

Raumplanung mit Java (OOP)

Ziele:

1. Einen vorgegebenen Raumplaner zunächst zu verstehen (alle gegebenen Strukturen nachvollziehen und erläutern können)
2. Diesen Raumplaner aber auch nach gegebenen Vorstellungen erweitern können.

Kompetenzen, die am Ende erreicht sein müssen

Fülle folgenden Fragebogen zu Beginn des Themas und am Ende einmal aus:

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|--|---|---|---|
| Ich kenne den Unterschied zwischen Pixelgrafik und Vektorgrafik | | | |
| Ich kann mit grundlegenden Begriffen zu dem Thema OOP umgehen: | | | |
| 1. Modell | | | |
| 2. Klasse | | | |
| 3. Objekt | | | |
| 4. Methode | | | |
| 5. Instanzvariable (Eigenschaft, Attribut) | | | |
| 6. Konstruktor | | | |
| 7. Datentyp | | | |
| Ich beherrsche die Grundlagen objektorientierter Modellierung: | | | |
| 1. Klassenaufbau: Name, Instanzvariablen, Konstruktoren, Methoden | | | |
| 2. UML-Diagramme zu gegebenen Projekten aufstellen. | | | |
| 3. Vererbung sinnvoll in den Projektentwurf einbauen. | | | |
| 4. Abstrakte Klassen und Methoden sinnvoll einsetzen. | | | |
| 5. Kohäsion | | | |
| 6. Kopplung | | | |
| 7. Überladen | | | |
| 8. Überschreiben | | | |
| Ich kann Objekte mit dem „new“ Operator erzeugen | | | |
| Ich verstehe die Punktnotation und kann sie in Klassen und der main-Methode richtig einsetzen | | | |
| Ich kann Beziehungen zwischen Klassen korrekt herstellen und benennen | | | |
| 1. Vererbung | | | |
| 2. Aggregation | | | |
| 3. Komposition | | | |
| 4. Delegation | | | |
| 5. „Hat-ein“ und „ist-ein“ - Beziehungen | | | |
| Ich kann grundlegende Programmstrukturen in mein Projekt sinnvoll einbauen: | | | |
| 1. Variablen | | | |
| 2. Schleifen („while“ und „for“, erweiterte „for“) | | | |
| 3. Programmverzweigungen | | | |
| Ich kann weiterführende Programmstrukturen in mein Projekt einfügen und beherrsche den Umgang mit ihnen: | | | |
| 1. Arrays | | | |

| | | | |
|--|--|--|--|
| 2. ArrayList | | | |
| 3. Iteratoren | | | |
| 4. Zusatz Linked List | | | |
| Ich kenne Polymorphie als Grundsatz der OOP, kann sie in einem Projekt erkennen, ihre Vorteile aufführen und ein Projekt entsprechend dieser Vorteile implementieren | | | |
| Ich weiß was eine „main“-Methode ist und kann sie sinnvoll zum Erstellen eines geplanten Raumes einsetzen | | | |

Anleitung – Wie gehe ich mit einem Skript um?

- Im Folgenden wird genau beschrieben, wie du mit einem Skript arbeiten kannst und worauf du dabei achten musst. Du kannst dir dies zusätzlich – oder stattdessen – auch in einem Video erklären lassen:
- Schau dir dazu auf www.youtube.com/alexkueck11 das Video mit dem Namen „Aufbau eines Skriptes“ an.
- Um dir das Thema „Objektorientierte Programmierung (OOP)“ zu erarbeiten, kannst du grundsätzlich zwei verschiedene Wege gehen:
 - **Weg A** – Du bearbeitest **eigenständig die Projektaufgabe**. Dazu kannst du dir aus den Materialien raussuchen, was du gebrauchen kannst oder dir auch ganz eigene erklärende Materialien suchen.
 - **Weg B** – Du gehst die **einzelnen Module Schritt für Schritt** durch und erarbeitest dir so nach und nach die einzelnen Module (hier lernst du sozusagen Stück für Stück, wie die Projektaufgabe gelöst werden kann. Die Projektaufgabe wurde daher auch in „kleine Häppchen“ aufgeteilt und am Ende von jedem Modul wird ein Stück der Projektaufgabe gelöst).
- Mit beiden Wegen kannst du die geforderten Kompetenzen erwerben. Wenn du schon einiges über die funktionale Programmierung weißt und gerne an etwas knobelst, kannst du Weg A wählen. Behalte dabei aber immer auch im Auge, was du am Ende der Einheit können musst.
- Wenn du in diesem Bereich aber noch unsicher bist und das Thema lieber Schritt für Schritt erklärt bekommen möchtest, um es zu begreifen, wähle zunächst lieber Weg B. Auch hier löst du die Projektaufgabe, aber eben Schritt für Schritt und es wird dir vorgegeben, wie der Lösungsweg aussehen kann.
- Wenn du einen der beiden Wege eingeschlagen hast, bedeutet das allerdings nicht, dass du darauf festgelegt bist! Natürlich kannst du vom Projekt auch wieder auf die Module umsteigen, zum Beispiel, wenn du bei einer Sache nicht weiterkommst. Ebenso kannst du auch zur Projektaufgabe wechseln, wenn du nach ein paar Modulen merkst, dass du jetzt schon gut im Thema drin bist, und versuchen möchtest, eigenständig weiter zu knobeln.
- Lege dir eine Mappe an, in der du alle Lösungen und (Zwischen-) Ergebnisse zu den Aufgaben bzw. dem Projektvorhaben festhältst.

Wichtig: Du kannst deine Ergebnisse immer zwischendurch mit dem Lehrer abgleichen, um zu sehen, ob du auf dem richtigen Weg bist.

Gerade wenn du an dem Projekt arbeitest (aber auch wenn du mit dem Skript eigenverantwortlich durch das Thema gehst), ist es wichtig, dass du festhältst, wie du vorgegangen bist. Das tust du bitte in einem Blog oder einer Mappe. Dort hältst du fest, in welche Probleme du gelaufen bist und wie du sie gelöst hast – und vor allem, was du dadurch gelernt hast.

Wichtige Ergebnisse, Erkenntnisse, Merksätze oder Lösungsstrategien gehören hier ebenfalls hin. Am besten ist es, wenn du das in deinen eigenen Worten oder auch durch eine Skizze ausdrücken kannst. Selbst wenn das dann nicht ganz richtig ist, ist es besser so, als etwas Fertiges abzuschreiben. Der Lehrer kann so drauf schauen und dir helfen, wenn du etwas noch nicht vollständig verstanden hast.

Problemlösestrategien stehen bei diesem Projekt im Vordergrund nicht die Inhalte! Wenn du nicht genau weißt, was du aufschreiben sollst, lasse es dir vom Lehrer erläutern. Vorgefertigte Bögen, wo du Hilfestellung zu den Inhalten bekommst, kannst du beim Lehrer abholen.

Weg A – Bearbeitung der Projektaufgabe:

- Wie du die Projektaufgabe löst, bleibt dir und deiner Kreativität ganz allein überlassen. Du kannst selbst im Internet recherchieren und nachsehen, ob es dort Erklärungen oder Videos gibt, die dir weiterhelfen. Du kannst aber auch die in diesem Skript angegebenen Materialien weiter hinten verwenden – denn sie passen zu der Projektaufgabe.
- Ein Anhaltspunkt, um dich mit dem Thema auseinanderzusetzen, sind die Begriffe, die bei den zu erreichenden Kompetenzen am Anfang der Beschreibung angegeben sind. Damit könntest du z.B. anfangen. Wenn du die Begriffe verstehst und was für Ideen damit verknüpft sind, bist du meistens schon voll mit dem Thema beschäftigt. Vielleicht hast du ja auch schon für dich einen Weg gefunden, wie du an neue Projekte herangehst, dann solltest du diesen Weg hier auch gehen.
- Wichtig ist unbedingt, dass du im Auge behältst, was du am Ende der Einheit können musst. Die Projektaufgaben sind so formuliert, dass du am Ende alles, was gefordert ist, auch kannst. Aber es gibt ja viele Lösungswege und vielleicht kannst du am Ende das ein oder andere noch nicht so gut. Dann frage bitte nach zusätzlichen Übungsaufgaben zu dem jeweiligen Thema oder Begriff - bis du das Gefühl hast, es gut genug anwenden zu können.

Weg B – Bearbeitung der Module

- Gehe die Module Schritt für Schritt durch. Um die in jedem Modul angegebenen Aufgaben bearbeiten zu können, musst du vorher lernen, wie das geht. Nicht der Lehrer erklärt dir das, du entscheidest, auf welchem Weg du die Informationen haben möchtest und musst sie dann selbst so für dich zusammenstellen, dass du die Aufgaben lösen

kannst. In der Regel kannst du wählen zwischen einem erklärenden Text, Webseiten, auf denen du passende Informationen findest oder erklärenden Videos. Diese kannst du dir so oft ansehen, wie du es brauchst und magst. Wenn du dennoch weitere Erklärungen benötigst, notiere dir deine Fragen und wende dich damit an deinen Lehrer oder suche im Internet selbst nach weiteren erklärenden Texten oder Videos. Der Lehrer ist da, um dich in deinem Lernen zu unterstützen, aber du musst aktiv werden und nachfragen, wenn etwas unklar ist.

- Es ist wichtig, dass du alle neuen Begriffe, die du nicht kennst, klärst und richtig verstehst. Du musst sie in eigenen Worten beschreiben oder sie in einer Skizze darstellen können.
- Gehe bei jedem der Kapitel wie folgt vor:
 1. Zu Beginn jedes Kapitels findest du Verweise auf Materialien, die dir helfen sollen, das Thema zu verstehen, damit du später die dazugehörigen Aufgaben lösen kannst. Das können zum Beispiel sein:
 - erklärende Videos,
 - Infotexte (parallel zu den Videos),
 - Seiten in einem Schulbuch und
 - Texte in Zeitschriften oder auch
 - Internetseiten
 2. Eventuell brauchst du trotz der Materialien eine zusätzliche Erklärung, dann frage beim Lehrer nach. Eventuell haben andere ja auch diese Frage, dann kannst du auch einen kurzen Lehrvortrag dazu bekommen oder ein zusätzliches erklärendes Video.
 3. Die Videos und Dateien, auf die in diesem Skript verwiesen werden, findest du auf dem YouTube-Kanal: youtube.com/alexxkueck11
Die Videos beginnen alle mit „OOP-
 4. Falls du in den Materialien auf unbekannte Begriffe stößt, notiere diese. Das können auch einfache Worte sein. Versuche sie mithilfe der weiteren Materialien oder durch eigene Recherchen zu klären.
 5. Wenn du das Thema verstanden hast und alle darin enthaltenen Fachbegriffe in deinen eigenen Worten oder mittels einer Skizze erklären kannst, gehst du weiter zu den Aufgaben:
 - Die Aufgaben fordern dich auf, das Gelernte nun anzuwenden.
 - Gehe die Aufgabe, die im Skript angegeben sind der Reihe nach durch (die Aufgaben sind logisch aufeinander aufgebaut, daher der Reihe nach durchgehen).
 - Wenn du eine Aufgabe nicht bearbeiten kannst, gehe noch einmal über die Materialien oder schaue dir das erklärende Video erneut an. Vielleicht hast du einen neuen Begriff oder eine neue Idee noch nicht ganz verstanden – dann hole das nun nach.

- Wenn das nichts hilft, frage bei Mitschülern oder dem Lehrer nach. Lass dir aber nicht die ganze Aufgabe lösen. Wichtig ist, dass du eigenständig an einer Lösung arbeitest – auch wenn sie am Ende vielleicht nicht ganz richtig ist.
- Wenn du an deiner Lösung zweifelst, schaue in den Musterlösungen nach (falls vorhanden) oder frage den Lehrer, ob er sich deine Ergebnisse auch zwischendurch anschauen kann.

Falls du bei einer Aufgabe doch noch Schwierigkeiten hast, schaue dir noch einmal die erklärenden Materialien an.

Wichtig ist, dass du selbst an den Lösungen arbeitest und nicht andere das mache lässt.

6. Wenn Aufgaben, die mit einem **Zusatz** gekennzeichnet sind, brauchst du nicht bearbeiten. Diese Aufgaben sind schwieriger und gehen über das hinaus, was du als Minimum erreichen musst, um das Skript erfolgreich abzuschließen. **Für eine abschließende Note im Einser- oder Zweier-Bereich solltest du aber zumindest einige dieser Zusatzaufgaben bearbeiten.**
7. Es wird zwischendurch Tests geben, diese werden rechtzeitig angegeben. Auch welche Kompetenzen in den Tests angefragt werden.

Wichtig:

Wichtig ist, dass du bei der Arbeit mit dem Skript selbst aktiv wirst und deinen eigenen Lernprozess überwachst:

- Liege ich noch gut in der Zeit?
- Habe ich alles verstanden/begriffen oder brauche ich noch Hilfe oder zusätzliche Erklärungen?
- Wie kann ich Zusammenhänge besser verstehen/begreifen, die noch unklar sind?
- Wer kann mir bei der Bearbeitung der Aufgaben helfen?

Du musst selbst entscheiden, wo du dir weitere Informationen/Hilfen holen möchtest und von dir aus auf deinen Lehrer oder Mitschüler zugehen, um Fragen zu stellen, damit du die Themen und Begriffe besser verstehst und am Ende die geforderten Zielkompetenzen erreichst!

Es wird am Ende eine Klausur geben, die du bestehst, wenn du alle Aufgaben bearbeitet und verstanden/begriffen hast und die Kompetenzen erreicht hast.

