Java - Fehler im Code

Softwarefehler

- Programmierfehler entstehen beim Schreiben des Programmcodes. Der Code entspricht nicht der Syntax der Programmiersprache.
- Logische Fehler können durch Denkfehler bei der Umsetzung der Aufgabe in ein Programm erzeugt werden. Das Programm wird fehlerfrei ausgeführt, aber das Ergebnis ist nicht korrekt.
- Laufzeitfehler treten während der Ausführung des Programms auf. Zum Beispiel wird die Obergrenze eines Arrays überschritten.

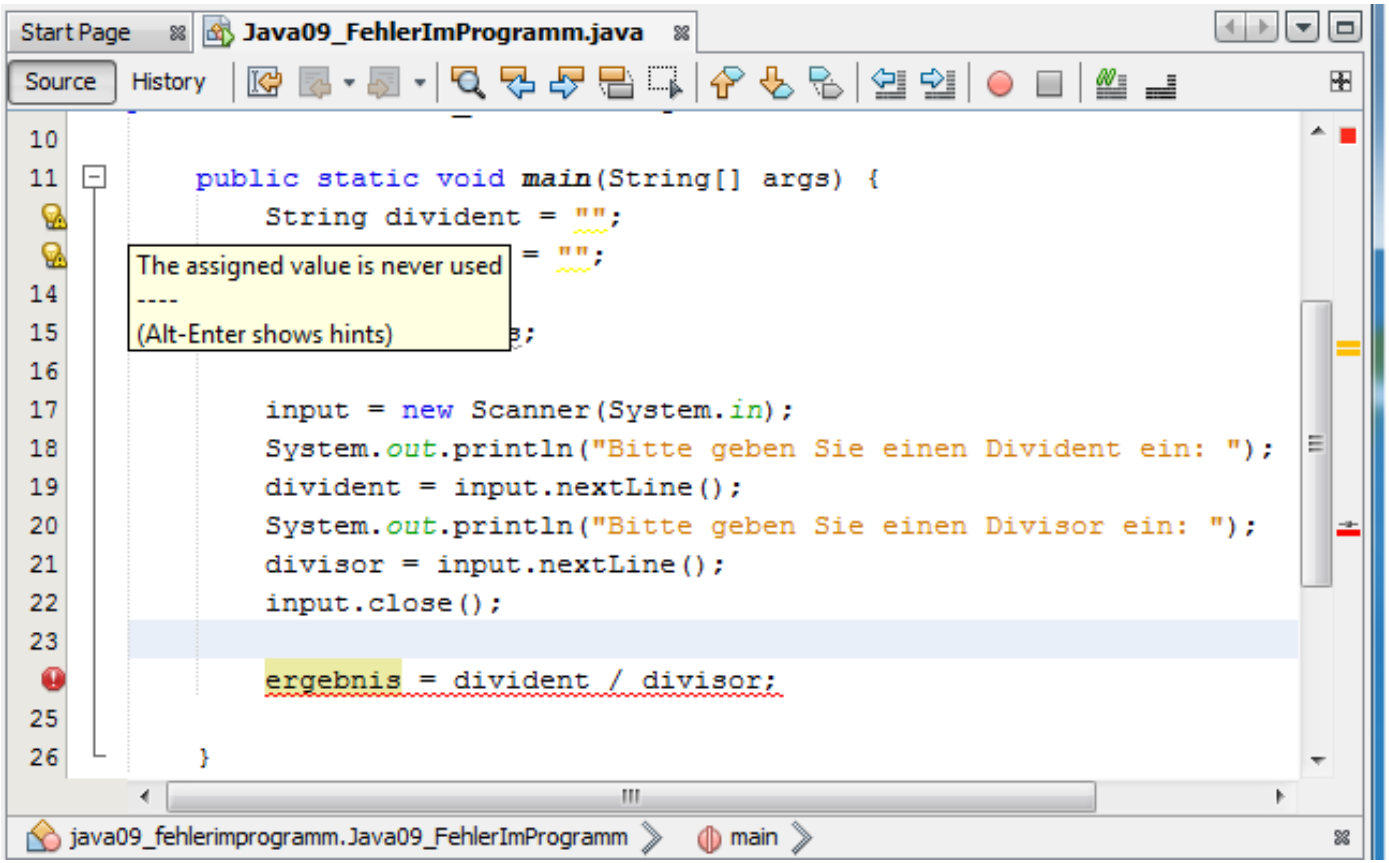
Syntaxfehler

- Verstoß gegen die Regeln einer Programmiersprache.
- Verletzung der Syntax einer Programmiersprache.
- Ein Ausdruck oder eine Anweisung hat einen falschen Datentyp.
- Der Interpreter erkennt Syntaxfehler. Sobald ein Syntaxfehler auftritt, wird die Ausführung des Programms abgebrochen.

