

# Programmierung und Programmiersprachen

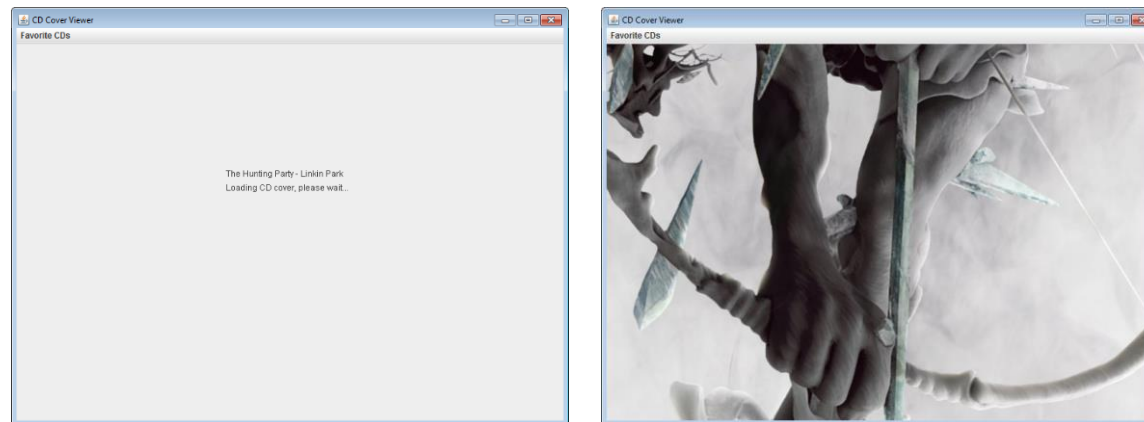
## Sommersemester 2023

### Proxy-Muster

## Proxy-Muster

### Remote Zugriff

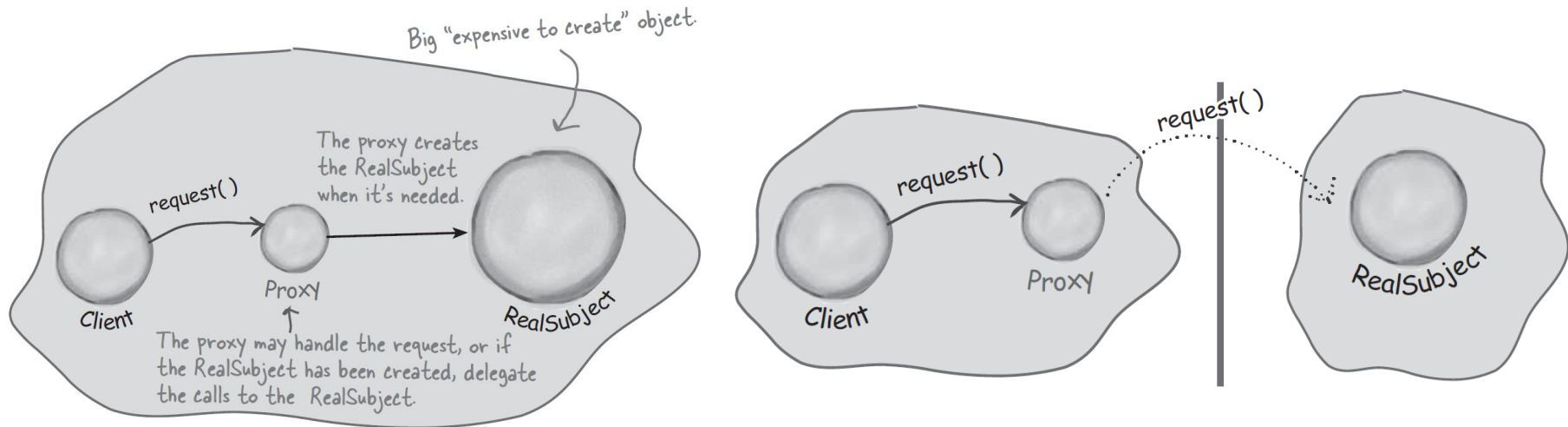
Es soll eine Anwendung zum Laden von CD-Covern von einer Webseite implementiert werden. Es soll eine Menü mit CD-Titel erstellt und das Abrufen von Bildern von einem Onlinedienst umgesetzt werden. Sie sollen jedoch dabei beachten, dass eventuell das Bild nicht mehr verfügbar ist oder das Laden über das Netz etwas länger dauern könnte. Während auf das Bild gewartet wird, sollte die Anwendung deshalb etwas anderes anzeigen.



## Proxy-Muster

### Proxy-Objekte

Unter einem Proxy-Objekt wird ein Objekt verstanden, welches als eine Art von Stellvertreter agiert. Das Proxy-Objekt kann im Laufe der Zeit auch durch das richtige Objekt ausgetauscht werden, wenn es verfügbar ist. Dann wird von einem Virtuellen Proxy gesprochen. Es kann jedoch auch während der gesamten Laufzeit als Stellvertreter für ein Remote-Objekt agieren (Remote-Proxy).

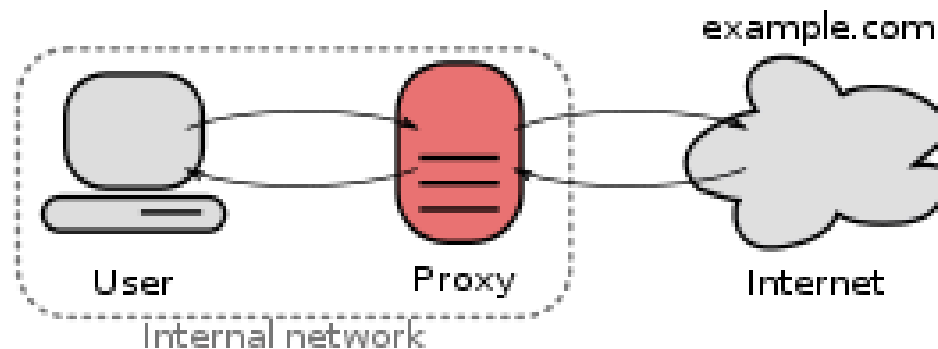


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

## Proxy-Muster

# Proxy Server

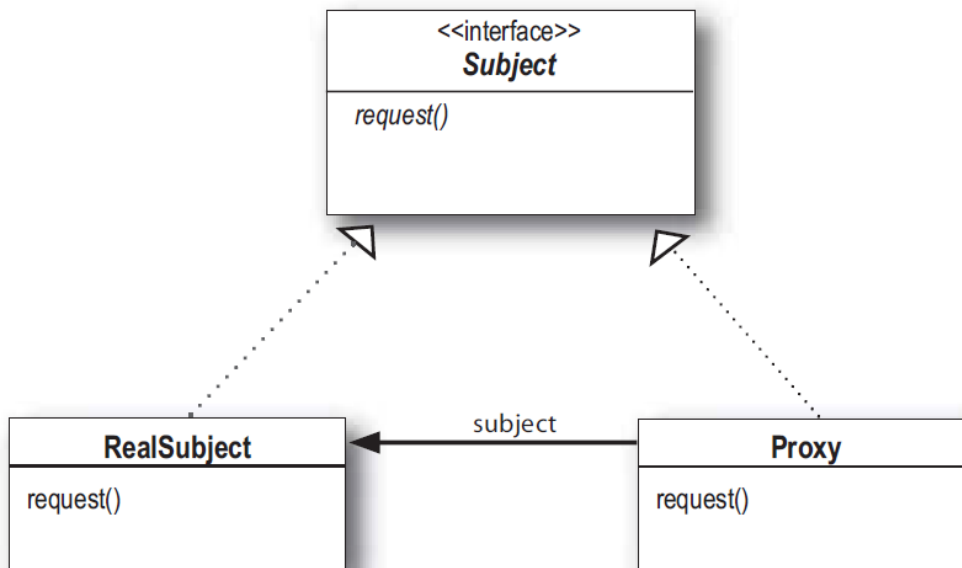
Ein Proxy (Server) ist eine Kommunikationsschnittstelle in einem Netzwerk. Er arbeitet als Vermittler, der auf der einen Seite Anfragen entgegennimmt, um dann über seine eigene Adresse eine Verbindung zur anderen Seite herzustellen. Wird der Proxy als Netzwerkkomponente eingesetzt, bleibt einerseits die tatsächliche Adresse eines Kommunikationspartners dem jeweils anderen Kommunikationspartner verborgen, was eine gewisse Anonymität schafft. Häufig werden Proxy Server auch im Intranet verwendet, um eine Kommunikation mit dem Internet zu überwachen.



