Wahlpflichtfach Design Pattern

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- miwilhelm@hs-harz.de
- http://www.miwilhelm.de
- Raum 2.202
- Tel. 03943 / 659 338

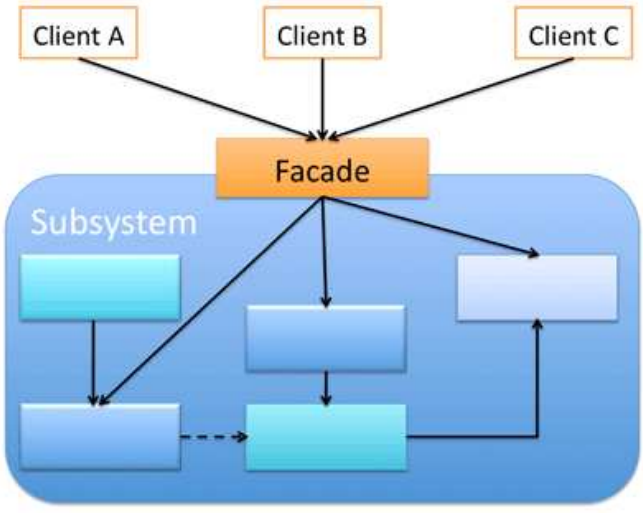
Inhalt

Entwurfsmuster: „Facade“

- **Facade-Pattern**
  - Dieses Pattern erlaubt die Kapselung von mehreren Klassen.
  - Man gruppiert diese unter einer Hauptklasse und definiert den Zugriff über eine oder mehrere Methoden. Damit bietet man einen zentralen Zugriffpunkt.
  - Man kann auch „feste“ Parameter vordefinieren, so dass die Anzahl der Parameter kleiner wird.
  - **Die Subklassen dürfen nicht von außen aufrufbar sein.**
  - Beispiel:

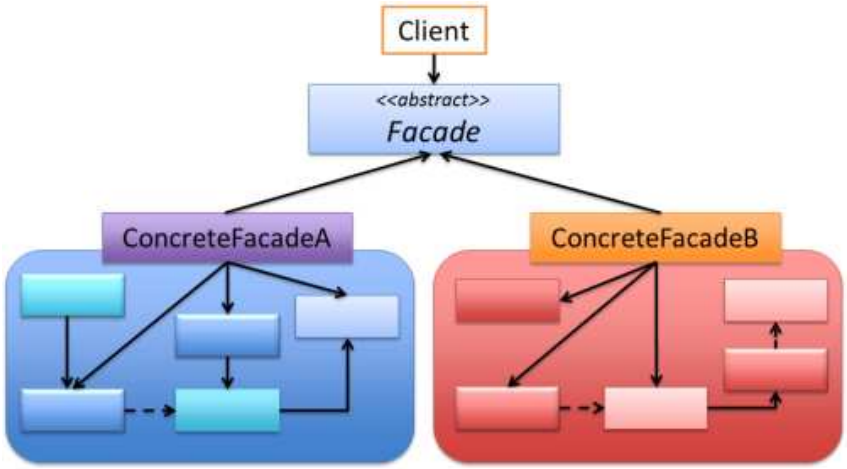
```
public static void ErrorMsg( String sTitle, String sMessage) {
   JOptionPane.showConfirmDialog(null,sMessage,sTitle,
     JOptionPane.DEFAULT_OPTION,
     JOptionPane.ERROR_MESSAGE);
 } // Error
```


