Zusammenfassung PM2

## Paradigmen

* Ausführung von Bytecode auf einer VM
* Plattformunabhängigkeit
* Optimierung durch JIT-Compilation
  + Übersetzen von Teilprogrammen in Maschinencode zur Laufzeit
* Kompiliert und Interpretiert
* Garbage-Collected: Nicht referenzierte Objekte finden und ab und zu entfernen
* Keine Header-Dateiei, da der Compiler die Informationen direkt aus den Klassen abliest
* Trennzeichen: Leer, Tab, Zeilen- und Seitenvorschub
* Seperatoren: Semikolon, Komma, Punkt, 3 Klammertypen
* 50 Keywords
* Leere Blöcke ohne Keyword sind erlaubt: { … }
* Imperativ: Variablen zum Ablegen von Daten
* Methoden: Call-by-value: Speicherinformationen warden nicht an Methoden als Argument übergeben, nur der Wert

## Widersprechen der objekt-orientierten Prinzipien:

* Konstruktoren sind nicht polymorph und werden nicht vererbt
* Operatoren sind keine Methoden für Referenztypen
* Primitive Datentypen sind keine Objekte

## Konsoleneingabe

String eingabe = new Scanner(System.in).nextLine(); (nextInt, nextDouble, …)

## Arrays

* Arrays implementieren das Interface Cloneable

Verschachtelung vergleichen

* Arrays.deepEquals(array\_1, array\_2) -> Arrays.equals prüft nur flach
* array\_1.equals(array\_2) auf Identität

Kopien

* Referenzkopie: int[][] sc = s;
* Flache Kopie: Neu initialisieren und Werte übernehmen
* Tiefe Kopie: Auch innere Werte (Array im Array) übernehmen

Initialisieren

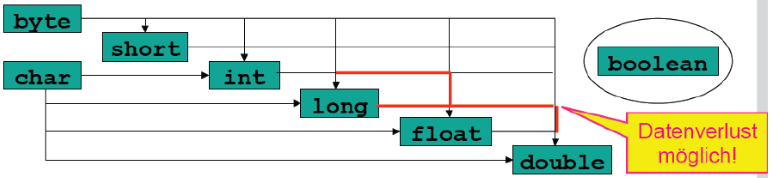
* Int[] primes, matrix[], m[][]; ist gleich int[] primes, int[][] matrix, int[][][] m;

## Kovarianz / Kontravarianz / Invarianz

Ob z.B. eine Typdeklaration gleichartig / entgegen der Vererbungsrichtung ist. Liegt in der Unterklasse keine Änderung gegenüber der Oberklasse vor, wird das als Invarianz bezeichnet

wenn eine Klasse U eine Unterklasse einer Klasse O ist, ist auch U[] ein Untertyp von O[]

## Coercion

Typumwandlung von einem Typ in einen anderen:  


* implizit (Nicht im Quelltext, Macht Java nur, wenn nicht verlustbehaftet) / explizit (Programmierer)
* werterhaltend (widening, 16 Bit 🡪 32 Bit) / verlustbehaftet (narrowing, 32 Bit 🡪 16 Bit)
* benutzerdefiniert / vordefiniert

byte und short werden bei Rechenoperationen automatisch in int umgewandelt. Ist ein Operant long, werden alle Operanden auf long erweitert

## Dangling-Else-Problem

Ein else von nicht geklammerten if-Anweisungen gehört immer zum innersten if

## Signatur

Methodenname und Parameterliste

## Parameter & Argument

void print(double a, double b): a, b 🡪 Parameter  
print(1.0, 2.0): 1.0, 2.0 🡪 Argumente

## Liskovsches Substitutionsprinzip (LSP, Ersetzbarkeitsprinzip)

Kriterium der OOP, das die Bedingung zur Modellierung eines Datentyps für seinen Untertyp angibt. Es besagt, dass ein Programm, das Objekte einer Basisklasse T verwendet, auch mit Objekten der davon abgeleiteten Klasse S korrekt funktionieren muss, ohne dabei das Programm zu verändern.

## Autoboxing / Unboxing

Autoboxing: Automatische Konvertierung vom Compiler zwischen primitiven Typen und ihren entsprechenden Objekt-Wrapper-Klassen. Z.B. Character ch = ‘a‘;

Unboxing: Konvertierung einer Objekt-Wrapper-Klasse in den entsprechenden primitiven Typen. Z.B. List<Double> list = new ArrayList<>(); double d = list.get(0);

## Wildcard-Typen

Kompatibilitätsregel

Lesen

Schreiben

## Statisches Binden

Es steht zur Compilezeit fest, welche Methodendefinitionen ausgeführt werden soll.

