Einsatz agiler Methoden der Softwaretechnik bei Projektarbeit

Workshop bei der 12. Landestagung der Informatiklehrkräfte am 16. April 2016 an der Universität Rostock

Ulrich Kiesmüller
Simon-Marius-Gymnasium Gunzenhausen
Kiesmueller@simon-marius-gymnasium.de

Die Idee

- Objektorientierte Modellierung und Programmierung im gymnasialen Lehrplan

Den Schülern wird deutlich, dass in der objektorientierten Sichtweise alle bisher angewandten Techniken zielgerichtet und miteinander verzahnt zur Lösung umfangreicherer Aufgabenstellungen genutzt werden können.

Die neuen Inhalte begegnen den Schülern im Rahmen von ausbaufähigen Aufgabenstellungen, wobei die praktische Arbeit einen großen Anteil des Unterrichts umfasst.

- Entwicklung kleiner Softwareprojekte

Gleichzeitig erfahren die Jugendlichen auch schrittweise die grundlegenden Vorgehensweisen bei der Planung von Softwareprojekten. Sie erkennen, dass erst durch sorgfältig geplante Teamarbeit in Verbindung mit einem soliden Fundament an Wissen und einer klar strukturierten Vorgehensweise die Lösung von schwierigen, für den Einzelnen zu umfangreichen Aufgabenstellungen möglich wird.

Die Idee

- Geringe zur Verfügung stehende Zeit (ca. 10 Unterrichtsstunden)
- Keine Softwareprojekte im eigentlichen Sinn (z. B. Tic-Tac-Toe)
- Keine Teamarbeit, sondern Pair-Programming einer nicht-trivialen Software
- Einbindung von Lehrplaninhalten der objektorientierten Modellierung und Programmierung in Projektarbeitsphase

Die Idee

- Möglichkeit des schrittweise Arbeitens über das Schuljahr hinweg an gleichem Thema (z. B. Supermarkt)
  - Modellierung (Klassendiagramm, Zustandsdiagramm, Nassi-Shneiderman-Diagramm, Sequenzdiagramm)
  - Implementierung in Java (Klassen, Konstruktoren, Methoden, Felder, Zufallszahlen)
  - Vererbung und Polymorphismus
  - Abstrakte Klassen und Interfaces
  - Graphische Benutzungsoberfläche
- Verschwindende Motivation wegen nicht selbst gewählten Themas

Die Idee

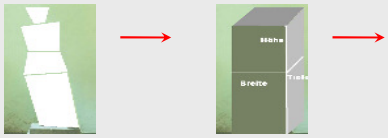
- Freie Themenwahl mit Pair-Programming
- „Turnschuhdidaktik“ – Lehrkraft völlig erledigt nach jeder Stunde
- Lernende empfinden Unterrichtsstunden überwiegend als Wartezeit auf Unterstützung
- Frustration wegen kaum vorhandener Erfolgserlebnisse, die sich erst spät einstellen
- Nur geringe Verbesserung bei Arbeit in größeren Gruppen
- Einsatz agiler Methoden
  - Betreuung weniger größerer Gruppen (ca. 8 Personen)
  - Interessensbasierte Einteilung und Themenwahl
  - Förderung des selbstständigen Arbeitens
  - Theorievermittlung bedarfsorientiert
  - Ermöglichung rascher und häufiger, wiederkehrender Erfolgserlebnisse

Theorieeinheiten

- Vererbung (abstrakte Klassen)
- Datenstruktur Feld
- Zufallszahlen
- Button-Steuerung
- Textausgabe
- Eingaberoutinen
- Dokumentation

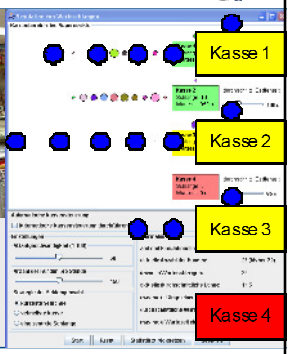
Modellierung

- Ein *Modell* ist eine,
  - abstrahierte Beschreibung
  - eines realen oder geplanten Systems,
  - das die für eine bestimmte Zielsetzung
  - wesentlichen Eigenschaften des Systems erhält.
- Möglichkeit der *Modellierung durch Abstraktion*
- Modellierung ist nicht eindeutig



- Modellierung abhängig vom Ziel

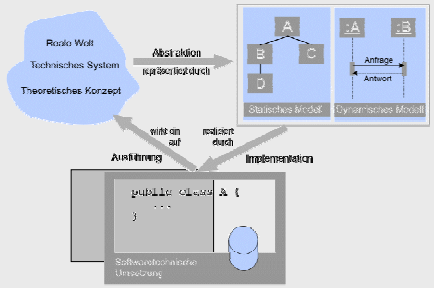
Modellierung → Simulation



- Reale Situation
- Modellierung
- Simulation

OOM-Einführung

- **Objektorientierte Modellierung**



OOM-Einführung

- Grundidee: Die Welt besteht aus Objekten

Objekte sind eigenständige, eindeutig identifizierbare Einheiten der Modellierung mit individuellen Merkmalen und dienen der Repräsentation von Gegenständen oder Konzepten.
Sie besitzen durch Werte erfassbare Eigenschaften, die ***Attribute***. Die *Attributwerte* charakterisieren den aktuellen Zustand des Objektes. Sie können während der Lebensdauer der Objekte unterschiedliche Werte annehmen. Die Attribute selbst sind unveränderliche Merkmale des Objekts. Als Bezeichnung werden Substantive verwendet.
Operationen (sog. ***Methoden***): Die Objekte stellen selbst die Operationen bereit, mit denen die Werte der Eigenschaften ausgelesen oder verändert werden können. Operationen werden durch den Infinitiv oder Imperativ von Verben mit „()“ benannt.

OOM-Einführung

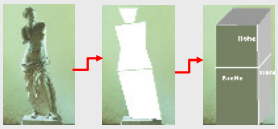
- Wichtigstes Grundprinzip: Datenkapselung

Zustand und Verhalten eines Objekts bilden eine Einheit. Man sagt: Ein Objekt kapselt Zustand (Daten, Attributwerte) und Verhalten (Operationen, Methoden). Der direkte Zugriff auf den Zustand eines Objekts (Bild: *innere blaue Kugel*) wird unterbunden und kann nur über die vom jeweiligen Objekt zur Verfügung gestellten außen sichtbaren Operationen (Bild: *äußerer Mantel*) erfolgen.