## Proxy-Muster

# Proxy Muster

Das Proxy Muster kontrolliert den Zugriff auf ein Objekt mit Hilfe eines vorgelagerten Stellvertreterobjektes. Hierzu wird eine Schnittstelle definiert, die sowohl vom Objekt als auch vom Stellvertreter implementiert wird.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

## Proxy-Muster

# Icons in Java

Das Paket `javax.swing` stellt ein Interface für Icons bzw. Bilder zur Verfügung. Die Methode `paintIcon` zeichnet das Icon an einer bestimmten Stelle unter Verwendung des gegebenen Grafikkontextes.

```
public interface Icon
```

A small fixed size picture, typically used to decorate components.

**See Also:**

`ImageIcon`

### Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method and Description	
int	<code>getIconHeight()</code>	Returns the icon's height.
int	<code>getIconWidth()</code>	Returns the icon's width.
void	<code>paintIcon(Component c, Graphics g, int x, int y)</code>	Draw the icon at the specified location.

## Proxy-Muster

# Icons in Java

Eine Implementierung der Schnittstelle ist die Klasse ImageIcon. Es werden Konstruktoren angeboten, die ein Icon auf Basis einer URL erzeugen können.

### Constructors

#### Constructor and Description

**ImageIcon()**

Creates an uninitialized image icon.

**ImageIcon(URL location)**

Creates an ImageIcon from the specified URL.

**ImageIcon(URL location, String description)**

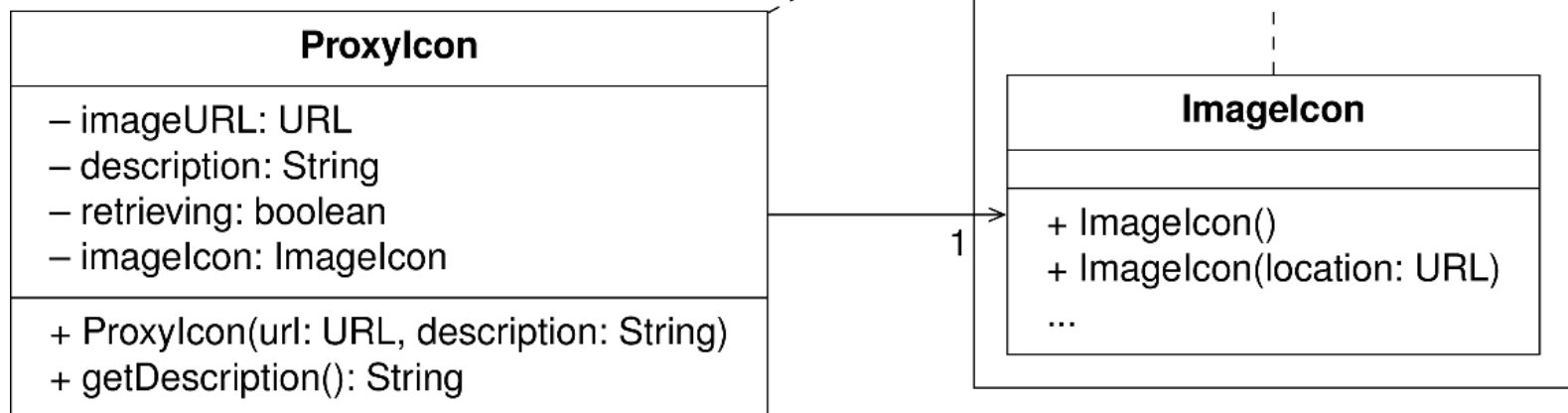
Creates an ImageIcon from the specified URL.

## Proxy-Muster

### Proxy für Icons

Der Proxy erzeugt ein ImageIcon und beginnt, das Bild von einer Netzwerk-URL zu laden.

Während des Ladens zeigt das ProxyIcon einen Text an. Wenn das Bild vollständig geladen ist, delegiert der Proxy alle Methodenaufrufe an das ImageIcon.






## Proxy-Muster

# Proxy Icon - Implementierung

```
public class ImageProxy implements Icon {  
    private URL imageURL;  
    private String description;  
    private ImageIcon imageIcon;  
    private boolean retrieving = false;  
  
    public ImageProxy(URL url, String description) {  
        this.imageURL = url;  
        this.description = description;  
    }  
    public int getIconWidth() {  
        if (imageIcon != null) {  
            return imageIcon.getIconWidth();  
        }  
        return 400;  
    }  
    public int getIconHeight() {  
        if (imageIcon != null) {  
            return imageIcon.getIconHeight();  
        }  
        return 400;  
    }  
    public String getDescription() {  
        return description;  
    }  
}
```

Es muss ein Attribut definiert werden, um zu prüfen, ob das Bild noch geladen wird.



## Proxy-Muster

# Proxy Icon - Implementierung

...

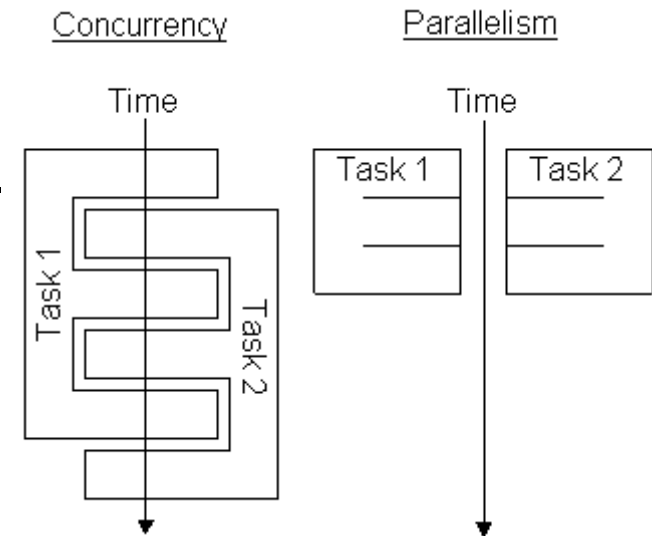
```
public void paintIcon(final Component c, Graphics g, int x, int y) {  
  
    if (this.imageIcon != null) {  
        this.imageIcon.paintIcon(c, g, x, y);  
    } else {  
  
        // Alternativen Text anzeigen  
        g.drawString(this.description, x, y);  
        g.drawString("Loading CD cover, please wait...", x, y + 20);  
  
        if (!this.retrieving) {  
            this.retrieving = true;  
            this.imageIcon = new ImageIcon(this.imageURL, this.description);  
  
            // Neuzeichnen der Komponente, in welche das Icon eingefügt wird  
            c.repaint();  
            this.retrieving = false;  
        }  
    }  
}
```

## Proxy-Muster

# Java Swing & Threads

Bei der Implementierung der Methode `paintIcon` tritt das Problem auf, dass die Oberfläche so lange blockiert wird, bis die Methode beendet ist. Dies ist erst der Fall, wenn das Icon vollständig geladen ist. Der alternative Text des Proxy würde gar nicht angezeigt werden.

Ein Thread ist wörtlich ein einzelner Ausführungsfaden eines Programms. Die Aktualisierung einer Swing-Oberfläche erfolgt in einem eigenen Thread. Damit die Aktualisierung und das Laden des Icons parallel verlaufen können, muss für das Laden ein eigener Thread erzeugt werden. Diese Threads laufen dann parallel und die Oberfläche wird nicht mehr blockiert.



## Proxy-Muster

# Threads in Java

In Java gibt es verschiedene Möglichkeiten mit Threads zu arbeiten. Hierzu kann beispielsweise die Klasse Thread verwendet werden.

Die Klasse Thread erzeugt für den aktuellen Prozess einen neuen Thread. Wesentlich sind dabei die Methoden `run()` und `start()`.

In der Methode `run()` werden die Anweisungen programmiert, die in diesem Thread ausgeführt werden sollen.

Die Methode `start()` startet dann den Thread, der anschließend parallel ausgeführt wird.