Inhalt

| | |
|--|----|
| Raumplanung mit Java (OOP) | 1 |
| Anleitung – Wie gehe ich mit einem Skript um? | 2 |
| Weg A - Projekt Raumplanung | 7 |
| <i>Mit oder ohne Skript weitermachen</i> | 10 |
| <i>Begleitende Materialien</i> | 11 |
| Weg B – Skript | 14 |
| Modul 1 Raumplaner - Motivation des Projektes | 14 |
| Modul 2 Raumplaner: Einführung in die Arbeit mit dem Editor und grundlegende Begriffe der OOP | 16 |
| Kapitel 1: <i>Basics</i> | 16 |
| Modul 3 Raumplaner: Erweiterungen des Raumplaner Projektes | 22 |
| Kapitel 1 <i>Objekte zu Klassen erzeugen</i> | 22 |
| Kapitel 2 <i>Punktnotation</i> | 22 |
| Kapitel 3 <i>Main-Methode</i> | 23 |
| Kapitel 4 <i>Überladen und alternative Konstruktoren</i> | 23 |
| Kapitel 5 <i>Erweiterung der Funktionalität durch weitere Methoden</i> | 24 |
| Kapitel 6 <i>Komplexere Erweiterungen der Funktionalität</i> | 26 |
| Kapitel 7 <i>Eine neue Klasse einfach anlegen</i> | 29 |
| Kapitel 8 <i>Eine Figur auf der Leinwand zeichnen</i> | 29 |
| Modul 4 Raumplaner: Beziehungen zwischen Klassen I | 32 |
| Kapitel 1 <i>Vererbung</i> | 32 |
| Kapitel 2 <i>UML-Diagramme – „ist-ein“ - und „hat-ein“ - Beziehungen</i> | 33 |
| Kapitel 3 <i>Kohäsion und Kopplung</i> | 34 |
| Modul 5 Raumplaner: Beziehungen zwischen Klassen II | 37 |
| Kapitel 1 <i>Nutzerbeziehungen (hat-ein-Beziehungen) eine praktische Anwendung</i> | 37 |
| Kapitel 2 <i>Nutzerbeziehungen zwischen Klassen</i> | 38 |
| Kapitel 3 <i>Verschiedenartige Nutzer-Beziehungen (hat-ein-Beziehungen)</i> | 39 |
| Modul 6 Raumplaner: Sammlungsklassen | 41 |
| Kapitel 1 <i>Lösung mit dem Array</i> | 41 |
| Kapitel 2 <i>Lösung mit ArrayList</i> | 42 |
| Kapitel 3 <i>Iteratoren (nur erNi und Prüfline auf gruNi)</i> | 42 |
| Zusatz: LinkedList | 43 |
| Modul 7: Raumplaner (nur erNi): Polymorphie | 45 |
| Zusatz <i>Polymorphe Variablen</i> | 46 |
| Modul 8: Raumplaner (nur erNi): Entwurfsmuster (Design-Pattern) | 47 |

Weg A - Projekt Raumplanung

Vorausgabe

Bitte fülle die zugehörige Kompetenzübersicht vor dem Beginn des Skriptes einmal aus

Projektbeschreibung - Projektauftrag:

Möbelhaus GLOBAL.COM
Möbelweg 56,
D-22222 Hamburg

August 2023

An den
Kurs S1 Informatik
Pergamentweg 1
D-22117 Hamburg
Betr.: Auftrag 86645

Sehr geehrte Damen und Herren,

hiermit beauftragen wir Sie, einen Prototyp für den Einrichtungsplaner zu erstellen. Der erste Prototyp ist innerhalb von 2 Monaten fertigzustellen. Der Prototyp muss in der Lage sein, uns einen ersten Eindruck hinsichtlich des Einrichtungsplaners zu geben. Folgende Anforderungen haben wir an den Einrichtungsplaner:

- Der Kunde soll mit Hilfe des Einrichtungsplaners unsere Möbel am Bildschirm in ein Zimmer stellen können.
- Folgende Möbel sollen zur Verfügung stehen:
- Sofa 'Komfort', Maße 2 m x 1 m
- Sessel 'deLuxe', Maße 1 m x 1 m
- Sofatisch 'Admiral', Maße 1 m x 1 m, eckig
- Sofatisch 'Monte Carlo', Durchmesser 1 m, rund
- Bett 'Le France', 2 Personen, 2 m x 2 m
- Bett 'Le Petit France', 1 Person, 1 m x 2 m
- Stuhl 'Nomade', 50 cm x 50 cm
- Esstisch 'Wikinger', 1,2 m x 1,2 m
- Schrank 'Billy', 1 Teil, 1,5 m x 2 m
- Schrankwand 'Big Billy', mehrere (beliebig viele) Schränke vom Typ Billy werden zusammengestellt (wichtig: die Schränke können separat, also auch unabhängig von der Schrankwand existieren)

Alle Möbel sollen in den Farben schwarz, weiß, rot, gelb und blau dargestellt werden können.

Die Möbel sollen auf der Leinwand bewegt werden können und zwar vertikal, horizontal und diagonal. Die Verschiebung soll schrittweise auf der Leinwand beobachtet werden können.

Es soll möglich sein, sich Möbelgruppen zusammenzustellen, die man dann zusammen verschieben oder neu einfärben kann.

Der gesamte Preis der zusammengestellten Einrichtung soll errechnet werden können – dazu soll eine beobachtende Klasse angelegt werden, die registriert, wenn ein Objekt auf der Leinwand hinzugefügt wird und somit immer im Überblick hat, welche Objekte vorhanden sind (hier soll nach dem Design Pattern „Observer“ eine passende Klasse implementiert werden).

Das fertige Projekt sollte ausreichend dokumentiert sein und incl. eines UML-Diagramms abgegeben werden. Siehe dazu auch nächste Seite (Video erstellen).

Mit freundlichen Grüßen
(Vorstandsvorsitzender Wilfried Meybohm)

Für die Planung der Einrichtung einer Wohnung bzw. eines Hauses soll der oben im Auftrag beschriebene Raumplaner entwickelt werden.

Zu dem Raumplaner gibt es bereits erste einfache Ansätze eines Java-Projektes - hier: 01-Raumplaner-Anfang

Projekt-Aufgabe

Mache dich mit dem vorgegebenen Projekt vertraut, erweitere es dann entsprechend des Auftrages und berücksichtige dabei die wichtigen grundlegenden Konzepte der OOP, wie sie in der folgenden Kompetenzübersicht angegeben sind.

Die Klasse Leinwand kann größtenteils als eine Art „black box“ angesehen werden, die nicht im Einzelnen verstanden werden muss.

Erstelle ein kurzes, Video, in welchem du deine Überlegungen und Implementierungen dazu erläuterst. Füge dies deinem Portfolio hinzu.

Fülle folgenden Fragebogen am Ende des Projektes zu deiner Sicherheit aus:

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|---|---|---|---|
| Ich kenne den Unterschied zwischen Pixelgrafik und Vektorgrafik | | | |
| Ich kann mit grundlegenden Begriffen zu dem Thema OOP umgehen: | | | |
| 3. Modell | | | |
| 4. Klasse | | | |
| 8. Objekt | | | |
| 9. Methode | | | |
| 10. Instanzvariable (Eigenschaft, Attribut) | | | |
| 11. Konstruktor | | | |
| 12. Datentyp | | | |
| Ich beherrsche die Grundlagen objektorientierter Modellierung: | | | |
| 9. Klassenaufbau: Name, Instanzvariablen, Konstruktoren, Methoden | | | |
| 10. UML-Diagramme zu gegebenen Projekten aufstellen. | | | |
| 11. Vererbung sinnvoll in den Projektentwurf einbauen. | | | |
| 12. Abstrakte Klassen und Methoden sinnvoll einsetzen. | | | |
| 13. Kohäsion | | | |
| 14. Kopplung | | | |
| 15. Überladen | | | |
| 16. Überschreiben | | | |
| Ich kann Objekte mit dem „new“ Operator erzeugen | | | |
| Ich verstehe die Punktnotation und kann sie in Klassen und der main-Methode richtig einsetzen | | | |
| Ich kann Beziehungen zwischen Klassen korrekt herstellen und benennen | | | |
| 6. Vererbung | | | |
| 7. Aggregation | | | |
| 8. Komposition | | | |
| 9. Delegation | | | |
| 10. „Hat-ein“ und „ist-ein“ - Beziehungen | | | |
| Ich kann grundlegende Programmstrukturen in mein Projekt sinnvoll einbauen: | | | |

| | | | |
|--|--|--|--|
| 4. Variablen | | | |
| 5. Schleifen („while“ und „for“, erweiterte „for“) | | | |
| 6. Programmverzweigungen | | | |
| Ich kann weiterführende Programmstrukturen in mein Projekt einfügen und beherrsche den Umgang mit ihnen: | | | |
| 5. Arrays | | | |
| 6. ArrayList | | | |
| 7. Iteratoren | | | |
| 8. Zusatz Linked List | | | |
| Ich kenne Polymorphie als Grundsatz der OOP, kann sie in einem Projekt erkennen, ihre Vorteile aufführen und ein Projekt entsprechend dieser Vorteile implementieren | | | |
| Ich weiß was eine „main“-Methode ist und kann sie sinnvoll zum Erstellen eines geplanten Raumes einsetzen | | | |

Mit oder ohne Skript weitermachen

Vorgehen:

Am Anfang des Skriptes wird dir eine **Projektaufgabe** gestellt.

1. du wechselst auf das Projekt, indem du nur diese Aufgabe bearbeiten musst (behalte dabei im Auge, was du am Ende können musst - siehe "Kompetenzen")!
2. In diesem Skript wird das Projekt Stück für Stück erarbeitet und du bekommst detaillierte Erklärungen und einen Weg, wie du vorgehen muss, vorgegeben. Es gibt dir gezielt Hilfestellung mit erklärenden Texten und Videos, um die Projektaufgabe Stück für Stück zu lösen.

Wenn du unsicher bist, welchen Weg du einschlagen sollst, komme zu mir und wir klären das.

Wichtig: Du kannst deine Ergebnisse immer zwischendurch mit dem Lehrer abgleichen, um zu sehen, ob du auf dem richtigen Weg bist, aber du musst in einem Blog oder einer Mappe dokumentieren, in welche Probleme du gehabt und wie du sie gelöst hast – und vor allem, was du dadurch gelernt hast. Problemlösestrategien stehen bei diesem Projekt im Vordergrund nicht die Inhalte!

Begleitende Materialien

Achtung:

Beim Schauen der Videos unbedingt Notizen machen. Haltet die Aussagen in eigenen Worten fest, die euch wichtig erscheinen!

Falls nach einmaligem Gucken die Anwendung schwerfällt, nehmt eine zum Video passende Aufgabe und löst sie parallel zum erneuten Gucken des Videos – Schritt für Schritt nach den Angaben im Video!

Videos:

Die Videos sind alle auf dem YouTube-Kanal „**alexkueck11**“ zu finden und fangen fast alle mit “OOP – ” an.

- ➔ **Video: OOP - Editor BlueJ**
<http://youtu.be/AnFNjcbW-DM>
- ➔ **Video: OOP - Grundlegende Begriffe**
<http://youtu.be/0VWs57eGuDU>
- ➔ **Video: OOP - Grundlegende Begriffe I**
<https://www.youtube.com/watch?v=WM6LwfZbfcw>
- ➔ **Video: OOP - Grundlegende Begriffe II**
<https://www.youtube.com/watch?v=tP6vuV81uIE>
- ➔ **Video: Variablen**
<http://youtu.be/WAPbONf7QJI>
- ➔ **Video: OOP - Methoden I**
https://www.youtube.com/watch?v=_Orq4wawPhU
- ➔ **Video: OOP - Methoden II Methoden mit Rückgabewert**
<https://www.youtube.com/watch?v=1yrELWX9h1U>
- ➔ **Video: OOP - Rückgabewerte bei Methoden (alt)“**
<https://www.youtube.com/watch?v=dNn8aSKAtWA>
- ➔ **Video: OOP - Objekte erzeugen**
<https://www.youtube.com/watch?v=kkMm0ShpdRk>
- ➔ **Video: OOP - Punktnotation Einführung**
https://www.youtube.com/watch?v=CTHDNc_g7dA
(Spiele) <https://www.youtube.com/watch?v=-nSx1uon9WM&t=3s>
- ➔ **Video: OOP - Punktnotation 2**
<https://www.youtube.com/watch?v=gUKqBYOJaTk>
- ➔ **Video: OOP – Main Methode**
<https://www.youtube.com/watch?v=Ods960lCeoY>
- ➔ **Video: OOP – Überladen und mehrere Konstruktoren**
<https://www.youtube.com/watch?v=JLcpiCANLwE>
- ➔ **Video: OOP – Figuren zusammenstellen**
https://www.youtube.com/watch?v=s3JS_kGFJ1A
- ➔ **Video: OOP – Zeichnen auf die Leinwand**
<https://www.youtube.com/watch?v=KehJahS-zYg>
- ➔ **Video: OOP - Vererbung**
<https://www.youtube.com/watch?v=TXRS2Nj4FSs>
- ➔ **Video: OOP – Vererbung 2**
https://www.youtube.com/watch?v=qC_dj8d4ZiE
- ➔ **Video: OOP – Vererbung - abstrakte Klassen und Methoden**