Diese bilden damit die Schnittstelle zwischen dem Objekt und seiner Umgebung. Die Schnittstellen bestimmen, auf welche Weise mit dem Objekt interagiert werden kann.

OOM-Einführung

- Objekte: Beschreiben



Modellierung durch Abstraktion

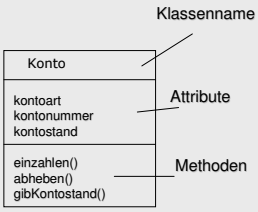
Nur die für die spätere Programmanwendung relevanten Attribute und Methoden der Objekte sind relevant.

Mögliche Modellierungssprache: UML (Unified Modeling Language)

OOM-Einführung

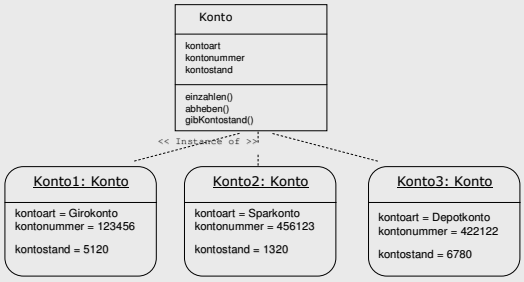
- Klasse: Notation in UML

Beispiel: Klasse Konto



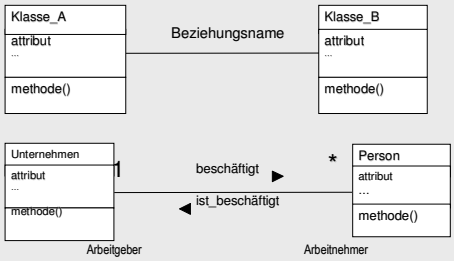
OOM-Einführung

- Eine Klasse – Viele Instanzen



OOM-Einführung

- Assoziationen (ungerichtet)



OOM-Einführung

- Assoziationen (gerichtet)

OOM-Einführung

- Klassen identifizieren

- Eine heuristische Vorgehensweise zur Identifikation von Klassen in Texten stellt *Abbotts Noun Approach* dar, der besagt, dass Substantive in einer Problemstellung Klassenkandidaten sind.
- Substantive, die allerdings keine zusammengesetzte, strukturierte Einheit repräsentieren, sondern lediglich einen einzelnen, atomaren Wert (Text, Zahl, Wahrheitswert o. ä.), sind Attributkandidaten.
- Sie werden dem Klassenkandidaten zugeordnet, in dessen Zusammenhang sie in der Problemstellung erwähnt werden (vgl. Oestereich et al. 1999, S. 250): „WENN das Hauptwort einen Wert annehmen kann, DANN handelt es sich (mit sehr hoher Wahrscheinlichkeit) um ein Attribut und nicht um eine Klasse."

OOM-Einführung

- Beispiel
  - Klassenkandidaten identifizieren

Eine Bank benötigt Software zur Abwicklung von Autokrediten. Dabei liegt folgendes Szenario zugrunde:

Besitzer eines oder mehrerer Autos können Personen oder Firmen sein. Alle Besitzer haben Namen und Adresse. Bei Personen wird zusätzlich das Geburtsdatum, bei Banken die Bankleitzahl benötigt. Personen können bei Firmen beschäftigt sein. Wichtiges Merkmal der Autos sind Modell und Baujahr. Ein Auto kann nur einen Besitzer haben. Zur Finanzierung eines Autos kann ein Besitzer einen Kredit (Kontonummer, Zinssatz, Kontostand) bei einer Bank aufnehmen. Kann ein Besitzer seinen Kreditforderungen nicht nachkommen, so hat die Bank das Recht, das Auto zu pfänden.

OOM-Einführung

- Beispiel
  - Klassenkandidaten identifizieren
  - Mögliche Attribute identifizieren

Eine Bank benötigt Software zur Abwicklung von Autokrediten. Dabei liegt folgendes Szenario zugrunde:

Besitzer eines oder mehrerer Autos können Personen oder Firmen sein. Alle Besitzer haben Namen und Adresse. Bei Personen wird zusätzlich das Geburtsdatum, bei Banken die Bankleitzahl benötigt. Personen können bei Firmen beschäftigt sein. Wichtiges Merkmal der Autos sind Modell und Baujahr. Ein Auto kann nur einen Besitzer haben. Zur Finanzierung eines Autos kann ein Besitzer einen Kredit (Kontonummer, Zinssatz, Kontostand) bei einer Bank aufnehmen. Kann ein Besitzer seinen Kreditforderungen nicht nachkommen, so hat die Bank das Recht, das Auto zu pfänden.

OOM-Einführung

- Beispiel
  - Klassenkandidaten identifizieren
  - Mögliche Attribute identifizieren
  - Mögliche Assoziationen identifizieren

Eine Bank benötigt Software zur Abwicklung von Autokrediten. Dabei liegt folgendes Szenario zugrunde:

Besitzer eines oder mehrerer Autos können Personen oder Firmen sein. Alle Besitzer haben Namen und Adresse. Bei Personen wird zusätzlich das Geburtsdatum, bei Banken die Bankleitzahl benötigt. Personen können bei Firmen beschäftigt sein. Wichtiges Merkmal der Autos sind Modell und Baujahr. Ein Auto kann nur einen Besitzer haben. Zur Finanzierung eines Autos kann ein Besitzer einen Kredit (Kontonummer, Zinssatz, Kontostand) bei einer Bank aufnehmen. Kann ein Besitzer seinen Kreditforderungen nicht nachkommen, so hat die Bank das Recht, das Auto zu pfänden.

