

11. Ausnahmebehandlung

- Motivation
- Behandeln mit try-Anweisungen
- Werfen von Ausnahmen
- Weiterreichen mit throws



11.1 Motivation

- Unterscheidung zwischen zwei Arten von Fehlern
 - **Übersetzungsfehler**
 - Der Compiler erkennt fehlerhafte Anweisungen/Deklarationen im Programm und erzeugt deshalb auch keinen ausführbaren Code.
 - Der Programmentwickler wird über die entsprechenden Fehler informiert.
 - **Laufzeitfehler**
 - Die Ausführung eines Programms erzeugt unerwartete Ergebnisse bzw. bricht völlig ab.
 - Der Benutzer des Programms bekommt eine unverständliche Fehlermeldung und kann diesen Fehler nicht selbst beheben.



```
Run TestRealFunc
Exception in thread "main" 2.0 + 3.0*x^1 + 4.0*x^2 + 5.0*x^3
8.0 + 30.0*x^1
Wert des Polynoms p(42.0) = 377624.0
2.0e^(3.0*x)
6.0e^(3.0*x)
java.lang.ClassCastException: Exp cannot be cast to Polynom
    at TestRealFunc.main(TestRealFunc.java:21) <5 internal calls>
Process finished with exit code 1
```

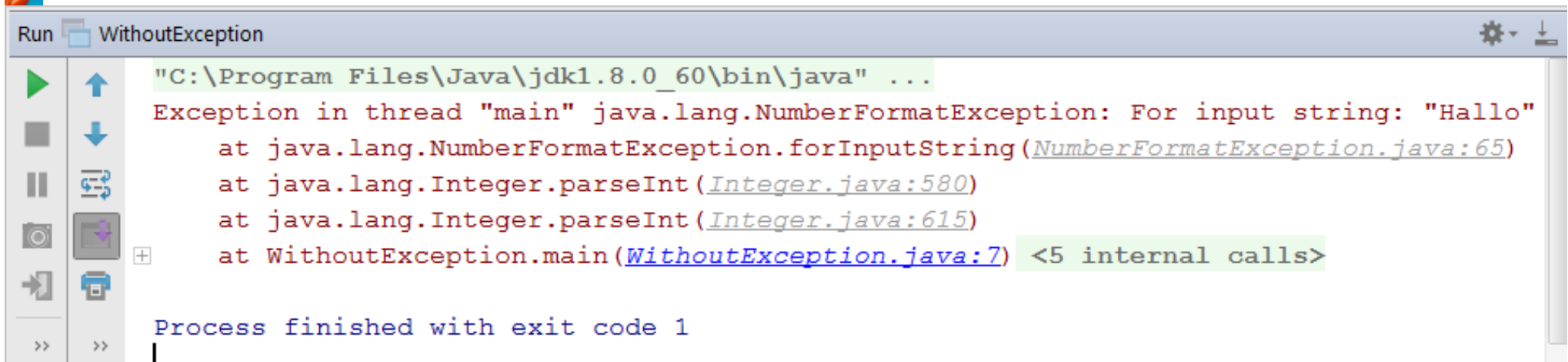


**Und solche Laufzeitfehler
führen zur Verzweiflung!**

Schon mal gesehen?

```
public class WithoutException {  
  
    public static void main(String[] args) {  
        int i = Integer.parseInt(args[0]);  
        System.out.println("Mein Zahl: " + i);  
    }  
}
```

- Zwei Ausgaben des Programms
 - Beim Start des Programms mit dem Argument 42.
Mein Zahl: 42
 - Beim Start des Programms mit dem Argument "Hallo Welt".



