getGender(){…}
    public String getInterests(){…}
    public int getHotOrNotRating() {
        if (ratingCount == 0) return 0;
        else return (rating/ratingCount);
    }
    public void setName(String name) {…}
    public void setGender(String gender) {…}
    public void setInterests(String interests) {…}
    public void setHotOrNotRating (int rating) {
        this.rating += rating; ratingCount++;
    }
}
```

## Probleem?

31

- Op jezelf stemmen
- Gegevens van iemand anders veranderen
- Iedereen kan elke methode oproepen
- Nood aan Protection Proxy
  - Toegang tot object regelen
  - Op basis van toegangsrechten
- Voorbeeld
  - Manager kan salaris van werknemer aanpassen
  - Personeelsdienst kan alle informatie van de werknemer opvragen
  - Werknemer kan persoonlijke informatie toevoegen



## Toegangsrechten in datingservice

32

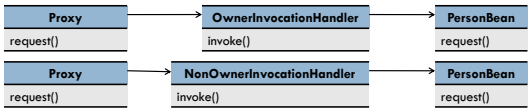
- Klant
  - Info over zichzelf instellen en wijzigen
    - Kan geen informatie van anderen wijzigen
  - Kan andere klanten beoordelen
    - Mag zichzelf niet beoordelen
  - Kan alle informatie van zichzelf en anderen opvragen
- Oplossing
  - Twee proxy's
    - Toegang tot je eigen PersonBean-object
    - Toegang tot ander PersonBean-objecten



## Overzicht oplossing

33

1. Maak twee InvocationHandlers
   - Implementeren gedrag proxy
   - Toegang tot eigen object
   - Toegang tot andere objecten
2. Code om dynamic proxy's te genereren
3. Omwikkel elke PersonBean met de juiste proxy

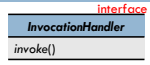





## Stap 1: InvocationHandlers maken

34

- Methodeaanroep proxy
  - Doorsturen naar InvocationHandler
  - Via oproepen methode invoke()
    - Stuurt aanvraag door naar PersonBean
- proxy.setHotOrNotRating(9);
  - invoke(Object proxy, Method method, Object[] args)
    - return method.invoke(person,args);






## OwnerInvocationHandler

35

```
import java.lang.reflect.*;
public class OwnerInvocationHandler implements InvocationHandler {
    PersonBean person;
    public OwnerInvocationHandler(PersonBean person) {this.person = person;}
    public Object invoke (Object proxy, Method method, Object[] args)  throws IllegalAccessException {
        try {
            if (method.getName().startsWith("get")) {
                return method.invoke(person,args);
            } else if (method.getName().equals("setHotOrNotRating")) {
                throw new IllegalAccessException();
            } else if (method.getName().startsWith("set")) {
                return method.invoke(person,args);
            }
        } catch (InvocationTargetException e) {…}
        return null;
    }}
```



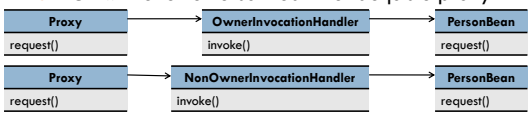
## NonOwnerInvocationHandler

36

## Overzicht oplossing

37

1. Maak twee InvocationHandlers
   - Implementeren gedrag proxy
   - Toegang tot eigen object
   - Toegang tot andere objecten
2. Code om dynamic proxy's te genereren
3. Omwikkel elke PersonBean met de juiste proxy






## Stap 2: proxyklasse maken en proxyobject instantiëren

38

```
PersonBean getOwnerProxy(PersonBean person) {
    return (PersonBean) Proxy.newProxyInstance(
        person.getClass().getClassLoader();
        person.getClass().getInterfaces(),
        new OwnerInvocationHandler(person));
}
```

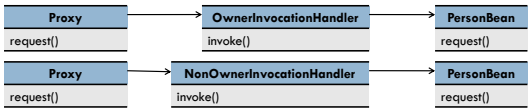
- getNonOwnerProxy?



## Overzicht oplossing

39

1. Maak twee InvocationHandlers
   - Implementeren gedrag proxy
   - Toegang tot eigen object
   - Toegang tot andere objecten
2. Code om dynamic proxy's te genereren
3. Omwikkel elke PersonBean met de juiste proxy






## Stap 3: Omwikkel PersonBean met de juiste Proxy

40

```
public class MatchMakingTestDrive {
  public static void main(String[] args) {
    MatchMakingTestDrive test = new MatchMakingTestDrive();
    test.drive();
  }
  public MatchMakingTestDrive() {initializeDatabase();}
  public drive() {
    PersonBean joe = getPersonFromDatabase("Joe Bean");
    PersonBean ownerProxy = getOwnerProxy(joe);
    … // wat kan, wat kan niet?
    PersonBean nonOwnerProxy = getNonOwnerProxy(joe);
    … // wat kan, wat kan niet?
  }
  …
}
```



## Opmerkingen

41

- Testen of een klassen een proxyklasse is?
  - Methode isProxyClass()
- public static Object newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h) throws IllegalArgumentException
  - Beperking interfaces?
    - Interfaces, geen klassen
    - Niet-publieke interfaces behoren tot zelfde package
    - Geen conflicterende methodenamen (met zelfde signatuur)
    - …



## Oefening

42

- p. 473

## Andere proxy's

43

- Firewall Proxy
  - Regelt toegang tot netwerkresources
  - Beschermt tegen slechte clients
- Smart Reference Proxy
  - Extra acties bij een referentie
    - Tellen aantal referenties
- Cache Proxy
  - Tijdelijke opslag "dure" operaties
  - Delen resultaten tussen meerdere clients



## Andere proxy's

44

- Synchronization Proxy
  - Veilige toegang vanuit meerdere threads
- Complexity Hiding Proxy
  - Verbergt de complexiteit
  - Regelt toegang tot complexe verzameling klassen
  - Facade Proxy
- Copy-on-Write Proxy
  - Variant Virtual Proxy
  - Regelt kopiëren object
    - Uitstellen totdat het expliciet opgevraagd wordt



## Samenvatting kennis

45

- OO-principes
  - Isoleer wat verandert
  - Verkies compositie boven overerving
  - Programmeer naar een interface en niet naar een implementatie
  - Streef naar ontwerpen met een zwakke koppeling tussen interagerende objecten
  - Klassen moeten open zijn voor uitbreiding maar gesloten voor verandering
  - Wees afhankelijk van abstracties, niet van concrete klassen
  - Hoe minder je moet weten, hoe beter
  - Bel ons niet, wij bellen jou
  - Een klasse dient maar één reden te hebben om te veranderen
- OO-patterns
  - Strategy, Observer, Decorator, Abstract Factory, Factory Method, Command Pattern, Adapter, Facade, Template Method Pattern, Iterator Pattern, Composite Pattern, State Pattern
  - Het Proxy Pattern zorgt voor een surrogaat of een plaatsvervanger voor een ander object om de toegang tot dit object te regelen.