

Einführung in die Programmierung für NF

Fehler und Ausnahmen

Ziele

- Fehlerquellen in Programmen und bei der Programmausführung verstehen
- Das Java-Konzept der Ausnahmen als Objekte kennenlernen
- Ausnahmen auslösen können
- Ausnahmen behandeln können

ROBUSTE PROGRAMME

Robuste Programme

- Wir beschäftigen uns hier nicht damit, wie man logische Fehler in einem Programm finden kann.
- Wir konzentrieren uns hier auf das Erkennen, Vermeiden und Behandeln von Ausnahmesituationen, die zu Laufzeitfehlern führen können.
- Unser Ziel ist es robuste Programme zu schreiben.

Definition:

Ein Programm heißt **robust**, falls es auch beim Auftreten von Fehlern sinnvoll reagiert.

Robuste Programme

- Einführung von zusätzlicher Fallunterscheidung und Fehlermeldung

```
int x = Eingabe.intEinlesen("Geben Sie eine Zahl ein:");
if(x<0){
    System.out.println("Falscher Eingabewert");
}
else{
    int y = Eingabe.intEinlesen("Geben Sie noch eine Zahl ein");
    while(x>0)...
```

meidet Fehler durch Seiteneffekt

Robuste Programme

- Besser: Kontrolliertes Auslösen von Ausnahmen

```
int x = Eingabe.intEinlesen("Geben Sie eine Zahl ein:");  
if(!x>0) throw new IllegalArgumentException("NegativerEingabewert");  
int y = Eingabe.intEinlesen("Geben Sie noch eine Zahl ein");  
while(x!=0)...
```

Löst bei negativem x eine Ausnahme aus („aprupte“ Terminierung! – nicht robust)

```
Exception thread "main"
```

```
java.lang.IllegalArgumentException: Negativer  
Eingabewert
```

FEHLER

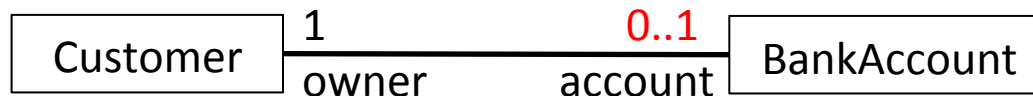
Fehlerarten

Ein Programm kann aus vielerlei Gründen fehlerhaft sein. Man unterscheidet u.a.:

- **Entwurfsfehler:** Der Entwurf entspricht nicht den Anforderungen
- **Programmierfehler:** Das Programm erfüllt nicht die Spezifikation

Beispiel Entwurfsfehler:

- Anforderung: ...Ein Kunde kann **mehrere** Bankkonten besitzen...
- Fehlerhafter Entwurf:



Fehlerarten

Programmierfehler können auch unterschiedlicher Art sein:

- **Syntaxfehler:** Die kontextfreie Syntax des Programms ist nicht korrekt
Beispiel: `whill(x >= 0)`
- **Typfehler:** Ein Ausdruck oder eine Anweisung des Programms hat einen falschen Typ
Beispiel: `while(x > true)`
- **Ein/Ausgabefehler:** z.B. wenn eine Klassendatei nicht gefunden wird
- **Logischer Fehler:** Das Programm erfüllt nicht die (Entwurfs-) Spezifikation
Beispiel: Falsche Implementierung einer Sortierfunktion