- https://www.youtube.com/watch?v=zW3Y_ozwm60
- ➔ **Video: OOP - UML-Diagramme**
Neu: <https://www.youtube.com/watch?v=sL95ZOKMxgY>
Alt: https://www.youtube.com/watch?v=Lf04Z_HTIhQ
 - ➔ **Video: OOP - UML-Diagramme 2**
<https://www.youtube.com/watch?v=LNz6yeNnmDs>
 - ➔ **Video: OOP – Kohäsion und Kopplung**
<https://www.youtube.com/watch?v=abNSLLgYcvG>
 - ➔ **Video: OOP – Komposition - Aggregation**
(Spiele): <https://www.youtube.com/watch?v=9RDmZ8Gfnzo>
 - ➔ **Video: OOP - Delegation**
<https://www.youtube.com/watch?v=-NJ9JqP31NM>
 - ➔ **Video: OOP – Arrays einfach**
<https://www.youtube.com/watch?v=pHU9g4ZoVNw>
 - ➔ **Video: OOP – Arrays aus Objekten**
https://www.youtube.com/watch?v=ZfF-EGMBI_Q&t=17s
 - ➔ **Video: OOP – Arraylist**
https://www.youtube.com/watch?v=VEitbCh_mrA
 - ➔ **Video: OOP – Array und Arraylist im Vergleich**
<https://www.youtube.com/watch?v=btGhbfJcMi4>
 - ➔ **Video: OOP – Erweiterte for-Schleife**
https://www.youtube.com/watch?v=icnHmlb5_wg
(Spiele) <https://www.youtube.com/watch?v=6JqQAY-mnM8>
 - ➔ **Videos: OOP - Iteratoren**
<https://www.youtube.com/watch?v=pfn9nbPAkdG>
(Spiele) <https://www.youtube.com/watch?v=ZnyfvIGkZ1A>
 - ➔ **Videos: OOP - LinkedList**
<https://www.youtube.com/watch?v=ol2xQQo7jvg>
 - ➔ **Videos: OOP - Polymorphie**
<https://www.youtube.com/watch?v=NQ3Wdmzr7no>
(Spiele) https://www.youtube.com/watch?v=c6G2Dm_J-1c

Erklärende Texte:

Skript Raumplanung - Kompetenzübersicht
Skript Raumplanung -Text Grafikarten
Einschub I zum Skript RaumplanerEinfach
Einschub II zum Skript RaumplanerEinfach
Einschub III zum Skript RaumplanerEinfach
Skript Raumplanung -Text Editor Grundlagen
Skript Raumplanung -Text Objekte erzeugen
Skript Raumplanung -Text Punktnotation
Skript Raumplanung -Text Main-Methode
Skript Raumplanung -Text Überladen
Skript Raumplanung -Text Zeichnen von Figuren
Skript Raumplanung -Text auf die Leinwand zeichnen
Skript Raumplanung -Text Beziehungen I
Skript Raumplanung -Text Vererbung
Skript Raumplanung -Text Kohäsion und Kopplung
Skript Raumplanung -Text Schrankwand aus Schränken

Skript Raumplanung -Text Aggregation und Komposition
Skript Raumplanung -Text Delegation
Skript Raumplanung -Text Schrank-Arrays
Skript Raumplanung -Text Schrank-ArrayList
Skript Raumplanung -Text Iteratoren
Skript Raumplanung -Text Linked List
Skript Raumplanung -Text Polymorphe Variablen
Skript Raumplanung -Text Entwurfsmuster

Erklärendes Buch (beim Lehrer erhältlich):

Barnes / Kölling: Java lernen mit BlueJ

Weg B – Skript

Modul 1 Raumplaner - Motivation des Projektes

Warum Raumplanung als Projekt?

Grafiksysteme haben inzwischen in der Arbeitswelt einen wesentlichen Anteil in Anwendungen. Zum einen gibt es große CAD-System (AutoCAD, ...), zum anderen aber auch viele kleine Anwendungen im Bereich des Internetauftritts (Beispiele für Raumplaner finden sich mehrere auf Internetseiten von Firmen und Institutionen:

- <http://www.everyday-feng-shui.de/wohnung-planen.html>
- <https://www.schoener-wohnen.de/einrichten/40207-rtkl-schoener-wohnen-raumplaner>

Vor allem aber geht es auch darum programmierte Zusammenhänge sofort zu visualisieren und damit für den Anwender zugänglich zu machen - anstatt nur irgendwelche Listen oder Zahlen auszugeben. Ein echter Programmierer freut sich natürlich schon darüber, wenn ein Programm eine „4“ ausgeben soll und das auch tut – aber ich hoffe, dass du hiermit etwas mehr Spaß und Motivation hast.

Vorweg: Was ist der Unterschied zwischen einer Vektorgrafik und einer Pixelgrafik

Paint.net

Aufgabe 1.1

Öffne das Grafikprogramm „Paint.net“, erstelle ein Farbbild von mindestens einer Größe von 300 mal 300 Pixeln, Hintergrund weiß und mache dich mit der Oberfläche vertraut. Zeichne in dieses Bild ein nicht zu kleines, schwarzes Rechteck mit einer Linienbreite von 5 Pixeln. Zeichne weitere Rechtecke, Ellipsen und andere Figuren und verändere dabei die verschiedenen Einstellungen für die zu zeichnenden Objekte. Versuche, eines der gezeichneten Rechtecke zu verschieben, zu löschen oder seine Farbe zu ändern.

Inkscape

Aufgabe 1.2.

Öffne das Grafikprogramm Inkscape, erstelle ein Farbbild von mindestens einer Größe von 300 mal 300 Pixeln, Hintergrund weiß und mache dich mit der Oberfläche vertraut. Zeichne in dieses Bild ein nicht zu kleines, schwarzes Rechteck mit einer Linienbreite von 5 Pixeln. Zeichne weitere Rechtecke, Ellipsen und andere Figuren und verändere dabei die verschiedenen Einstellungen für die zu zeichnenden Objekte. Versuche eines der gezeichneten Rechtecke zu verschieben, zu löschen oder seine Farbe zu ändern.

Was soll das – warum soll ich zweimal dasselbe machen?

Es geht in dieser einführenden Übung nicht um unterschiedliches Handling bzw. um Unterschiede in der Oberfläche der Programme.

Die beiden Grafikprogramme unterscheiden sich in ihrer Grundkonzeption:

Paint.net ist ein **Pixel – Zeichenprogramm**,

Inkscape ist ein **Vektor – Zeichenprogramm**.

Wir interessieren uns in diesem Semester aber für die besonderen Eigenschaften von Vektor – Zeichenprogrammen. Wir werden dabei lernen, dass man heute sicher nicht mehr den Namen Vektor – Grafik wählen würde, sondern **Objektgrafik**.

Zusatz – Erklärung: Vektorgrafiken sind Objektgrafiken

Um das zu klären und die folgenden Aufgaben lösen zu können, kannst du aus den folgenden Quellen wählen:

➔ **Video: ---**

➔ **Eigenständige Recherche im Internet:**

<http://www.northern-concepts.de/glossar/Grundlagen/vektorgrafik>

➔ **Erklärender Text:**

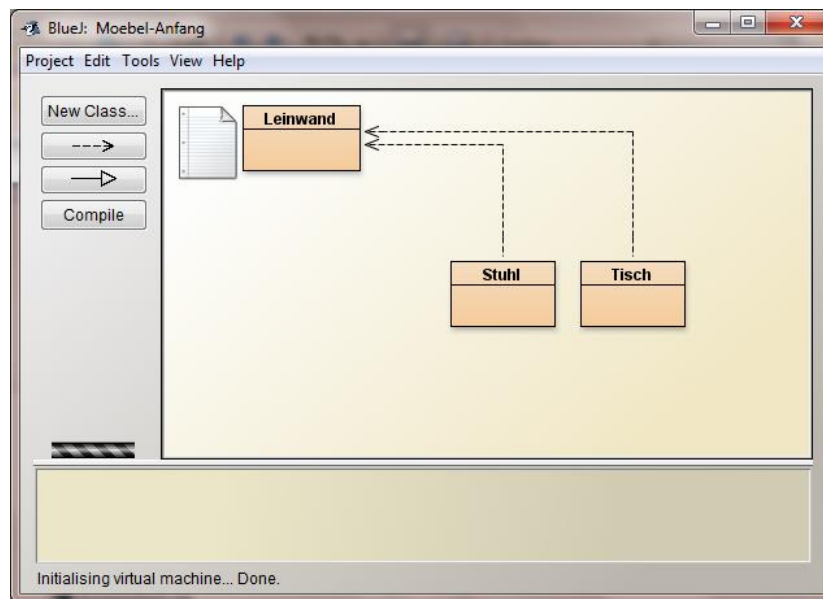
Skript Raumplanung -Text Grafikarten

Wir wollen uns auf den Weg zu einer Lösung machen und dabei die wesentlichen Methoden der Modellierung und Entwicklung eines Objektorientierten Softwaresystems auf der Basis der Programmiersprache JAVA kennen lernen.

Modul 2 Raumplaner: Einführung in die Arbeit mit dem Editor und grundlegende Begriffe der OOP

Kapitel 1 Basics:

Zunächst musst du dich mit dem Editor BlueJ abertraut machen.



Um das zu klären und die folgenden Aufgaben lösen zu können, kannst du aus den folgenden Quellen wählen:

➔ Videos:

„OOP – Editor BlueJ“ - <http://youtu.be/AnFNjcbW-DM>

„OOP – Grundlegende Begriffe“ - <http://youtu.be/0VWs57eGuDU>

„OOP – Grundlegende Begriffe I“ -

<https://www.youtube.com/watch?v=WM6LwfZbfcw>

„OOP – Grundlegende Begriffe II“ –

<https://www.youtube.com/watch?v=tP6vuV81uIE>

„Variablen“ - <http://youtu.be/WAPbONf7QJI>

„OOP – Methoden I“ - https://www.youtube.com/watch?v=_Orq4wawPhU

„OOP – Methoden II Methoden mit Rückgabewert“ -

<https://www.youtube.com/watch?v=1yrELWX9h1U>

„OOP – Rückgabewerte bei Methoden (alt)“ -

<https://www.youtube.com/watch?v=dNn8aSKAtWA>

➔ Erklärender Text:

Skript Raumplanung -Text Editor Grundlagen

Weiterführende Erklärungen: Einschub I zum Skript RaumplanerEinfach

Aufgabe 2.1.

1. Probiere alles aus. Spiele so lange mit dem BlueJ Projekt herum, bis du mit dem Editor und dem vorgegebenen Projekt vertraut bist und alle „Schaltflächen“ verstehst.
2. Kläre folgende Vokabeln, bis du eine konkrete eigene Vorstellung dazu hast, d.h. sie in eigenen Worten formulieren kannst und auch eine Skizze dazu anfertigen kannst (lies dazu den Text oder schaue die Videos als Hilfe):
 - Klasse
 - Objekt/ Instanz
 - Instanzvariable (Attribut)
 - Methode
 - Konstruktor
 - Parameter
 - Datentyp (speziell String, Integer, boolean)

Denke daran, alle Ergebnisse schriftlich in deinem Wordpress-Account zu sammeln.

3. **Lege deine Ausarbeitungen zur Durchsicht dem Lehrer vor.** Du musst hier bereits wichtige grundlegende Konzepte klären, daher ist es wichtig, dass du diese genau begreifst, da noch viel darauf aufbauen wird. Dies ist nicht als Kontrolle gedacht, sondern als Hilfestellung.

Eine weitere wichtige Vereinbarung in der OOP solltest du dir unbedingt merken, denn es erleichtert das Verstehen von vorgegebenem Programmcode:

- **Alle Variablen** fangen mit einem kleinen Buchstaben an: test, test1
- **Alle Methoden** fangen mit einem kleinen Buchstaben an, haben aber nach dem Namen immer ein Klammer-Paar (zwischen den klammern können Parameter stehen oder nicht): test(), test1(10, 20),...
- **Alle Klassennamen** fangen mit einem Großbuchstaben an: Tisch, Stuhl,...

Aufgabe 2.2:

1. Erstelle zwei Tische mit jeweils vier Stühlen darum.
2. Beobachte dabei, was bei der Arbeit an dieser Aufgabe auf unbefriedigende Funktionalität hindeutet.
3. Formuliere Anforderungen an die Funktionalität.
4. **Zusatzaufgabe für Fortgeschrittene:** Versuche die Klassen um diese Funktionalität zu erweitern.

Du hast nun einige der **wichtigsten grundlegenden Konzepte bzw. Vokabeln der OOP** kennen gelernt.

Da diese so wichtig sind, sollst du nun schauen, ob du sie soweit verstanden hast, dass du sie auch anwenden kannst.

Bearbeiten dazu den folgenden Aufgabenkomplex – wenn du merkst, dass du eine Aufgabe noch nicht tun kannst, schaue entweder die passenden Videos erneut an oder schaue in folgendem weiterführenden erklärenden Text nach Hilfestellung:

Einschub I zum Skript RaumplanerEinfach

Erledige diese Aufgaben unbedingt eigenständig, denn die hierbei erlernten Konzepte sind so grundlegend wichtig, dass du es nicht übergehen solltest oder Lösungen einfach abschreiben solltest.

Melde dich beim Lehrer, wenn du nicht klarkommst.

Aufgabe 2.3:

1. Objekte erzeugen:

Skizziere ein eigenes Konzept dazu, was im Rechner passiert, wenn du über den Editor einen neuen Stuhl mit **new Stuhl()** erzeugst.

Wenn sich eine Klasse ändern soll, musst du den zu ihr gehörenden Quelltext verändern. Dazu doppelklickst du in BlueJ auf das Klassensymbol und es erscheint ein Textfenster mit dem Quelltext der Klassendefinition. Diesen kannst du nun bearbeiten.

Nach dem Bearbeiten musst du das Ergebnis übersetzen (compile) und dann kannst du – hoffentlich – mit der neuen Variante arbeiten.