OOM-Einführung

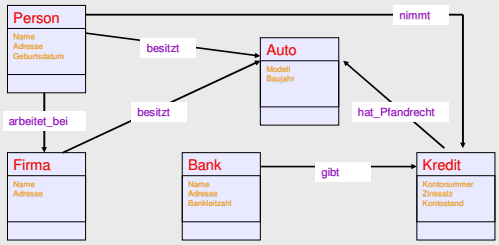
- Beispiel
  - Klassenkandidaten identifizieren
  - Mögliche Attribute identifizieren
  - Mögliche Assoziationen identifizieren

Eine Bank benötigt Software zur Abwicklung von Autokrediten. Dabei liegt folgendes Szenario zugrunde:

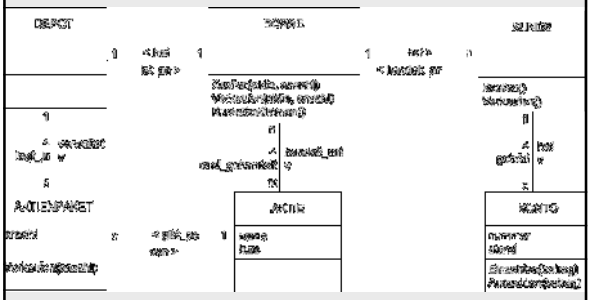
Besitzer eines oder mehrerer Autos können Personen oder Firmen sein. Alle Besitzer haben Namen und Adresse. Bei Personen wird zusätzlich das Geburtsdatum, bei Banken die Bankleitzahl benötigt. Personen können bei Firmen beschäftigt sein. Wichtiges Merkmal der Autos sind Modell und Baujahr. Ein Auto kann nur einen Besitzer haben. Zur Finanzierung eines Autos kann ein Besitzer einen Kredit (Kontonummer, Zinssatz, Kontostand) bei einer Bank aufnehmen. Kann ein Besitzer seinen Kreditforderungen nicht nachkommen, so hat die Bank das Recht, das Auto zu pfänden.

OOM-Einführung

- Beispiel
  - Mögliches statisches Modell

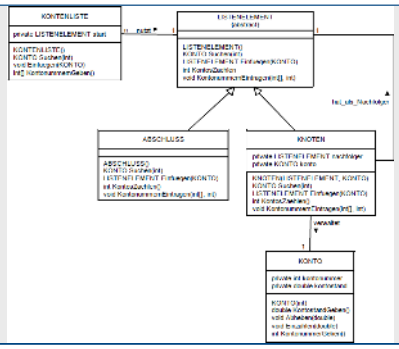


Klassendiagramm



Diagrammeditor Dia

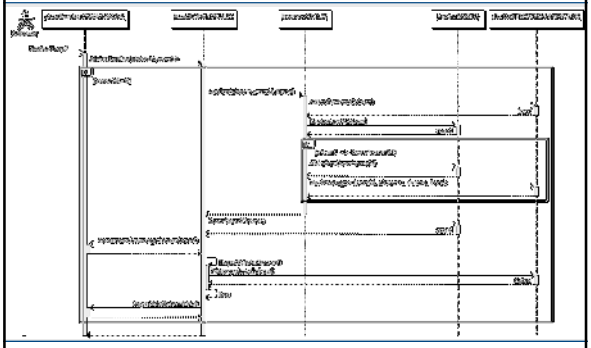
Erweitertes Klassendiagramm



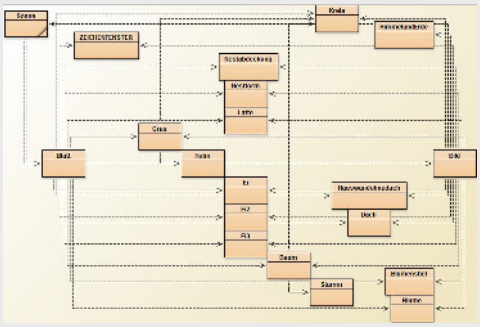
Klassenkarte

KONTO
int nummer double kontostand KUNDE besitzer
KONTO(int nummer, KUNDE besitzer) void Einzahlen(double betrag) void Auszahlen(double betrag) void KontostandGeben()

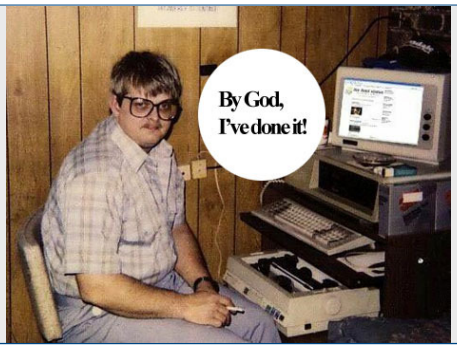
Sequenzdiagramm



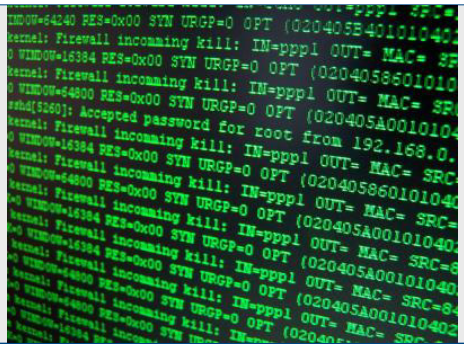
Teamwork notwendig



Softwareentwicklung



Softwareentwicklung



Softwareentwicklung

Ein verflixter Auftrag

Ziele bei der Softwareentwicklung

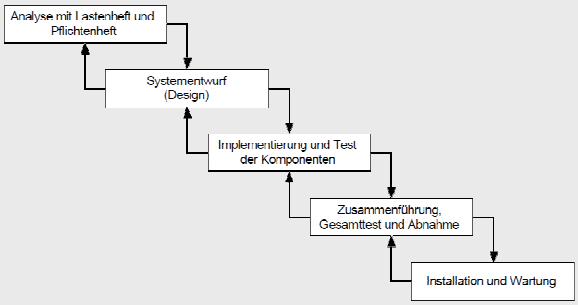
- Entwicklung qualitativ hochwertiger komplexer Software
- Berücksichtigung von Zeit und Budget
- Den wirklichen Bedürfnissen der Kunden entsprechend
- Unmöglich bei unstrukturiertem Vorgehen
  - Nur unvollständige Beschreibung der geforderten Leistungen
  - Fehlende Durchgängigkeit im Entwicklungsprozess
  - Viele unnötige Crash-Aktionen
  - Abstimmungsprobleme
  - Arbeiten auf Zuruf
  - Unzureichende entwicklungsbegleitende Dokumentation