Fehler- und Ausnahmenklassen in Java

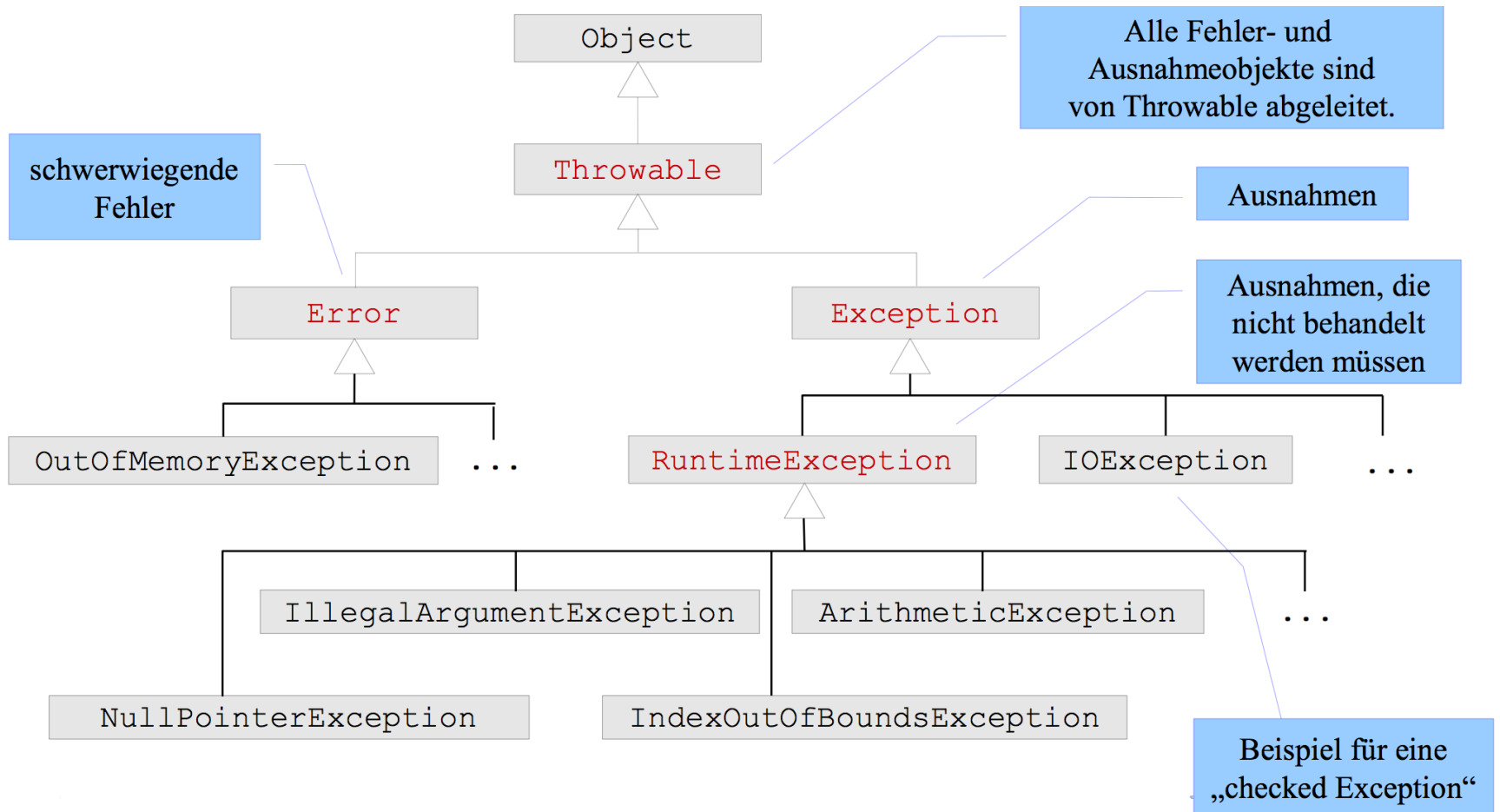
In Java unterscheidet man zwischen Fehlern und Ausnahmen, die beide durch **Objekte** repräsentiert werden.

- Fehler sind Instanzen der Klasse `Error`
- Ausnahmen sind Instanzen der Klasse `Exception`

Fehler deuten auf schwerwiegende Probleme der Umgebung hin und können nicht sinnvoll behandelt werden, z.B. `OutOfMemoryException`

Ausnahmen können vom Programmierer im Programm durch Ausnahmebehandlung abgefangen werden

Vererbungshierarchie der Fehlerklassen



Die Klasse Throwable

- Ausnahme- und Fehler-Objekte enthalten Informationen über Ursprung und Ursache des Fehlers.
- Die Klasse `Throwable`, von der alle Fehlerklassen abgeleitet sind, verwaltet solche Informationen, z.B.:
 - eine **Nachricht** zur Beschreibung des aufgetretenen Fehlers (`Throwable(String message)`)
 - einen **Schnappschuss** des Aufrufstacks zum Zeitpunkt der Erzeugung des Objekts
- Nützliche Methoden in `Throwable`:
 - `String getMessage()`: gibt die Fehlermeldung zurück
 - `void printStackTrace()`: gibt den Aufrufstack des Fehlers aus