2. Lege eine neue Klasse Test an: `public class Test{`
In der Klasse sollen folgende Datenfelder/Variablen definiert werden:
 1. Ein Datenfeld **name** vom Typ **String**
 2. Ein Datenfeld vom Typ **int**, das **alter** heißt.
 3. Eine Variable mit dem Namen **kreditrahmen** vom Typ **int**
 4. Eine Variable vom Typ **boolean** mit dem Namen **testing**
3. Initialisiere alle unter Aufgabenteil 2 angegebenen Variablen mit einem passenden Wert.
4. Anstatt die Variablen direkt zu initialisieren, schreibe in der Klasse Test, in der alle Variablen als Instanzvariablen angelegt wurden, nun einen Konstruktor und initialisiere darin alle oben angegebenen Variablen.
5. Schreibe den Konstruktor aus Aufgabe 4 nun so um, dass ihm ein **boolean** Wert übergeben werden kann, welcher dann in der Variablen **testing** direkt als initialer Wert gespeichert wird.
6. Beschreibe in eigenen Worten, welche Vorteile sich ergeben, wenn man in einem Konstruktor die Möglichkeit vorsieht, durch die Angabe von Parametern die Instanzvariablen zu initialisieren.

7. Eine **neue Eigenschaft (=Attribut)** soll der Klasse **Tisch** hinzugefügt werden, d.h. eine neue Instanzvariable zu Tisch soll eingefügt werden (incl. passendem Datentyp zu der Variablen):

Anmerkung: die Begriffe Eigenschaft, Attribut, Instanzvariable sind hier austauschbar, denn sie bezeichnen alle das Gleiche!

Ein Tisch soll zusätzlich folgende Eigenschaft besitzen: er soll einen **Namen** haben (Variablenname: **name**). Dieser Name soll ein String sein, der den Typ oder auch Kategorie des Tisches bezeichnet (ähnlich wie bei einem großen Möbelhaus, wo auch alle Möbel Namen haben...)

8. Den Konstruktor von Tisch so abändern, dass die neue Instanzvariable dort initialisiert wird mit dem Wert „Helmoland“

9. Methoden allgemein schreiben:

1. Schreibe Namen und Rückgabewert folgender Methoden auf und gib jeweils an, was die Methode tut. Gib auch an, welchen Namen mögliche Parameter haben.

```
a) public void test1 (){  
    int z =3;  
    System.out.println(z);  
}
```

```
b) public void test2 (int z){  
    System.out.println(z);  
}
```

```
c) public int test3 (){  
    int z =3;  
    return z;  
}
```

2. Schreibe eigenständig eine Methode, die auf dem Bildschirm "Hallo" ausgibt.

3. Schreibe eigenständig eine Methode, die einen ganzzahligen Wert in der Variablen x übergeben bekommt. Dann diesen Wert verdoppelt und den neuen Wert am Ende auf dem Bildschirm ausgibt.

4. Schreibe eigenständig eine Methode, die 2 verschiedene Strings übergeben bekommt, diese dann verbindet und das Ergebnis auf dem Bildschirm ausgibt.

5. Schreibe eigenständig eine Methode, die zwei Integer Zahlen als Parameter übergeben bekommt, diese addiert und dann das Ergebnis der Addition wieder an den Aufrufer zurückliefert (siehe Methoden mit Rückgabewert im Einschub I).

10. Methoden neu schreiben – eine **neue Fähigkeit, die ein Tisch tun** kann:

Nun soll in dem bisherigen Raumplaner-Projekt die Klasse Tisch um eine neue Me-

thode erweitert werden. Und zwar soll der Name, den jeder Tisch nun hat über einen Rückgabewert angegeben werden können. Man soll auch die Möglichkeit haben, den Namen in einer weiteren Methode zu ändern. Der neue Name soll dieser Methode als Parameter übergeben werden können.

11. Schreibe den Konstruktor von Tisch so um, dass beim Erzeugen eines Tisches schon ein Name gegeben werden kann, der dann in der Instanzvariablen gespeichert wird.
12. Erzeuge mehrere Instanzen von Tisch und Stuhl und ändere einige der Eigenschaften einer jeden Instanz. Lasse dir dann über den Objektinspektor die Eigenschaften anzeigen. Kopiere diese und füge das Ganze mit Erklärungen in deinen Blog ein.
13. Denke dir selbst eine neue Eigenschaft für Tisch oder Stuhl aus, füge diese entsprechen in die Klasse ein. Überlege dir dann auch eine neue Tätigkeit für Stuhl oder Tisch und füge eine passende Methode ebenfalls in die Klasse ein.

Solltest du Probleme mit diesen Aufgaben haben oder/und nicht genau begreifen, was die einzelnen Begriffe bedeuten und wie sie zusammenarbeiten, so verlange dazu einen Lehrvortrag, schaue dir die am Beginn des Teils angegebenen Videos (Grundlegende Begriffe) an. Eine umfassende Erklärung zu den Vokabeln und noch einigen wichtigen weiteren Begriffen der OOP findest du hier:

- in dem begleitenden Buch „BlueJ“
- Einschub I zum Skript RaumplanerEinfach.

Oder du fragst nach einem Lehrvortrag, in dem du die Vokabeln mit dem Lehrer zusammen klären kannst.

Wichtig ist, diese Vokabeln vollständig zu begreifen, da du sonst im Folgenden große Schwierigkeiten haben wirst. Nimm dir also Zeit dafür!

Aufgabe 2.4.

Überprüfe zu deiner Sicherheit, ob du folgende Dinge nun kannst (fordere gegebenenfalls weitere Aufgaben zum Üben beim Lehrer an):

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|---|---|---|---|
| Ich kann mit dem Editor BlueJ umgehen. | | | |
| Ich weiß, wie man neue Objekte erzeugt und sich den Quellcode der Klassen zu den Objekten anschaut. | | | |
| Ich kann mit grundlegenden Begriffen zu dem Thema OOP umgehen: | | | |
| 1. Klasse | | | |
| 2. Objekt/Instanz | | | |
| 3. Methode | | | |
| 4. Instanzvariable (Eigenschaft, Attribut) | | | |
| 5. Parameter | | | |
| 6. Konstruktor | | | |
| 7. Datentyp | | | |
| Ich kenne die wichtigen Konventionen in Bezug auf die Schreibweise von Variablen und Methoden | | | |

Was solltest du nun können:

1. Du solltest mit dem Editor in BlueJ umgehen können
2. Du solltest wissen, wie man neue Objekte erzeugt und sich den Quellcode der Klassen zu den Objekten anschaut.
3. Du solltest folgende Begriffe verstanden haben, so dass du ganz spielerisch damit umgehen kannst und im Programmkontext sofort verstehst, was damit gemeint ist:
 - a. Klasse
 - b. Objekt/Instanz
 - c. Instanzvariable (Attribut, Eigenschaft)
 - d. Methode (mit oder ohne Rückgabewert)
 - e. Konstruktor (mit oder ohne Parameter)
 - f. Parameter
 - g. Datentyp (speziell String, Integer, boolean)
4. Du solltest die wichtigen Konventionen in Bezug auf die Schreibweisen von Variablen und Methoden kennen.

Nach Teil 2 wird ein Test geschrieben, um zu sehen, ob die Inhalte ausreichend verstanden wurden. Das ist wichtig, damit du die weiteren Teile erfolgreich bearbeiten kannst.

Modul 3 Raumplaner: Erweiterungen des Raumplaner Projektes

Kapitel 1 Objekte zu Klassen erzeugen

Bisher hast du den Bauplan verschiedener Objekte verändert und erweitert. Mit einem Rechtsklick auf die Klasse in BlueJ konntest du zu dem Bauplan einfach Objekte erzeugen (BlueJ nimmt dir hier den Programmcode ab!) und dann über ein Rechtsklick auf die Objekte, Methoden auswählen, um diese dann auf die jeweiligen Objekte anzuwenden.

Ohne BlueJ muss man diesen Programmcode selbst schreiben. Diese Hilfe von BlueJ ist nicht immer sinnvoll, denn es macht schon Sinn zu verstehen, wie man sich Objekte zu Klassen erzeugt und auf diese dann die Methoden anwendet – das kommt nun:

➔ **Video: OOP - Objekte erzeugen**

<https://www.youtube.com/watch?v=kkMm0ShpdRk>

➔ **Erklärender Text:**

[Skript Raumplanung -Text Objekte erzeugen](#)

Aufgabe 3.1

1. Lege eine neue Klasse Test an. In dieser Klasse soll es nur eine Methode geben: „public static void main(String[] args“ – diese Signatur musst du noch nicht verstehen, aber in dieser Methode sollst du vier Objekte anlegen – drei Stühle und einen Tisch. Führe anschließend die Methode in BlueJ aus und beobachte, was passiert. Stelle eine Vermutung an, wieso nichts zu sehen ist auf der Leinwand.
2. Lege eine Klasse Tischgruppe an. In dieser Klasse sollen ein Tisch und 4 Stühle als Instanzvariablen existieren (jede Klasse definiert ja einen eigenen Datentyp) und im Konstruktor initialisiert werden. Erzeuge nun ein Tischgruppenobjekt in BlueJ. Klicke dazu auf „inspizieren“ des entstandenen Objektes.
Fertige eine Skizze an, in der du deutlich machst, wie ein solches Tischgruppen-Objekt im Speicher aussehen würde.

Kapitel 2 Punktnotation

Anders als in den meisten Programmen, die du bisher im Informatikunterricht kennengelernt hast, gibt es nun mehrere Objekte (in diesem Fall auf der Leinwand zu sehen) – vorher bezog sich ein Programm immer auf genau ein Objekt – z.B. den Roboter. Wollte man den Roboter z.B. bewegen, so musste man einfach einen entsprechenden Befehl angeben. Es war immer klar, dass sich der Befehl auf eben diesen einen Roboter bezieht. Nun gibt es aber verschiedene Objekte auf der Leinwand. Möchte man eines davon bewegen oder in anderer Form den Zustand eines bestimmten Objektes ändern, so muss man angeben, auf welches man die jeweilige Methode (z.B. den Bewegungsbefehl) anwenden möchte.

In BlueJ geschieht das einfach durch einen Rechtsklick auf das Objekt und dann durch die Auswahl einer Methode. Aber eigentlich muss man das als Code im Programm tun! Der

Rechtsklick entspricht sozusagen dem „.“ im Code.

➔ **Video: OOP - Punktnotation Einführung**

https://www.youtube.com/watch?v=CTHDNc_g7dA

(Spiele) <https://www.youtube.com/watch?v=-nSx1uon9WM&t=3s>

➔ **Video: OOP - Punktnotation 2**

<https://www.youtube.com/watch?v=gUKqBYOJaTk>

→ **Erklärender Text:**

Skript Raumplanung -Text Punktnotation

Aufgabe 3.2

1. Würde man in Stuhl die „gibAktuelleFigur()“ Methode folgendermaßen umschreiben:

```
private Shape gibAktuelleFigur() {  
    GeneralPath stuhl = new GeneralPath();  
    moveTo(0,0);  
   .lineTo(breite,0);  
    .....  
}
```

so würde der Compiler eine Fehlermeldung liefern. Probiere es aus und erkläre die Fehlermeldung, die der Compiler liefert. Erläutere im Detail in eigenen Worten, wieso das so nicht funktioniert. Was müsste man tun, damit keine Fehlermeldung mehr kommt, wenn der Code dieser Methode aber so bleiben soll.

Kapitel 3 Main-Methode

Und wo macht man das bzw. wie wendet man das an?

Dazu gibt es die sogenannte Main Methode – so muss man sich einen Raum nicht immer wieder neu „zusammenklicken“ – man stellt sich einmal einen Raum zusammen und kann den auf einen Klick immer wieder zusammenstellen

→ **Video: OOP – Main Methode**

<https://www.youtube.com/watch?v=Ods960lCeoY>

→ **Erklärender Text:**

Skript Raumplanung -Text Main-Methode

Aufgabe 3.3

1. Erweitere die Main-Methode aus der Aufgabe 3.1. nun so, dass die Objekte über Punktnotation die Methode zeige() ausführen.
2. Im letzten Modul solltest du eine Tischgruppe zusammenstellen, indem du einen Tisch und vier Stühle erzeugst (mit Rechtsklick auf die jeweilige Klasse) und die Stühle dann so um den Tisch positionierst, dass es hübsch aussieht. Das sollst du nun erneut tun – aber indem du den nötigen Code in der Main Methode zusammenstellst, so dass er jederzeit wieder ausgerufen werden kann.

Kapitel 4 Überladen und alternative Konstruktoren

Es lässt sich für die vorliegende Anwendung feststellen: Die Klassen Stuhl und Tisch haben für den Benutzer unzureichende Funktionalität.

Erzeugt man nämlich mehrere Exemplare dieser Klasse und stellt sie dar, erhält man das unbefriedigende Ergebnis, dass alle – in ihren Eigenschaften gleichen – Exemplare sich überdecken, wie eines aussehen, also optisch nicht unterscheidbar sind, obwohl sie selbstverständlich nicht dieselben Objekte sind.

→ **Video: OOP – Überladen und mehrere Konstruktoren**

<https://www.youtube.com/watch?v=JLcpiCANLwE>

→ **Erklärender Text:**

Skript Raumplanung -Text Überladen

Aufgabe 3.4

Durch die Möglichkeit des Überladens brauchen wir den bisherigen Konstruktor nicht zu ersetzen, wir fügen der Klasse einfach **einen weiteren**, sogar mit **demselben Namen** hinzu. Zum vorgegebenen Konstruktor gehört folgender Programmtext:

```
public Stuhl() {  
    xPosition = 160;  
    yPosition = 80;  
    farbe = "blau";  
    orientierung = 0;  
    istSichtbar = false;  
    breite = 40;  
    tiefe = 40;  
}
```

Die Parameter, die wir einbringen wollen, müssen in der Klammer nach dem Konstruktor-namen auftauchen. Wollen wir also die Stühle mit verschiedenen Startpositionen erzeugen können, müssen für diese Startwerte in der Klammer Parameter angegeben werden, die dann den Instanzvariablen `xPosition` und `yPosition` zugewiesen werden.