Ziele bei der Softwareentwicklung

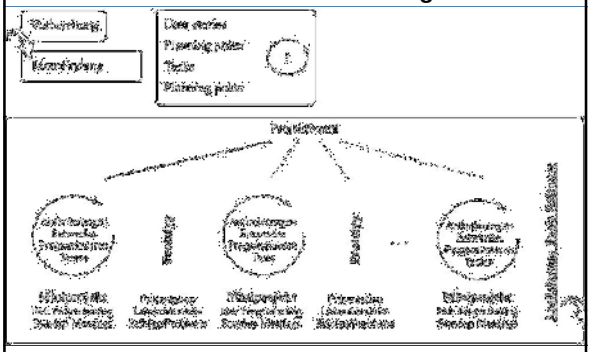
- Vermeidung von Fehlern
  - Ariane 5 Explosion
  - LH Airbus 320 Unglück bei Landung
  - Therac 25 Verstrahlung mehrerer Patienten
  - Scheitern der Mars Probe Mission
  - Rechnerabsturz der Berliner Feuerwehr
  - Scheitern des Patriot-Abwehrsystem bei Scud-Rakete
  - Explosion einer chemischen Fabrik

"Learn from the mistakes of others,...
you'll never live long enough to make them all yourself."

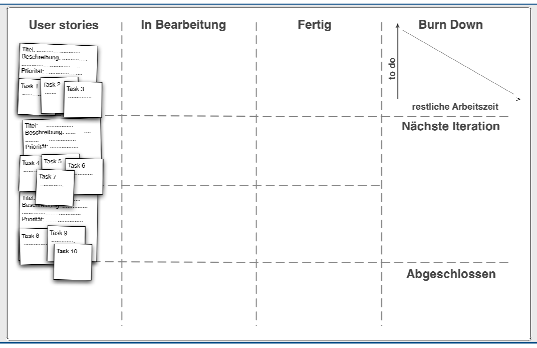
Wasserfallmodell



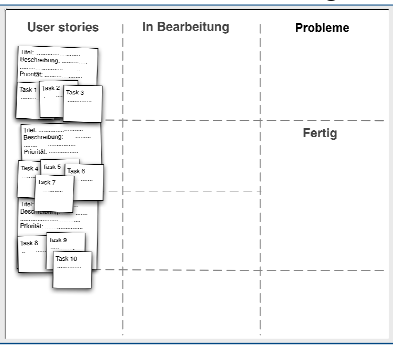
Agile Methoden der Softwareentwicklung



Agile Methoden der Softwareentwicklung



Agile Methoden der Softwareentwicklung



Agile Methoden der Softwareentwicklung

User Story

Titel: *Lebensenergie*

Beschreibung:

Wenn der Avatar mit einem giftigen Element in Berührung kommt, werden ihm Lebensenergiepunkte abgezogen.

Schätzung: Priorität: 30

Agiler Projektunterricht in der Schulinformatik

Wird auf einer Karteikarte festgehalten und besitzt Titel, Beschreibung sowie Platzhalter für Priorität und Schätzwert für Arbeitsaufwand.

1) Beschreibt eine Aktivität/Funktion, die das Programm anbieten soll.

2) Nimmt Sicht des Nutzers ein.

3) Ist kurz, enthält nicht mehr als drei Sätze.

4) Benutzt keine Fachausdrücke.

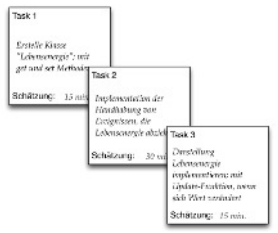
5) Legt keine Werkzeuge oder Technologien fest.

Die Priorität wird durch das Projektteam festgelegt.

Agiler Projektunterricht in der Schulinformatik

40

Tasks



Tasks sind elementare Aufgaben, die von einem Paar zu implementieren sind.
Tasks haben Titel, Beschreibung und Platzhalter für Schätzwert des Arbeitsaufwands.

1) Jede User Story wird in Tasks runtergebrochen.

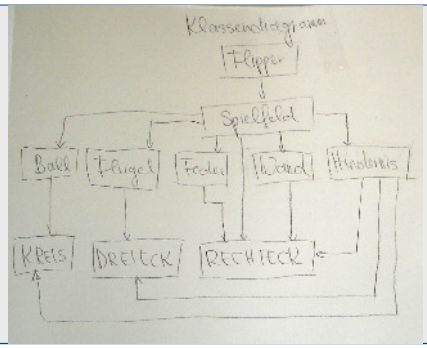
2) Sammlung von Tasks repräsentiert eine User Story.

3) Knapp verfasste Beschreibung, max. zwei Sätze.

4) Perspektive des Entwicklers.

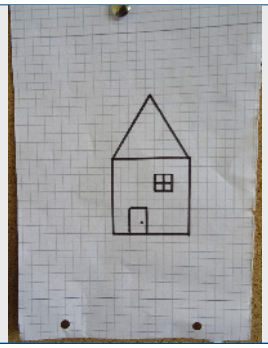
Dr.-Ing. Ulrich Kiesmüller
Kiesmueller@simon-marius-gymnasium.de

41



Dr.-Ing. Ulrich Kiesmüller
Kiesmueller@simon-marius-gymnasium.de

42

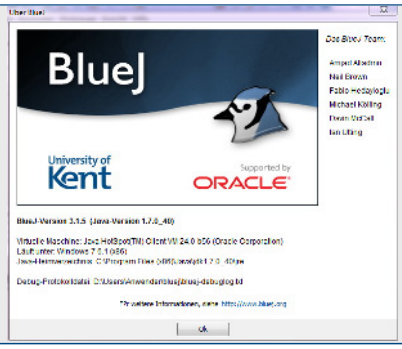


Dr.-Ing. Ulrich Kiesmüller
Kiesmueller@simon-marius-gymnasium.de

Ball Point Game

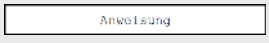
- Um einen Punkt zu bekommen, muss ein Tennisball von jedem Teammitglied einmal berührt worden sein.
- Der Ball muss jeweils „weiter geworfen“ werden (weiterreichen ist nicht erlaubt) – „the ball must have air time“.
- Der Ball darf nicht zur direkt benachbarten Person geworfen werden.
- Der Ball darf nicht zu Boden fallen und er muss zurück in den Sammelbehälter.