Entwurfsmuster: „Facade“

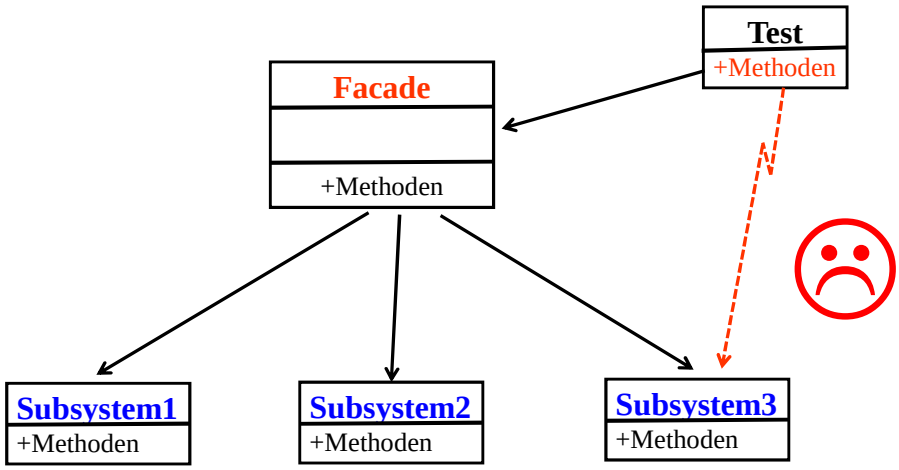


http://www.philipphauer.de/study/se/design-pattern/facade.php

Entwurfsmuster: „Facade“: mit Interface



Entwurfsmuster: „Facade“



Entwurfsmuster: „Facade“

Wann benutzen:

- Mehrere Klassen sind unübersichtlich und haben keine „guten“ Parameter“.
- Die Reihenfolge der Benutzung ist wichtig.
- Dient auch der Entkopplung der Systeme.
- Es ist nicht gewollt, dass man direkt auf das System zugreift (keine Sicherheitsüberprüfung in den Subklassen), aber in der Schnittstelle.
- Man muss nicht alle Funktionen veröffentlichen!!
- Man kann „Makromethoden“ implementieren.

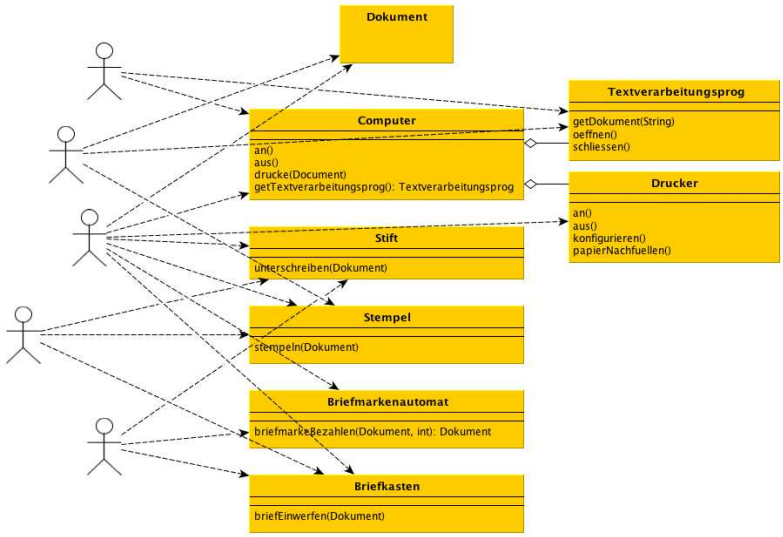
Konsequenzen:

- Die Schnittstelle, eine oder mehrere Klassen, erleichtern den Zugriff, sie muss aber MIT den Subsystemen wachsen.
- Nicht alle Funktionen sind eventuell in der Schnittstelle implementiert.

Vorteile:

- **Vereinfachte Schnittstelle.**
  - Der Client kann ein komplexes System einfacher verwenden, ohne die Klassen des Systems zu kennen und sich mit ihren mannigfaltigen Schnittstellen auseinander zu setzen. Stattdessen kann eine einzige wohldefinierte Schnittstelle der Fassade genutzt werden.
- **Entkopplung des Client vom Subsystem.**
  - Da der Client nur noch gegen die Fassade arbeiten, ist er **unabhängig** von Änderungen im Subsystem.
  - Wartungen und Modifikationen am Subsystem sind einfacher. Die Schnittstelle der Fassade nach außen bleibt davon unbetroffen. Kein Code der Clients bricht.
- **Änderungen**
  - Änderungen im Subsystem pflanzen sich **nicht mehr** fort. Die Facadeschnittstelle bleibt konstant.
- **Abhängigkeiten**
  - Eine Fassade vereinfacht die Anzahl Klassen pro Client.
- **Profi**
  - Jedoch können wenn gewünscht anspruchsvolle Clients weiterhin die Fassade übergehen und die Objekte des Subsystems direkt verwenden, sollte die bereitgestellte Schnittstelle der Fassade einmal nicht ausreichen.