Beispiele

- Tippfehler bei der Eingabe von Variablennamen oder Schlüsselwörtern.
- Mischung von numerischen und nicht-numerischen Operatoren.
- Syntaxfehler in Schleifen oder bedingten Anweisungen.
- Falsche Übergabe von Parametern.

Warnungen

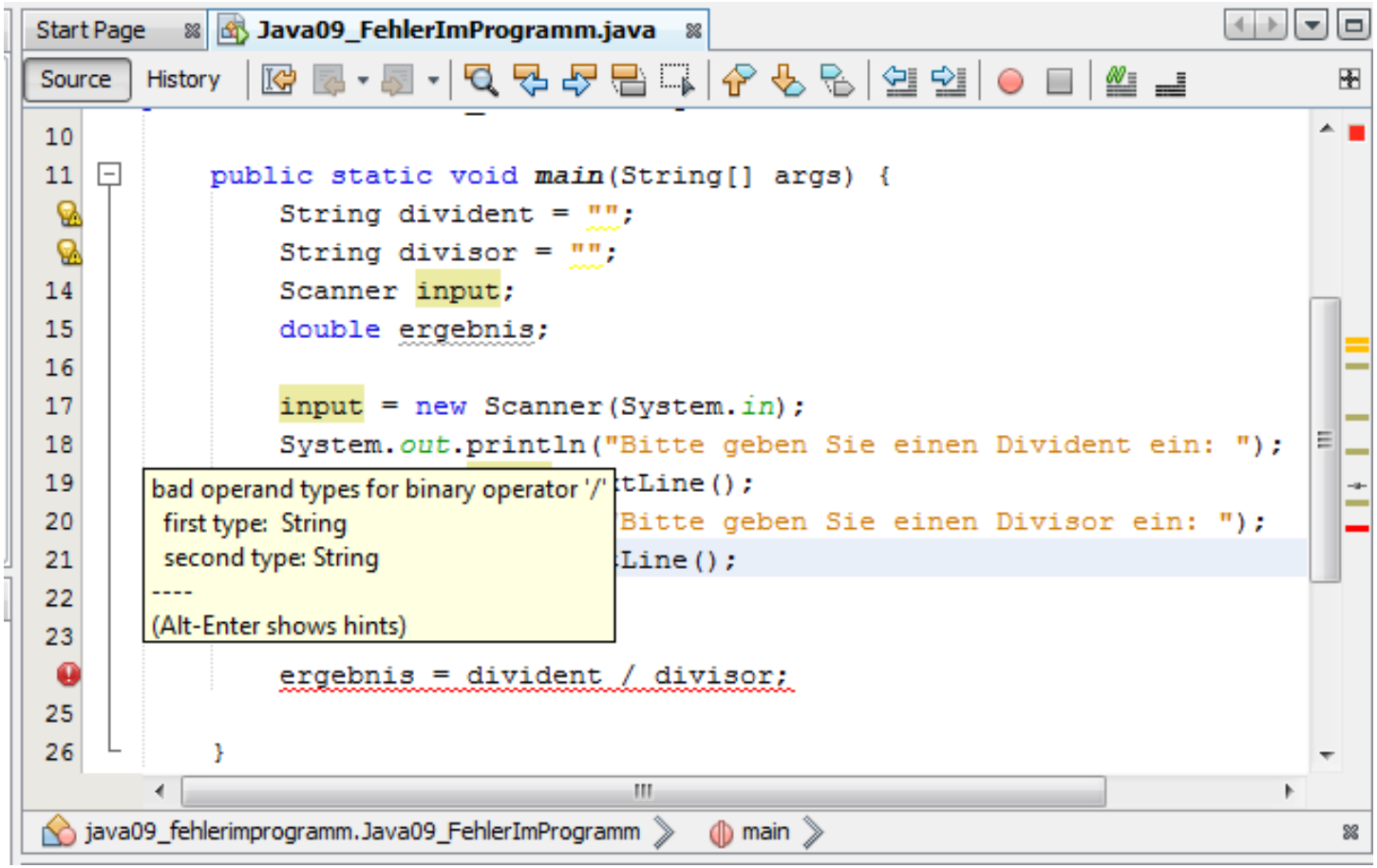

Start Page
Java09_FehlerImProgramm.java
Source
History
10
11
public static void main(String[] args) {
String divident = "";
= "";
The assigned value is never used