The screenshot shows a Java IDE window titled "Run WithoutException". The command line at the top is "C:\Program Files\Java\jdk1.8.0_60\bin\java" ... The main output area displays a red exception message: "Exception in thread "main" java.lang.NumberFormatException: For input string: "Hallo" at java.lang.NumberFormatException.forInputString (NumberFormatException.java:65) at java.lang.Integer.parseInt (Integer.java:580) at java.lang.Integer.parseInt (Integer.java:615) at WithoutException.main (WithoutException.java:7) <5 internal calls>". Below the exception, it says "Process finished with exit code 1". The left sidebar contains standard IDE icons for running, stepping through, and debugging the code.

```
Run WithoutException  
"C:\Program Files\Java\jdk1.8.0_60\bin\java" ...  
Exception in thread "main" java.lang.NumberFormatException: For input string: "Hallo"  
    at java.lang.NumberFormatException.forInputString (NumberFormatException.java:65)  
    at java.lang.Integer.parseInt (Integer.java:580)  
    at java.lang.Integer.parseInt (Integer.java:615)  
    at WithoutException.main (WithoutException.java:7) <5 internal calls>  
  
Process finished with exit code 1
```

Was sieht man in der Ausgabe?

- Bei einem Laufzeitfehler im Java-Programm, der nicht behandelt wird, wird der sogenannte **Stack-Trace** ausgegeben.
- Der Stack-Trace ist die Liste der **Methodenaufrufe**, die beim Zeitpunkt des Programmabbruchs aktiv waren, mit der **Zeilennummer des letzten Befehls** in der Methode.
 - In unserem Fall bestand die Methode aus folgenden Aufrufen.
 - main-Methode unserer Klasse WithoutException (Zeile 7)
 - parseInt-Methode der Klasse Integer (Zeile 615)
 - parseInt-Methode der Klasse Integer (Zeile 580)
 - forInputString-Methode der Klasse NumberFormatException (Zeile 65)
 - Diese Methode hat zum Abbruch des Programms geführt.

Typische Ursachen für Laufzeitfehler

- Arrayzugriff mit einem Index außerhalb des erlaubten Bereiches
 - Zugriff auf nicht richtig initialisierte Variablen
 - Zugriff auf ein Objekt wird versucht, aber Referenz ist „null“
 - Konvertieren von Daten mit verschiedenen Repräsentationen
 - Cast-Operation eines Objekts in eine nicht-erlaubte Klasse
 - Schreibversuch in eine schreibgeschützte Datei oder Zugriff auf eine nichtvorhandene Datei
 - Missachtung von Vor- und Nachbedingungen
 - Inkorrekte Spezifikation von Methodenparametern
- ➔ Tritt während der Programmausführung ein Laufzeitfehler auf, so gerät das Programm in einen **Ausnahmezustand**.

Ausnahmen und Fehler

- Fehler (Error)
 - ist ein nicht reparierbarer Laufzeitfehler oder ein Hardware-Problem, das zum Absturz des Programms führt.
- Ausnahme (Exception)
 - ist meistens kein eigentlicher Fehler, sondern ein unerwarteter Fall, auf den das Programm reagieren sollte.

Erster Lösungsansatz

- In Programmiersprachen ohne explizite Unterstützung für die **Ausnahmebehandlung** wird typischerweise ein Fehlercode bei Methoden zurückgegeben.
- In dem folgenden Programmfragment ist das der Wert **-1**.

```
/** Die Methode durchsucht ein Array nach einem Element.
 * @param arr Array mit ganzen Zahlen
 * @param was Das zu suchende Element in dem Array
 * @return Index im Array, an dem das gesuchte Element liegt.
 */
int search(int[] arr, int was){
    int n = arr.length;
    for (int i = 0; i < n; i++)
        if (arr[i] == was) return i;
    return -1;
}
```

Nachteile von Fehlercodes (1)

- Vermischung von Programmlogik und Ausnahmebehandlung führt zu unleserlichen Programmen mit vielen bedingten Anweisungen
- Hier ein Beispiel als Pseudocode, wie so etwas aussehen könnte:

```
int readFile {  
    int errorCode = 0;  
  
    open the file;  
    if (theFileIsOpen) {  
        compute length of file;  
        if (gotTheFileLength) {  
            allocate memory;  
            if (gotEnoughMemory) {  
                read file into memory;  
                if (readFailed) {  
                    errorCode = -1;  
                }  
            }  
            else {  
                errorCode = -2;  
            }  
        }  
    }  
}
```

```
        else {  
            errorCode = -3;  
        }  
        close the file;  
        if (fileDidntClose && errorCode == 0) {  
            errorCode = -4;  
        }  
        else {  
            errorCode = errorCode &&-4;  
        }  
    }  
    else {  
        errorCode = -5;  
    }  
    return errorCode;  
}
```


