## Abstraktion

Abstraktion bedeutet, Details, wie z.B. die Interna einer Methode, also wie genau eine Methode implementiert ist, zu ignorieren. Die Interna sind für die Benutzung der Methode nicht relevant.

## Modulation

Modularisierung bedeutet, dass man ein Programm in einzelne Komponenten aufteilt. Aufteilung in Verständliche Teilprobleme

## List of Errors Summary: (unchecked)

* NullPointerException. (not Instantiated array list e.g)
* ArrayIndexOutOfBoundsException.
* IllegalStateException. (e.g. Remove from Array in Loop)
* ClassCastException. (wrong Class hierarchy, to cast)
* ArithmeticException. (div. By Zero)
* IllegalArgumentException. (invalid entry as Parameters)

## Iterator :

* Ein Iterator deklariert nur Methoden, die zum Iterieren benötigt werden, und eine Methode zum Löschen des aktuellen Elements.
* Den Iterator hole ich mir mittels: Iterator<Auto> iter = autos.iterator();
* Im Gegensatz zu einer foreach Schleife, können beim Iterieren über die Sammlung mit einem Iterator während dem Iterieren auch Elemente entfernt werden.
* for(Iterator<Integer> it = list.iterator(); it.hasNext(); ) geht auch um alle Objekte zu entfernen.

## Generische Klasse

*class Test<T> {*

 *// An object of type T is declared*

 *T obj;*

 *Test(T obj) { this.obj = obj; } // constructor*

 *public T getObject() { return this.obj; }*

*}*

Eine generische Klasse definiert potentiell viele Typen.

Im Code sehen Sie die Typdeklaration PatriciaTrie<Person>. Sie schliessen daraus: PatriciaTrie ist eine generische Klasse.

*public class SchluesselWertPaare<S,W>*

*{*

 *public void hinzufuegen(S schluessel, W wert) {*

 *}*

*}*

in hinzufügen können : Objekte der Typen, die beim Erzeugen des SchluesselWertPaare-Objekts für S und W festgelegt wurden.

## Deprecated

Mit "Deprecated" Gekennzeichnetes sollte nicht mehr verwendet werden. Typischerweise weil es gefährlich ist oder eine bessere Alternative existiert.

## Dokumentation

### Klasse:

* + Den Klassennamen
	+ Einen Kommentar, der den allgemeinen Zweck und die Eigenschaft der Klasse beschreibt
	+ Eine Versionsnummer
	+ Den (oder die) Autorennamen
	+ Eine Dokumentation für jeden Konstruktor und jede Methode (mindestens für die, die public sind)

## Tests

Modultests vs. Akzeptanztests

* Modultests testen typischerweise Klassen oder Methoden.
* Akzeptanztests testen eine ganze Anwendung.
* Akteptanztest sind nicht auch Modultests

Regressionstests

* Regressionstests sollen garantieren, dass bestehende Funktionalitäten bei Änderungen weiter wie bis anhin funktionieren.
* Automatische Regressionstests sind zu bevorzugen
* Zum Vermeiden einer Situation, in der während der Modifikation eines Programms Fehler in bestehende Funktionalität eingefügt wird.
* JUnit Testfälle sind Regressionstests.
* Bei Änderungen / Refactoring einer Applikation sollten Regressionstests bereits vorhanden sein.

## Kopplung / Kohäsion



Enge Kopplung: (nicht möglich Aussage über innerhalb einer Klasse) Code Duplizierung = Hohe Kopplung

Was ist der grösste Vorteil, wenn man die Änderungen im Code lokal behält? => Es gibt weniger Folgefehler in andern Klassen.

Hohe Kohäsion löst keine Kopplungsprobleme.

Wenn alle Methoden in einer Klasse eine hohe Kohäsion aufweisen, dann weisst die Klasse selber **nicht** auch eine hohe Kohäsion auf.

## Switch vs. If/Else

Nicht jede Kombination von if- und else-Anweisungen kann durch eine einzige Switch-Anweisung ersetzt werden.

Die Verwendung von if-else sollte grundsätzlich zugunsten switch-Konstrukten vermieden werden.

Eine switch-Anweisung kann teilweise eine Sequenz von if- und else-Anweisung ersetzen und hat obendrein den Vorteil, dass sie die Übersichtlichkeit verbessert.

## Superklasse / Subklasse

Aus der Subklasse kann indirekt auf private Datenfelder der Superklasse zugegriffen werden, indem die Superklasse geeignete Methoden anbietet.

Eine Superklasse kann mehrere direkte Subklassen haben, aber Eine Subklasse kann nicht smehrere direkte Superklassen haben

Hund ist Subklasse von Haustier und Dogge von Hund. =>

Haustier tier = new Dogge();

if (tier instanceof Hund) => TRUE

## Überschreiben vs. Überladen

Um eine Methode zu überschreiben, muss dieselbe Signatur wie bei der überschriebenen Methode verwendet werden

## Equals / ==

Falls nicht für eine Spezifische Klasse überschrieben tun sie exakt dasselbe.

## Statischer vs. Dynamischer Typ

Der dynamische Typ einer Variablen muss derselbe Typ oder ein Subtyp (Subklasse) des statischen Typs sein.

Zur Kompilierzeit ist im Allgemeinen nicht bekannt, welche Methode zur Laufzeit aufgerufen wird.

Es wird immer die Methode des dynamischen Typs verwendet. Diese Methode kann direkt in der entsprechenden Klasse implementiert bzw. überschrieben oder von einer Superklasse geerbt sein.

## Abstrakte Klassen und Interfaces

Kann extends von 1 Abstrakte Klasse und beliebig vielen Interfaces (mit Komma)

* Interface und Abstrakte Klassen können deklariert werden aber dann nicht benutzt
* Implementiert eine Subklasse alle abstrakten Methoden der Superklasse, so darf sie trotzdem selbst als abstrakt deklariert werden.
* Implementiert eine Subklasse nicht alle abstrakten Methoden der Superklasse, so muss sie selbst als abstrakt deklariert werden.
* Enthält eine Klasse eine abstrakte Methode, so muss die Klasse selbst als abstrakt deklariert werden
* Es ist möglich abstrakte Klassen zu schreiben, die keine abstrakten Methoden besitzen.