Programmierumgebung BlueJ



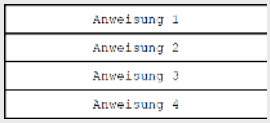
Algorithmische Grundstrukturen

- Anweisung




```
anweisung;
```

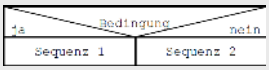
- Sequenz




```
anweisung1;
anweisung2;
anweisung3;
anweisung4;
```

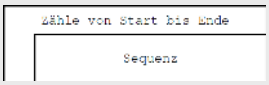
Algorithmische Grundstrukturen

- Bedingte Anweisung




```
if (bedingung)
{    sequenz1;      }
else
{    sequenz2;      }
```

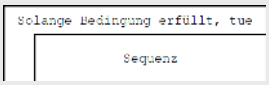
- Wiederholung (mit Zähler)




```
for (int zaehler = start; zaehler <= ende; zaehler = zaehler + 1)
{    sequenz;       }
```


Algorithmische Grundstrukturen

- Wiederholung (mit Bedingung)

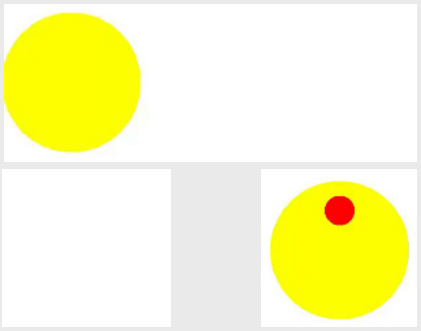



```
while (Bedingung)
{    sequenz;       }
```

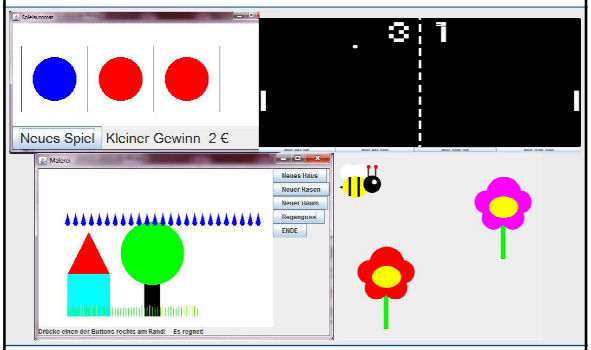

Kundenwünsche

- Kunde 1
  „Ich möchte, dass sich ein Ball quer über den Bildschirm bewegt.“
- Kunde 2
  „Ich möchte, dass sich ein Kreisring von links nach rechts über den Bildschirm bewegt.“
- Kunde 3
  „Ich möchte, dass sich ein Ball in einem Kreis auf dem Bildschirm bewegt.“

Kundenwünsche



Themenfindung



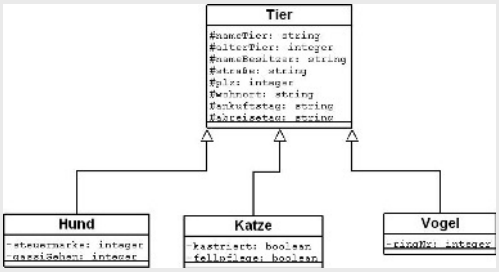
Vererbung
Beispiel *Tierpension*

Es soll die Verwaltung einer Tierpension modelliert werden. Diese nimmt Hunde, Katzen und Vögel auf. Von allen soll Name und Alter des Tiers sowie Name und Anschrift des Besitzers gespeichert werden. Außerdem muss gespeichert werden, wann das Tier zur Pflege gegeben wurde (Ankunftstag und Abreisetag). Hunde besitzen eine Steuermarkennummer und Vögel eine Nummer am Fußring. Für Hunde ist festzulegen, wie oft sie jeden Tag Gassi geführt werden müssen. Bei Katzen wird gespeichert, ob sie kastriert sind und ob das Fell regelmäßig gepflegt werden muss.

Modelliere diesen Zusammenhang in einem UML-Klassendiagramm

Vererbung
Beispiel *Tierpension*

Modelliere diesen Zusammenhang in einem UML-Klassendiagramm



Vererbung
Beispiel *Tierpension*

Implementiere die obige Tierpension in Java. Schreibe für die Klasse Tier *einen Konstruktor, der alle Parameter hat und verwende diesen in den anderen Klassen.*

```
public abstract class Tier
{
    protected String nameTier;
    protected int alterTier;
    protected String nameBesitzer;
    protected String strasse;
    protected int postleitzahl;
    protected String wohnort;
    protected String ankunftstag;
    protected String abreisetag;
```


Vererbung
Beispiel *Tierpension*

Implementiere die obige Tierpension in Java. Schreibe für die Klasse Tier *einen Konstruktor, der alle Parameter hat und verwende diesen in den anderen Klassen.*

```
public Tier(String nT, int aT, String nB, String st, int plz, String wo, String
ankunft, String abreise)
    {   nameTier = nT;
        alterTier = aT;
        nameBesitzer = nB;
        strasse = st;
        postleitzahl = plz;
        wohnort = wo;
        ankunftstag = ankunft;
        abreisetag = abreise;
    }
```

Vererbung
Beispiel *Tierpension*

Implementieren Sie die obige Tierpension in Java. Schreiben Sie für die Klasse Tier *einen Konstruktor, der alle Parameter hat und verwenden Sie diesen in den anderen Klassen.*

```
public class Hund extends Tier
{   private int steuermarke;
    private int gassiGehen;

    public Hund(String nT, int aT, String nB, String st, int plz, String wo,
String ankunft, String abreise, int steuer, int gassi)
    { super (nT, aT, nB, st, plz, wo, ankunft, abreise);
     steuermarke = steuer;
     gassiGehen = gassi;
    }
}
```


Vererbung
Beispiel *Tierpension*

Implementiere die obige Tierpension in Java. Schreibe für die Klasse Tier *einen Konstruktor, der alle Parameter hat und verwende diesen in den anderen Klassen.*

```
public class Katze extends Tier
{   private boolean kastriert;
    private boolean fellpflege;

    public Katze(String nT, int aT, String nB, String st, int plz, String wo,
              String ankunft, String abreise, boolean kastriert, boolean fell)
    { super (nT, aT, nB, st, plz, wo, ankunft, abreise);
        this.kastriert = kastriert;
        fellpflege = fell;
    }
}
```