Nachteile von Fehlercodes (2)

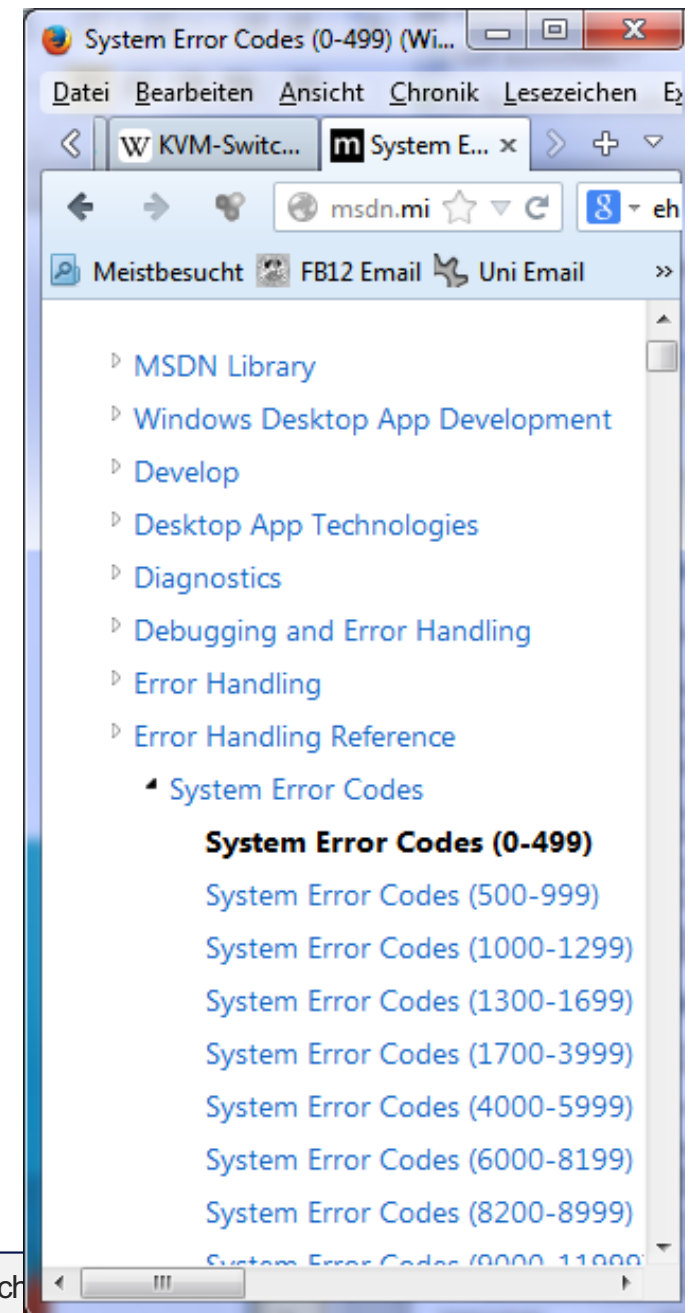
- Aufwändiges Durchreichen von Ausnahmen
- Hier ein Beispiel als Pseudocode, wie so etwas aussehen könnte:

```
void method1 () {  
    ErrorCodeType error = method2 ();  
    if (error < 0)  
        doErrorProcessing;  
    else  
        proceed1(); // Pseudocode  
}  
  
int method2() {  
    int error = method3();  
    if (error < 0)  
        return error;  
    else  
        proceed2(); // Pseudocode  
}
```

```
int method3() {  
    int error;  
    error = readFile( ....) ;  
    if (error < 0)  
        return error;  
    else  
        proceed3(); // Pseudocode  
}
```

Nachteile von Fehlercodes (3)

