Praxis der Programmierung

Namensräume, Ausblick auf Java

Institut für Informatik und Computational Science
Universität Potsdam

Henning Bordihn

Namensräume

Erinnerung: Namensräume

- definieren Bereiche, in denen Namen / Bezeichner eindeutig sein müssen
- in verschiedenen Namensräumen kann der gleiche Name verwendet werden
- *Beispiele*:
  - Telefonnummern in Vorwahlbereichen: `0331 123456`
    `030 123456`
  - Telefonnummern mit Vorwahl in Ländernetzen: `+49 30 123456`
    `+36 30 123456`
  - Dateinamen in Ordnern; diese in übergeordneten Ordnern, ...
  - in Netzwerken absolute Pfadnamen auf Hosts (Rechnernamen), ...

Erinnerung: Qualifizierte Namen

- unqualifizierte Namen: die Bezeichner selbst

  Beispiel: `meineDatei`

- qualifizierte Namen: mit Angabe des Namensraums

  Beispiel: `/home/rlehre/meineDatei`

- in **C++**: `namensraum::bezeichner`

  Beispiel: `std::cout`, `std::cin`, `std::endl`

- Namensraum `std` enthält Bezeichner der Standardbibliothek

- Definition eigener Namensräume möglich ... *später*

Benutzerdefinierte Namensräume

- Zuordnung von Definitionen zu einem Namensraum:

```
namespace name {
   void function1();     // Definitionen, die dem Namensraum 'name'
   int function2();      // zugeordnet werden
}
```

- Zugriff mit `name::function1();`

  oder durch

```
using namespace name;
// ...
function1();
```

Aufgabe 1

1. Kopieren Sie aus `/home/rlehre/W14` die Datei `namespaces.tar.gz` und analysieren Sie die Quellcodes.

2. Kommentieren Sie in `testgeometry.cpp` den Aufruf von `diameter()` ein. Was stellen Sie fest?

3. Ordnen Sie die Definitionen in `square_ext_point.h` einem neuen Namensraum `noabstraction` zu. Treffen Sie alle nötigen Maßnahmen, damit die Methode `diameter()` in `testgeometry.cpp` getestet werden kann.

Anonyme Namensräume

- Namensräume ohne Namen: `namespace { // ... }`
- Zugriffe sind auf Funktionen beschränkt, die in derselben Quelltextdatei definiert sind.
- Alternative: alle Funktionen `static` definieren
  - `static` definierte Funktionen werden dem Linker nicht bekannt gegeben
    $\rightsquigarrow$ stehen nur innerhalb der Quelltetdatei mit ihrer Definition zur Verfügung
  - *aber:* statische Methoden von Klassen sind Klassenmethoden

Ausblick auf Java

Java

Applikation: eigenständiges Programm, das in Java geschrieben wurde und ohne Browser ausgeführt werden kann

Applet : Java-Programm, das in eine HTML-Seite eingebunden wird und in einem Browser ausgeführt werden kann

