Systementwurf und Design

Beim ***Systementwurf*** legt man die konkrete Umsetzung des geplanten Produkts fest. Hierbei definiert man Datenflüsse, Algorithmen, Schnittstellen,… für das gesamte Produkt. Dabei nutzt man die schon bekannten Designprinzipien wie Abstraktion (z.B. abstrakte Oberklasse), Kapselung (public / private …), Modularisierung, Interfaces, ….

Des Weiteren ist es hilfreich, wenn man beim Designen der Software ***Diagramme und Muster*** nutzt. ***Muster*** sind bewährte Lösungsschablonen für wiederkehrende Probleme und stellen damit eine wiederverwendbare Vorlage zur Problemlösung dar. Neben den ***Algorithmenmustern*** gibt es auch ***Entwurfsmuster***, die Standardstrukturen für Systemarchitekturen darstellen.

Diagramme (Oldenburg S. 133-144)

Zusammenfassung der formalisierten grafischen Darstellungen für Systementwürfe:

- Klassenkarten geben Auskunft über die Attribute und Methoden einzelner Klassen
- Klassendiagramme beschreiben die Beziehungen der Klassen untereinander. Dabei wird der Fokus insbesondere auf die Referenzattribute gelegt.
- Ein Objektdiagramm beschreibt den aktuellen Zustand (Attributwertskombinationen aller Objekte) eines System bzw. einzelner Objekte.
- Sequenzdiagramme beschreiben die zeitlich dargestellten Aufrufe von Methoden unter Einbeziehung der dabei teilnehmenden Objekte eines Codeabschnitts.
- Zustandsdiagramme beschreiben Abläufe über Zustände und ihre Wechsel, die Ereignisse, die einen Zustandswechsel auslösen und die durch den Wechsel ausgelösten Aktionen.
- Struktogramme beschreiben Abläufe als Algorithmen.
- Datendiagramme beschreiben den Aufbau persistenter (dauerhaft gespeicherter) Daten. Für relationale Datenbanken sind dies Tabellenschemata.
- Datenflussdiagramme gibt die Art der Verwendung, die Bereitstellung und Veränderung von Daten innerhalb eines Programms dar. Hierbei wird allerdings nicht die konkrete Berechnung der Daten beleuchtet. → Blackbox-Sicht

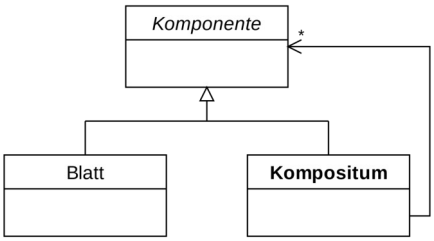
Entwurfsmuster

Entwurfsmuster sind - wie bereits erwähnt - Lösungsschablonen für wiederkehrende Entwurfsprobleme in der Softwarearchitektur und -entwicklung. In den folgenden Kapiteln werden die am häufigsten verwendeten Muster vorgestellt.

Entwurfsmuster Kompositum

Das Kompositum ist bereits von den rekursiven Datenstrukturen bekannt.

Das Kompositum wird angewendet um eine Teil-Ganzes-Hierarchie zu repräsentieren. Man verbirgt hierbei die Unterschiede zwischen einzelnen Objekten und zusammengesetzten Objekten (Kompositionen).



Exkurs: Komplexe Zahlen und Fraktale (optional)

Fraktal ist ein von Benoît Mandelbrot geprägter Begriff für geometrische Muster mit einem hohen Grad an Selbstähnlichkeit. Eine geometrische Figur wird ***selbstähnlich*** genannt, wenn sie aus verkleinerten Kopien ihrer selbst besteht (wie beim Kompositum: Ein Teil des Ganzen sieht genauso aus, wie das Ganze selbst.).

https://de.wikipedia.org/wiki/Fraktal

Einschub: Schlüsselwort static in Java

Das Schlüsselwort ***static*** in Java bindet ein Attribut oder eine Methode ***an die Klasse*** anstatt - wie üblich - ans Objekt.

- Das bedeutet, dass Methode ohne ein existierendes Objekt aufgerufen werden können. Bei der Punktnotation wird hier anstatt des Objektnames der Klassenname hingeschrieben: ***Klassenname.statischeMethode().***
- Statische Attribute existieren nur genau ein Mal; auch ohne Objekte dieser Klasse. Falls Objekte dieser Klasse existieren, ***teilen sich alle Objekte*** dieser Klasse das statische Attribut.