Vererbung
Beispiel *Tierpension*

Implementiere die obige Tierpension in Java. Schreibe für die Klasse Tier *einen Konstruktor, der alle Parameter hat und verwende diesen in den anderen Klassen.*

```
public class Vogel extends Tier
{   private int ringNr;
    public Vogel(String nT, int aT, String nB, String st, int plz, String wo,
String ankunft, String abreise, int ring)
    { super (nT, aT, nB, st, plz, wo, ankunft, abreise);
     ringNr = ring;
    }
}
```

Vererbung
Beispiel *Tierpension*

Schreibe in der Klasse Tier *eine Methode* ausgabe(), *die alle Attribute ausgibt.*

```
public void ausgabe(){
        System.out.println("Name des Tiers: "+nameTier);
        System.out.println("Alter des Tiers: "+alterTier);
        System.out.println("Name des Besitzers: "+nameBesitzer);
        System.out.println("Anschrift: "+strasse+", "+postleitzahl+" "+wohnort);
        System.out.println("Ankunftstag: "+ankunftstag);
        System.out.println("Abreisetag: "+abreisetag);
}
```


Vererbung
Beispiel *Tierpension*

Passe diese Methode durch Überschreiben für die einzelnen Tierarten an.

```
public void ausgabe(){
    System.out.println("Name des Hundes: "+nameTier);
    System.out.println("Alter des Hundes: "+alterTier);
    System.out.println("Name des Besitzers: "+nameBesitzer);
    System.out.println("Anschrift: "+strasse+", "+postleitzahl+" "+wohnort);
    System.out.println("Nummer der Steuermarke: "+steuermarke);
    System.out.println("Muss täglich "+gassiGehen+" mal Gassi geführt werden");
    System.out.println("Ankunftstag: "+ankunftstag);
    System.out.println("Abreisetag: "+abreisetag);
}
```


Vererbung
Beispiel *Tierpension*

Passe diese Methode durch Überschreiben für die einzelnen Tierarten an.

```
public void ausgabe(){
    System.out.println("Name der Katze: "+nameTier);
    System.out.println("Alter der Katze: "+alterTier);
System.out.println("Name des Besitzers: "+nameBesitzer);
    System.out.println("Anschrift: "+strasse+", "+postleitzahl+" "+wohnort);
    System.out.println("Die Katze ist kastriert: "+kastriert);
    System.out.println("Das Fell muss regelmäßig gepflegt werden: "+fellpflege);
    System.out.println("Ankunftstag: "+ankunftstag);
    System.out.println("Abreisetag: "+abreisetag);
}
```

Vererbung
Beispiel *Tierpension*

Passe diese Methode durch Überschreiben für die einzelnen Tierarten an.

```
public void ausgabe(){
    System.out.println("Name des Vogels: "+nameTier);
    System.out.println("Alter des Vogels: "+alterTier);
    System.out.println("Name des Besitzers: "+nameBesitzer);
    System.out.println("Anschrift: "+strasse+", "+postleitzahl+" "+wohnort);
    System.out.println("Der Vogel trägt die Ringnummer: "+ringNr);
    System.out.println("Ankunftstag: "+ankunftstag);
    System.out.println("Abreisetag: "+abreisetag);
}
```


Vererbung
Überschreiben – Überladen

Enthalten eine *Unterklasse* und die dazugehörige *Oberklasse* eine Methode des gleichen Namens, so gilt:

		Rückgabetypen gleich	Rückgabetypen verschieden
Signaturen der Eingabeparameter	gleich	***überschreiben***	**VERBOTEN!!!**
	verschieden	***überladen***	nebeneinander existierende Methoden

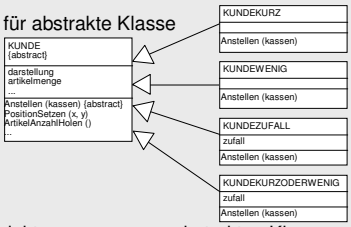
Überladen

Enthält eine *einzelne* Klasse zwei Methoden des gleichen Namens, so gilt:

		Rückgabetypen gleich	Rückgabetypen verschieden
Signaturen der Eingabeparameter	gleich	**VERBOTEN!!!**	
	verschieden	***überladen***	nebeneinander existierende Methoden

Abstrakte Klassen

- Beispiel für abstrakte Klasse



- keine Objekterzeugung von abstrakten Klassen möglich
- kein Methodenrumpf bei abstrakten Methoden existent
- abstrakte Methoden nur in abstrakten Klassen existent

Interface-Klassen

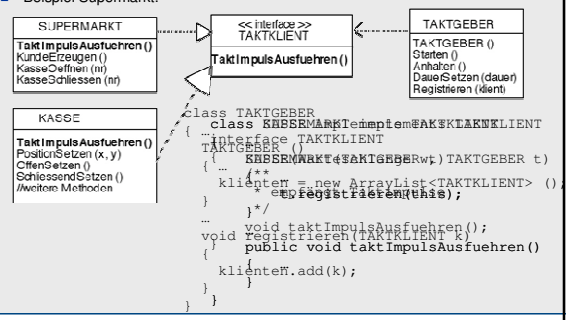
- Spezielle abstrakte Klassen
- Möglichst hohe Reduktion der Information, die eine Klasse über eine andere hat, für die Erreichung besserer Wartbarkeit und Flexibilität
- Beispiel Supermarkt:
  - Taktgeber bedient Objekt der Klasse SUPERMARKT mit Taktimpulsen
  - für Taktgeber unerheblich, welche Aufgabe Objekt besitzt
  - für Taktgeber ausreichend, dass SUPERMARKT-Objekt Methode `taktImpulsAusfuehren()` besitzt
  - auch Aufruf von Objekten der Klasse KASSE möglich
  - Ohne Interface-Konzept Bedienung von Objekten unterschiedlicher Klassen unmöglich