## Proxy-Muster

### Thread – Implementierung I

Eine mögliche Implementierung ist die Erstellung einer privaten Klasse innerhalb der Klasse ProxyIcon.

```
...
private class LoadIconThread extends Thread {

    private ImageProxy proxy;
    private Component c;

    LoadIconThread(ImageProxy proxy, Component c) {
        this.proxy = proxy;
        this.c = c;
    }

    public void run() {
        this.proxy.imageIcon = new ImageIcon(this.proxy.imageURL, this.proxy.description);
        this.c.repaint();
        this.proxy.retrieving = false;
    }
}
```

## Proxy-Muster

### Thread – Implementierung I

Zum Laden des Icons wird jetzt jeweils ein neuer Thread erstellt und gestartet. Die Methode `paintIcon` wird im anderen Thread weitergeführt.

...

```
public void paintIcon(final Component c, Graphics g, int x, int y) {  
    if (this.imageIcon != null) {  
        this.imageIcon.paintIcon(c, g, x, y);  
    } else {  
        g.drawString(this.description, x, y);  
        g.drawString("Loading CD cover, please wait...", x, y + 20);  
  
        if (!this.retrieving) {  
            this.retrieving = true;  
            Thread retrievalThread = new LoadIconThread(this, c);  
            retrievalThread.start();  
        }  
    }  
}
```

## Proxy-Muster

# Threads in Java mit Lambda Ausdrücken

Der Klasse Thread kann im Konstruktor auch ein sogenanntes Runnable-Objekt (Interface Runnable) übergeben werden. Dieses Objekt implementiert nur die Methode run() und wird dann implizit vom Thread aufgerufen bzw. der Aufruf wird delegiert. Die Implementierung von Interfaces mit einer Methode (FunctionalInterface) können effizient mit Lambda Ausdrücken umgesetzt werden. In diesem Fall kann sogar auf das Interface Runnable verzichtet werden, da die Klasse Thread einen entsprechenden Konstruktor bereitstellt.

### Constructors

#### Constructor and Description

**Thread()**

Allocates a new Thread object.

**Thread(Runnable target)**

Allocates a new Thread object.

```
@FunctionalInterface  
public interface Runnable
```

#### All Methods

#### Instance Methods

#### Abstract Methods

#### Modifier and Type Method and Description

void

**run()**

When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

## Proxy-Muster

# Thread – Implementierung II

Verwendung eines Lambda Ausdrucks zur Implementierung eines Threads

...

```
public void paintIcon(final Component c, Graphics g, int x, int y) {  
    if (this.imageIcon != null) {  
        this.imageIcon.paintIcon(c, g, x, y);  
    } else {  
        g.drawString(this.description, x, y);  
        g.drawString("Loading CD cover, please wait...", x, y + 20);  
  
        if (!this.retrieving) {  
            this.retrieving = true;  
  
            new Thread( () -> {  
                ImageIcon = new ImageIcon(imageURL, description);  
                c.repaint();  
                retrieving = false;  
  
            }).start();  
        }  
    }  
}
```



## Proxy-Muster

# GUI Implementierung

Es wird eine Komponente implementiert, die ein Icon aufnehmen und an einer bestimmten Position darstellen soll.

```
public class ImageComponent extends JComponent {  
  
    private Icon icon;  
    public ImageComponent(Icon icon) {  
        setIcon(icon);  
    }  
    public void setIcon(Icon icon) {  
        this.icon = icon;  
    }  
  
    public void paintComponent(Graphics g) {  
  
        super.paintComponent(g);  
        int w = icon.getIconWidth();  
        int h = icon.getIconHeight();  
        int x = (800 - w)/2;  
        int y = (600 - h)/2;  
        icon.paintIcon(this, g, x, y);  
    }  
}
```

## Proxy-Muster

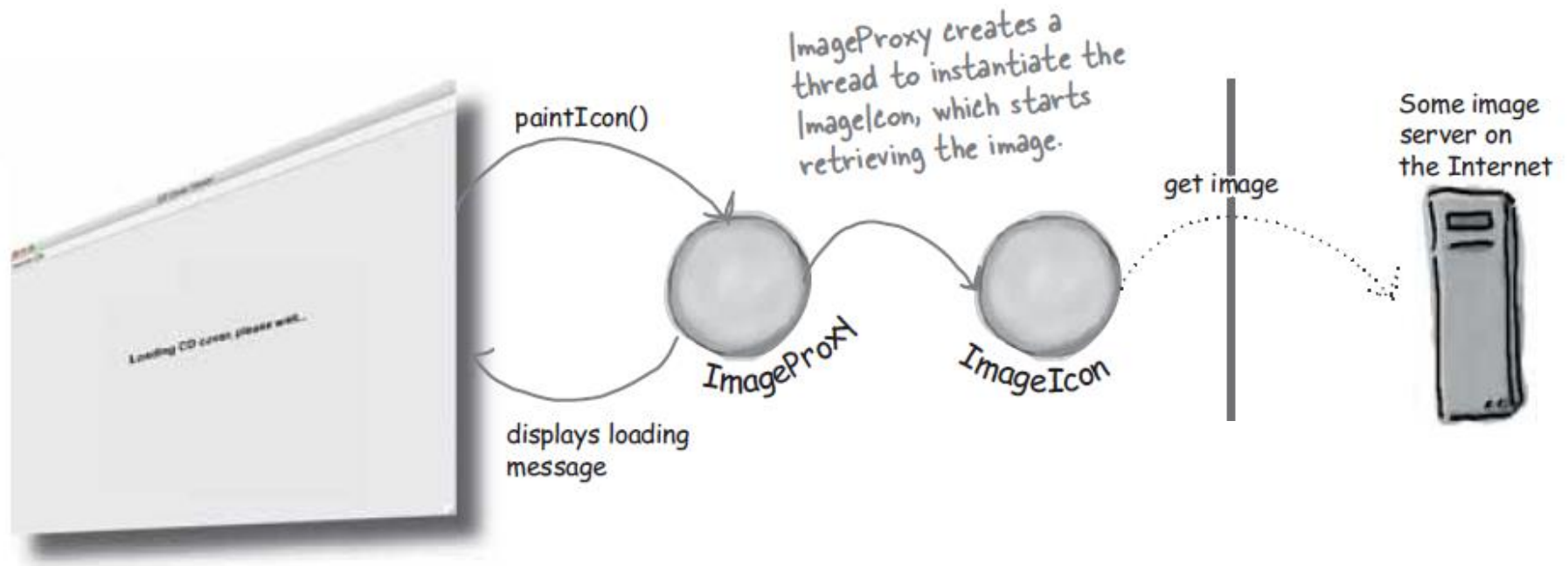
# GUI Implementierung

Die Klasse zum Testen der Anwendung erstellt ein Anwendungsfenster mit einer Komponente zum Anzeigen eines Icons. Es wird ein Proxy Icon erzeugt und gesetzt.

```
public class ImageProxyTest {  
  
    public static void main(String[] args) throws Exception {  
  
        Icon icon = new ImageProxy(  
            new URL(  
                "http://ecx.images-amazon.com/images/I/81vzdIaKNOL._SL1500_.jpg"),  
                "The Hunting Party - Linkin Park");  
        ImageComponent c = new ImageComponent(icon);  
  
        JFrame frame = new JFrame("CD Cover Viewer");  
        frame.getContentPane().add(c);  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setSize(800, 600);  
        frame.setVisible(true);  
    }  
}
```