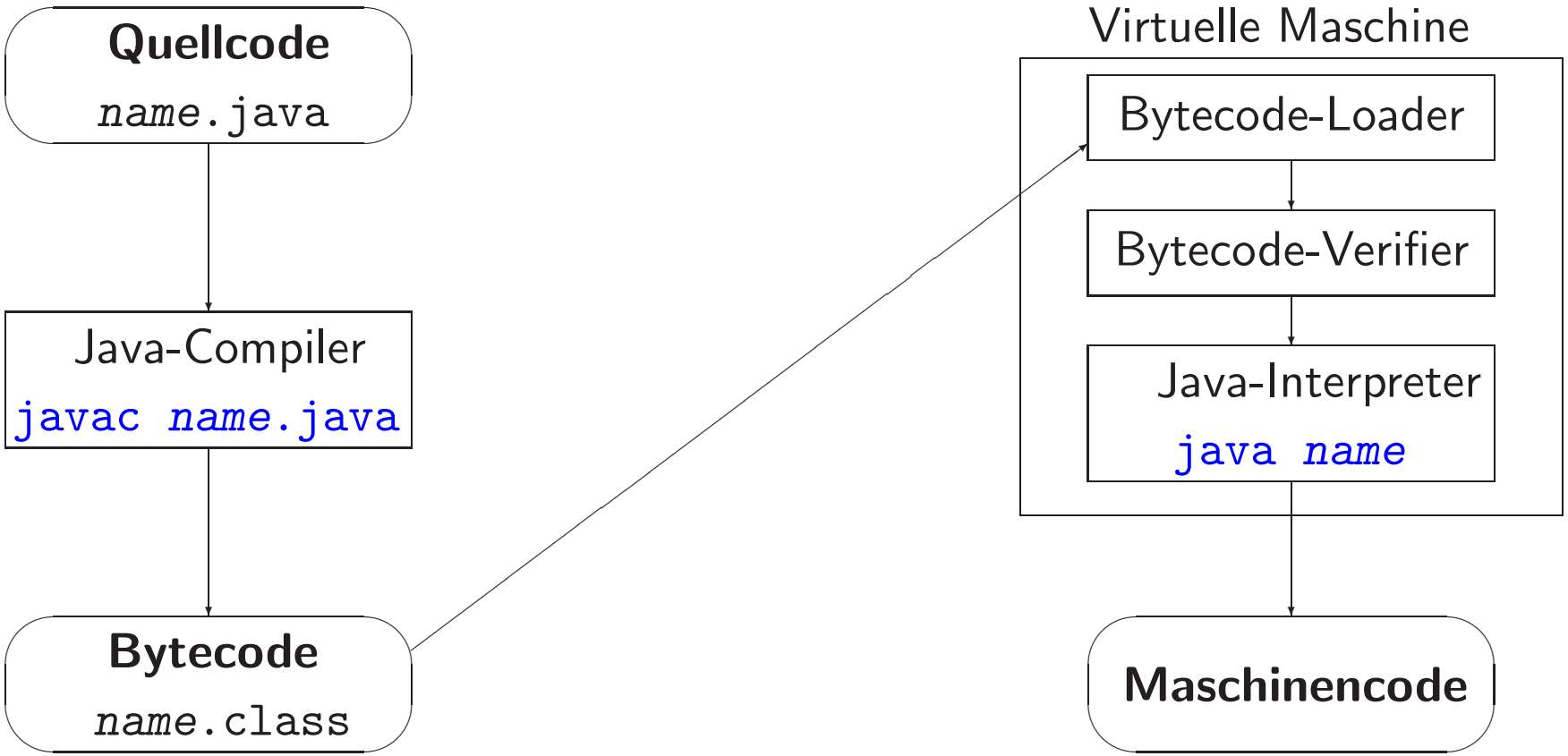
Java und das Internet

- *Idee:* Übertragung von Programmcode vom Server zum Client, der im Internet-Browser ausgeführt wird