- Fehlercodes bieten **keinerlei Strukturierung** an.
 - Keine Gruppierung und Klassifizierung von Codes

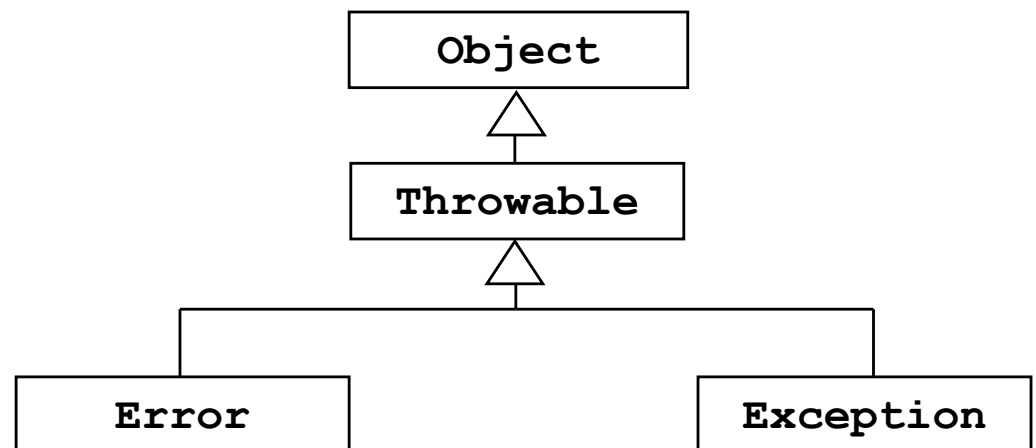


Ziele

- Bereitstellung einer Methodik, um Ausnahmen eleganter zu behandeln als mit Fehlercodes
 - z. B.: Ausgabe eines aussagekräftigen Texts zum Fehler
- Ausnahmen sollten nicht zum Abbruch des Programms führen.
 - Effektive Behandlung von Ausnahmen im Programm
- Trennung vom normalen Programmcode und dem Code zur Ausnahmebehandlung.
 - Vermeidung stark verschachtelter if-else-Anweisungen