## Proxy-Muster

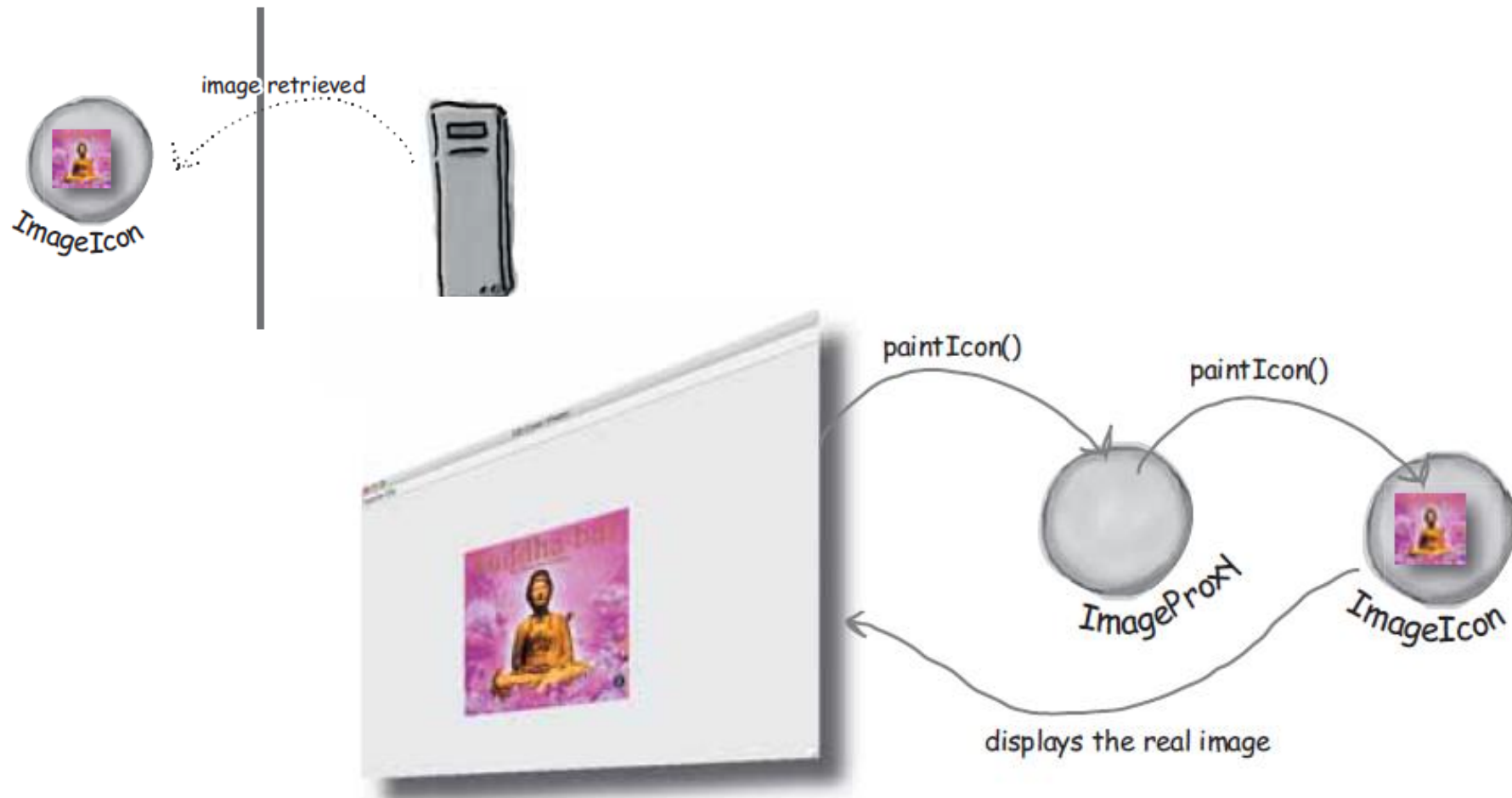
# Aufbau der Anwendung



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

## Proxy-Muster

### Aufbau der Anwendung



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

## Proxy-Muster

# Proxy Muster

Das Proxy Muster kontrolliert den Zugriff des Clients auf ein anderes Objekt durch einen vorgelagerten Stellvertreter. Es gibt verschiedene Möglichkeiten, wie dieser Zugriff gesteuert wird.

Remote Proxy	Steuert die Wechselwirkungen zwischen einem Client und einem entfernten Objekt
Virtueller Proxy	Kontrolliert den Zugriff auf ein Objekt, dessen Instanziierung aufwendig ist
Schutz-Proxy	Kontrolliert den Zugriff auf die Methoden eines Objektes abhängig davon, von wem der Aufruf kommt

## Proxy-Muster

# Proxy Muster

Das Proxy Muster kontrolliert den Zugriff des Clients auf ein anderes Objekt durch einen vorgelagerten Stellvertreter. Es gibt verschiedene Möglichkeiten, wie dieser Zugriff gesteuert wird.

**Cache Proxy**      Ermöglicht die vorübergehende Speicherung der Ergebnisse von aufwendigen Operationen in einem Puffer. Er kann auch mehreren Clients die gemeinsame Nutzung der Ergebnisse erlauben, um die Latenzzeiten zu verringern

**Synchronisierungs-Proxy**

Ermöglicht einen sicheren Zugriff auf ein Subjekt aus mehreren Threads heraus

# Programmierung und Programmiersprachen

## Sommersemester 2023

### Adapter-Muster

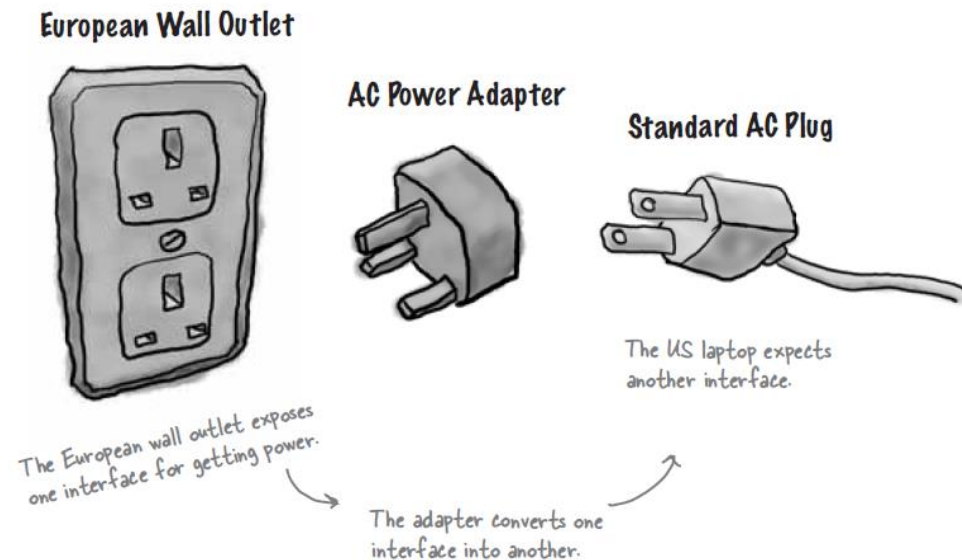
## Adapter-Muster

### Bestehende Systeme verbinden

Adapter sind in der realen Welt sehr weitverbreitet. Ein typisches Beispiel sind Stecker und Steckdosen.

Einige Stromadapter ändern nur die Gestalt der Steckdose und reichen den Strom einfach durch.

Andere Adapter sind jedoch komplexer und wandeln den Strom um, damit er den Anforderungen des Geräts entspricht.



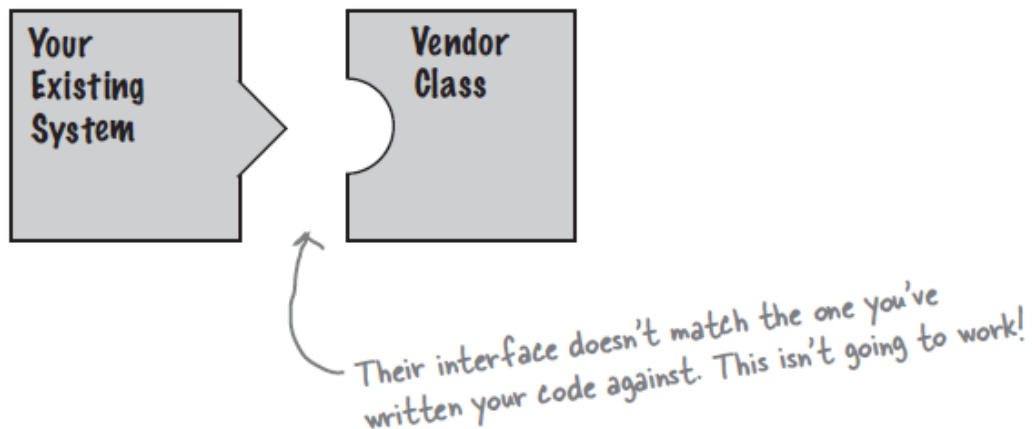
Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly



## Adapter-Muster

### Bestehende Systeme verbinden

Adapter müssen auch sehr häufig im Rahmen der Software-Entwicklung verwendet werden. Dies ist dann der Fall, wenn zwei bestehende Softwaresysteme (oder Klassen-Bibliotheken) zusammen eingesetzt werden sollen.

