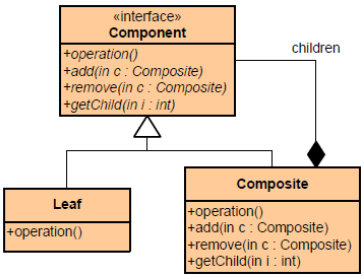
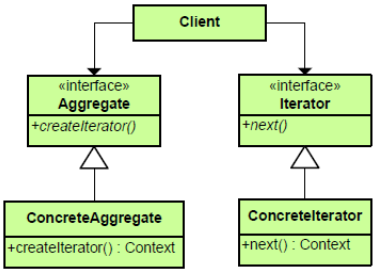
Composite




```
Interface Component {}
class Composite implements Component {
   private List<Component> list=new
   ArrayList<Component>;
   public void add(Component k) {
      list.add(k); }
   public void remove(Component k) {
      list.remove(k); }
   void methode1() {
      for(Komponente k: list) { methode1();
   }
} }
```

- Komponente definiert als (abstract / interface) Basisklasse das gemeinsame Verhalten aller Teilnehmer
- Blatt repräsentiert Einzelobj.; besitzt keine Kindobjekte
- Das Kompositum enthält Komponenten, also weitere Komposita oder auch Blätter, als Kindobjekte

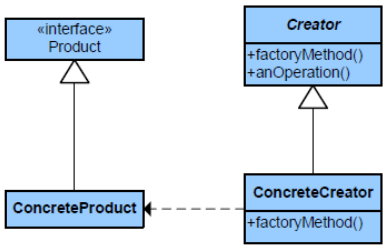
Iterator




```
Class Iterator{
    private String[] source;
    private int current=0;
    Iterator(String[] source) {
       this.source=source }
    String getCurrent(){
       return source[current]; }
    boolean hasNext() {
       if(curret<source.length) {
          return true; }
       return false; }
    void next() {
       if(current+1<source.length) {
          current++; }
} }
```

- Ziel: durch Elemente einer Klasse (Array, Liste, Map…) durchzugehen ohne die Klasse konkret zu kennen
- Klasse Iterator mit dem Attribut *source* vom Typ der Klasse sowie dem Attribut *current* (meist Integer) erstellt
- Methoden sind meist *hasNext(), next(),last(), getCurrent()*

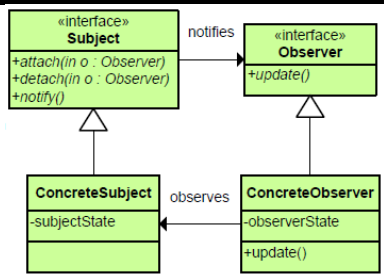
Factory Method




```
class Fabrik{
    Auto = Auto.prodRotesAuto();
    Auto = Auto.prodBlauesAuto();
}

class Auto{
    String farbe;
    private Auto(String farbe){
        this.farbe=farbe; }

    public Auto prodBlauesAuto(){
        return new Auto(„blau“); }
    public Auto prodRotesAuto(){
        return new Auto(„rot“); }
}
```

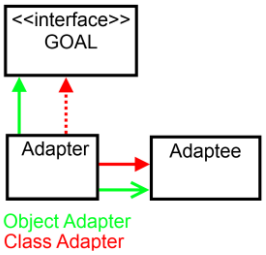
- Meth. die wie Konstruktor Objekte der Klasse erzeugt

Observer



- Überprüft Statusveränderungen eines Subjektes
- Beobachtete-Klassen erben von Observable
- Beobachter implementieren Observer

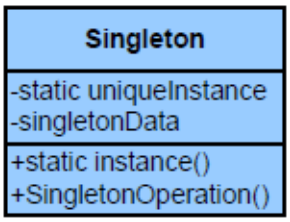
Class/Object Adapter




```
class usingClass {
   Adapter adapter=new Adapter();
   Typ state=adapter.translate(offeringClass.state)
}

class offeringClass {
   // Typdeklaration whatever }

class Adapter {
   Datentyp translate(Input) {
      // Übersetzung des Datentyps }
   }
}
```

- Konvertiert das Interface einer Klasse in das des Klienten
- ermöglicht Zusammenarbeit von Klassen, die ansonsten aufgrund inkompatibler Interfaces nicht zusammenarbeiten