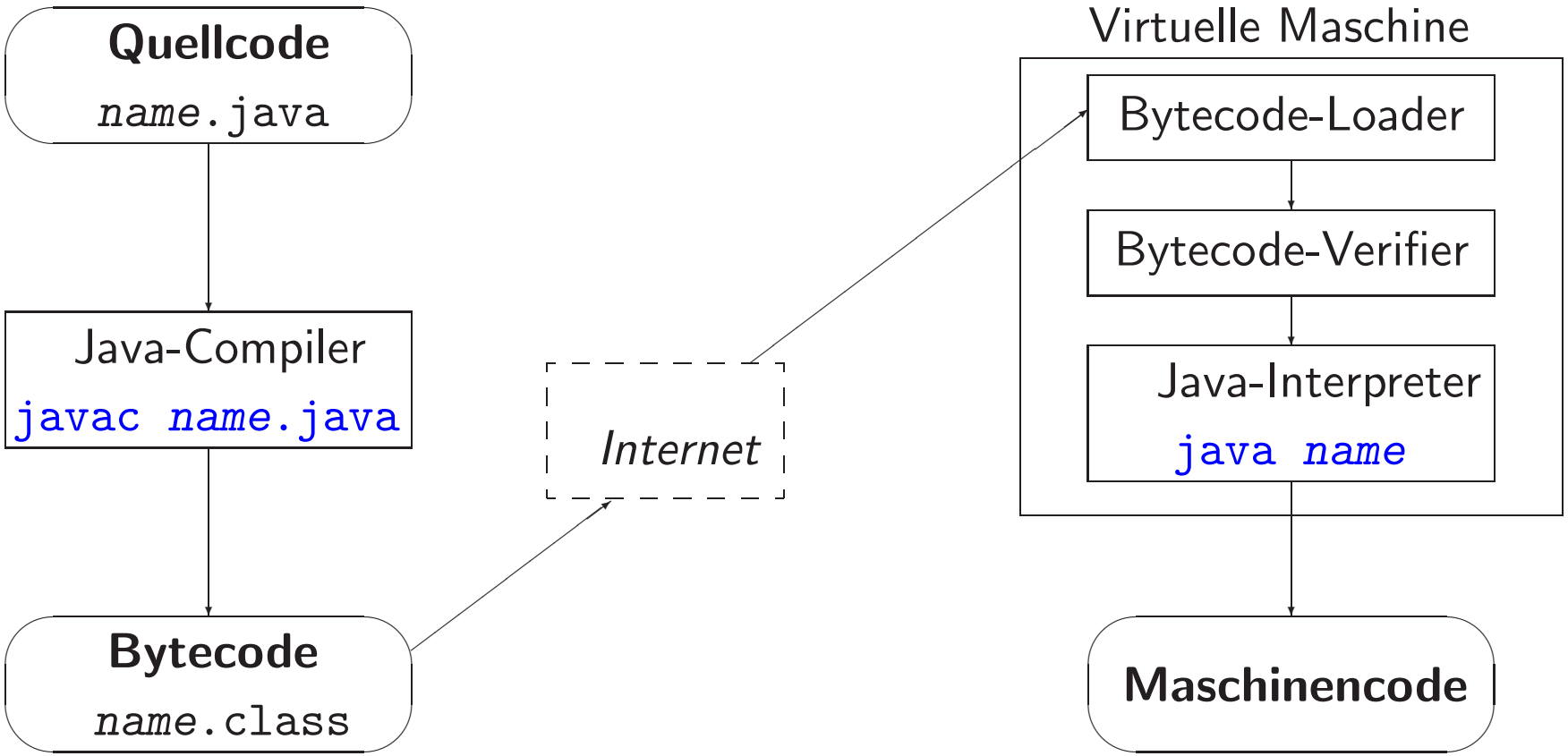
$\Longrightarrow$ **Anforderungen an die Programmiersprache:**

  - plattformunabhängiger Programmcode (*Portabilität*)
  - gegen einfache Änderungen der Ablaufumgebung *robuster* Programmcode
  - *Sicherheit* z.B. vor Viren sowie vor dem Zugriff auf das Dateisystem, die Hardware und den Browser
  - Vielfältigkeit/Universalität
  - modernes Programmier-Paradigma

Quellcode — Bytecode — Maschinencode



Applets im Internet



Eigenschaften von Java

- portabel
  - virtuelle Maschine
  - plattformunabhängige Datentypen
- robust
  - keine Pointer, Garbage Collector
  - stenge Objektorientiertheit, strenge Typenprüfung
- sicher
  - virtuelle Machine, Bytecode-Verifizierung
  - digitale Signaturen
- multithreaded
  - parallele, kommunizierende Prozesse
- verteilt
  - Remote Method Invocation (RMI)
- objektorientiert

Einfache Datentypen

Datentyp	mögliche Werte	Wrapper-Klasse
boolean	true, false	Boolean
byte	-128..127 (8 Bit)	Byte
short	-32768..32767 (16 Bit)	Short
int	-2147483648..2147483647 (32 Bit)	Integer
long	-9223372036854775808..922$\cdots$807 (64 Bit)	Long
float	Gleitkommazahl (32 Bit)	Float
double	Gleitkommazahl (64 Bit)	Double
char	einzelne Unicode-Zeichen (16 Bit)	Character