1. Beispiel der Facade



http://www.philipphauer.de/study/se/design-pattern/facade.php

1. Beispiel der Facade

- Modellierung einer Firmenverwaltung.
- Technik:
  - Dokumente, Computer, Textverarbeitungsprogramme, Drucker, Stifte, Stempel, Briefkästen,Briefmarkenautomaten.
- User:
  - nur den Computer für Texte
  - Text mit einem Stift
  - Frankieren und Versenden von Briefen

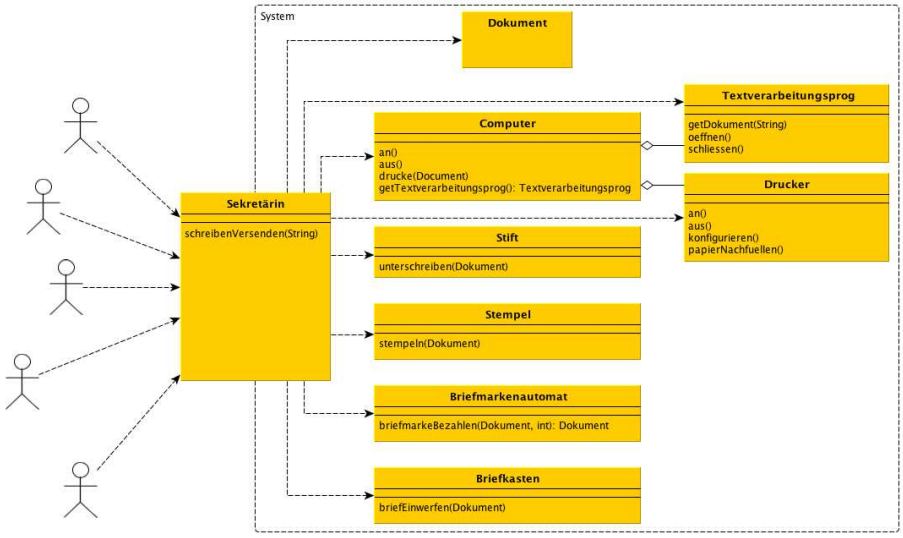
1. Beispiel der Facade

- Computer muss angeschaltet werden.
- Zugriff auf das Textverarbeitungsprogramm über den Computer.
- Das Textverarbeitungsprogramm muss geöffnet werden.
- Mit dem Textverarbeitungsprogramm muss das Dokument erstellt werden
- Drucker muss angeschaltet werden
- Drucker muss konfiguriert werden
- Papier muss in den Drucker gelegt werden
- Das Dokument muss gedruckt werden
- Der Drucker sollte ausgeschaltet werden
- Der Computer sollte ausgeschaltet werden
- Das Dokument muss mit dem Stift unterschrieben werden
- Das Dokument muss mit dem Stempel gestempelt werden
- Das Dokument muss mit Hilfe des Briefmarkenautomaten frankiert werden
- Das Dokument muss schließlich in den Briefkasten geworfen werden.

1. Beispiel der Facade

- String text = "Dieser Text soll verschickt werden";
- computer.an();
- Textverarbeitungsprog textprog = computer.getTextverarbeitungsprog();
- textprog.oeffnen();
- Dokument dokument = textprog.getDokument(text);
- drucker.an();
- drucker.konfigurieren();
- drucker.papierNachfuellen();
- computer.drucke(dokument);
- drucker.aus();
- stift.unterschreiben(dokument);
- stempel.stempel(dokument);
- briefmarkenautomat.briefmarkeBezahlen(dokument, 2);
- briefkasten.briefEinwerfen(dokument);