In Java werden Konstruktoren, Attribute, finale Methoden, Statische Methoden und Attribute, und private Methoden statisch gebunden.

## Dynamisches Binden

Umgang des Compilers mit polymorphen Methoden. Ein Methodenaufruf zur Laufzeit wird anhand des tatsächlichen Typs eines Objektes aufgelöst.

Erst zur Laufzeit kann entschieden werden, welche Version einer überlagerten Methode aufgerufen werden soll 🡪 Verschlechterung der Performanz, da der Compiler zusätzlichen Code generieren muss.

Erst zur Laufzeit wählt die Laufzeitumgebung dynamisch die entsprechende Objektmethode aus passne dzum tatsächlichen Typ. Die VM weiß, welcher Typ hinter den Objekten steht

## Checked / Unchecked Exceptions

## Sichtbarkeit

* public: Außerhalb der Klasse zugreifbar
* private: Innerhalb der Klasse zugreifbar
* protected: Innerhalb des Packages und über Subklassen zugreifbar

## Main

Lässt man die Parameter der main-Methode weg, produziert der Compiler keine Fehlermedlung. Allerdings braucht der Java-Interpreter eine Methode mit der Signatur   
public static void main(String[] args), um ein Programm auszuführen zu können. Findet er diese nicht, wird zur Laufzeit ein Fehler geworfen.

## Equals

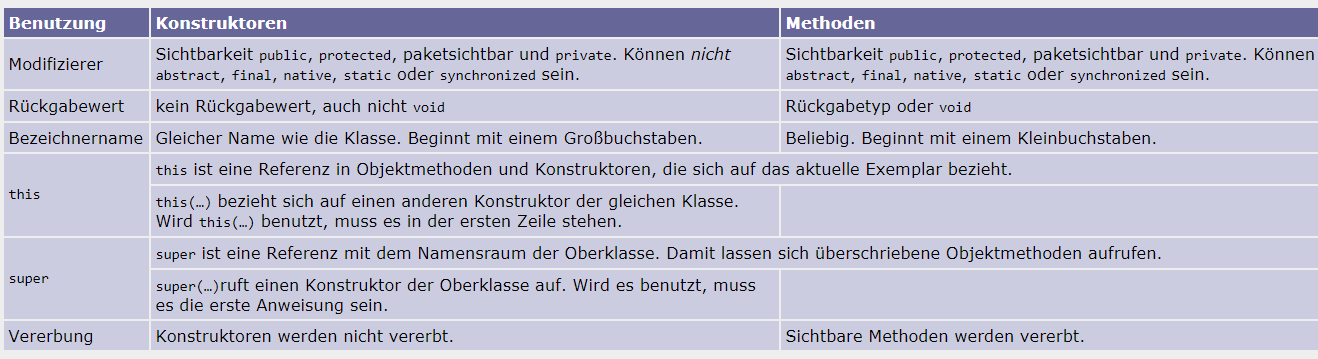
@Override  
public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 SubTest subTest = (SubTest) o;  
 return id == subTest.id;  
}

## Referenzvariable

Das Ergebnis eines new ist eine Referenz auf das neue Objekt. Die Referenz wird in einer Referenzvariable zwischengespeichert, um Eigenschaften vom Objekt nutzen zu können.

## Konstruktor

Normalerweise werden Objektvariablen initialisiert, sodass sichergestellt ist, dass das neue Objekt einen sinnvollen Anfangszustand aufweist.



## Default-Package

Klassen aus dem unbenannten Paket sind nicht für Unterpakete sichtbar

## Import

Das import informiert den Compiler über Pakete, sodass ein Typ nicht mehr voll qualifiziert werden muss.

## Statischer Import

Gibt die Möglichkeit die statischen Methoden oder Variablen ohne Typnamen sofort zu nutzen

## Referenzen

3 spezielle Referenzen: null, this, super

## Identität

Referenzieren zwei Variablen denselben Typ sind sie identisch. == überprüft auf Identität. Der Inhalt von Objekten spielt bei == keine Rolle

## Gleichheit

In der Klasse ist festgelegt, wann Objekte gleich(wertig) sind (equals-Methode)

## Generics

Type Erasure: Compiler löscht alle Typinformationen (Alles was in spitzen Klammern steht) und jede Typvariable wird zu Object. <T> fällt weg, T wird zu Object.

Nicht vorhanden:

* new
* instanceof
* generische Exceptions
* statische Eigenschaften
* Überladung

Bounds

* Extends: <T extends Comparable<T>>

Wildcards