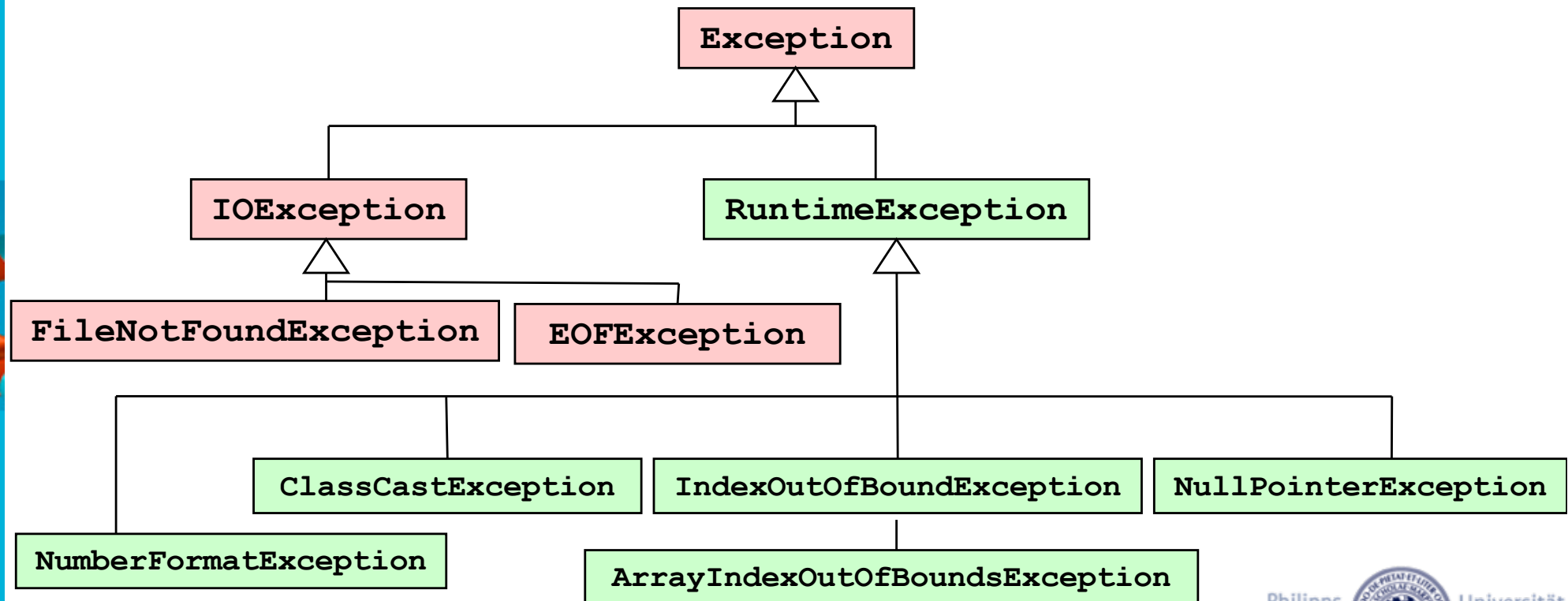
11.2 Ausnahmebehandlung in Java

- Java stellt für die Erkennung von (erwarteten) Fehlern, eine sogenannte **Ausnahmebehandlung** zur Verfügung.
 - Mittels den Oberklassen **Throwable**, **Error** und **Exception** und davon abgeleiteten Unterklassen werden Ausnahmeobjekte zur Verfügung gestellt.
 - Ausnahmen können **erzeugt** („geworfen“) werden: „to throw an exception“.
 - Falls eine Ausnahme auftritt, kann dieses Ereignis **abgefangen** werden: „to catch an exception“.
- Die Klasse **Throwable** ist die Oberklasse aller für Fehler und Ausnahmen zuständigen Klassen.
 - **Geworfen** werden können nur Objekte dieser Klasse bzw. ihrer Unterklassen.
 - Analog: Nur Throwable-Objekte können **abgefangen** und **behandelt** werden.



Die Klassenhierarchie Exception

- ▶ In dieser Übersicht sind einige aber nicht alle Unterklassen von **Exception** aufgeführt.
- ▶ Die Klasse `RuntimeException` und deren Unterklassen haben einige Besonderheiten, auf die wir später noch eingehen werden.



Ein erstes Beispiel

```
public class WithException {  
  
    public static void main(String[] args) {  
        int i;  
        try {  
            i = Integer.parseInt(args[0]);  
        }  
        catch (NumberFormatException except) {  
            i = 7;  
            System.out.println("Defaultzahl = " + i);  
        }  
        System.out.println("Meine Zahl: " + i);  
    }  
}
```

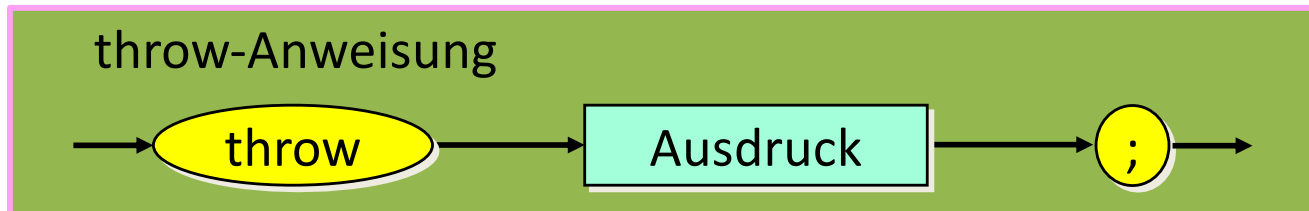
- Verwendung der try-catch-Anweisung
 - try-Block besteht aus dem eigentlichen Programm
 - catch-Block besteht aus der Fehlerbehandlung.
 - Parameter des catch-Blocks ist ein Objekt einer Exception-Klasse