Auslösung einer RuntimeException und Ausgabe des Aufrufstacks

```
public class Div0 {
    /** Die Methode m loest wegen der Division durch 0 eine
     * ArithmeticException aus: */
    public static void m() {
        int d = 0;
        int a = 42 / d;
        System.out.println("d= „+ d);
        System.out.println("a= „+ a);
    }
    public static void main(String args[]) {
        this.m();
    }
}
```

Java-Ausgabe mit Aufrufstack:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Div0.m(Div0.java:6)
at Div0.main(Div0.java:11)
```

Der Aufrufstackenthält die Folge der Methodenaufrufe, die zum Fehler geführt haben

AUSNAHMEN

Kontrolliertes Auslösen von Ausnahmen

- Mittels der `throw`-Anweisung kann man eine Ausnahme auslösen.
- **Syntax:** `throw exp;`
- Der Ausdruck `exp` muss eine Instanz einer von `Throwable` abgeleiteten Klasse (d.h. eine Ausnahme oder ein Fehlerobjekt) bezeichnen.
- **Beispiel:** `throw new IllegalArgumentException("...");`
- Die Ausführung einer `throw`-Anweisung stoppt den Kontrollfluss der Methode und löst die von `exp` definierte Ausnahme aus. Die nachfolgenden Anweisungen im Rumpf der Methode werden nicht mehr ausgeführt (wie bei `return`).
- Es kommt zu einem Abbruch des Programms, wenn die Ausnahme nicht in einer übergeordneten Methode abgefangen und behandelt wird.

Geprüfte Ausnahmen (Checked Exceptions)

- Geprüfte Ausnahmen sind in Java alle Instanzen der Klasse `Exception`, die nicht Objekte der Klasse `RuntimeException` sind.
- Gibt es in einem Methodenrumpf eine `throw`-Anweisung mit einer geprüften Ausnahme, dann **muss** das im Methodenkopf mit `throws` explizit deklariert werden.
- Geprüfte Ausnahmen **müssen** vom Aufrufer der Methode entweder behandelt werden oder wieder im Methodenkopf deklariert werden.

- Beispiele:

```
import java.io.IOException;
...
public void m() throws IOException{
    if(...) {
        throw new IOException();
    }
}
public void n() throws IOException{
    this.m();
}
```

Man muss deklarieren, dass in dieser Methode die Ausnahme `IOException` auftreten kann.

Da die `IOException`, die beim Aufruf von `m()` auftreten kann, hier nicht behandelt wird, muss die Methode selbst wieder eine `throws`-Klausel deklarieren.

Ungeprüfte Ausnahmen

- Ungeprüfte Ausnahmen sind genau die Instanzen von `RuntimeException`.
- Beispiele: `ArithmeticException`, `NullPointerException`, ...
- Ungeprüfte Ausnahmen müssen nicht im Methodenkopf explizit deklariert werden (sie können es aber).
- Ungeprüfte Ausnahmen müssen nicht behandelt werden (sie können es aber).

Benutzerdefinierte Ausnahmeklassen

Mittels Vererbung kann man eigene Ausnahmeklassen definieren.

Beispiel:

- Klassen BankKonto und SparKonto. Es soll nicht möglich sein, ein SparKonto zu überziehen.
- Wir definieren dazu eine (checked) Exception, die beim Versuch das SparKonto zu überziehen, geworfen werden soll:

```
public class KontoUngedecktException extends Exception{
    private double abhebung;

    public KontoUngedecktException(String msg, double abhebung) {
        super(msg);    // Konstruktor von Exception nimmt Nachricht
        this.abhebung = abhebung;
    }

    public double getAbhebung() {
        return abhebung;
    }
}
```

Auslösen einer benutzerdefinierten Ausnahme

```
public class BankKonto{
    ...
    public void abheben(double x) throws KontoUngedecktException{
        kontoStand = kontoStand-x;
    }
}

public class SparKonto extends BankKonto{
    ...
    public void abheben(double x) throws KontoUngedecktException{
        if(getKontoStand() < x) {
            throw new KontoUngedecktException("Sparkonten dürfen nicht
            überzogen werden.", x);
        }
        super.abheben(x);
    }
}
```

Behandlung von Ausnahmen

Ausnahmebehandlung geschieht in Java mit Hilfe der `try`-Anweisung. Damit können Ausnahmen **abgefangen** werden.