1. Beispiel der Facade



class Sekretaerin {

```
public void schreibenVersenden(String pString){
    String text = pString;
    computer.an();
    Textverarbeitungsprog textprog = computer.getTextverarbeitungsprog();
    textprog.oeffnen();
    Dokument dokument = textverarbeitungsprog.getDokument(text);
    drucker.an();
    drucker.konfigurieren();
    drucker.papierNachfuellen();
    computer.drucke(dokument);
    drucker.aus();
    computer.aus();
    stift.unterschreiben(dokument);
    stempel.stempel(dokument);
    briefmarkenautomat.briefmarkeBezahlen(dokument, 2);
    briefkasten.briefEinwerfen(dokument);
}

//Instanzvariable der genutzten Objekte Computer, Drucker, Stift etc.
}
```

1. Beispiel der Facade

Aufruf:

- Sekretaerin sekretaerin = new Sekretaerin();
- String text = "Dieser Text soll verschickt werden„;
- sekretaerin.schreibenVersenden(text);

2. Beispiel der Facade

```
public class Class1 {
        public int doSomethingComplicated(int x) {
            return x * x * x;
        }
}
public class Class2 {
        public int doAnotherThing(Class1 class1, int x) {
            return 2 * class1.doSomethingComplicated(x);
        }
}
public class Class3 {
        public int doMoreStuff(Class1 class1, Class2 class2, int x) {
            return class1.doSomethingComplicated(x) * class2.doAnotherThing(class1, x);
        }

}
```

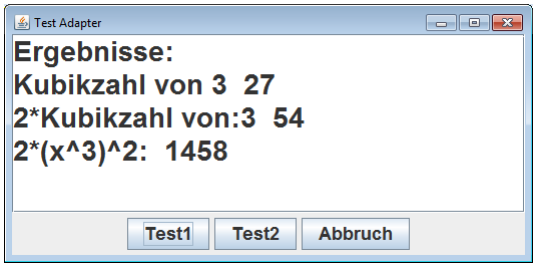
Völlig übersichtlich!

2. Beispiel der Facade

```
public class Facade {

        public int cubeX(int x) {
                Class1 class1 = new Class1();
                return class1.doSomethingComplicated(x);
        }

        public int cubeXTimes2(int x) {
                Class1 class1 = new Class1();
                Class2 class2 = new Class2();
                return class2.doAnotherThing(class1, x);
        }

        public int xToSixthPowerTimes2(int x) {
                Class1 class1 = new Class1();
                Class2 class2 = new Class2();
                Class3 class3 = new Class3();
                return class3.doMoreStuff(class1, class2, x);
        }
```

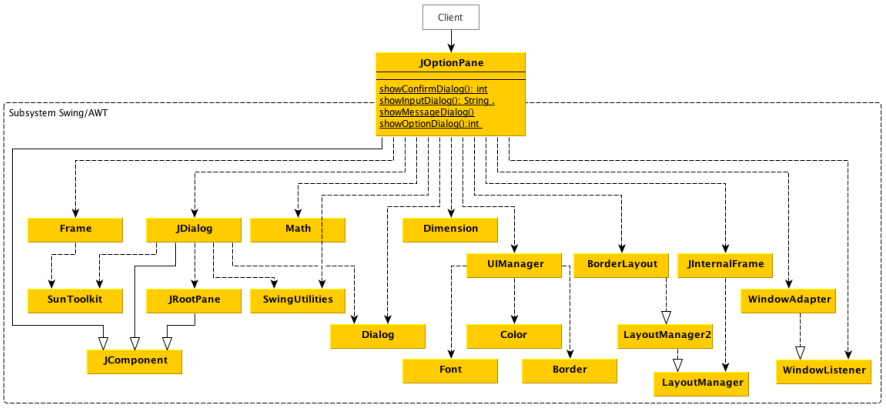

2. Beispiel der Facade

```
public void test() {
        Facade facade = new Facade();
        int x = 3;
        Syso(„x^3 " + x + ":" + facade.cubeX(3));
        Syso(" 2*x^3 " + x +  facade.cubeXTimes2(3));
        Syso("2*(x^3)^2:" + facade.xToSixthPowerTimes2(3));
}
```



Die Facade in Java

javax.swing.JOptionPane



http://www.philipphauer.de/study/se/design-pattern/facade.php