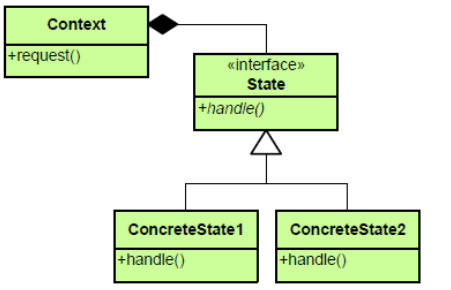
Singleton




```
public class Singleton {
    private static Singleton instance = null;
    private Singleton() {}
    public static Singleton getInstance() {
        if (instance==null) {
          instance = new Singleton();
        }
        return instance;
    }
}
```

- Sorgt dafür, dass immer nur EINE Instanz einer Klasse instanziiert ist

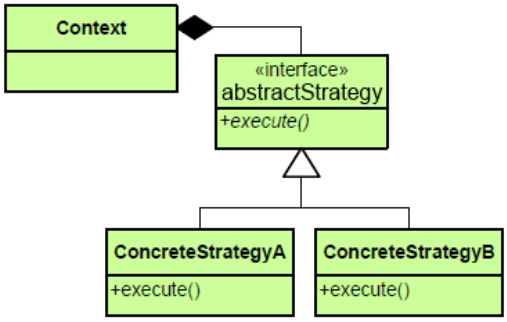
State




```
Interface AbstractState{
void writeStatus();
}
class StateA implements Abstract State{
   void writeStatus(){
      System.out.printLn("Status:A");
   } }
class StateB implements Abstract State{
   void writeStatus(){
      System.out.printLn("Status:B");
   } }

class StateUser {
private State myState;

StateUser() {
   setState(new ConcreteStateA()); }

public setState(AbstractState newState){
   myState=newState; }

public String writeStatus() {
   myState.writeStatus(); }
}
```

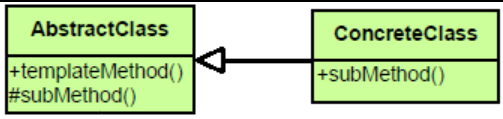
- Erlaubt einem Objekt sein Verhalten zu ändern, wenn interne Statusänderungen geschehen
- State oft als Interne-Klasse implementiert

Strategy



- ermöglicht Auswahl zwischen verschiedenen “Modi”
- concreteStrategies erben von Interface Strategy
- concreteStrategies haben eine Variante des Algorithmus
- Kontextklasse hat Attribut vom Typ abstractStrategy, dass auf die jeweilige conreteStrategy gesetzt wird

Template Method



- In Superklasse wird Skelett einer Methode definiert, die erst in einer Unterklasse vervollständigt wird

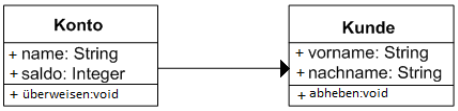
Junit Test

Import org.junit.* | Public class *Klassenname* extends TestCase{}

- assertTrue([message], booleanCondition) und assertFalse([message], booleanCondition)
- assertEquals([message], expected, actual, [Tolerance (*double & float*)])
- assertNull([message], object) und assertNotNull([message], object)
- fail([message]) lässt den Test augenblicklich scheitern

Analyse-Klassendiagramm / Domänenmodell

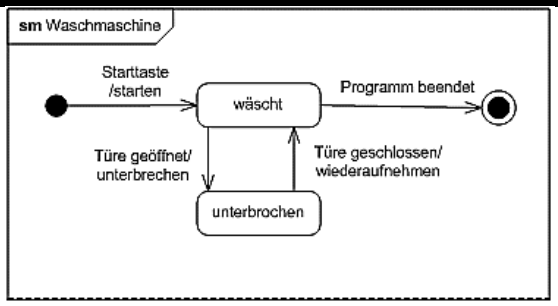
Erweitertes Analyse-Klassendiagramm



CRC - Diagramm

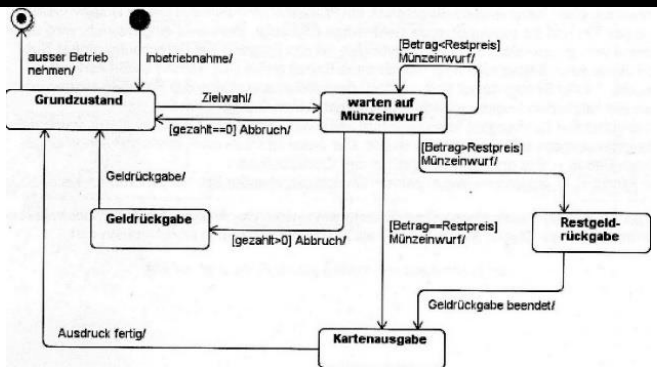
Klassenname: Auction	
Verantwortlichkeiten	**Mithelfer**
openAuction	
registerItem	
createItem	Item
createPerson	Person
bidItemBy	Item
closeAuction	
showItemList	
showAuctionResult	