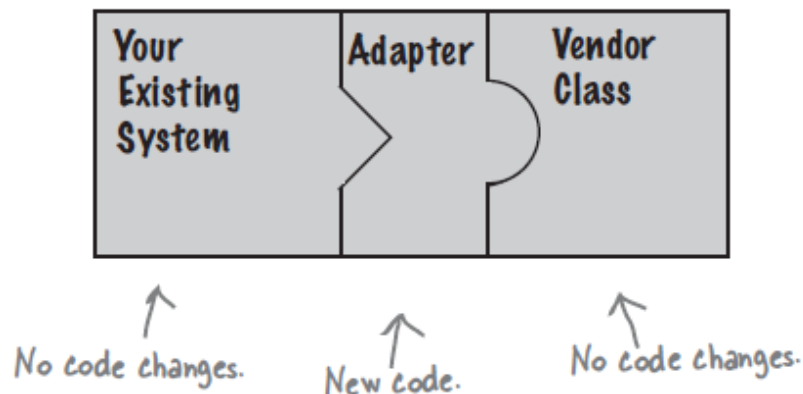


Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

## Adapter-Muster

### Bestehende Systeme verbinden

In der Regel ist es nicht möglich den bestehenden Code zu ändern. Es kann eine Klasse implementiert werden, die die Schnittstelle des einen Codes an die des anderen Codes anpasst. Die Adapter-Klasse fungiert somit als Vermittler.



Freeman et al. (2005) Entwurfsmuster von Kopf bis Fuß. O'Reilly

## Adapter-Muster

### Beispiel

Gegeben ist eine Software-Bibliothek zur interaktiven Erzeugung von geometrischen Objekten (z.B. Punkte, Linien, Kreise, etc.). Diese Bibliothek soll um Funktionen der algorithmischen Geometrie ergänzt werden (z.B. Abstand, Schnittpunkte, Durchschnitt, Vereinigung berechnen, etc.).

Um den Implementierungsaufwand zu reduzieren, soll eine andere Bibliothek verwendet werden. Es werden jedoch unterschiedliche Klassen verwendet (z.B. Rechteck und Rectangle).

Rechteck
<ul style="list-style-type: none"><li>– x: double</li><li>– y: double</li><li>– b: double</li><li>– h: double</li></ul>
<ul style="list-style-type: none"><li>+ Rechteck(x, y, b, h: double)</li><li>+ getX(): double</li><li>+ setX(x: double): void</li><li>+ getY(): double</li><li>+ setY(y: double): void</li><li>+ getB(): double</li><li>+ setB(b: double): void</li><li>+ getH(): double</li><li>+ setH(h: double): void</li></ul>

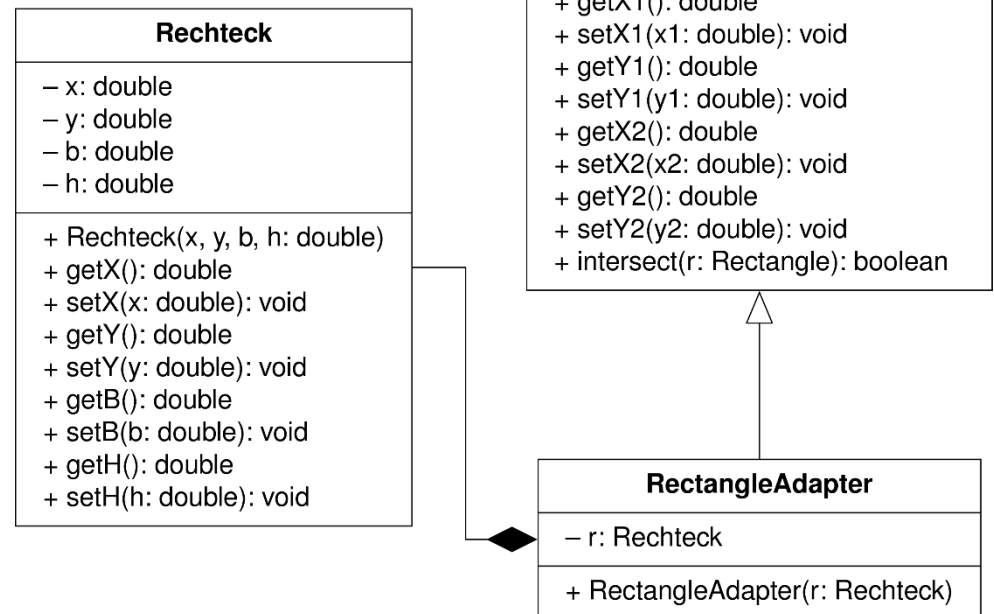
Rectangle
<ul style="list-style-type: none"><li>– x1: double</li><li>– y1: double</li><li>– x2: double</li><li>– y2: double</li></ul>
<ul style="list-style-type: none"><li>+ Rectangle(x1, y1, x2, y2: double)</li><li>+ getX1(): double</li><li>+ setX1(x1: double): void</li><li>+ getY1(): double</li><li>+ setY1(y1: double): void</li><li>+ getX2(): double</li><li>+ setX2(x2: double): void</li><li>+ getY2(): double</li><li>+ setY2(y2: double): void</li><li>+ intersect(r: Rectangle): boolean</li></ul>

## Adapter-Muster

### Objektadapter

Es wird ein Adapter implementiert, der ein Rechteck-Objekt in ein Rectangle-Objekt umwandelt.

Des Weiteren sollen Änderungen durch die Rectangle-Bibliothek auch an das Rechteck-Objekt weitergegeben werden.



# Adapter-Muster

## Beispiel - Implementierung

```
public class RectangleAdapter extends Rectangle {  
    private Rechteck r;  
  
    public RectangleAdapter(Rechteck r) {  
        super(r.getX(), r.getY(), r.getX()+r.getB(), r.getY()+r.getH());  
        this.r = r;  
    }  
    @Override  
    public void setX1(double x1) {  
        super.setX1(x1);  
        r.setX(x1);  
    }  
    @Override  
    public double getX1() {  
        return r.getX();  
    }  
    @Override  
    public void setX2(double x2) {  
        super.setX2(x2);  
        r.setB(x2 - r.getX());  
    }  
    @Override  
    public double getX2() {  
        return r.getX()+r.getB();  
    }  
}
```

Was sind die Nachteile  
dieser Implementierung?

## Adapter-Muster

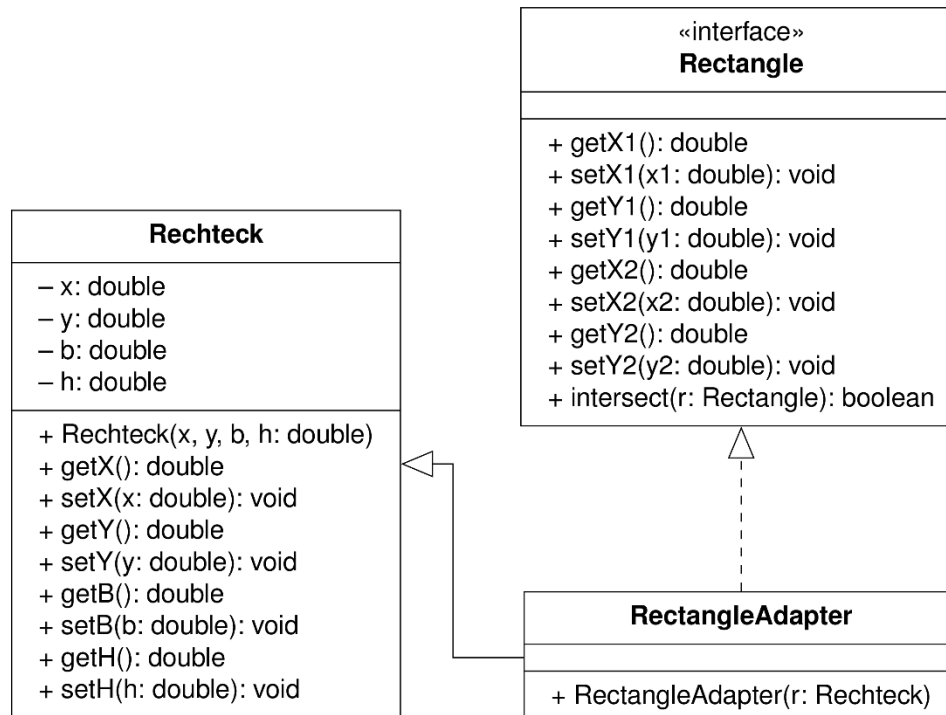
### Beispiel - Implementierung

```
public class AdapterTest {  
  
    public static void main(String[] args) {  
  
        Rechteck r1 = new Rechteck(1,1,4,2);  
        Rechteck r2 = new Rechteck(2,2,4,4);  
  
        RectangleAdapter ra1 = new RectangleAdapter(r1);  
        RectangleAdapter ra2 = new RectangleAdapter(r2);  
  
        System.out.println("Schneiden sich die Rechtecke? "+ ra1.intersects(ra2));  
    }  
}
```

