Technische Hochschule Rosenheim
Technical University of Applied Sciences

Objektorientierte Programmierung
Kapitel 8 – Exceptions

Prof. Dr. Kai Höfig

Motivation: Exception

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Ziel: Robuste** Programme, die
  - auf Laufzeitfehler vorbereitet sind
  - und kontrolliert darauf reagieren.

- Auftreten von Fehler
  - **beim Kompilieren:** Syntax/Fehler bei Typüberprüfung durch Compiler (Generics!)
  - **zur Laufzeit**: Logische Fehler im Programm, fehlerhafte Bedienung durch Benutzer, Datei- oder Netzwerkoperationen

- **Java Exception**
  - Sprachmittel zur kontrollierten Reaktion auf Laufzeitfehler.
  - Ohne den Programmfluss zu stören!

- Angemessene **Reaktion** abhängig von Fehlerart
  - Logischer Fehler → Programm anhalten
  - Bedienungsfehler → Aufforderung zur Korrektur
  - Problem in JVM → Kaum sinnvolle Maßnahmen möglich

Umgang mit Fehlern in C

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Returncode**
  - Fehler in C Routine wird durch einen Fehlercode signalisiert.
  - Beispiel: Wert ungleich 0 signalisiert Fehler.
  - Beispiel: Zu einem bestimmten Namen wird keine Telefonnummer gefunden.

- **Nachteile** von Returncodes
  - "*Rückgabewert* = *Ergebnis* + *Fehlercode*" → Rückgabewert überladen.
  - Prüfung liegt in Verantwortung des Aufrufers.

- **Faustregel**
  - *Returncodes* für Fehler, die **legale** Situation der Anwendung darstellen.
  - *Returncodes* eher nicht geeignet, um falsche Benutzereingabe abzufangen.

```
public Contact findPhoneNumber (String name) {
    . . .
    if (found)
        return current;
    else
        return null;
. . .
```

Returncodes sinnvoll!

```
public Contact findPhoneNumber (String name) {
    . . .
    if (CustomerDB.connect()) {
        // search in database
    } else
        return null; // error with connection
    } . . .
```

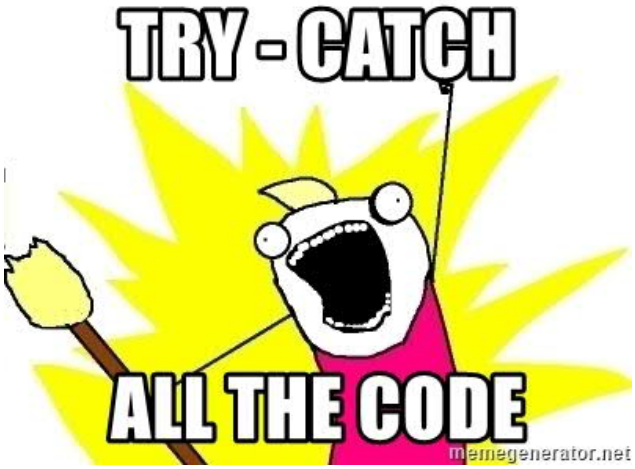
Exception besser geeignet, Returncode würde Programmfluss unterbrechen!

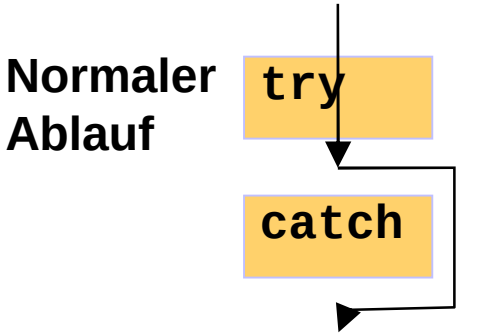
Exceptions mit `try` und `catch`

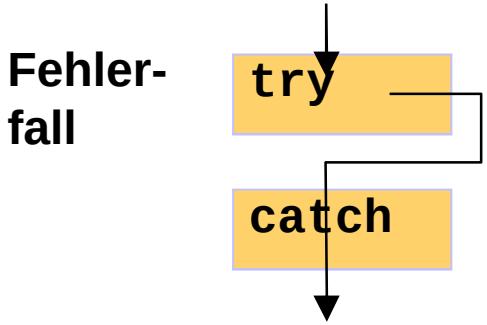
Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Bei Fehler in ***überwachtem Programmbereich*** (`try`) wird spezieller Programmcode zur Fehlerbehandlung (`catch`) aufgerufen.

```
try {
    // Programmcode, der eine Ausnahme auslösen kann
}
catch {
    // Programmcode zum Behandeln der Ausnahme
}
// es geht normal weiter, Ausnahme wurde behandelt
```