Verhaltenszustandsmaschine



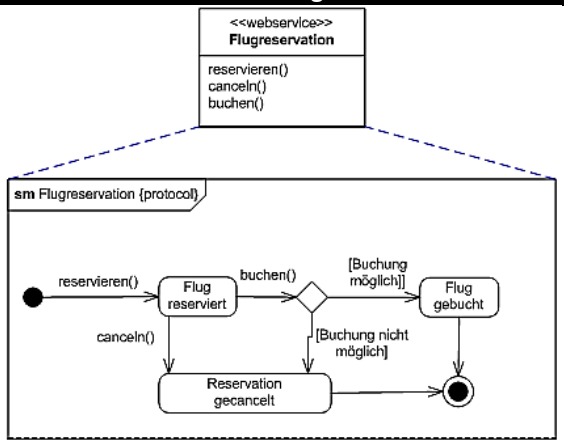
- Methoden die jederzeit ausgeführt werden können außen
- An gerichteten Pfeil Notiz „Trigger [Condition] / Action“
- Definiert die Reaktion des Systems auf Signale und Ereignisse von außerhalb

Protokollmaschine



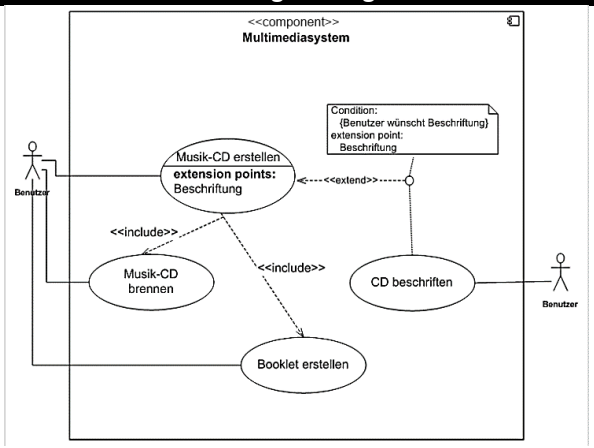
- [Vorbedingung] Trigger / [Nachbedingung]
- Spezifikation zulässiger Reihenfolgen von Operationen

Aktivitätsdiagramm



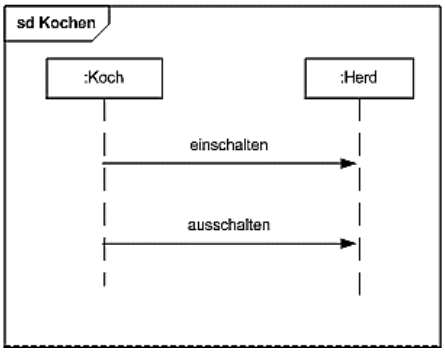
- Stellt die Vernetzung von elementaren Aktionen dar

Anwendungsfalldiagramm



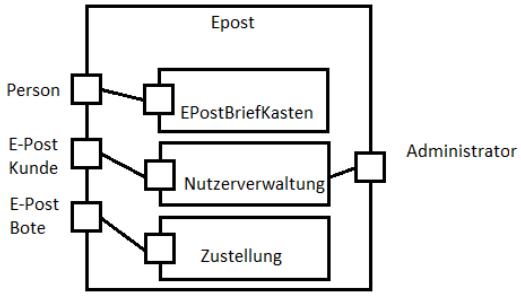
- Systemgrenzen in Rechtecken | Vorgänge in Ellipsen

Sequenzdiagramm



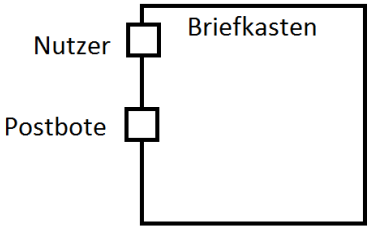
- Stellt konkrete Interaktion zwischen Klassen dar

Top-Level Architektur



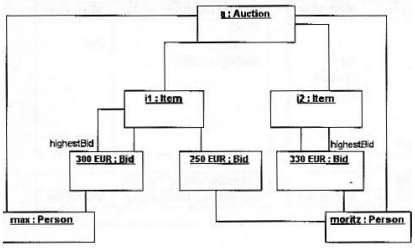
- Erweiterung des Kontextmodells
- Zeigt Interaktion zw. Schnittstelle und Teilkomponente

Kontextmodell



- Stellt Nutzer dar, und welche Klassen sie ansprechen

Objektdiagramm



List und Set sind iterierbar | über Maps lässt sich mithilfe des KeySets iterieren