1. Schreibe einen zweiten Konstruktor für die Klasse **Tisch und Stuhl**, so dass beide an unterschiedlichen Positionen erzeugt werden können.
2. Füge die neuen Konstruktoren jeweils in die Klasse **Tisch und Stuhl** ein und übersetze jede Klasse neu.
3. Lege nun mit BlueJ jeweils 2 Stühle und zwei Tische neu an. Was hat sich beim Aufruf mit `new...`, um die Objekte zu erzeugen geändert? Begründe warum das so ist.
4. Erläutere, was der Vorteil mehrerer Konstruktoren sein kann.

Wer eine Hilfestellung braucht:

Der Kopf unseres Konstruktors könnte dann so aussehen

(Namen der Parameter in den Klammern kann man frei wählen):

```
public Stuhl(int initX, int initY)
```

Kapitel 5 Erweiterung der Funktionalität durch weitere Methoden

Für weitere Methoden lassen sich viele Beispiele finden.

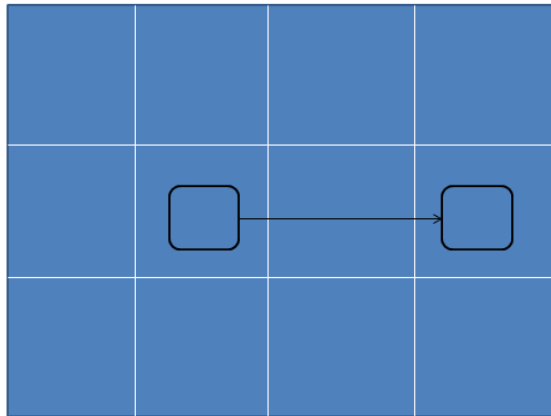
Wichtig ist an dieser Stelle, dass du lernst, mit BlueJ und den Grundelementen der Sprache JAVA umzugehen.

Aufgabe 3.5

1. **Vorübungen**, wenn du mit **Verzweigungen und Schleifen** in Java noch nicht vertraut bist: Lies **Einschub II zum Skript RaumplanerEinfach** und bearbeite die dort angegebenen Aufgaben.

Wenn du mit den grundlegenden Strukturen aus Aufgabe 1 vertraut bist und sie beliebig in bereits bestehende Klassen einbauen kannst, kannst du weiter machen!!!

2. Man könnte z.B. für den Raum ein Raster vorgeben, an dem sich alle Möbelstücke orientieren müssen. Ein Verschieben wäre dann nur um Vielfache dieses Rasters möglich.



Hier z.B. würde der Stuhl um 2 Raster-Werte in positive x-Richtung verschoben. Das Raster selbst soll auf der Leinwand nicht zu sehen sein. Es geht lediglich darum, dass ein Stuhl nicht mehr an beliebige Pixel-Positionen gesetzt werden kann, sondern nur an bestimmte Stellen. Wenn du Probleme hast, diese Aufgabe auszuführen, hilft dir folgendes Video weiter:

Video: OOP - Skript Raumplaner Aufgabe 3 2

https://www.youtube.com/watch?v=_Ywxsp_AdQs

Interessant wäre an dieser Stelle auch, das Überladen bei Methoden wie z.B. **bewegeHorizontal(...)** einzubauen, indem diese Methode ohne Parameter aufgerufen werden kann und dann um eine vorgegebene Weite weitersetzt.

1. Nimm alle nötigen Veränderungen vor, um ein solches Raster, wie oben beschrieben wurde „anzulegen“ (wie gesagt nicht auf der Leinwand zu sehen). Wähle eine geeignete Rastergröße.
2. Lasse es nun bei den verschiebe Methoden zu, dass ein Verschiebe-Parameter angegeben wird, aber diesmal nicht die Pixelzahl, sondern die Anzahl der im Raster weiterzuschiebenden Blöcke soll angegeben werden.
3. Lasse in diesem Raster auch diagonale Verschiebungen zu, indem du eine Methode **bewegeDiagonal(int x, int y)** schreibst, welche gleichzeitig um x Blöcke in x und um y Blöcke in y Richtung verschiebt.
4. Verhindere, dass man den Stuhl und Tisch durch Angabe zu großer Abstände aus der Leinwand verschiebt (über if-Abfragen zu den x und y Werten).
5. Ändere das Verschieben so ab, dass der Stuhl und der Tisch Schritt für Schritt über die Leinwand geschoben werden (wie bei einer Animation), bis zu dem eingegebenen Endpunkt.

Tipp: Dies erreichst du durch das Einfügen einer Schleife, die die Position pixelweise in x – und y- Richtung bis zum Endwert der neuen Position verändert.

Zusatz: Auch hierbei sollen die Möbel nicht über den Rand der Leinwand hinausgeschoben werden können.

6. Denke dir selbst zwei weitere Veränderungen aus und implementiere sie im Quelltext. Dokumentiere, warum du diese Veränderungen gewählt hast und in welche Probleme du bei der Umsetzung gelaufen bist – und wie du sie gelöst hast.
7. *Erstelle ein kurzes, Video, in welchem du deine Überlegungen und Implementierungen zu Teil 6. erläuterst. Füge dies deinem Portfolio hinzu.*

Gib das als kleines Projekt beim Lehrer ab.

Tipp

Wenn du etwas neu in einer Klasse einfügst oder änderst – und es dann nicht sofort funktioniert in einer eventuell etwas komplexeren Umgebung – kann es sinnvoll sein, sich eine Testklasse anzulegen. Hier wird dann nur der zu testende Code eingefügt. Hat man sichergestellt, dass der Code in dieser einfachen Testumgebung funktioniert, so kann er in komplexere Umgebungen eingebunden werden.

So kann man Fehler leichter eingrenzen!

Mit den grundlegenden Strukturen (Schleifen und Verzweigungen) von Java und dem grundlegenden Aufbau der Klasse Stuhl und Tisch solltest du nun vertraut sein. Falls du hier noch Zweifel hast, lasse dir vom Lehrer gern weitere Aufgaben geben, **denn es ist wichtig, dass du die Grundlagen gut beherrscht, bevor du weiter gehst.**

































Kapitel 6 Komplexere Erweiterungen der Funktionalität

Nun zu etwas größeren Veränderungen. Du sollst das Projekt um weitere Klassen von Möbelstücken erweitern. Dazu müssen wir uns auch damit beschäftigen, wie die Objekte auf die Leinwand gezeichnet werden. Wir werden und im Folgenden also die Klasse Stuhl und Tisch genauer anschauen.

Wichtig: Den Quelltext der Leinwand selbst musst du an dieser Stelle nicht verstehen, behandle sie einfach als „black box“, mit der du aber arbeiten kannst.

Aufgabe 3.6

1. Erstelle mindestens 2 weitere Möbelklassen mit einem Erscheinungsbild, das den Normzeichen entspricht: Schrank, Bett, Regal, Sessel, Sofa, usw. Wähle zunächst einfache Figuren.
Folgende Normzeichen gibt es:

| Wohnen | | | Bad | | |
|---|---------------------------------|----------------|---|--------------------------------------|---------------|
|  | Tisch | 80x120 |  | Badewanne | 180x80 |
|  | Runder Tisch | 100 Ø |  | Dusche | 80x80 |
|  | Schreibtisch | 60x180 |  | Spülklosett | 40x60 |
|  | Schreibtischstuhl | 40x40 |  | (mit Spülkasten) | 40x70 |
|  | Sofa | 80x150 |  | Handwaschbecken | 60x50 |
|  | Stuhl, Sessel | 45x45 60x60 |  | Waschmaschine | 60x60 |
|  | Hocker | 40 Ø |  | Heizkörper | 100x10 |
|  | Schränkelement (Hochschrank) | 60x110 | Kochen | | |
|  | Sidebord, Anrichte | 50x100 |  | Geschirrspülmaschine | 60x60 |
|  | Klavier | 60x140 |  | Elektroherd (mit Backofen) | 60x60 |
|  | Pflügel | 200x150 |  | Gasherd (mit Backofen) | 60x60 |
| Schlafen | | |  | Kühlschrank | 60x60 |
|  | Franz. Bett | 160x220 |  | Gefrierschrank | 60x60 |
|  | Doppelbett | 210x210 |  | Arbeitsfläche mit 2 Oberschränken | 30-60x100 |
|  | Nachtisch | 40x40 |  | Spülbecken mit Abtropffläche | 100x60 |
|  | Fernseher | 45x50 |  | Warmwasserbereiter (Elektro/Gas) | 50 Ø |
|  | Garderobe | 25x140 |  | Blumentische | 40x40 40 Ø |

Vorgehen für die folgenden Aufgaben:

- Du kannst entweder selbst versuchen herauszufinden, wie ein Tisch und ein Stuhl genau gezeichnet werden und wendest das auf zwei neue Symbole an und legst dazu entsprechende neue Klassen, deren Objekte dann auch auf der Leinwand erscheinen und beweglich sind, wie die bereits bestehenden Objekte.
- du bearbeitest folgenden Teilaufgaben, die dich dann etwas mehr begleiten, um zu dem Ziel zu gelangen.

Die Aufgabe in kleinen Schritten erledigen:

Ziel ist es zu verstehen, wie das Zeichnen von Formen mit Java grundsätzlich funktioniert. Wenn du das verstanden hast, ist es relativ einfach, zwei neue Möbelklassen zu entwerfen. Schauen wir uns dazu zunächst die Methode `gibAktuelleFigur()` von der Klasse `Stuhl` an, denn sie ist für das Aussehen des Stuhls verantwortlich:

Eine Erklärung dazu gibt es hier:

→ Video: OOP – Figuren zusammenstellen

https://www.youtube.com/watch?v=s3JS_kGFJ1A

→ Erklärender Text:

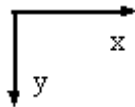
Skript Raumplanung -Text Zeichnen von Figuren

Aufgabe 3.7

- a) Suche über folgenden Link <http://docs.oracle.com/javase/6/docs/api/> alle in der Klasse `Stuhl` importierten Klassen und finde heraus, welche Methoden dadurch zur Verfügung stehen, wenn ein Objekt einer solchen Klasse angelegt wurde. Halte das in deinen schriftlichen Ausarbeitungen fest.
- b) Untersuche die Methoden, die die Klasse `GeneralPath` zur Verfügung stellt genauer und beschreibe, was die Methoden `moveTo(...)` und `lineTo(...)` genau tun. Mache dazu eine kleine Skizze und vollziehe so genau nach, was in der Methode `gibAktuelleFigur()` passiert (nur der erste Teil bis „Transformation“ soll hier betrachtet werden).

Beachte:

In der Informatik sehen die Koordinatensysteme in der Regel folgendermaßen aus (die Konvention gibt solche Koordinatensysteme vor – leider hält sich nicht jeder daran):



Es gibt also nur eine positive x und eine positive y- Achse. Der Ursprung liegt in der linken oberen Ecke.

Bisher ist zwar ein Objekt entstanden, jedoch sind die Eckpunkte bzw. der Umriss nur in dem Objekt selbst gespeichert – existieren sozusagen im Speicher, aber noch nicht auf der Leinwand, wir können sie daher noch nicht sehen.

Zwei Fragen bleiben daher:

- Wie erstelle ich die neue Klasse?
- Wie wird der Tisch auf die Leinwand gezeichnet?

Wenn Du beides selbst beantworten kannst, versuche den Rest der Aufgabe allein zu lösen, ansonsten hier ein paar weitere Hilfestellungen:

Kapitel 7 Eine neue Klasse einfach anlegen

Die neue Klasse erstellt man am einfachsten in zwei Schritten. Zunächst einmal fordert man BlueJ auf, eine neue Klasse zu erstellen. Im Dialogfenster gibt man den Namen, z.B. Schrank an, behält die Einstellungen bei und drückt dann OK. Im Klassenfenster taucht nun ein weiteres Rechteck als Symbol für die neue Klasse auf, das als Schrank beschriftet ist. Nun öffnet man die Programmtexte der Klassen Stuhl und Schrank, markiert bei Stuhl den ganzen Text, kopiert ihn, markiert in Schrank den gesamten Text und überschreibt ihn durch Einfügen mit dem von Stuhl. Bevor man etwas anderes macht, korrigiert man in dieser Klasse nun die Namen in der Klassendefinition und in den Konstruktoren auf den neuen Namen Schrank. (Sinnvollerweise korrigiert man auch die Namen in den Kommentaren.) Erst jetzt geht man daran, den Text der Methode `gibAktuelleFigur()` zu korrigieren.

Aufgabe 3.8

Lege für zwei neue Elemente deiner Wahl jeweils eine neue Klasse an.