Beispiel für eine Ausnahme / Stack Trace

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
public static void divideBy500(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Durch welchen Integer soll ich 500 dividieren?");
    int div = scanner.nextInt();
    System.out.println("Ergebnis: " + (500 / div));
}
```

- Welche Ausnahmen können hier auftreten?
  - Benutzer tippt kein Integer ein, sondern z.B. "19$s%" → `InputMismatchException`
  - Benutzer tippt 0 ein → `ArithmeticException`

- **Stack Trace**
  - VM merkt sich auf Stack, von welcher Methode aktuelle Methode aufgerufen wurde.
  - Information über Name der `Exception` und Stelle im Programm, an dem Fehler auftrat.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionTest.main(ExceptionTest.java:12)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at com.intellij.rt.execution.application.AppMain.main(AppMain.java:144)
```

Abfangen von Ausnahmen: `try` / `catch`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
try {
    int div = scanner.nextInt();
    System.out.println("Ergebnis: " + (500 / div));
} catch (InputMismatchException e) {
    e.printStackTrace();
} catch (ArithmeticException e) {
    System.out.println("Name of exception: 0" + e.getClass().getName());
    System.out.println("Message content: " + e.getMessage());
    System.out.println("String representation: c" + e.toString());
}
```

- Java Dokumentation: Hinweis, was welche Ausnahmen auslöst.
- `Exception` Objekt `e` speichert zahlreiche Informationen
  - Welche Ausnahme? → `e.getClass().getName()`
  - Fehlernachricht? → `e.getMessage()`
  - Wie sieht Stack Trace aktuell aus? → `e.printStackTrace()`
- Abfangen verschiedener Fehlertypen möglich
  - Ein `try`-Block kann **mehreren** `catch`-Klauseln zugeordnet sein!
  - **Hinweis**: Es wird nur die **erste passende** catch-Klausel ausgeführt!
  - Falls Fehlerbehandlung für mehrere Exceptions gleich ist:
    - `catch (E1 | E2 | … | En e) {. . .}`

Abfangen von Ausnahmen: `finally`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Finally**-Block (optional)
  - Anweisungen in diesem Block werden **immer** ausgeführt, unabhängig davon ob Ausnahme auftritt oder nicht.
  - Steht am Ende der Liste der `catch`-Blöcke.
  - Abwicklung von Aufräumarbeiten.
  - Vermeidet Code-Redundanz.

```
try …

catch …

finally {
   … Anweisungen …
}
```

```
try {
    customerDB.connect();
    // some DB queries
    customerDB.close();
} catch (Exception e) {
    // do some error handling
    customerDB.close();
}
```

Code-Redundanz!
Ohne `finally`, DB-Verbindung sowohl im `try`- als auch im `catch`-Teil schließen!

```
try {
    customerDB.connect();
    // some DB queries
} catch (Exception e)  {
    // do some error handling
} finally {
    customerDB.close();
}
```

Weiterleiten der Ausnahme an Aufrufer

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- `try / catch`
  - Einzäunen der problematischen Bereiche und lokale Reaktion auf Ausnahme.
- `throws`
  - Weiterleiten der Ausnahme an den Aufrufer.
  - `throws`-Klausel in Signatur einer Methode verdeutlicht, was passieren kann.
  - Aufrufer muss sich kümmern, wird zur Reaktion **gezwungen**.