Übersicht zu Exceptions in Java

- Eine Ausnahme kann **geworfen** werden.
 - Durch eine throw-Anweisung wird ein Objekt einer Exception-Klasse erzeugt.
- Eine Ausnahme kann **gefangen** werden.
 - Hierfür gibt es den try-catch-Block
- Eine Ausnahme kann **weitergereicht** werden.
 - Für sogenannte **ungeprüfte Ausnahmen** (Objekte der Klasse RuntimeExceptions oder einer Unterklasse) ist dies ohne weitere Angaben möglich.
 - Für alle anderen Exceptions (**geprüfte Ausnahmen**) wird noch das throws-Schlüsselwort im Methodenkopf benötigt.

11.2.1 Werfen von Ausnahmen

- Das Werfen einer Ausnahme erfolgt durch die **throw-Anweisung**.



- In dem Ausdruck muss (in der Regel) ein **Konstruktor der Exception-Klasse** oder einer Klasse, welche die Klasse Exception erweitert, aufgerufen werden.
- Bei der Ausführung der Anweisung wird ein Objekt der entsprechenden Exception-Klasse erzeugt und der übliche **Ablauf der Methode gestoppt**.
 - Die Fortführung der Programmausführung hängt davon ab, ob die Ausnahme
 - **gefangen**
 - **oder weitergereicht** wird.

Beispiel

- Beim Erzeugen eines Objekts der Klasse Edge können zwei Runtime-Exceptions geworfen werden:
 - Falls einer der Punkte den Wert null hat.
 - NullPointerException
 - Falls die Punkte gleich sind.
 - EqualPointException ist eine eigene RuntimeException-Klasse.

```
public class Edge{  
    ...  
    Edge(Point2D p, Point2D q) {  
        if (p == null || q == null)  
            throw new NullPointerException("Eingabe: null");  
        if (p.equals(q))  
            throw new EqualPointException(q);  
  
        start = p;  
        end = q;  
    }  
    ...  
}
```

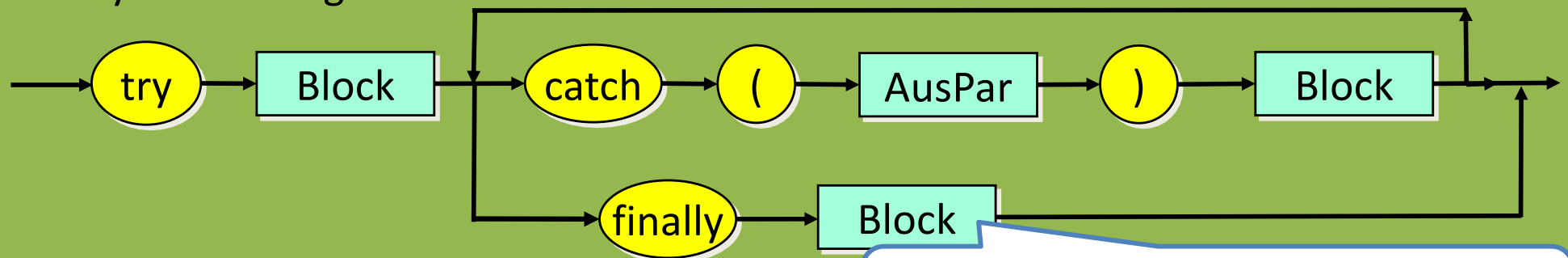
Wie eigene Exception-Klassen definiert werden, sehen wir später.

Falls p oder q null sind, wurde eine Exception geworfen. Der weitere Code wird nicht mehr ausgeführt. Daher ist es OK auf p eine Methode aufzurufen.