Interface-Klassen

- Beispiel Supermarkt:
  - Lösung mit Konzept der Sichten
  - Beschreibung von Methoden zur Kommunikation zwischen Objekten zweier Klassen ohne vollständig bekannte Funktionsweise
  - Sicht TAKTKLIENT beschreibt für Taktgeber Methode `taktImpulsAusfuehren()`
  - Klassen sichern Implementierung dieser Methode zu
  - Taktgeber stützt sich auf diese Sicht
  - andere Sicht der Klassen beschreibt eigentliche Aufgabe
  - Interfaces stellen Sichten in Java dar
  - Interface beschreibt alle für Sicht zur Verfügung stehende Methoden
  - keine Angabe von Attributen möglich
  - Schlüsselwort `implements` zur Implementierung von Interfaces
  - Methoden in Interfaces sind automatisch(!) `abstract`

Interface-Klassen

- Beispiel Supermarkt:




```
class TAKTGEBER
{ …
  TAKTGEBER ()
  { …
    klienten = new ArrayList<TAKTKLIENT> ();
  }
  …
  void registrieren(TAKTKLIENT k)
  {
    klienten.add(k);
  }
}
```

```
class SUPERMARKT implements TAKTKLIENT
{ …
  SUPERMARKT (TAKTGEBER t)
  { …
    /**
     * t.registrieren(this);
     */
    void taktImpulsAusfuehren();
    public void taktImpulsAusfuehren()
    {
    }
  }
```

```
interface TAKTKLIENT
```


Datenstruktur Feld

```
public class Zeichenblatt
{ private ZEICHENFENSTER hintergrund;
  private Baum baum1;
  private Halm[] rasen;                                   Deklaration

  public Zeichenblatt()
  { int farbe;
    hintergrund = new ZEICHENFENSTER("Malerei",800,500);
    baum1 = new Baum(200,280,30,90);
    rasen = new Halm[50];
    for (int i = 0; i < 50 ; i = i + 1)
    { rasen[i] = new Halm(20, baum1.gibXStartStamm()+i*5 - 100,
                 baum1.gibYStartStamm(), 2); }}             Initialisierung

    public void zeichneGrundstueck()
    { …
      for (int i = 0; i <50 ; i = i + 1)
        { hintergrund.fuelleDreieck(rasen[i].gibX1Halm(),
          rasen[i].gibY1Halm(),rasen[i].gibX2Halm(),
          rasen[i].gibY2Halm(),rasen[i].gibX3Halm(),
          rasen[i].gibY3Halm(),rasen[i].gibFarbnrHalm()); } }   Verwendung
```


Zufallszahlen

```
public class Zeichenblatt
{ private ZEICHENFENSTER hintergrund;
  private Baum baum1;
  private Halm[] rasen;

  public Zeichenblatt()
  { int farbe;
    hintergrund = new ZEICHENFENSTER("Malerei",800,500);
    baum1 = new Baum(200,280,30,90);
    rasen = new Halm[50];
    for (int i = 0; i < 50 ; i = i + 1)
    { int zufall = (int) (Math.random()*10);
      if (zufall >= 8)
      { farbe = 6; //gelber Halm }
      else
      { farbe = 2; //grüner Halm }
       rasen[i] = new Halm(20+zufall, baum1.gibXStartStamm()+i*5-100,
        baum1.gibYStartStamm(), farbe); }

   …
```

Ein- und Ausgabe von Texten

```
import java.io.*;

public class Testklasse
{ private Kartenstapel stapel, teilstapel1, teilstapel2, teilstapel3;
  private Kartenfeld feldGesamt;

  public Testklasse()
  { … }

  public void main()
  { …
    spos = 0;
    while (spos == 0)
    { System.out.print("In welcher Spalte - 1 bis 3 - befindet sich
      die von Ihnen gemerkte Karte? ");
      spos = eingabe();
      if (spos < 1 || spos > 3) spos = 0; }

static int eingabe ()
    { … } }
```


Ein- und Ausgabe von Texten

```
import java.io.*;

…

 static int eingabe ()
 {
  String zeile= "";
  int wert;
  BufferedReader in =
              new BufferedReader(new InputStreamReader(System.in) );
  try
  {
    zeile = in.readLine();
  }
  catch (IOException ioe) {}
  wert= Integer.parseInt(zeile);
  return wert;
 }
}
```


Graphische Benutzungsoberflächen

- Aufbau aus Komponenten
  - Schaltknopf (JButton)
  - Textanzeige (JLabel)
  - Texteingabe
  - Auswahlliste aufgebaut
- geeignete Klassen im Paket javax.swing
- Einfügen der Komponenten in Graphikfenster
  - Methode der Klasse ZEICHENFENSTER
  - `komponenteHinzufuegen(JComponent element, String position)`
  - Einfügen rechts neben oder unter der eigentlichen Zeichenfläche

Graphische Benutzungsoberflächen

```
import javax.swing.*;
public class SPIELAUTOMAT
{ private ZEICHENFENSTER hintergrund;
  private JButton schaltknopf;                          Deklaration
  …
  public SPIELAUTOMAT()
  { …
    schaltknopf = new JButton("Neues Spiel");           Initialisierung
    schaltknopf.addActionListener(new ActionListener()
       { public void actionPerformed(ActionEvent e)
           { spiele(); }                                Zuordnung von spiele() zu action performed()
       }                          );
    hintergrund.komponenteHinzufuegen(schaltknopf, "unten");
    …                                                   Einfügen des Buttons auf dem Bildschirm
  } … }
```


Graphische Benutzungsoberflächen

```
import javax.swing.*;
import java.awt.event.*;
public class GUIchef
{
   // Variablendeklaration
   private ZEICHENFENSTER hintergrund;
   private JButton knopf1;
   private JButton knopf2;
   private JLabel anzeige1;
   private JLabel anzeige2;
   private boolean komponente1;
   private boolean komponente2;
   private boolean programmlauf;
   private boolean ablauf;
```


Graphische Benutzungsoberflächen

```
// Konstruktor für einen GUI-Hintergrund mit fester Größe
public GUIchef()
{ // Variableninitialisierung
 hintergrund = new ZEICHENFENSTER("GUI",800,500);
 programmlauf = true;
 ablauf = false;
 komponente1 = false;
 komponente2 = false;
 hintergrund.komponenteHinzufuegen(knopf1, "rechts");
 hintergrund.komponenteHinzufuegen(knopf2, "rechts");
 anzeige1 = new JLabel(" ");
 hintergrund.komponenteHinzufuegen(anzeige1, "unten");
 anzeige1.setText("Drücke einen der Buttons rechts am Rand! ");
 anzeige2 = new JLabel(" ");
 hintergrund.komponenteHinzufuegen(anzeige2, "unten");
```