```
public static void divideBy500() throws InputMismatchException, ArithmeticException {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Durch welchen Integer soll ich 500 dividieren?");
    int div = scanner.nextInt();
    System.out.println("Ergebnis: " + (500 / div));
}
public static void main(String[] args) {
    try {
        divideBy500();
    }
    catch (ArithmeticException e){
        System.out.println("OK, ich kümmere mich um Division durch 0")
    }
    catch (InputMismatchException e) {
        ...
}}
```

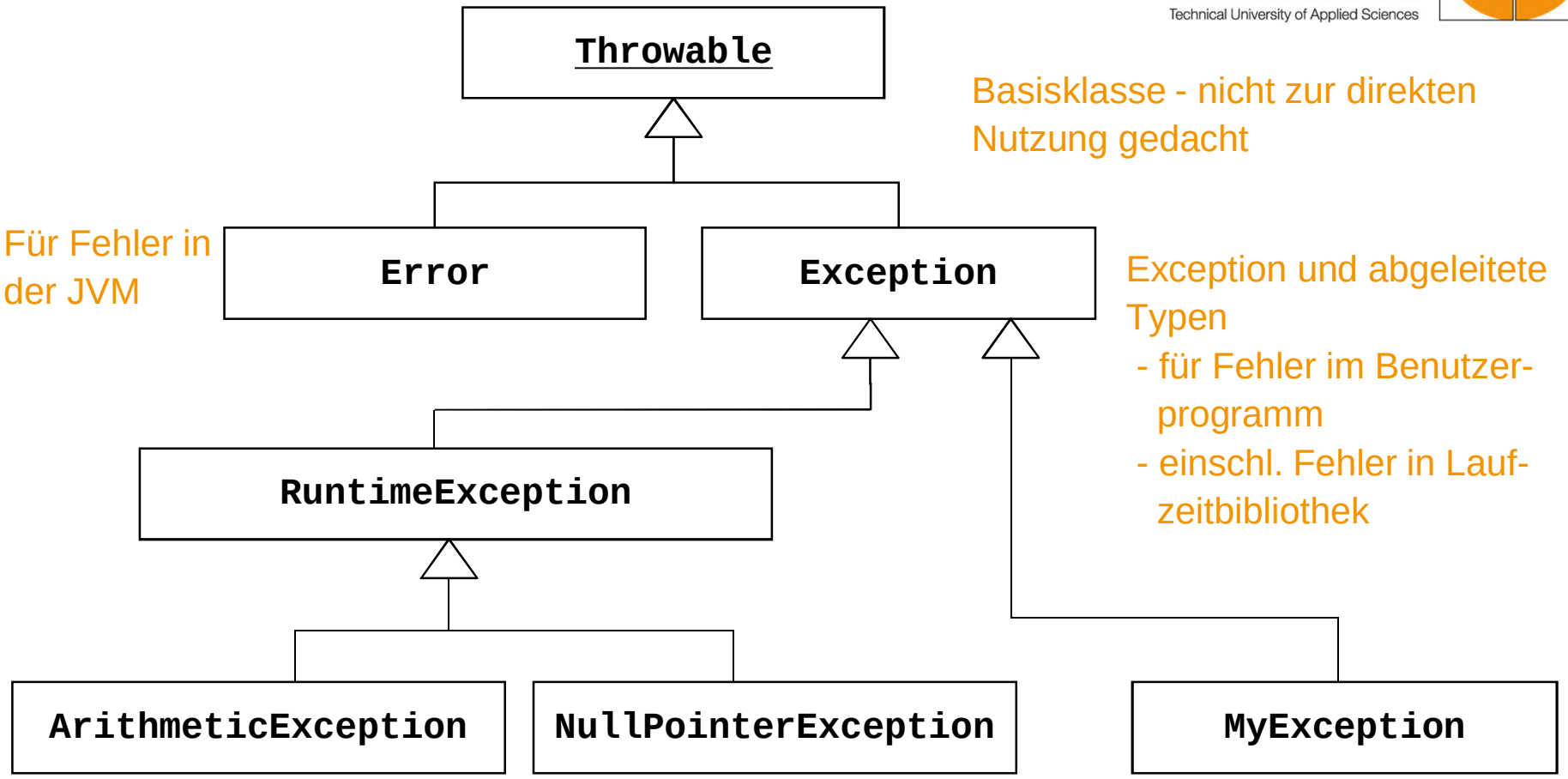
Geprüfte vs. Ungeprüfte Ausnahmen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Geprüfte ("checked") Ausnahmen**
  - Compiler besteht auf Abfangen der Ausnahme.
  - Abfangen durch `try/catch` bzw. Delegation an Aufrufer durch `throws`.
  - Beispiel: `IOException`
    - Falls Datei nicht existiert → kein sinnvoller, weiterer Programmverlauf. Der Fehler muss abgefangen werden!

- **Ungeprüfte Ausnahmen**
  - Ausnahmebehandlung optional, Compiler besteht ***nicht*** auf Abfangen der Ausnahme.
  - Ansonsten: Ungeprüfte Ausnahmen müssten in Signatur von fast jeder Methode stehen (`throws`)
  - Ursache von ungeprüften Ausnahmen oft: Denkfehler des Programmiers.
  - Beispiel: **Alle Unterklassen** von `RunTimeException`
    - `ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, IllegalArgumentException, ClassCastExcpetion`

Exception: Auszug aus der Klassenhierarchie



Hinweis: Es genügt einen Fehler der Oberklasse abzufangen!

"Harte" Ausnahmen: `Error`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Bedeutung**
  - `Error` und abgeleitete Typen vorgesehen für **Fehler der JVM**
  - Benutzerprogramm sollte nicht explizit **`Error`** auslösen
- Meist keine sinnvolle Reaktion möglich
  - Benutzerprogramm sollte daher `Error` nicht fangen und behandeln
  - `Error` ist grundsätzlich eine **ungeprüfte Exception**.
- **Beispiele**
  - `OutOfMemoryError`
    - JVM hat allen verfügbaren Speicher verbraucht.
  - `ClassFormatError`
    - Versuch, defekten Bytecode zu laden
  - `VirtualMachineError:`
    - Interner Fehler der JVM

Auslösen von Ausnahmen mit `throw`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wie löst man **selbst** eine Ausnahme aus?
  - Erzeugen eines Objekts der Klasse (oder Unterklasse) von `Exception`.
  - Standardfälle können durch vorhandene Exception Klassen der Java API abgedeckt werden.
  - Starten der Ausnahmebehandlung: Schlüsselwort `throw`
  - Nicht verwechseln mit `throws`!

- Beispiel: Man darf nur *volljährige* Objekte der Klasse **Person** erzeugen.