Beim Klassendiagramm werden statische Attribute oder Methoden unterstrichen!

Aufgabe 1:

Teste das Verhalten vom Schlüsselwort static im dazugehörigen BlueJ-Projekt.

Einschub: main-Methode

Bisher haben wir in BlueJ das Programm über das Erzeugen eines Objekts der „Hauptklasse“ gestartet. Dies ist bei anderen Programmen ja nicht der Fall. Hier kann man die Datei mit Doppelklick starten.

Die main-Methode ist die Methode, die beim Starten eines Programms (Doppelklick auf

die Datei) ausgeführt wird. Sie ***muss*** folgende Signatur besitzen:

public static void main(String[ ] args)

- Sie muss ***public*** sein, ansonsten kann sie der Nutzer, der das Programm starten will, nicht ausführen.
- Die Methode muss ***statisch*** sein, da sie als allererstes aufgerufen wird, bevor ein Objekt erzeugt wurde.
- Sie besitzt ***keinen Rückgabewert***.
- Sie bekommt ein ***String-Array übergeben***, das die Kommandozeilenparameter enthält. Beispiel:

  shutdown.exe -s -f -t 60

  -s, -f, -t und 60 sind die Kommandozeilenparameter, die das Programm mit gewünschten Werten des Nutzer starten: -s steht für Shutdown also Herunterfahren, -f steht für force also das Herunterfahren erzwingen, -t steht für Timer also eine Angabe, nach wie viel Sekunden heruntergefahren werden soll und 60 steht für die Anzahl der Sekunden.

Aufgabe 2: (optional)

Vollständige das Projekt Fraktale mithilfe der dazugehörigen Präsentation.

Einschub: ausführbares Programm mit BlueJ erstellen

- Schreibe eine main-Methode (am besten in der Hauptklasse), die das Programm passend startet.
- Projekt → Als jar-Archiv speichern… → Gib die Klasse mit der main-Methode an → Gib der Datei einen Namen
- Die .jar-Datei lässt sich nun mit Doppelklick ausführen, sofern eine passende Java-Version installiert ist.

  https://www.oracle.com/java/technologies/javase-downloads.html

Aufgabe 3:

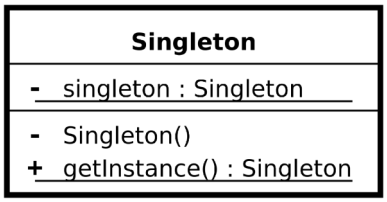
Such dir aus deinen Projekten ein beliebiges aus und mache dieses ausführbar.

Entwurfsmuster Singleton

Das ***Singleton*** (dt. ***Einzelstück***) ist ein Entwurfsmuster, das zusichert, dass es von der Singleton-Klasse nur genau ein Objekt existiert und in der Regel global verfügbar ist.

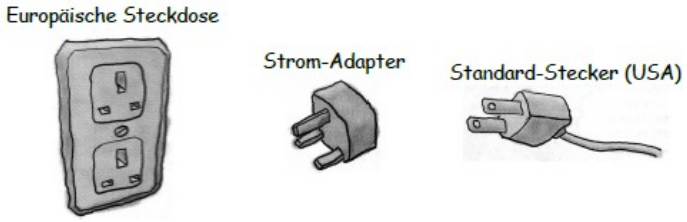
Aufgabe 4:

Schreibe anhand des Klassendiagramms und mithilfe des Schüsselworts static deine Singleton-Klasse.