(Alt-Enter shows hints)
14
15
;
16
17
input = new Scanner(System.in);
18
System.out.println("Bitte geben Sie einen Divident ein: ");
19
divident = input.nextLine();
20
System.out.println("Bitte geben Sie einen Divisor ein: ");
21
divisor = input.nextLine();
22
input.close();
23
ergebnis = divident / divisor;
25
26
}
java09_fehlerimprogramm.Java09_FehlerImProgramm
main

… in NetBeans

- Gelber Warnhinweis am linken Rand des Codefensters.
- Warnungen sind Hinweise auf mögliche Fehler.
- Warnungen können ignoriert werden. Wenn möglich sollten aber der Code so umgeschrieben werden, dass Warnungen nicht mehr angezeigt werden.

Syntaxfehler



… in NetBeans

- Roter Kreis am linken Rand des Codefensters.
- Wenn der Mauszeiger über die Kennzeichnung liegt, wird ein Hinweis zu dem Syntaxfehler eingeblendet.
- Der Code wird nicht vollständig ausgeführt. Die Ausführung wird abgebrochen.
- Syntaxfehler müssen behoben werden.

Logische Fehler

- Treten bei der Ausführung der Software auf.
- Falsche Berechnung von Werten.
- Fehlerhaftes Design der Software.
- Falsche Nutzung durch den Benutzer.

Strategien

- Testen des Programms mit Hilfe des Debuggers.
- Der Entwickler kann Zeilen, die in Verdacht stehen einen Fehler zu verursachen, auskommentieren. Falls das Programm anschließend fehlerfrei läuft, ist der Fehler lokalisiert und kann behoben werden.
- Nach einem erfolgreichen Test wird das Programm gespeichert. Für die Weiterentwicklung wird die Programmdatei kopiert. Falls Fehler nicht gefunden werden, kann auf eine alte fehlerfreie Version zurückgegriffen werden.

Debuggen in NetBeans

- *Debug* – *Debug Project* startet den Debugger für das aktive Projekt.
- *Debug* – *Finish Debugger Session* beendet den Debugger. Das Programm kann mit Hilfe von *Run* – *Run Project* vollständig durchlaufen werden.

Informationen im Internet

- https://netbeans.org/project_downloads/usersguide/nbfieldguide/Chapter5-Debugging.pdf
- https://netbeans.org/kb/docs/java/debug-visual.html

… bis zum Mauszeiger

- *Debug – Run to Cursor* startet das Programm und durchläuft es bis zu der Zeile in der sich die Einfügemarke befindet.

… in Einzelschritten durchlaufen

- Alle nachfolgenden Menübefehle führen eine Zeile aus und springen zur nächsten ausführbaren Zeile.
- *Debug* – *Step Into* . Bei einem Methoden-Aufruf wird zur Deklaration der Methode gesprungen.
- *Debug* – *Step Over.* Wenn die ausführbare Zeile einen Methoden-Aufruf enthält, wird dieser nur ausgeführt.
- *Debug* – *Step Out Of* durchläuft eine Methode vollständig und springt dann in die Zeile nach dem Aufruf der Methode.