11.2.2 Fangen von Ausnahmen

- Eine Ausnahme kann durch eine **try-Anweisung** gefangen werden.
 - Dabei steht das eigentliche Programm in dem **try-Block**.
 - In dem Block können auch Exceptions geworfen werden.
 - Das Fangen der Ausnahmen erfolgt in den **catch-Blöcken**.
 - Im Kopf vor dem catch-Block wird eine Variable vom Typ einer Ausnahmeklasse deklariert.
 - Optional können in einem **finally-Block** noch die letzten Schritte der try-Anweisung (nach einem catch-Block oder try-Block) angegeben werden.

try-Anweisung:



AusPar:



Ablauf einer try-Anweisung

- Der Rumpf der try-Anweisung wird entweder komplett oder bis zum Werfen der ersten Ausnahme ausgeführt.
- Beim Werfen einer Ausnahme wird ein Exception-Objekt erzeugt. Dann werden die catch-Böcke von oben nach unten daraufhin überprüft, ob sie die Ausnahme behandeln können.
 - *Eine Behandlung ist dann möglich, wenn das geworfene Exception-Objekt der Variable im Kopf des catch-Blocks zugewiesen werden kann.*
 - *Wenn so der erste catch-Block gefunden wurde, wird das Exception-Objekt der Variablen zugewiesen und dann die Anweisungen im eigentlichen Block ausgeführt.*
 - *Falls kein passendes catch-Konstrukt gefunden wurde, wird die Ausnahme an die rufende Methode weitergereicht. (→ Details später)*
- Falls der finally-Block vorhanden ist, wird dieser immer am Schluss ausgeführt.

Beispiel (mehrere catch-Blöcke)

- Wir behandeln in unserem bisherigen Programm noch das Problem, wenn ein Benutzer kein Argument liefert.

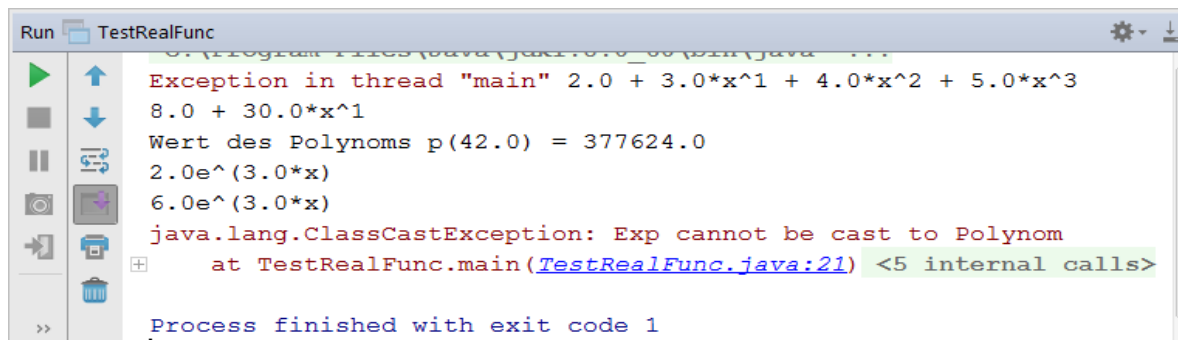
```
public class WithException {  
  
    public static void main(String[] args) {  
        int i;  
        try {  
            i = Integer.parseInt(args[0]);  
        }  
        catch (NumberFormatException abc) {  
            i = 7;  
            System.out.println("Zahl (bei falscher Eingabe) " + i);  
        }  
        catch (ArrayIndexOutOfBoundsException except) {  
            i = 0;  
            System.out.println("Zahl (bei keiner Eingabe) " + i);  
        }  
        System.out.println("Meine Zahl: " + i);  
    }  
}
```

Reihenfolge der catch-Blöcke

- Die Reihenfolge der catch-Blöcke ist wichtig, da die Fehlerbehandlung beim ersten Block stattfindet, bei dem die geworfene Ausnahme passt.
 - Ausnahmeobjekt kann der Parametervariablen im catch-Block zugewiesen werden.
- In der Liste der catch-Blöcke sollte ein Block mit einer speziellen Exception-Klasse vor dem einer allgemeinen Klasse stehen.
 - Weshalb?