## Adapter-Muster

### Verwendung von Schnittstellen

Eine bessere Implementierung wäre, wenn die Rectangle-Bibliothek eine Schnittstelle bereitstellen würde. Eine Adapter-Klasse sollte wenn möglich immer eine Schnittstelle implementieren



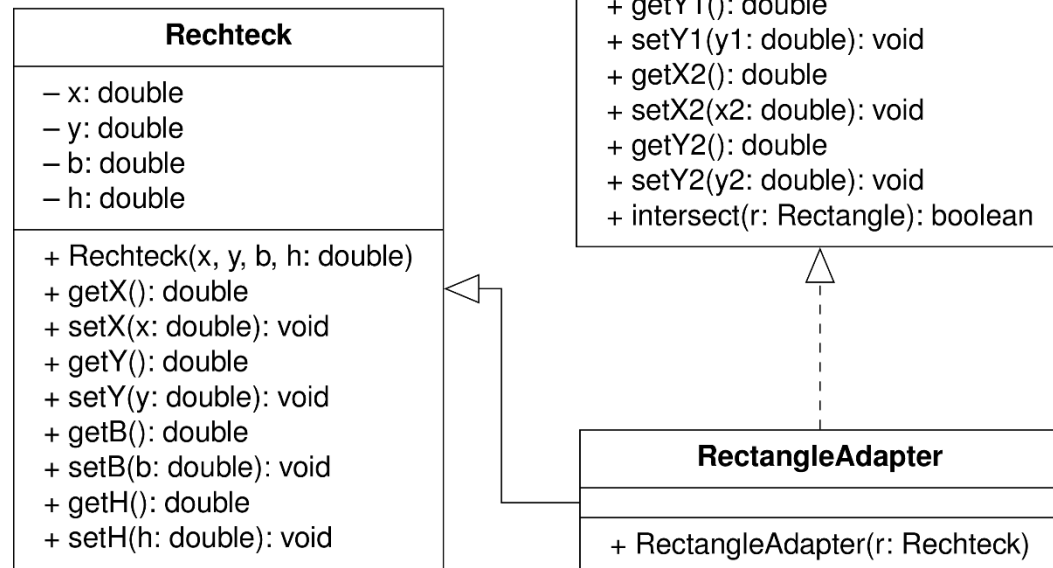
## Adapter-Muster

### Klassenadapter

Alternative: Ist das Ziel des Adapters ein Interface, kann von der Adapterquelle auch geerbt werden.

In Sprachen mit Mehrfachvererbung (z.B. C++) kann auch von beiden Klassen geerbt werden.

In Java werden generell nur Objektadapter benutzt.

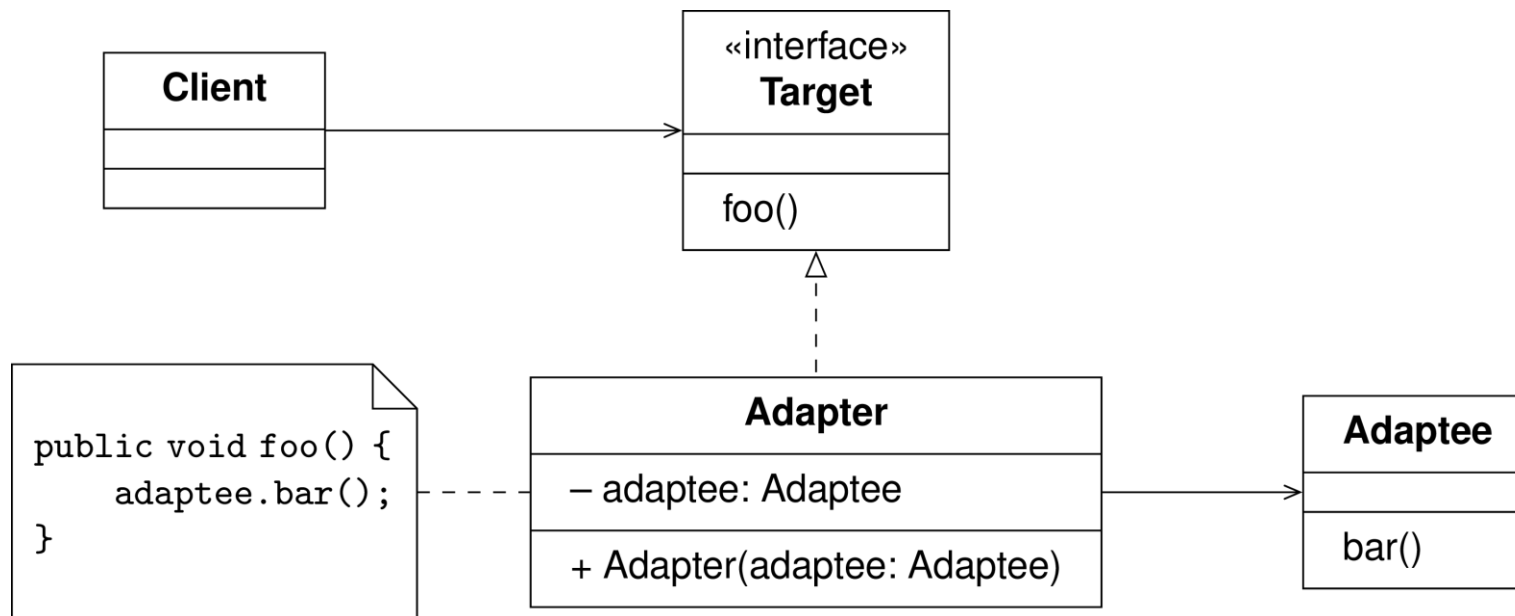




## Adapter-Muster

# Adapter-Muster

Das Adapter-Muster konvergiert die Schnittstelle einer Klasse in die Schnittstelle, die ein Client erwartet. Somit wird die Zusammenarbeit von inkompatiblen Schnittstellen ermöglicht.



## Adapter-Muster

### Adapter vs. Decorator

Adapter- und Decorator-Muster umhüllen jeweils ein anderes Objekt. Was sind die Unterschiede?