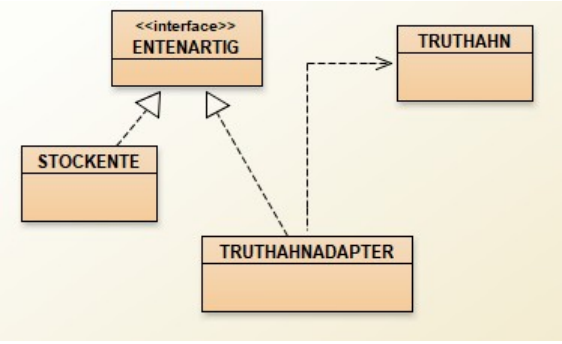


Entwurfsmuster Adapter

Dieses Entwurfsmuster wird verwendet um eine Brücke zu schlagen zwischen zwei verschiedenen Schnittstellen. Dadurch können Klassen zusammenarbeiten, die sonst nicht in der Lage wären. Ähnlich eines Hardware-Adapters:



Adapter werden meist in der Form eines Interfaces festgelegt. Elemente der einen Klasse, die die andere beherrschen soll, werden in das Interface gelagert. Die Adapter-Klasse implementiert dieses Interface und referenziert gleichzeitig ein Objekt der anderen Klasse. Da die Adapter-Klasse nun die Methoden des Interfaces überschreiben muss, kann entsprechend umgesetzt werden, wie Objekte der anderen Klassen in diesen Fällen agieren sollen.



Beispiel:

Hier sollen Truthähne zu Enten adaptiert werden. Die Fähigkeiten der Ente (quaken und fliegen) werden in das Interface ENTENARTIG eingebettet. Der Truthahnadapter repräsentiert (referenziert) einen Truthahn und implementiert gleichzeitig das Interface, muss also umsetzen, wie der referenzierte Truthahn quakt und fliegt.

```
public void quaken()
{
    truthahn.schnattern();
}

public void fliegen()
{
    // ein Truthahn kann nur kurze Distanzen bewältigen,
    truthahn.fliegen();
    truthahn.fliegen();
    truthahn.fliegen();
    truthahn.fliegen();
}
```

Nun können Objekte der Klasse Truthahnadapter wie „entenartige“ behandelt werden.

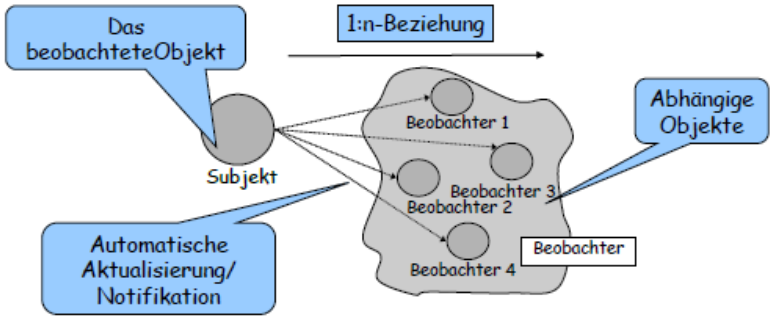
Aufgabe 5:

Öffne das Projekt Adapter und beschreibe präzise die Funktionsweise des Adapters.

Entwurfsmuster Observer

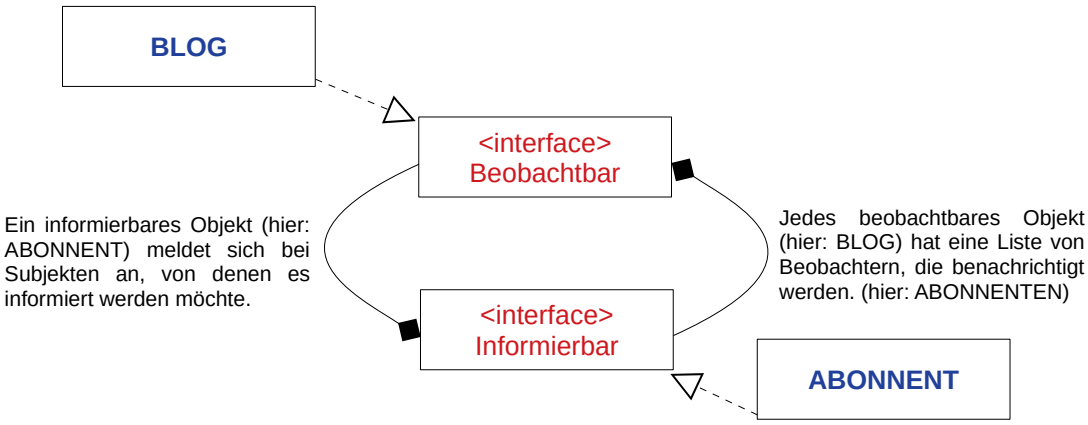
Dieses Prinzip beruht darauf, dass ein Objekt sich bei einem anderen Objekt (***Subjekt***) als „Beobachter“ (***Observer***) anmeldet und von nun an von diesem über gewisse Zustandsänderungen informiert wird. Es wird eine ***Eins-zu-viele-Abhängigkeit*** umgesetzt.