Kapitel 8 Eine Figur auf der Leinwand zeichnen

Nun ist zwar eine Figur entstanden, doch existiert sie zunächst nur im Speicher, man kann auf der Leinwand noch nichts sehen. Wie das geht, findest du hier **beschrieben**:

➔ **Video: OOP – Zeichnen auf die Leinwand**

<https://www.youtube.com/watch?v=KehJahS-zYg>

➔ **Erklärender Text:**

Skript Raumplanung -Text auf die Leinwand zeichnen

Aufgabe 3.9

1. Stelle nun beide Klassen der von dir gewählten Möbelemente (entsprechend der Normzeichen in Aufgabe 3.3) fertig und lasse sie zur Probe auf der Leinwand zeichnen.
2. Überlege dir mindestens zwei weitere (nicht triviale) Erweiterungen für deine neuen Möbel, die du dann in Java umsetzt und in die Klassen einbaust. Begründe die Wahl deiner Erweiterungen und halte in deinem Portfolio fest, wie du vorgegangen bist, welche Probleme aufgetreten sind und wie du sie gelöst hast. Konntest du alle deiner anfänglichen Änderungs Ideen umsetzen?
3. **Fertige ein kurzes Video an**, in welchem du den Mitschülern erläuterst, welche Änderungen du wieso vorgenommen hast (was war deine Zielsetzung dabei, was ist der Mehrwert deiner Änderung für den Kunden) und welche Probleme du dabei wie gelöst hast.

Zusatzaufgabe: Shape ist eigentlich gar keine Klasse, sondern ein Interface. Informiere dich, was genau ein Interface ist und was Shape eigentlich genau in diesem Zusammenhang ist. Dabei wird auch der Begriff abstrakte Klasse auftauchen, der auch zu klären ist.

Zusatzaufgabe:**Nun dürfen die Figuren etwas komplexer werden:**

Einige der Möbelzeichnungen lassen sich nur aufwändig aus mehreren Teilfiguren zusammensetzen. Für solche Figuren ist die Klasse **GeneralPath** sehr hilfreich, da sie nicht nur elementare Zeichenmethoden wie **moveTo(...)** und **lineTo(...)** bereitstellt, sondern über die **append(...)** – Methode auch andere **Shapes** aufnehmen kann.

Als Beispiel verwenden wir die Badewanne



die man aus mehreren Teilfiguren zusammensetzen kann, nämlich

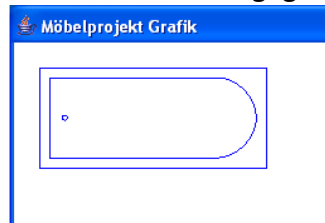
- einem vollständigen Rechteck,
- einem unvollständigen Rechteck, das wir aus drei Linien mit einem Halbkreis (Klasse **Arc**) daran zusammensetzen werden
- und einem kleinen Kreis für den Ablauf.

Für die zu verwendenden Maße nehmen wir relative Werte, also Angaben, die abhängig von den Werten **breite** und **tiefe** sind. Vorgegebene Maße aus der Normzeichnung sind 180 x 80, das definiert uns die Werte für das volle Rechteck. Für das innere, unvollständige Rechteck lassen wir auf jeder Seite gleich viel, z.B. ca. 10% der Tiefe weg, so dass dessen Position in der Figur nicht durch die linke obere Ecke (0;0), sondern durch (0,*breite; 0.1*tiefe) beschrieben wird, seine Tiefe durch 0.8*tiefe (oben und unten weniger!). Die Breite ist komplizierter zu bestimmen, da auf der rechten Seite auch der Radius des Halbkreises abgezogen werden muss.

Man muss bei einer solchen Figur sicher etwas probieren, bis man passende Werte findet.

Zusatz-Aufgabe 3.10

Lege eine Klasse **Badewanne** an und implementiere die Methode **gibAktuelleFigur()** entsprechend der oben angegebenen Musterzeichnung einer Badewanne.



Beachte, dass die Badewanne aus verschiedenen Teil-Shapes zusammengesetzt wird:

```
GeneralPath badewanne = new GeneralPath();
```

Ein Anfang könnte sein:

```
Rectangle2D umriss = new Rectangle2D.Double(0 , 0, breite, tiefe);
```

```
badewanne.append(umriss, false);
```

Beim **append** muss als zweiter Parameter jeweils **false** angegeben werden, damit zwischen den einzelnen Teilen der Figur keine Verbindungslinien gezeichnet werden.

Aufgabe 3.11.

Überprüfe zu deiner Sicherheit, ob du folgende Dinge nun kannst (fordere gegebenenfalls weitere Aufgaben zum Üben beim Lehrer an):

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|---|---|---|---|
| Ich kenne das Konzept des Überladens (mehrere Methoden oder Konstruktoren) | | | |
| Ich weiß, wie man Objekte einer Klasse erzeugt, so dass sie gleich zu Beginn unterschiedliche Zustände haben (Instanzvariablen) | | | |
| Ich kann Objekte mit dem „new“ Operator erzeugen | | | |
| Ich verstehe die Punktnotation und kann sie in Klassen und der main-Methode richtig einsetzen | | | |
| Ich kenne das Prinzip der Delegation und kann sie in Projekten sinnvoll einsetzen | | | |
| Ich kenne mich aus bzgl. der Java-Klassenbibliotheken und kann Teile davon sinnvoll in mein Projekt einbinden. | | | |
| Ich kann mit grundlegenden Programmstrukturen in Java umgehen | | | |
| 7. Schleifen („while“ und „for“) | | | |
| 8. Programmverzweigungen | | | |
| Ich verstehe, wie Möbelstücke auf die Leinwand gezeichnet werden | | | |

Was solltest du nun können?

1. Wissen, dass eine Klasse mehrere Konstruktoren haben kann, die sich aber in ihrer Parameterliste unterscheiden müssen (überladen von Methoden).
2. Wissen, wie man Objekte erzeugt, die gleich zu Beginn unterschiedliche Anfangszustände haben (Übergabeparameter an den Konstruktor haben).
3. Allgemein wissen, was überladen von Methoden heißt und wozu man es verwendet.
4. Das Prinzip der Delegation kennen.
5. Die Java-Klassenbibliothek kennen und sie verwenden können, um Methoden anderer Klassen einsetzen zu können.
6. Beliebige Änderungen in bestehenden Möbelklassen ausführen können (incl. Schleifen und Verzweigungen einbauen können).
7. Eine neue Möbelklasse anlegen können und dabei verstehen, wie die Möbelstücke auf der Leinwand gezeichnet werden.

Modul 4 Raumplaner: Beziehungen zwischen Klassen I

Kapitel 1 Vererbung

Einstiegsaufgabe

1. Verändere die Methode `bewegeHorizontal(..)` in jedem Möbelobjekt so, dass immer die zehnfache Schrittweite weiterbewegt wird für jeden angegebenen Zahlwert (so wie bei dem Raster in Teil 3, nur für alle neuen Möbelstücke auch).
2. Beschreibe, was bei der Erledigung der Aufgabe lästig ist. Was könnte in dem Projekt also verbessert werden?

Vererbung ist ein ganz grundlegendes aber auch sehr wichtiges Prinzip der OOP.

Eine Erklärung dazu gibt es hier:

➔ **Video: OOP - Vererbung**

<https://www.youtube.com/watch?v=TXRS2Nj4FSs>

➔ **Video: OOP – Vererbung 2**

https://www.youtube.com/watch?v=qC_dj8d4ZiE

➔ **Video: OOP – Vererbung - abstrakte Klassen und Methoden**

https://www.youtube.com/watch?v=zW3Y_ozwm60

➔ **Erklärender Text:**

Skript Raumplanung -Text Vererbung

Beachte:

1. Die Methodensuche fängt immer in der Klasse des gerade betrachteten Objektes an und arbeitet sich dann von Oberklasse zu Oberklasse nach oben hin vor, bis sie eine Methode gefunden hat, die passt. Findet man auf dem Vererbungsweg keine passende Methode (also sowohl in der Klasse, in der sie aufgerufen wurde, als auch in einer der Oberklassen), so gibt es eine Fehlermeldung.
2. Durch die Vererbung wird zwischen Klassen eine sogenannte „**ist ein Beziehung**“, aufgebaut. Ein `Stuhl` ist ein `Moebel`.
3. Eine Oberklasse, die als „abstract“ gekennzeichnet ist, muss **mindestens eine** abstrakte Methode haben – d.h. eine Methode, die nicht in dieser Klasse selbst implementiert wird, dafür aber durch ihre Kennzeichnung als „abstract“ **erzwingt**, dass jede Unterklasse sie implementiert (siehe Kapitel Probleme...)t.
Bsp.: `public abstract void test();` <-ohne geschweifte Klammer, nur die Signatur und dann ein Semikolon

Aufgabe 4.1.

1. Ändere selbst das Anfangsprojekt (oder dein Projekt, soweit du es bis jetzt geschrieben hast), indem du eine **abstrakte** Oberklasse `Moebel` anlegst und dort alles wie beschrieben hineinpackst. Ändere `Tisch` und `Stuhl` und deine eigenen Möbelklassen entsprechend ab. Überlege genau, was du mit der Methode „`gibAktuelleFigur()`“

machst – entscheide dich für eine der zwei Möglichkeiten.

2. Halte schriftlich fest, warum diese Methode zu Problemen führt, wenn man `Moebe1` als Oberklasse einführt.

Erstelle ein kurzes, Video, in welchem du deine Überlegungen und Implementierungen dazu erläuterst. Füge dies deinem Portfolio hinzu.

Kapitel 2 UML Diagramme – „ist-ein“ - und „hat-ein“ - Beziehungen

➔ Video: OOP - UML-Diagramme

Neu: <https://www.youtube.com/watch?v=sL95ZOKMxgY>

Alt: https://www.youtube.com/watch?v=Lf04Z_HTIhQ

➔ Video: OOP - UML-Diagramme 2

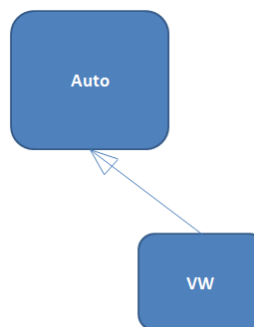
<https://www.youtube.com/watch?v=LNz6yeNnmDs>

➔ Erklärender Text:

Skript Raumplanung -Text Beziehungen I

Aufgabe 4.2

1. Durch das Einfügen der Oberklasse `Moebe1` hat sich die Darstellung der Klassen und der Beziehungen zwischen den Klassen in BlueJ geändert. Beschreibe, was für Veränderungen du beobachtet hast.
2. Gib für jede der beiden Beziehungen („ist-ein“ und „hat-ein“) drei anschauliche Beispiele in der Form „ein VW ist ein Auto“ und skizziere dazu das passende UML Diagramm:



Beachte: Die Nutzerbeziehung (z.B. hat-ein-Beziehung) muss in den Instanzvariablen der Klasse auftauchen, die andere Klassen nutzen will. Oben bei Mensch z.B. die Nase – sie ist in den Instanzvariablen definiert. Was später in den Methoden „genutzt“ wird, wird nicht dargestellt.

Aufgabe 4.3

1. Finde den Unterschied zwischen den beiden Beziehungstypen im Programmtext und beschreibe ihn. Halte das in deinem Portfolio schriftlich und grafisch fest.
2. Stelle das Möbel-Projekt Beispiel in einem UML-Diagramm dar (also mit allen Attributen und Methoden entsprechend aufgeführt).
3. Kennzeichne alle „hat-ein“- und „ist-ein“- Beziehungen
4. *Erstelle ein kurzes, Video, in welchem du deine Überlegungen dazu erläuterst.*

Tipp:

Wenn du fit genug bist in UML zeichnen, dann kannst du das auch ein Programm für dich machen lassen (aber bitte erst verwenden, wenn du verstanden hast, wie es geht): [EssModel.exe](#)

Kapitel 3 Kohäsion und Kopplung

Mit diesen beiden Begriffen werden Eigenschaften von Klassenentwürfen beschrieben, bei denen man die Qualität von Entwürfen beschreibt.

→ Video: OOP – Kohäsion und Kopplung

<https://www.youtube.com/watch?v=abNSLLgYcvg>

→ Erklärender Text:

Skript Raumplanung -Text Kohäsion und Kopplung

Kopplung zwischen Klassen

Es ist immer gut, wenn die einzelnen Klassen möglichst wenig gekoppelt sind, d.h. möglichst unabhängig voneinander sind, sie also nichts Gemeinsames haben – auch keinen gemeinsamen Code. Wenn ich dann in einer Klasse etwas ändere, dann nur in dieser einen Klasse, ich muss die anderen Klassen nicht ändern – was dann sehr überschaubar ist!

Konzept lose Kopplung:

Der Begriff beschreibt den **Grad der Abhängigkeit** zwischen Klassen. Wir streben eine möglichst lose Kopplung an – also ein System in dem die Klassen möglichst unabhängig voneinander definiert sind.

Kohäsion innerhalb einer Methode

Kohäsion in der Physik z.B. beschreibt, wie sehr etwas „zusammenhängt“. In der OOP hat es eine ähnliche Bedeutung z.B. bzgl. Methoden. Beinhaltet eine Methode mehrere Aufgaben, so könnte man die Methode noch leicht aufteilen, um die Methoden einzeln erledigen zu lassen durch jeweils eine Methode – d.h. es liegt in der zusammengesetzten Methode eine geringe Kohäsion vor, da es leicht ist, sie aufzuspalten. Wird aber nur eine „atomare“ Handlung ausgeführt, so kann man die Methode nicht weiter aufteilen, d.h. die Kohäsion ist sehr hoch.

Konzept hohe Kohäsion:

Der Begriff beschreibt, wie gut eine Programmeinheit (z.B. eine Methode) **eine logische Aufgabe** oder Einheit abbildet. In einem System mit hoher Kohäsion ist jede Programmeinheit verantwortlich für **genau eine** wohldefinierte Aufgabe.

Nun ein wenig Anwendung dazu:

Aufgabe 4.4. (nur erNi)

Die Methode `gibAktuelleFigur()` der konkreten Möbelklassen verstößt gegen das Konzept der Kohäsion, da sie zwei Aufgaben erledigt:

- den Shape zusammensetzen.
- die Transformation an eine bestimmte Stelle im Koordinatensystem.