*Decorator*                      *Hier werden Klassen bzw. Objekten weitere Funktionalität hinzugefügt. Die Schnittstellen werden jedoch nicht geändert.*

*Adapter*                        *Wandelt eine Schnittstelle in eine andere Schnittstelle um. Es wird keine neuen Funktionalität implementiert.*

# Programmierung und Programmiersprachen

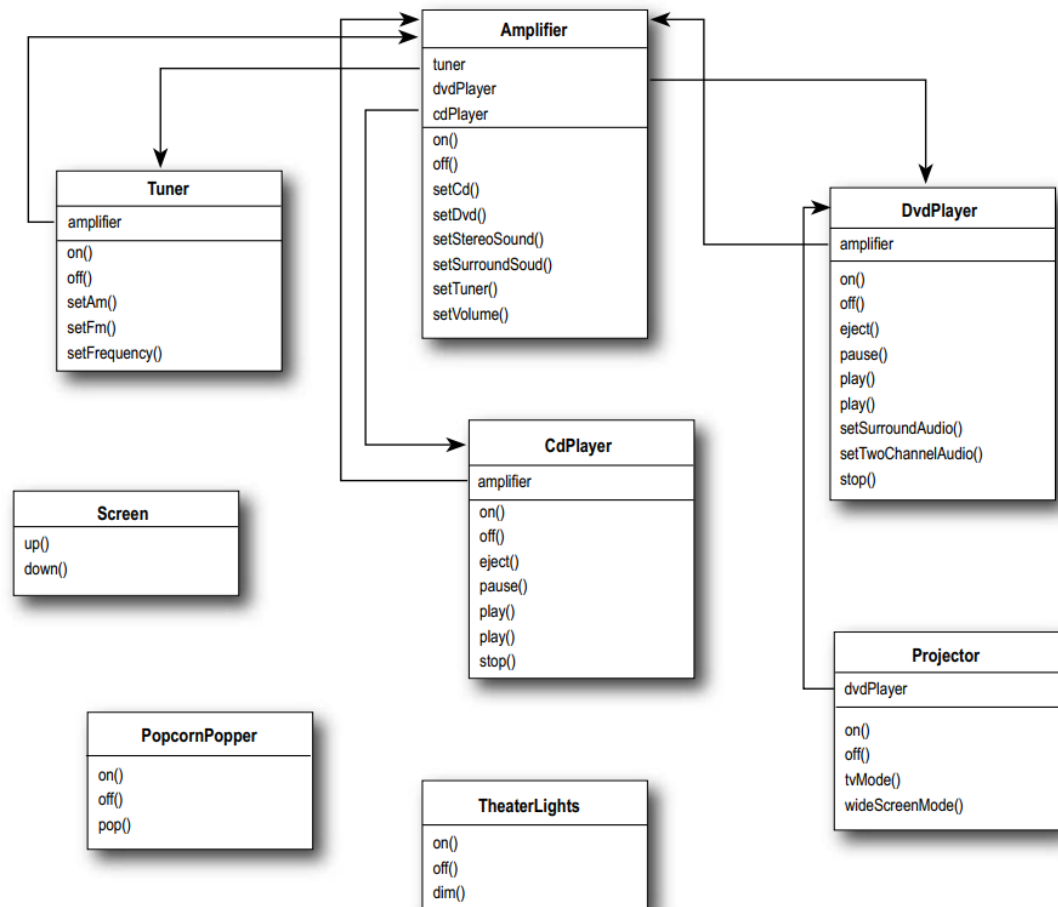
## Sommersemester 2023

### Facade-Muster

## Facade-Muster

# Vereinfachung von Schnittstellen

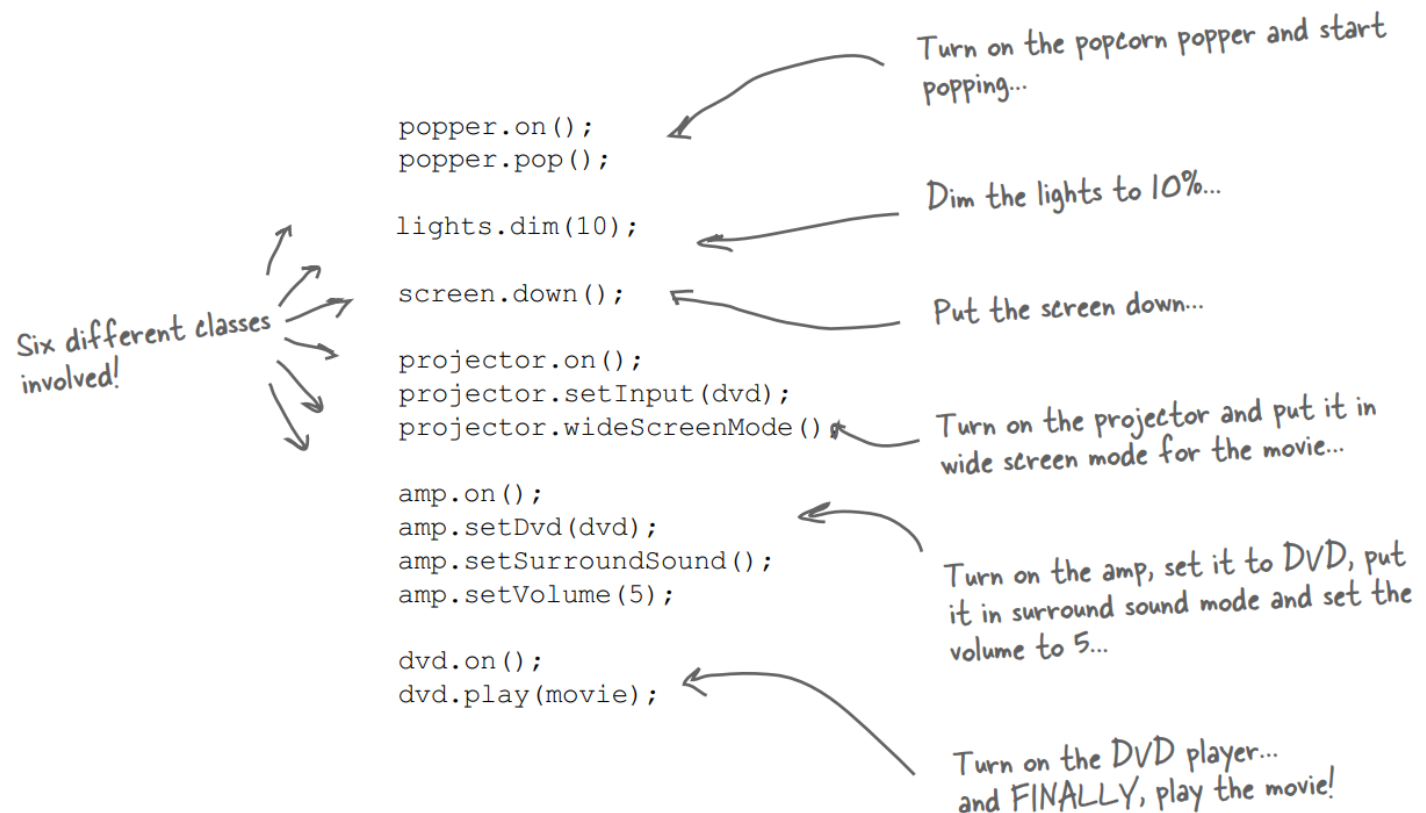
Gegeben sind eine Vielzahl von Klassen zur Steuerung und Verwendung eines Heimkino-Systems.



## Facade-Muster

### Vereinfachung von Schnittstellen

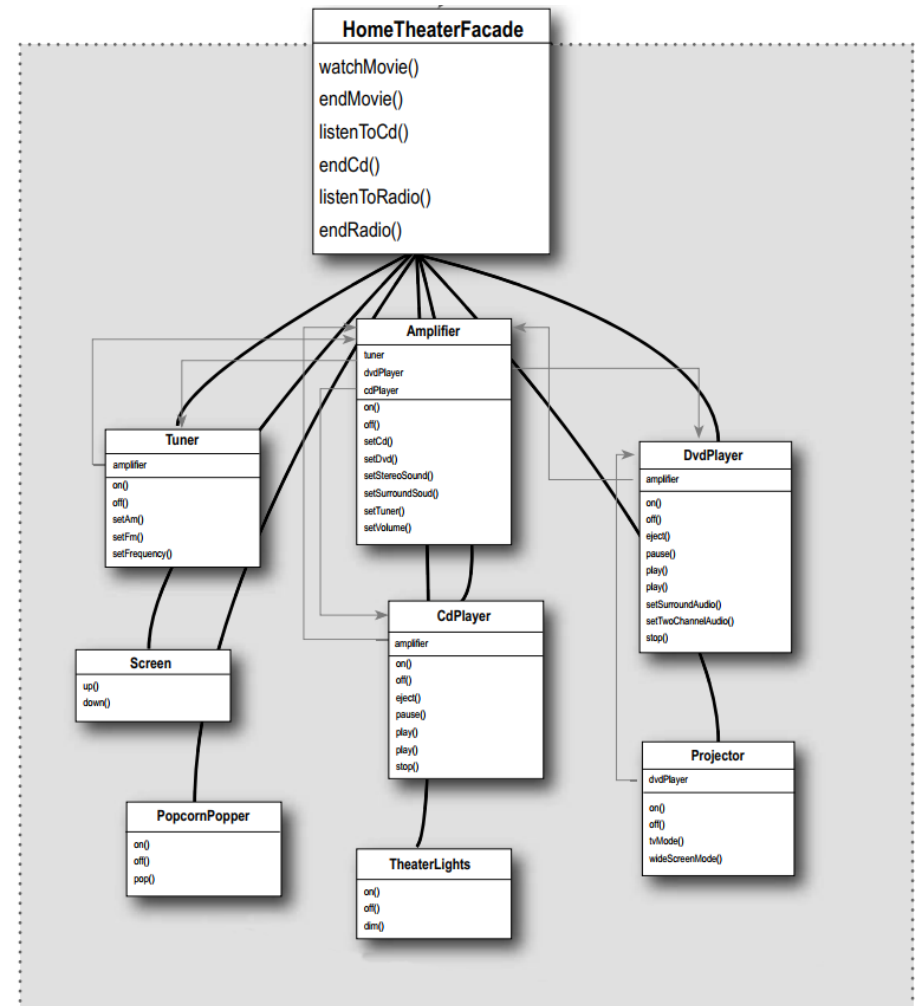
Damit ein Film angesehen werden kann, müssen folgende Aufgaben erledigt werden:



## Facade-Muster

# Facade-Muster

Mit dem Facade-Muster kann ein komplexes (Basis)-System eingebettet und die Verwendung vereinfacht werden. Die Facade-Klasse stellt nur andere Schnittstellen zur Verwendung zur Verfügung. Der Basis-Klassen werden nicht verändert. Das (Basis)-System steht auch weiterhin zur Verfügung.



## Facade-Muster

### Beispiel - Implementierung

```
public class HomeTheaterFacade {  
    Amplifier amp;  
    Tuner tuner;  
    DvdPlayer dvd;  
    CdPlayer cd;  
    Projector projector;  
    TheaterLights lights;  
    Screen screen;  
    PopcornPopper popper;  
  
    public HomeTheaterFacade(Amplifier amp, Tuner tuner, DvdPlayer dvd, CdPlayer cd,  
        Projector projector, Screen screen, TheaterLights lights, PopcornPopper popper) {  
  
        this.amp = amp;  
        this.tuner = tuner;  
        this.dvd = dvd;  
        this.cd = cd;  
        this.projector = projector;  
        this.screen = screen;  
        this.lights = lights;  
        this.popper = popper;  
    }  
    ...  
}
```

# Facade-Muster

## Beispiel - Implementierung

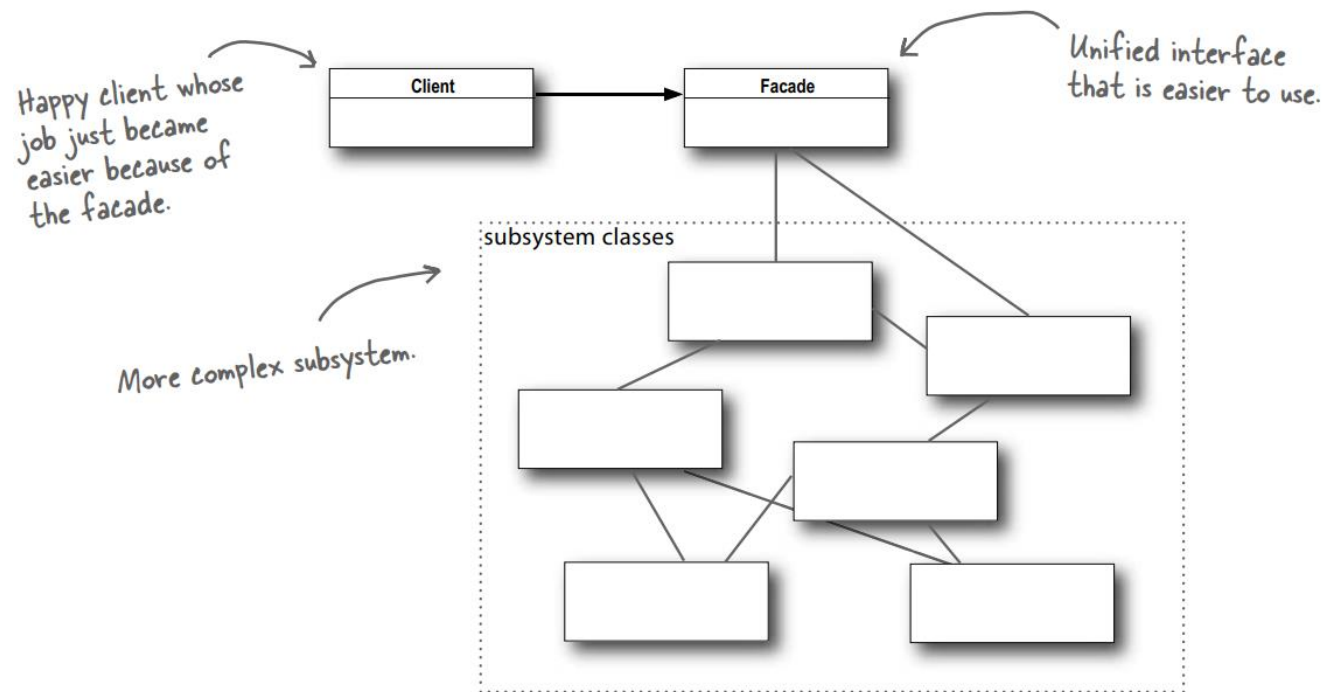
```
public class HomeTheaterFacade {  
    ...  
    public void watchMovie(String movie) {  
        System.out.println("Get ready to watch a movie...");  
        popper.on();  
        popper.pop();  
        lights.dim(10);  
        screen.down();  
        projector.on();  
        projector.wideScreenMode();  
        amp.on();  
        amp.setDvd(dvd);  
        amp.setSurroundSound();  
        amp.setVolume(5);  
        dvd.on();  
        dvd.play(movie);  
    }  
    ...  
}
```



## Facade-Muster

### Facade- vs Adapter-Muster

Fassaden und Adapter können mehrere Klassen einpacken. Der Zweck einer Fassade ist jedoch die Vereinfachung einer Schnittstelle. Hingegen soll ein Adapter eine Schnittstelle in eine andere umwandeln.



# Facade-Muster

## Kopplung

Gegeben sind einige Klassen zum Betrieb einer Wetterstation.

```
public class Thermometer {  
    private double temperatur;  
  
    public double getTemperatur() {  
        return temperatur;  
    }  
    public void setTemperatur(double temperatur) {  
        this.temperatur = temperatur;  
    }  
}  
  
public class WeatherStation {  
    private Thermometer thermometer;  
  
    public WeatherStation(Thermometer thermometer) {  
        this.thermometer = thermometer;  
    }  
    public Thermometer getThermometer() {  
        return this.thermometer;  
    }  
}
```

```
public class House {  
  
    private WeatherStation station;  
  
    public House(WeatherStation station) {  
        this.station = station;  
    }  
  
    public double getTemperatur() {  
        return this.station.getThermometer()  
            .getTemperatur();  
    }  
}
```

Was könnte das Problem sein?

# Facade-Muster

## Kopplung

Es sollte vermieden werden, dass eine große Anzahl von Klassen gekoppelt wird. Bei Änderungen kann es zu vielen Anpassungen kommen.

Methoden eines Objektes sollten nur Methoden aufrufen, die

- zum Objekt selbst,
- zu Objekten, die der Methode als Parameter übergeben wurden,
- zu Objekten, die die Methode erstellt oder instanziiert, sowie
- zu Komponenten des Objekts

gehören.

## Facade-Muster

# Das Prinzip der Verschwiegenheit

Die Klassen zum Betrieb einer Wetterstation müssten somit wie folgt angepasst werden:

```
class WeatherStation {  
    private Thermometer thermometer;  
  
    public WeatherStation(Thermometer thermometer) {  
        this.thermometer = thermometer;  
    }  
    public Thermometer getThermometer() {  
        return this.thermometer;  
    }  
    public double getTemperatur() {  
        return this.thermometer.getTemperatur() ;  
    }  
}
```

```
public class House {  
  
    private WeatherStation station;  
  
    public House(WeatherStation station) {  
        this.station = station;  
    }  
  
    public double getTemperatur() {  
        return this.station.getTemperatur();  
    }  
}
```