Graphische Benutzungsoberflächen

```
knopf1 = new JButton("Einzelaktion");
knopf2 = new JButton("Verlauf");
knopf1.addActionListener(new ActionListener()
{ public void actionPerformed(ActionEvent e)
 { machwas();
   schreibeStatus("Einzelaktion erledigt!");
   komponente1 = true;
 }} );
knopf2.addActionListener(new ActionListener()
{ public void actionPerformed(ActionEvent e)
 { ablauf = true; }
} );}
```


Graphische Benutzungsoberflächen

```
/** Methode zur Textausgabe in einem JLabel
* @param text Text (String), der angezeigt werden soll
*/
public void schreibeStatus(String text)
{ anzeige2.setText(text); }
/** Methode zur Durchführung einer Einzelaktion
* @param irgendwas
*/
public void machwas()
{ … // hier steht die zugehörige Sequenz }
/** Methode zur Durchführung eines kontinuierlichen Vorgangs
* @param irgendwas
*/
public void durchfuehren()
{ … // Sequenz des Ablaufs
 schreibeStatus("Jetzt passiert was!"); }
```

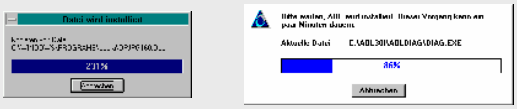

Graphische Benutzungsoberflächen

```
/** Hauptmethode - diese muss zuerst aufgerufen werden, bevor
* Programm in gewünschter Form mit Buttons gesteuert werden kann
* @param programmlauf solange TRUE, läuft Programm in
* Dauerschleife (boolean)
* @param ablauf löst mit TRUE (über Button 2 gesetzt) kontinuierliche
* Aktion aus - wird anschließend auf FALSE zurückgesetzt (boolean)
*/

public void main()
{ while (programmlauf)
  { if (ablauf)
    { durchfuehren();
      schreibeStatus("Aktion läuft!");
      ablauf = false;
} } }
```

“Perlen des GUI-Designs”

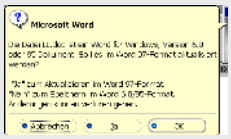
“Perlen des GUI-Designs”

“Perlen des GUI-Designs”

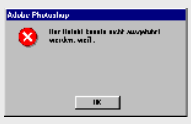


“Perlen des GUI-Designs”

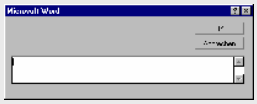
“Perlen des GUI-Designs”



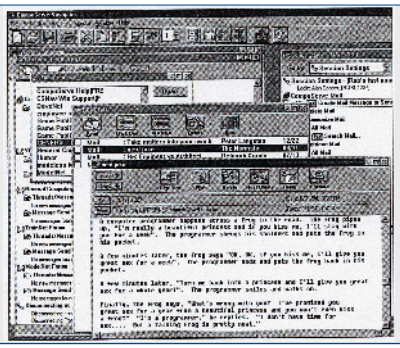
“Perlen des GUI-Designs”



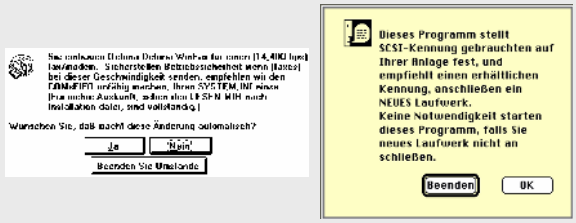
“Perlen des GUI-Designs”



“Perlen des GUI-Designs”

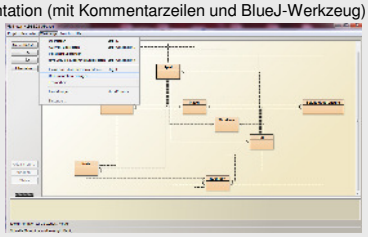


“Perlen des GUI-Designs”



Dokumentation

- „Keiner ist jemals arbeitslos!“
- Zusammenfügen von Teilen der einzelnen Programmierpaare
- Testen (mit Objektinspektor und Testklassen)
- Dokumentation (mit Kommentarzeilen und BlueJ-Werkzeug)



Dr.-Ing. Ulrich Kiesmüller
Kiesmueller@simon-marius-gymnasium.de

Impressionen früherer Durchläufe

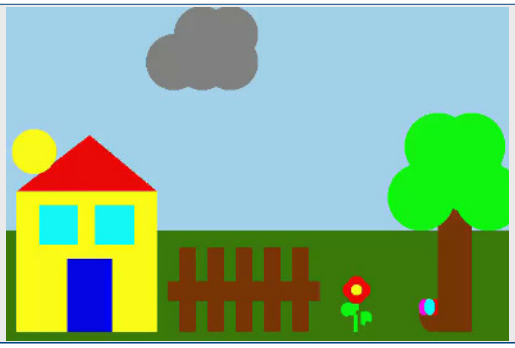
Dr.-Ing. Ulrich Kiesmüller
Kiesmueller@simon-marius-gymnasium.de

Objektorientierte Programmierung

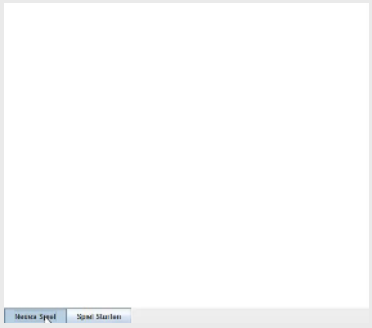
```
// bearbeitet von Maxi und Lukasz
// laufähig

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Image;
import java.applet.Applet;
import java.lang.Math;
import java.util.Random;
import java.io.*;
```

Dr.-Ing. Ulrich Kiesmüller
Kiesmueller@simon-marius-gymnasium.de

Ostersonne



Kreativität, selbstgesteuertes Lernen



Agile Methoden – ein Gewinn für alle