```
public class Person extends Object implements Comparable<Person>{
    private String name;
    private int age;

    // constructor(s)
    public Person(String n, int a) {
        if (age < 18) {
            throw new IllegalArgumentException("Minderjährige Person nicht erlaubt!");
        }
        name = n;
        age = a;
    }
. . .
```

Definition eigener Ausnahmeklassen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Nötig, falls vordefinierte Ausnahmeklassen nicht passend.
- **Vorteil:** Eigene Ausnahme hat eigenen Datentyp `MyException`.
- **Vorgehen:** Ableiten von vorhandener `Exception`-Klasse
  - Entscheidung ob *geprüfte* oder *ungeprüfte Ausnahme.*
  - *Geprüfte Ausnahme:* Unterklasse von Exception
  - *Ungeprüfte Ausnahme*: Unterklasse von `RuntimeException`.

```
public class MyException extends Exception {

    public MyException() {};

    public MyException(String s) {
        super(s);
    }
}
```

Verwendung

```
if (age < 18) {
    throw new MyException(
        "Minderjährige Person nicht erlaubt!");
}
```

Erinnerung: Testen von Ausnahmen / Exceptions

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
@Test(expected = ExceptionType.class)
public void name() {
    ...
}
```

- Ergibt "Pass" falls die Ausnahme / Exception tatsächlich eintritt.
- Wichtig, falls im eigenen Programm Exceptions mit `throw` ausgelöst werden.

```
@Test(expected = ArrayIndexOutOfBoundsException.class)
public void testBadIndex() {
    int[] array = new int[4];
    int i = array[4];   // should fail
}
```

Zusammenfassung: Umgang mit Fehlern

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- **Vermeiden** von Fehlern
  - Tests mit JUnit

- **Exception**
  - Sprachmittel zur kontrollierten Reaktion auf Laufzeitfehler
  - Vorteile: Laufzeitfehler können nicht ignoriert werden und Code für regulären Programmablauf ist textuell getrennt vom Code zur Fehlerbehandlung.
  - Schritt 1: Ausnahmensituationen werden abgeprüft und ggf. geworfen (**throw**)
  - Schritt 2: Behandeln von Fehlern/Ausnahmesituationen (**try … catch**)
  - Geprüfte und ungeprüfte Exceptions

- **Returncodes**
  - Zur Meldung von „legalen Fehlern“

Ein seltsames Programm …

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Was gibt die folgende Methode zurück?

```
static String getIsbn() {
    try {
        return "3821829877";
    }
    finally {
        return "";
    }
}
```