11.2.3 Weiterreichen von Ausnahmen

- Werden ausgelöste Ausnahmen innerhalb einer Methode nicht behandelt, so werden diese an die aufrufende Methode weitergereicht.
 - In der **main-Methode** führt eine nicht-behandelte Ausnahme zum **Abbruch des Programms und Ausgabe des Stack-Trace**.



```
Run TestRealFunc
Exception in thread "main" 2.0 + 3.0*x^1 + 4.0*x^2 + 5.0*x^3
8.0 + 30.0*x^1
Wert des Polynoms p(42.0) = 377624.0
2.0e^(3.0*x)
6.0e^(3.0*x)
java.lang.ClassCastException: Exp cannot be cast to Polynom
    at TestRealFunc.main(TestRealFunc.java:21) <5 internal calls>
Process finished with exit code 1
```

- Beim Weiterreichen von Ausnahmen ist die Unterscheidung zwischen zwei Arten von Ausnahmen wichtig.
 - **Ungeprüfte** Ausnahmen
 - **Geprüfte** Ausnahmen

Geprüfte und ungeprüfte Ausnahmen

- Java unterscheidet zwei Arten von Ausnahmen
 - **Ungeprüfte Ausnahmen** sind Objekte der Klasse **RuntimeException** und ihrer Unterklassen.
 - Wird eine solche Ausnahme nicht in einer Methode behandelt, wird diese automatisch an die aufrufende Methode weitergeleitet.
 - Im Programm ist hierfür **kein zusätzlicher Code** erforderlich.
 - **Geprüfte Ausnahmen** sind Objekte der Klasse **Exception** und ihrer Unterklassen, die **nicht** eine RuntimeException sind.
 - z. B. Öffnen einer Datei, die nicht existiert.
 - In der Methode, in der diese Aufnahme geworfen wird, muss diese Ausnahme entweder behandelt oder **explizit** für die Weiterreichung **angemeldet werden**.
- Entwickler entscheiden, ob sie **neue Ausnahmen als geprüfte oder ungeprüfte Exceptions** implementieren.

Weiterreichen geprüfter Ausnahmen

- Werden **geprüfte Ausnahmen** innerhalb einer Methode nicht behandelt, so müssen diese weitergereicht werden.
 - Mit Hilfe der **throws-Klausel** muss **im Methodenkopf** festgelegt, welche geprüften Ausnahmen von einer Methode weitergereicht werden.
 - Beim **Überschreiben von Methoden** ist darauf zu achten, dass die Ausnahmen bei der überschriebenen Methode angegeben werden.
 - Eine Spezialisierung von Exceptions in der throws-Klausel ist erlaubt.
- Beispiel
 - Nehmen wir an, dass die Klasse `EqualPointException` eine geprüfte Ausnahme erzeugt (also keine Unterklasse von `RuntimeException` ist).

```
Edge(Point2D p, Point2D q) throws EqualPointException {  
    // Beste Lösung in diesem Fall!  
    if (p.equals(q))  
        throw new EqualPointException(q);  
    start = p;  
    end = q;  
}
```


11.2.4 Eigene Exception-Klasse

- Regel Nr. 1
 - Bevor eine eigene Exception-Klasse geschrieben wird, sollte man prüfen, ob eine der bereits in Java existierenden Klassen benutzt werden kann.
- Eine eigene Exception-Klasse muss von Exception bzw. von RuntimeException erben.
 - Klasse Exception
 - Konstruktoren
 - Exception()
 - Exception(String str)
 - Methoden (von der Klasse Throwable geerbt)
 - getMessage, printStackTrace, toString,

Beispiel

```
package geo;
```

```
public class EqualPointsException extends RuntimeException {  
    private Point p;
```

```
    public EqualPointsException(Point in){  
        super("Punkte sind gleich: " + in);  
        p = in;  
    }  
}
```

Zusammenfassung

- Ausnahmen in Java
 - Nutzen vorhandener Exception-Klassen
 - Neuentwicklung eigener Exception-Klassen
- Unterscheidung
 - Geprüfte Ausnahmen
 - Ungeprüfte Ausnahmen
- Wichtige Funktionalität
 - Werfen von Ausnahmen mit throw
 - Fangen von Ausnahmen mit try-catch-finally
 - Weiterreichen von geprüften Ausnahmen mit throws-Liste