Quellen

- Java-Entwicklungsumgebung:

  `http://www.oracle.com/technetwork/java/index.html`
  `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
  `http://www.oracle.com/`

- Literatur:

  http://docs.oracle.com/javase/8/docs/api/index.html
  http://www.javabuch.de/
  http://openbook.galileocomputing.de/javainsel/

Quellcode einer Java-Applikation

```
// Name.java
//

public class Name { // muss mit dem Dateinamen uebereinstimmen

     public static void main (String[] args) {

          Anweisung;
          Anweisung;
          ...
          Anweisung;
     }
}

// Name
```

Aufgabe 2

Erstellen Sie eine Java-Applikation zur Ausgabe von `Hello World!`.

Hinweis: Zur Ausgabe eines Strings s benutzen Sie die Anweisung

`System.out.println(`s`);`

`System.out.print(`s`);` erzeugt die Stringausgabe ohne Zeilenvorschub.

Aufgabe 3

1. Kopieren Sie aus `/home/rlehre/W14` die Dateien `Lab2_1.java` und analysieren Sie den Quellcode. Führen Sie die Applikation aus.

2. Sehen Sie sich die Klasse `String` der Java-API an:
   `http://docs.oracle.com/javase/8/docs/api/index.html`
   Benutzen Sie String-Methoden, um den zweiten und dritten Buchstaben des Vornamens zu isolieren und diese in Großbuchstaben auf die Konsole auszugeben.

Java-Klassen und Applikaionen

- Es gibt nur Klassen.
  (streng objektorientiert) (teilweise Spezialfälle)

- Alle Funktionen sind Methoden (ggf. statisch),
  alle globalen Variablen sind statische Datenelemente.

- Es gibt keine Header-Dateien.
  (Um Schnittstellen festzulegen, werden Interfaces benutzt.)

- Eine Java-Applikation ist eine Java-Klasse mit einer `main`-Methode.

Definition von Datenelementen und Methoden

- wie in **C++**, *aber*:
- Modifikatoren für die Sichtbarkeit stehen in jeder Definition als erstes Schlüsselwort

```
private int x;
public static void methode() { }
```

Standardinitialisierung von Datenelementen

Alle Datenelemente, die nicht durch die Parameter des Konstruktors initialisiert werden, erhalten standardmäßige Initialwerte wie folgt:

`byte, short, int`	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
boolean	false
Verweistypen	null

Datentypen

einfache Datentypen

```
boolean
byte, short, int, long
float, double
char
```

Wrapper-Klassen:

```
Boolean
Byte, Short, Integer, Long
Float, Double
Character
```

Referenz-/Verweisdatentypen

```
String
Point
Integer
...
```

haben KEINE Wrapper-Klassen

einfache Datentypen

Referenz-/Verweisdatentypen

Variablendefinition

```
int num1, num2;
boolean b1, b2;
```

```
Point p1, p2;
String str1, str2;
```

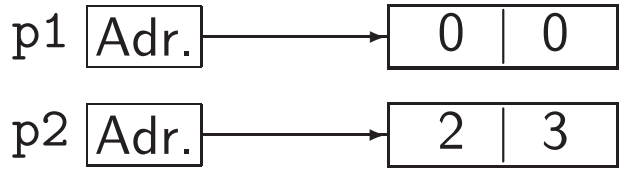
Variableninitialisierung

```
num1 = -12;
num2 = 4;
b1 = true;
b2 = false;
```

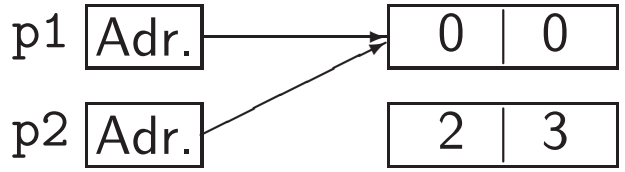
```
p1 = null;
p2 = new Point();
str1 = new String("Hallo!");
str2 = "Hallo?";
```

im Hauptspeicher:

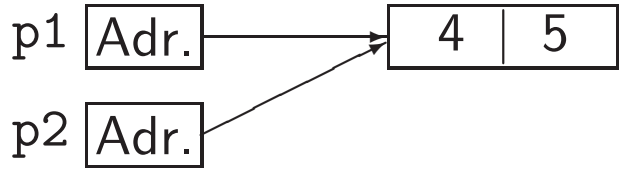
```
p1 = new Point();
p2.moveTo(2,3);
```



```
p2 = p1;
```



```
p1.moveTo(4,5);
```



```
num2 = num1;
```

```
num1 = 99;
```

Schlüsselwörter zur Zugriffsmodifikation

- Datenelemente/Methoden/Konstruktoren mit dem Modifier

`public`	sind für alle Klassen sichtbar;
`private`	sind nur für die Klasse sichtbar, in der sie vereinbart sind;
ohne Modifier	sind für Klassen aus demselben Paket (Verzeichnis) sichtbar
`protected`	für alle Unterklassen und Klassen aus demselben Paket sichtbar

- Der Modifier ist das *erste* Schlüsselwort in der Definition/ Signatur.
- Klassen dürfen auch Modifier haben. (`public class ...`)

Parameterübergabe

Der Parameter der `main`-Methode

```
String[] args
```

wird beim Programmstart mit jenen Zeichenketten initialisiert, die auf der Kommandozeile als Argumente nach dem Namen der Applikation angegeben sind.

```
java BspApplikation das sind 4 Parameter
```

bewirkt die Initialisierung

```
args[0] = "das"      args[1] = "sind"
args[2] = "4"        args[3] = "Parameter"
```

Vererbung

- keine Mehrfachvererbung
- Syntax: `public class Unterklasse extends Oberklasse`
- alle Methoden sind implizit virtuell
- Aufruf von Konstruktoren der Oberklasse: `super(...)`
- Aufruf von Methodenimplementierungen der Oberklasse (beim Überschreiben): `super.method(...);`
- abstrakte Klassen und Methoden mit Schlüsselwort `abstract` definieren, z.B.: `public abstract class { public abstract void method(); }`

Casting von Verweisdatentypen

Situation:

- Variable `varA` vom Typ Verweis auf Instanzen der Klasse `A`
- Zuweisung eines Exemplars `exemplarB` einer Unterklasse `B` von `A`

```
A varA = exemplarB;
```

1. **Indikation:** Zuweisung an eine Variable vom Typ `B`

   Syntax: `B varB = (B)varA;`

2. **Indikation:** Aufruf einer Methode `methode()`, die in `B`, nicht aber in `A` existiert

   Syntax: `((B)varA).methode();`

Interfaces

- Interfaces sind reine Schnittstellen, die keinerlei Implementierung enthalten.
- Alle Methoden sind implizit `abstract`, alle Datenelemente sind implizit final static (ohne Angabe dieser Schlüsselwörter!).

```
public interface Bewegliches {

public void beschleunigeAuf(int speed);
public void stoppe();

}

public interface Lebend {} // flag-Interface
```

Interfaces (2)

- Eine Klasse kann ein oder mehrere Interfaces implementieren.

  ```
  public class Tier implements Bewgliches, Lebend { ... }
  ```

  Dann müssen alle Methoden dieser Interfaces überschrieben werden.

- Die Eigenschaft einer Klasse, ein Interface zu implementieren, wird an ihre Unterklassen vererbt.