```
try{
    // Block fuer „normalen“ Code
}
catch(Exception1 e) {
    // Ausnahmebehandlung fuer Ausnahmen vom Typ Exception1
}
catch(Exception2 e) {
    // Ausnahmebehandlung fuer Ausnahmen vom Typ Exception2
}
```

- Zunächst wird der `try`-Block normal ausgeführt.
- Tritt im `try`-Block keine Ausnahmesituation auf, so werden die beiden Blöcke zur Ausnahmebehandlung ignoriert.
- Tritt im `try`-Block eine Ausnahmesituation auf, so wird die Berechnung dieses Blocks abgebrochen.
 - Ist die Ausnahme vom Typ `Exception1` oder `Exception2`, so wird der Block nach dem jeweiligen `catch` ausgeführt.
 - Ansonsten ist die Ausnahme unbehandelt.

Behandlung von Ausnahmen: Beispiel Konto

```
public static void main(String[] args) {  
  
    SparKonto konto = new SparKonto(5, 1); // 5 Euro, 1% Zinsen  
  
    String einleseBetrag = JOptionPane.showInputDialog("Betrag zum Abheben?");  
    double betrag = Double.parseDouble( einleseBetrag);  
  
    try{  
        konto.abheben(betrag);  
    }  
    catch(KontoUngedecktException e) {  
        System.out.println(e.getMessage());  
        System.out.println("Der Abhebungsbetrag " + e.getAbhebung() +" war zu hoch.");  
    }  
}
```

Behandlung von Ausnahmen: finally

Manchmal möchte man nach der Ausführung eines try-Blocks bestimmte Anweisungen ausführen, egal ob eine Ausnahme aufgetreten ist.

- Beispiel: Schließen einer im try-Block geöffneten Datei.
- Das kann man mit einem finally-Block erreichen, der in jedem Fall nach dem try-Block und der Ausnahmebehandlung ausgeführt wird.

```
try{
    // Block fuer „normalen“ Code
}
catch(Exception1 e) {
    // Ausnahmebehandlung fuer Ausnahmen vom Typ Exception1
}
catch(Exception2 e) {
    // Ausnahmebehandlung fuer Ausnahmen vom Typ Exception2
}
finally{
    // Code, der in jedem Fall nach normalem Ende und nach
    // Ausnahmebehandlung ausgefuehrt werden soll.
}
```

Beispiel für finally

Ablauf in einem Geldautomaten:

```
public static void main(String[] args) {  
  
    SparKonto konto = new SparKonto(5, 1);           // 5 Euro, 1% Zinsen  
  
    String einleseBetrag = JOptionPane.showInputDialog("Betrag zum Abheben?");  
    double betrag = Double.parseDouble(einleseBetrag);  
  
    try{  
        konto.abheben(betrag);  
    }  
    catch(KontoUngedecktException e) {  
        System.out.println(e.getMessage());  
        System.out.println("Der Abhebungsbetrag " + e.getAbhebung() + " war zu hoch. ");  
    }  
    finally{  
        System.out.println("Bitte entnehmen Sie ihre Karte.");  
    }  
}
```

Zusammenfassung

- Ausnahmen werden in Java durch Objekte dargestellt.
- Methoden können Ausnahmen auslösen implizit durch einen Laufzeitfehler oder explizit mit `throw` und damit „abrupt“ terminieren.
- Ausnahmen können mit `catch` behandelt werden, so dass sie nicht zu einem Abbruch des Programms führen.
- Wir unterscheiden geprüfte und ungeprüfte Ausnahmen.
- Geprüfte Ausnahmen müssen abgefangen werden oder im Kopf der Methode wiederum deklariert werden.
- In jedem Fall ist es am Besten Ausnahmen zu vermeiden.
- Defensive Programme sehen auch für vermeidbare Ausnahmesituationen das Werfen von Ausnahmen vor (was dann hoffentlich nie nötig ist).

Vielen Dank für Ihre Aufmerksamkeit