Beispiele: Blogs, Newsletter, Push-Nachrichten auf dem Smartphone, …



Um das zu realisieren verwendet man in der Regel zwei Interfaces:

1. ***Interface Beobachtbar***
   Es bietet eine Möglichkeit, dass sich Beobachter anmelden können. Die implementierende Klasse (=Subjekt) muss die Beobachter verwalten und kann über die Methode aktualisieren() die angemeldeten Beobachtbar über eine Änderung informieren.
2. ***Interface Informierbar***
   Enthält eine Methode aktualisieren(). Da ein Subjekt seine Beobachter verwaltet, kann das Subjekt über diese Methode die Änderungen an jeden Beobachter senden.



Aufgabe 6:

Implementiere ein Entwurfsmuster Observer anhand des Beispiels (vgl. Präsentation)

Exkurs: GUI-Programmierung

Zur Programmierung einer ***GUI*** (**G**raphical **U**ser **I**nterface) in JAVA sind zwei Bibliotheken gegeben, die wir verwenden. Das sind ***AWT*** (Abstract Window Toolkit) und ***Swing***.

Wir binden die Bibliotheken folgendermaßen ein:

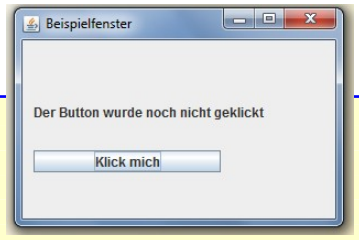
```
import javax.swing.*;
import java.awt.event.*;
```

Die Klasse, die die graphische Oberfläche modellieren soll, erbt von ***JFrame***. JFrames stellen die Hauptfenster dar und sind somit die Grundlage der GUI-Programmierung.

Nun stehen uns verschiedene Bedienelemente zur Verfügung um eine Interaktion mit dem Benutzer umzusetzen. Dies sind z.B. Buttons (***JButton***) oder Textfelder (***JLabel***).

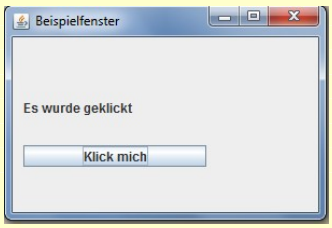
Um eine Reaktion auf Klicks etc. umzusetzen, wird der AWT- ***ActionListener*** implementiert und für die betreffenden Bedienelemente hinzugefügt. Die Methode ***actionPerformed(ActionEvent e)*** muss ausgehend vom Interface ActionListener immer implementiert und überschrieben werden.

Code-Beispiel:



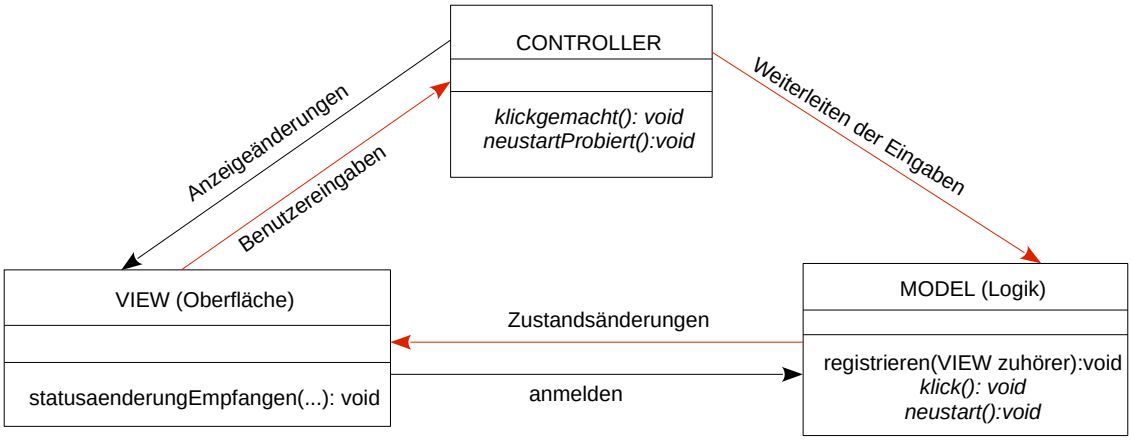

```
import javax.swing.*;
import java.awt.event.*;
public class GUI extends JFrame implements ActionListener
{
   private JButton button1;
   private JLabel textfeld;

   public static void main(String[] args)          // eine Main-Methode wird benötigt um das Programm als jar-Datei
   {                                               // unabhängig von BlueJ ausführen zu können
      new GUI();                                   // es wird nur der Konstruktor darin aufgerufen.
   }
```

```
GUI()                                          // Konstruktor: Bedienelemente „designen“
{
   super();

   button1 = new JButton("Klick mich");
   button1.setBounds(10,100,170,20);
   button1.addActionListener(this);


   textfeld = new JLabel("Der Button wurde noch nicht geklickt");
   textfeld.setBounds(10,50,280,30);

   super.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   //WICHTIG: wird benötigt, damit sich das Fenster wieder schließt
   super.setLocation(100,100);                             // Platzierung des Fensters auf dem Bildschirm→ oben links: (0|0)
   super.setSize(300,200);
   super.setTitle("Beispielfenster");
   super.setLayout(null);

   super.add(button1);                                     // Button und Textfeld dem Fenster hinzufügen
   super.add(textfeld);

   super.setVisible(true);                                 // das Fenster anzeigen lassen
}

public void actionPerformed(ActionEvent e)                 // diese Methode wird immer aufgerufen wenn eine Interaktion
{                                                          // (z.B ein Klick) passiert.
    if(e.getSource() == this.button1){                     // getSource() findet heraus welcher Button geklickt wurde
        textfeld.setText("Es wurde geklickt");
    }

 }
}
```



Enwurfsmuster Model-View-Controller (MVC)

Das Entwurfsmuster MVC bietet mit einer Model – View – Controller Struktur ein Softwaredesign und dient als Grundlage vieler Softwareprojekte mit grafischer Aus- und Eingabe. Es verwendet dabei das Observer- und evlt. das Strategie-Muster, um die einzelnen, unabhängigen Komponenten voneinander zu trennen.



MODEL

Das MODEL entspricht der ***Logik*** der Software. Sie verwaltet die Software in ihrem Inneren und nimmt Berechnungen vor:

z.B.: **klick(), neustart()**,...

Treten Änderungen auf, so muss das Model diese ***Zustandsänderungen*** an seine

Grafikschnittstelle weitergeben (siehe VIEW)

VIEW

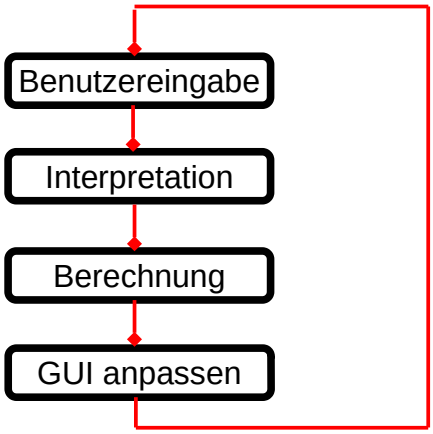
Die VIEW entspricht der ***graphischen Oberfläche (GUI)*** der Software. Sie nimmt Eingaben des Benutzers entgegen und leitet sie erst an den CONTROLLER weiter.

Außerdem muss sie sich beim MODELL (als Beobachter) ***anmelden***, um über Änderungen der inneren Struktur informiert zu werden. Darauf reagiert sie entsprechend mit graphischen Veränderungen.

CONTROLLER

Der CONTROLLER ist das ***Bindeglied*** zwischen VIEW und MODEL. Er interpretiert die Eingaben des Benutzers für eine VIEW und leitet sie weiter an das MODEL. Eventuelle Grafikeinstellungen, die keiner Berechnung bedürfen, nimmt er selber vor.

Es ergibt sich im Wesentlichen folgender Zyklus (im Diagramm rot dargestellt):



Aufgabe 7:

Implementiere einen Klickzähler gemäß der Vorlage mithilfe des Entwurfsmuster MVC.