Teile die Methode in zwei einzelne Methoden auf, so dass das Konzept der Kohäsion gegeben ist.

Dazu ein paar Tipps (wenn du es wünschst):

Die Methode `gibAktuelleFigur()` sollte das gleiche Endergebnis liefern wie vorher, da es in Bezug auf die Leinwand verwendet wird. Sie könnte die beiden Aufgaben Shape-Erzeugen und Transformieren vereinen:

```
protected Shape gibAktuelleFigur() {
    Shape shape = gibShape();
    shape = transformiere(shape, xPosition, yPosition, orientierung);

    return shape;
}
```

Da sie nun für alle Objekte gleich ist, kann sie in die Möbelklasse geschrieben werden. Ebenso die Methode `transformiere(shape, x, y, orientierung)`, die nur den Teil der Transformation übernehmen soll.

Es bleibt eine für jede Möbelklasse individuelle Methode `gibShape()`, die den Shape erzeugen soll und auch als Shape zurückliefert, selbst wenn es ein `GeneralPath` ist, wie bei Stuhl.

Aufgabe 4.5.

Überprüfe zu deiner Sicherheit, ob du folgende Dinge nun kannst (fordere gegebenenfalls weitere Aufgaben zum Üben beim Lehrer an):

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|---|---|---|---|
| Ich beherrsche die Grundlagen objektorientierter Modellierung: | | | |
| 1. UML-Diagramme zu gegebenen Projekten aufstellen. | | | |
| 2. Vererbung sinnvoll in den Projektentwurf einbauen. | | | |
| 3. Kohäsion | | | |
| 4. Kopplung | | | |
| Ich kenne und verstehe den Begriff „protected“ | | | |
| Ich verstehe, was es bedeutet, wenn man sagt „die Methodensuche beginnt immer von unten bzw. ausgehen von der Klasse, von der ein Objekt abgeleitet wurde“. | | | |
| Ich begreife die Auswirkungen, die diese Aussage hat. | | | |
| Ich weiß, was abstrakte Klassen und Methoden sind | | | |
| Ich kann abstrakte Klassen und Methoden in meinen Projektentwurf sinnvoll einsetzen. | | | |

Was solltest du nun können:

1. Wissen was Vererbung ist und wozu man sie verwendet (wann man sie sinnvoll einsetzt).
2. Wissen, was eine abstrakte Klasse ist und wozu man sie verwendet. Speziell den Vorteil kennen, dass man in einer abstrakten Klasse abstrakte Methoden definieren kann, wo nur der Methodenkopf angegeben werden muss, aber nicht die konkrete Implementierung (die kann in einer der Unterklassen erst vorgenommen werden).
3. „protected“ als Sichtbarkeit von Variablen kennen.
4. Verstehen, was es bedeutet, dass die Methodensuche immer „von unten“ losgeht.
5. UML-Diagramme mit **ist-ein** und **hat-ein** Beziehungen für beliebige Projekte zeichnen können.
6. **Zusatz:** Das Prinzip der Kopplung und Kohäsion kennen und es sinnvoll in dem Entwurf und der Erweiterung von Projekten einsetzen können.

Nach Teil 4 wird ein Test geschrieben, um zu sehen, ob die Inhalte ausreichend verstanden wurden. Das ist wichtig, damit du die weiteren Teile erfolgreich bearbeiten kannst.

Modul 5 Raumplaner: Beziehungen zwischen Klassen II

Kapitel 1 Nutzerbeziehungen (hat-ein-Beziehungen) eine praktische Anwendung

Schrank und Schrankwand

Wir wollen in unser Raumplanersystem eine Schrankwand mit aufnehmen, die aus mehreren gleichen Schrankelementen zusammenzustellen ist. Dazu sollte man zunächst einmal ein einzelnes Schrankelement zeichnen können, so dass wir uns nun eine Klasse Schrank konstruieren:



Wir verwenden wieder die abstrakte Klasse `Moebel`, so dass vom Text eigentlich nur die übliche Methode `gibAktuelleFigur()` interessant ist:

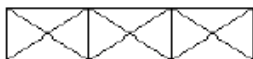
Aufgabe 5.1.

Lege eine Klasse Schrank an, die oben beschriebenen Schrank (zunächst als einteiligen Schrank) konstruiert. Der Schrank soll dabei aus einem Rechteck und zwei diagonalen Linien zusammengesetzt werden (über `append`). Der Anfang könnte folgendermaßen aussehen:

```
GeneralPath schrank = new GeneralPath();
Shape rahmen = new Rectangle2D.Double(0, 0, breite, tiefe);
...
schrank.append(rahmen, false);
...
```

Beachte, dass zunächst ein Schrank-Objekt vom Typ `GeneralPath` angelegt werden muss, da darüber dann über die Methode `append(..)` die einzelnen Zeichnungen (Rechteck und Linien) zusammengefügt werden können.

Nun lässt sich auf einfache Weise eine Schrankwand aus drei solchen Schränken zusammensetzen.



Aufgabe 5.2.

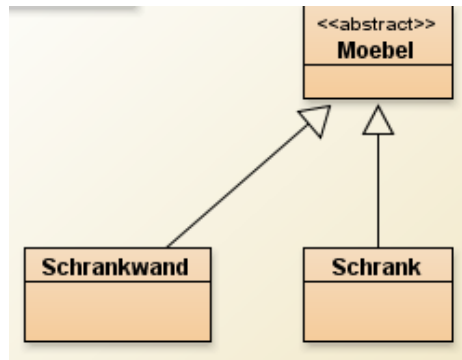
Lege eine neue Klasse Schrankwand an, indem du die Klasse Schrank so umschreibst, dass daraus eine Schrankwand aus den drei Schrank-Elementen entsteht. Dazu musst du nur die Methode `gibAktuelleFigur()` so umschreiben, dass über die Methode „`append`“ eben drei der Schränke aneinander gehängt werden.

Beachte, dass bei jedem neuen Schrankelement der Anfangspunkt des Rechtecks und der Diagonalen um die Breite des vorhergehenden Schrankelementes verschoben werden muss.

Beachte auch, dass die `breite` der Schrankwand das Dreifache der Breite eines Schrankelementes hat, damit das Drehzentrum genau in der Mitte der Schrankwand – also im mittleren Element – liegt.

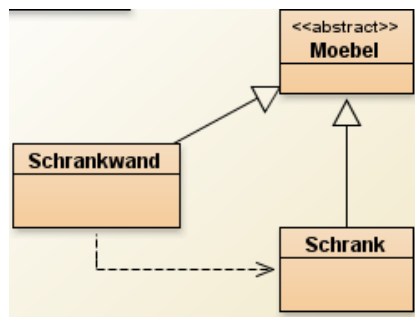
Wieder ist durch Codeduplizierung die Klasse Schrankwand erzeugt worden und nicht aus eigenem Code.

Statt Codeduplizierung durchzuführen gilt es, sich vorher über die Beziehungen zwischen den Klassen Gedanken zu machen. Im Klassendiagramm der derzeitigen Lösung gibt es aber keine Beziehung zwischen den beiden (Schrank und Schrankwand), sondern nur eine erbt – Beziehung zu **Moebel**:



Kapitel 2 Nutzerbeziehungen zwischen Klassen

Die beiden Klassen haben ganz offensichtlich eine Beziehung, man kann nämlich die Klasse Schrank nutzen, um eine Schrankwand zu erzeugen. Grundsätzlich tritt also eine Nutzerbeziehung auf (eine Schrankwand „**hat-einen**“ Schrank). Der Gedanke ist: **Wozu muss die Schrankwand wissen, wie ein einzelner Schrank aussieht. Das kann Schrank übernehmen.**



Die Lösung bereitet aber mehr Schwierigkeiten, als man zunächst vermutet.

Du kannst entweder eigenständig versuchen, die folgende Aufgabe zu lösen oder du schaust im folgendem erklärenden Text, welcher dich schrittweise zur Lösung führt:

➔ **Erklärender Text:**

Skript Raumplanung -Text Schrankwand aus Schränken

Aufgabe 5.3*

Schreibe die **Klasse Schrankwand** so um, dass sie das Erstellen der Schrankwand durch das Hinzufügen mehrerer Schränke durchführt, aber ohne selbst Einblick zu haben, wie genau die Schränke erstellt werden. Es sollen also Schränke als „Instanzvariablen“ – entsprechend einer „**hat-ein**“-Beziehung initialisiert werden:

```

public class Schrankwand extends Moebel{
    Schrank schr1;
    Schrank schr2;
    Schrank schr3;
    ...
  }
  
```

Tipps:

1. Benutze die Methode `append` von `GeneralPath`, um Schränke zusammenzufügen.
2. Achte auf die Rückgabetypen – die müssen stimmen. Die Methode `append` braucht ein `Shape`-Objekt, ein `Schrank` an sich ist aber erst Mal kein solches Objekt (erinnere dich an die zwei Darstellungen eines `Möbel`-Objektes – als Speicher und als Shape über `gibAktuelleFigur()`)

Kapitel 3 Verschiedenartige Nutzer-Beziehungen (hat-ein-Beziehungen)**→ Video: OOP - Vererbung**

<https://www.youtube.com/watch?v=TXRS2Nj4FSs>

→ Video: OOP – Vererbung 2

https://www.youtube.com/watch?v=qC_dj8d4ZiE

→ Video: OOP – Vererbung - abstrakte Klassen und Methoden

https://www.youtube.com/watch?v=zW3Y_ozwm60

→ Video: OOP - UML-Diagramme

Neu: <https://www.youtube.com/watch?v=sL95ZOKMxgY>

Alt: https://www.youtube.com/watch?v=Lf04Z_HtIhQ

→ Video: OOP - UML-Diagramme 2

<https://www.youtube.com/watch?v=LNz6yeNnmDs>

Assoziationen

Beziehungen zwischen Objekten sind ein wichtiges Thema bei der OO-Modellierung einer gewissen Realität. Wir wollen hier speziell den Typ Assoziation anschauen (auch **Nutzerbeziehung** oder „hat-ein-Beziehung“ genannt).

Hier sollen zwei wichtige Beziehungstypen (Spezialfälle der Assoziation) genauer betrachtet werden: **Aggregation und Komposition**


→ Video: OOP – Komposition - Aggregation

(Spiele): <https://www.youtube.com/watch?v=9RDmZ8Gfnzo>

→ Erklärender Text:

Skript Raumplanung -Text Aggregation und Komposition

Aufgabe 5.7.

1. Wenn du dein bisheriges Projekt betrachtest – welche Objekte sind Kompositionen, welche Aggregationen – welche gar nichts von beiden?
2. Wenn es bisher keine solche Objekte in deinem Projekt gibt, überlege dir eine mögliche Zusammenstellung, die eine Aggregation oder Komposition sein könnte.
3. Erkläre in eigenen Worten, was eine Aggregation und was eine Komposition ist.
4. Untersuche die folgenden Beispiele daraufhin, ob sie Aggregationen oder Kompositionen enthalten:
 - Eine Sitzgruppe, bestehend aus einem quadratischen Tisch und vier Stühlen.
 - Eine Duschwanne, bestehend aus drei Quadraten 
 - Ein Kochherd, der einen Benutzer benötigt.
 - Ein Kochherd, der eine Küche benötigt.

Zeichne die zugehörigen UML-Diagramme (auch in BlueJ – Variante) und begründe deine Zuordnungen!

5. Gib zwei eigene Beispiele, wo eine Assoziation auch eine Aggregation ist.
6. Gib zwei eigene Beispiele, wo eine Assoziation auch eine Komposition ist.

Delegation

Wenn ein Objekt einen Auftrag bzw. ein „Tun“ an ein anderes Objekt abgibt (weil das Objekt die passende Methode hat, es zu tun), so nennt man das Delegation.

➔ **Video: OOP - Delegation**

<https://www.youtube.com/watch?v=-NJ9JqP31NM>

➔ **Erklärender Text:**

[Skript Raumplanung -Text Delegation](#)

Aufgabe 5.8.

Schreibe folgendes Projekt so um, dass der Chef an den Manager und der wiederum an den Praktikanten delegiert.

<https://www.dropbox.com/s/dpz2nofolvx677z/Delegation.zip?dl=0>

Aufgabe 5.9.

Überprüfe zu deiner Sicherheit, ob du folgende Dinge nun kannst (fordere gegebenenfalls weitere Aufgaben zum Üben beim Lehrer an):

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|---|---|---|---|
| Ich kann Beziehungen zwischen Klassen korrekt herstellen und benennen | | | |
| 11. Vererbung | | | |
| 12. Aggregation | | | |
| 13. Komposition | | | |
| 14. Delegation | | | |
| Ich kann diese verschiedenen Beziehungstypen im UML-Diagramm korrekt darstellen | | | |

Was solltest du nun können:

1. Wissen, wie man Nutzerbeziehungen sinnvoll einsetzen kann, um Codeduplikation zu vermeiden.
2. Wissen, was eine Assoziation, eine Aggregation und eine Komposition ist und wie sie im UML-Diagramm dargestellt werden.

Modul 6 Raumplaner: Sammlungsklassen

Viele Schränke bilden eine Schrankwand

Das einfache Teil-Projekt Schrankwand aus Teil 5 könnte so erweitert werden, dass die Schrankwand eine „beliebige“ Zahl von Schränken enthalten kann. In diesem Fall brauchen wir statt der Objekte schrank1, schrank2 und schrank3 eine Variable (gesucht ist also ein **neuer Datentyp**) mit der wir alle Schränke gemeinsam verwalten können (in der also eine **beliebige Anzahl von Schrank-Objekten** gespeichert werden kann – die Anzahl ist vorher nicht bekannt, soll also variabel sein), also z.B. eine **Liste von Schränken** oder ein **Array von Schränken**. Wir werden uns mehrere Fälle ansehen, **zunächst das Beispiel Array**.