- Ob ein Objekt `objekt` ein Exemplar einer Klasse `klasse` ist, die ein Interface `interf` implementiert, kann zur Laufzeit mit dem `instanceof`-Operator geprüft werden:

  ```
  object instanceof interf   // true falls klasse interf implementiert
  ```

Pakete

- Namensräume werden durch Pakete definiert (kein `namespace`)
- hierarchische Definition von Paketen möglich
- vor dem Schlüsselwort `class`:
  - `package paketname;`
    ⇝ hier definierte Klassen gehören zum Paket `paketname`
    ⇝ Datei muss im Unterverzeichnis `paketname` liegen
  - `import paket.Klasse;`
    `import paket2.*;`
    ⇝ `Klasse` aus `paket` und alle Klassen aus `paket2` können hier benutzt werden
  - alle Klassen der Standardbibliothek außer jener in `java.lang` müssen importiert werden

Aufgabe

1. Erstellen Sie eine Java-Klasse `Point`, mit zwei Datenelementen für die x- und die y-Koordinate, einem Standard- und einem Initialisierungskonstruktor und einer Methode

Exceptions vs. Errors

Exceptions	Errors
eher leichte Laufzeitfehler	eher schwere Laufzeitfehler
können abgefangen werden	führen zum Programmabbruch
Beispiele:	
`ArithmeticException`	`NoClassDefFoundError`
`ArrayIndexOutOfBoundsException`	`OutOfMemoryError`
`StringIndexOutOfBoundsException`	`Internal Error`
`NumberFormatException`	
`EOFException`	

Exceptions sind Objekte

- Exception-Typ $\longrightarrow$ Klasse (Bsp.: `java.lang.NullPointerException`).
- Tritt ein Laufzeitfehler ein, wird ein Exemplar der jeweiligen Exception-Klasse erzeugt (*Throwing*).
- Alle Exception-Klassen sind Unterklassen von `java.lang.Exception`.
  - hierarchischer Aufbau der Exceptions
  - Verteilung auf verschiedene Pakete
- Sie besitzen daher alle gewisse Methoden, u.a.:
  - `getMessage()`, die bestimmte Informationen über den Fehlerfall liefert

    Beispiel: `java.lang.ArrayIndexOutOfBoundsException:` **-1**
  - printStackTrace(), die die Fehlermeldung und die dynamische Aufruf-hierarchie auf `stdout` ausgibt

Explizites Abfangen von Exceptions

```
try {
     // Anweisungen, in denen eine XYException
     // ausgelöst werden kann
}

catch (XYException e) {
     // Anweisungen, die beim Auftreten einer
     // XYException ausgeführt werden, z.B.
     System.out.println(e.getMessage());
}
```

mehrere Fehlerarten $\longrightarrow$ mehrere catch-Blöcke

Weitergeben von Exceptions

- Alternativ werden Exceptions zur Behandlung weitergegeben, und zwar
  - erst an übergeordnete Programmblöcke derselben Methode,
  - dann (entlang der dynamischen Aufrufhierarchie) an die Aufrufer der Methode,

  in der die Exception ausgelöst wurde.

  Wird sie nirgens explizit abgefangen, so bricht das Programm ab.
- die `throws`-Klausel signalisiert, welche Exceptions die jeweilige Methode nicht selbst behandelt:

  ```
  public void methode() throws IOException { ... }
  ```
- Auf alle Exceptions außer `RuntimeExceptions` muss der Programmierer reagieren!!!

Benutzerdefinierte Exceptions

- für Fehlersituationen im Zusammenhang mit benutzerdefinierten Klassen
- Auslösen mit `throw` <*Exemplar eine Exception-Klasse*>, z.B.:

```
if ( //Fehlersituation )
        throw new RuntimeException("Denominator is zero.");
```

- Jede Exception-Klasse besitzt Konstruktoren

XY`Exception()` und XY`Exception(String message)`

- benutzerdefinierte Exceptions als Unterklasse von einer Exception-Klasse:

```
MyException(){super()}   und
MyException(String msg){super(msg)}
```

Weitere syntaktische Besonderheiten

- Konstanten werden mit dem Schlüsselwort `final` vereinbart
- konstante Methoden können nicht überschrieben werden
- generische Typen werden als Klassen mit Typparametern definiert:

```
public class Pair<T,U>
```