Tipp

Wenn du etwas neu in einer Klasse einfügst oder änderst – und es dann nicht sofort funktioniert in einer eventuell etwas komplexeren Umgebung – kann es sinnvoll sein, sich eine Testklasse anzulegen. Hier wird dann nur der zu testende Code eingefügt. Hat man sichergestellt, dass der Code in dieser einfachen Testumgebung funktioniert, so kann er in komplexere Umgebungen eingebunden werden.

So kann man Fehler leichter eingrenzen!

Kapitel 1 Lösung mit dem Array

- ➔ **Video: OOP – Arrays einfach**
<https://www.youtube.com/watch?v=pHU9g4ZoVNw>
- ➔ **Video: OOP – Arrays aus Objekten**
https://www.youtube.com/watch?v=ZfF-EGMBL_Q&t=17s
- ➔ **Video: OOP – Arraylist**
https://www.youtube.com/watch?v=VEitbCh_mrA
- ➔ **Video: OOP – Array und Arraylist im Vergleich**
<https://www.youtube.com/watch?v=btGhbfJcMi4>
- ➔ **Erklärender Text:**
Skript Raumplanung -Text Schrank-Arrays

Die ersten Änderungen im Konstruktor sind also:

```
public Schrankwand(int x, int y, int anz)
{
    xPosition = 40;
    yPosition = 80;
    farbe = "blau";
    orientierung = 0;
    istSichtbar = false;
    int schrankbreite = 60;
    anzahl = anz;
    breite = schrankbreite*anzahl; //Breite der Schrankwand berechnen
    tiefe = 37;
    ...
    //hier fehlt die Deklaration und Initialisierung eines Schrank-Arrays
}
```

Aufgabe 6.1.

1. Vervollständige den neuen Konstruktor der Klasse Schrankwand
Beachte bitte, dass im Schleifenkopf die richtigen Werte für die Variable `i` verwendet werden. `i` wird zunächst auf den Startwert 0 gesetzt, da die erste Position im Array den Index 0 hat, die Zählvorschrift `i++` zählt jeweils um 1 weiter und die Laufbedingung, mit der man den Abbruch der Schleife festlegt, muss `i < anzahl` lauten, da der höchste zulässige Index den Wert `anzahl-1` hat. Die zulässigen Indizes gehen also von 0..anzahl-1.
Denke an die Änderungen in `gibAktuelleFigur()`. Hier muss das Einfügen in den **General-Path** über eine Schleife gesteuert werden.
2. Korrigiere die Klasse Schrankwand, so dass ein Schrankwand-Objekt beliebig viele Schränke enthalten kann. Probiere es in BlueJ aus.

Kapitel 2 Lösung mit ArrayList

Eine ArrayList ist eine Klasse, die List implementiert (s.u.) und uns von JAVA im Paket `java.util` bereitgestellt wird. Wir müssen sie zunächst importieren.

```
import java.util.ArrayList;
```

Im Kopf der Klasse heißt es nun:

```
private ArrayList schraenke = new ArrayList();
```

oder wenn ich sicherstellen möchte, dass in meine ArrayList auch nur Objekte vom Typ Schrank hineindürfen:

```
private ArrayList<Schrank> schraenke = new ArrayList<Schrank>();
```

➔ Video: OOP – Erweiterte for-Schleife

https://www.youtube.com/watch?v=icnHmlb5_wg

(Spiele) <https://www.youtube.com/watch?v=6JqQAY-mnM8>

➔ Erklärender Text:

Skript Raumplanung -Text Schrank-ArrayList

Aufgabe 6.2.

1. Füge alles zusammen und ändere die Klasse Schrankwand so, dass sie mit einer ArrayList arbeiten kann.
2. Erläutere, was der Vor- bzw. Nachteil einer solchen ArrayList sein könnte.
3. Füge statt einer normalen Schleife eine „erweiterte for-Schleife“ ein.

Kapitel 3 Iteratoren (nur erNi und Prüflinge auf gruNi)

➔ **Videos: OOP - Iteratoren**

<https://www.youtube.com/watch?v=pfn9nbPAkdg>

(Spiele) <https://www.youtube.com/watch?v=ZnyfvIGkZ1A>

➔ **Erklärender Text:**

Skript Raumplanung -Text Iteratoren

Aufgabe 6.3. nur erNi und Prüflinge auf gruNi

Schreibe die Klasse Schrankwand entsprechend um, so dass sie mit Iteratoren arbeitet.

Aufgabe 6.4. (Abschlussaufgabe für diesen Teil) - alle!!!

Suche dir ein weiteres Möbelement in dem Projekt aus, welches du ähnlich wie die Schrankwand aus anderen Möbelobjekten zusammensetzen kannst. Implementiere die neue Klasse entsprechend der „hat-ein-Beziehung“ mit einem Array, einer ArrayList und mittels eines Iterators.

Erstelle ein kurzes, Video, in welchem du deine Überlegungen und Implementierungen dazu erläuterst. Füge dies deinem Portfolio hinzu.

Gib deine Ausarbeitungen beim Lehrer ab.

Zusatz: LinkedList

➔ **Videos: OOP - LinkedList**

<https://www.youtube.com/watch?v=ol2xQQo7jvg>

➔ **Erklärender Text:**

Skript Raumplanung -Text Linked List

Zusatz-Aufgabe 6.7.

Schreibe das Projekt Raumplaner so um, dass statt ArrayList der Datentyp LinkedList verwendet wird.

Aufgabe 6.8.

Überprüfe zu deiner Sicherheit, ob du folgende Dinge nun kannst (fordere gegebenenfalls weitere Aufgaben zum Üben beim Lehrer an):

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|--|---|---|---|
| Ich kann weiterführende Programmstrukturen in mein Projekt einfügen und beherrsche den Umgang mit ihnen: | | | |
| 9. Arrays | | | |
| 10. ArrayList | | | |
| 11. Erweiterte „for“ - Schleifen | | | |
| 12. Iteratoren | | | |
| 13. Zusatz Linked List | | | |
| Ich weiß, wann sich diese Programmstrukturen sinnvoll einsetzen kann in einem Projekt | | | |

Was solltest du nun können:

1. Wissen, was ein **Array** ist und wann und wie man es in einem Programm sinnvoll einsetzt.
2. Wissen, was eine **ArrayList** ist und wann und wie man sie in einem Programm sinnvoll einsetzt.
3. Wissen, wann es sinnvoller ist, mit einem Array zu arbeiten und wann man lieber eine ArrayList verwenden sollte.
4. Wissen, was eine erweiterte-for-Schleife und ein **Iterator** ist und wann und wie man ihn in einem Programm sinnvoll einsetzt.
5. **Zusatz**: Wissen, wie man mit Linked-Listen umgeht und wie man sie im Programm einsetzt.

Nach Teil 6 wird ein Test geschrieben, um zu sehen, ob die Inhalte ausreichend verstanden wurden. Das ist wichtig, damit du die weiteren Teile erfolgreich bearbeiten kannst.

Modul 7: Raumplaner (nur erNi): Polymorphie

Das wohl **wichtigste Prinzip der OO-Modellierung** ist das der Polymorphie.

➔ **Videos: OOP - Polymorphie**

<https://www.youtube.com/watch?v=NQ3Wdmzr7no>

(Spiele) https://www.youtube.com/watch?v=c6G2Dm_J-1c

Dazu zunächst eine kleine Aufgabe, um dir das Prinzip zu verdeutlichen:

Aufgabe 7.1.

1. Schreibe für jede einzelne konkrete Möbelklasse eine individuelle Methode „kosten()“ (die Signatur der Methode muss in allen Klassen gleich sein: **public double kosten()**), die die Kosten für ein Möbelstück zurückgibt. Lege dazu auch in der Moebelklasse eine neue Instanzvariable **kostenMoebel** an, die in den konkreten Unterklassen aber jeweils mit anderen Werten belegt wird. Ebenfalls in der Oberklasse **Moebel** soll es eine neue Instanzvariable **kostenStunde** geben, die mit einem Wert von 30 belegt sein soll (für alle Möbelklassen also gleich ist), sie bezeichnet die Arbeitskosten, die pro Stunde anfallen (falls sie anfallen). Bei der Klasse **Stuhl** sollen die Kosten einfach dem Wert der Instanzvariablen **kostenMoebel** entsprechen. Bei der Klasse **Tisch** dagegen fließen in die Kosten für das Material noch die Arbeitsstunden ein mit einem Faktor von 20 pro Stunde. Die Methoden sind also vom Kopf her (Signatur) gleich, werden aber vollständig unterschiedlich implementiert. Denke dir für deine eigenen Möbelklassen eigene Berechnungsschemata für die Kosten aus und implementiere das jeweils in der **kosten()** Methode.

Beachte: Für die Klasse **Tischgruppe** musst du eine Methode **gesamtKostenTischgruppe()** anlegen, denn die **Tischgruppe** ist kein **Moebel** und die Kosten würden über die Methode **kosten()** ja sonst doppelt berechnet werden, wenn mal die Gesamtkosten bestimmt werden sollen (da die Elemente der **Tischgruppe** ja außerhalb der **Tischgruppe** existieren).

1. **Zusatz** Entsprechend des beobachter-Musters (siehe Kapitel Entwurs-Muster) soll nun eine abstrakte Klasse „**Beobachter**“ angelegt werden (abstrakt, weil von ihr ja kein Objekt erzeugt werden soll), welche alle Möbel „beobachtet“, die auf der Leinwand sichtbar sind.
D.h. sobald ein Möbelstück auf der Leinwand sichtbar gemacht wird, muss es auch in der Beobachter-Klasse angemeldet werden – wenn es wieder von der Leinwand gelöscht wird, muss es entsprechend beim Beobachter wieder abgemeldet werden. Zusätzlich soll es eine Methode geben, die die Gesamtkosten aller auf der Leinwand sichtbaren Möbelstücke berechnet. Alle drei Methoden sollten statische Methoden sein, da es wie gesagt kein Objekt gibt, über das sie aufgerufen werden können. Man muss sie aber von den Möbelklassen aufrufen können, um Objekte beim Beobachter an- und abzumelden.

Tipp: falls du Probleme hast, wie du damit anfangen sollst, hier ein Grundgerüst der Beobachter-Klasse:

```
import java.util.*;import java.util.*;
public abstract class Beobachter
{
    private static ArrayList<Möbel> ...
    private static int gesamtKosten = 0;

    public static void hinzufügen(Möbel möbel){
        ...
    }
    public static void entfernen(Möbel möbel){
        ...
    }

    public static int gesamtKosten(){
        ...
    }
}
```

2. **Zusatz** Erläutere nun, welche Methode jeweils aufgerufen wird, wenn der Beobachter die **gesamtKosten** berechnet.
3. Recherchiere, was in der OO-Modellierung „überschreiben“ bedeutet – und vergleiche das mit dem, was in Aufgabe 3 erläutert wurde.
4. Beschreibe nun in eigenen Worten, was Polymorphie bedeutet und welche Vorteile es liefert.
5. *Erstelle ein kurzes, Video, in welchem du deine Überlegungen und Implementierungen dazu erläuterst. Füge dies deinem Portfolio hinzu.*
6. *Gib deine Ausarbeitungen beim Lehrer ab.*

Nun hast du dir ganz allein erarbeitet, wie dieses wichtige Prinzip der OO-Modellierung funktioniert. Und das war auch schon alles dazu!

Zusatz Polymorphe Variablen

➔ **Erklärender Text:**

Skript Raumplanung -Text Polymorphe Variablen

Aufgabe 7.2.

Überprüfe zu deiner Sicherheit, ob du folgende Dinge nun kannst (fordere gegebenenfalls weitere Aufgaben zum Üben beim Lehrer an):

| Kompetenzen, die du nun erreicht haben solltest | + | 0 | - |
|---|---|---|---|
| Ich kenne Polymorphie als Grundsatz der OOP. | | | |
| Ich kann Polymorphie in einem Projekt erkennen | | | |
| Ich kenne die Vorteile, die Polymorphie mit sich bringt | | | |
| Ich kann Klassen bzw. Methoden entsprechend dieses Konzeptes der Polymorphie implementieren | | | |

Was solltest du nun können:

1. Wissen, was das Prinzip Polymorphie in der OO-Modellierung bedeutet und welche Vorteile es mit sich bringt.
2. Es in einem Projekt erkennen und auch selbst implementieren können.

Zum Abschluss wird ein Test geschrieben, um zu sehen, ob die Inhalte ausreichend verstanden wurden.

Zusatz: Modul 8: Raumplaner (nur erNi): Entwurfsmuster (Design-Pattern)

Entwurfsmuster sind ein Strukturierungsmittel, das hilft, nicht jedes Mal das „Rad neu zu erfinden“. Erkennt man in einem Entwurf, den man macht, ein solches Entwurfsmuster, dann braucht man sich nicht noch einmal zu überlegen, wie „man das eigentlich macht“. Nach dem „aha, das ist wieder ein ...“ schaut man entweder in einer Lösung, die man schon einmal gemacht hat oder in der Literatur dazu nach. In der Regel wird man dann sogar größere Programmblocke schon fertig haben.

→ **Erklärender Text:**

Skript Raumplanung -Text Entwurfsmuster

Was solltest du nun können:

1. Wissen, was ein Entwurfsmuster (design-Pattern) ist und wozu man sie verwendet bzw. wann man sie einsetzt.
2. Folgende Entwurfsmuster zumindest oberflächlich kennen:
 - Singleton
 - Fabrik-Methode
 - Iterator
 - Beobachter (Observer)
 - Composite Muster