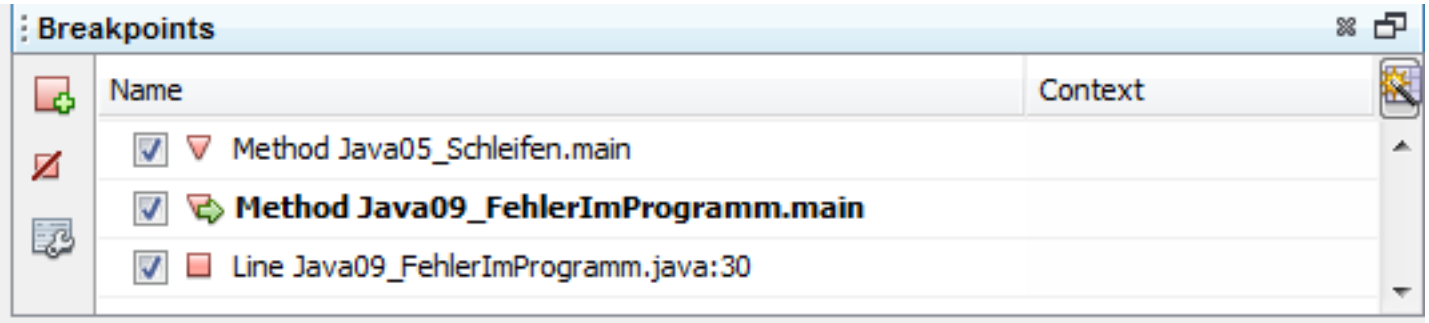
… bis zu einem Haltepunkt durchlaufen

- Das Programm stoppt einem bestimmten Haltepunkte.
- Ab diesem Punkt kann das Programm in Einzelschritten durchlaufen werden.

Setzen von Haltepunkten

- *Debug – Toggle Line Breakpoint*. In einer Zeile wird ein Haltepunkt gesetzt. Die Haltepunkte werden mit einem rosa Rechteck am rechten Rand gekennzeichnet.
- *Debug – New Breakpoint* beim Aufruf einer Methode oder Referenzierung einer Klasse.

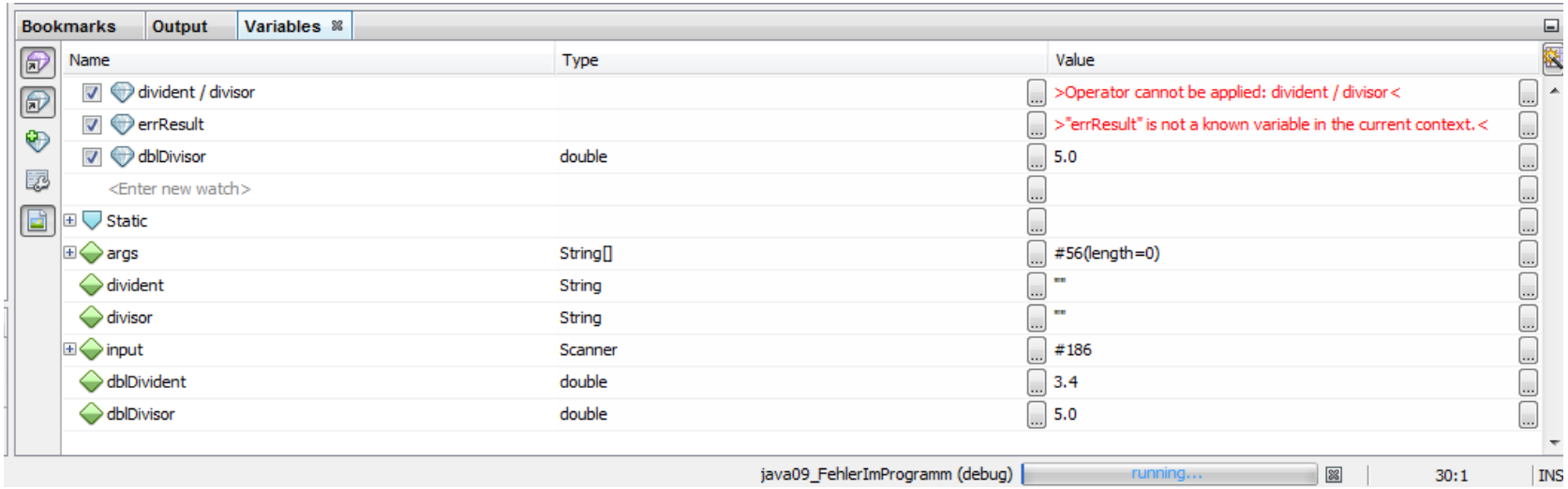
… anzeigen



- *Window – Debugging – Breakpoints*.
- Alle Haltepunkte werden angezeigt und können darüber entfernt werden.

Anzeige und Überwachung von Variablen

- *Window – Debugging – Variables*.



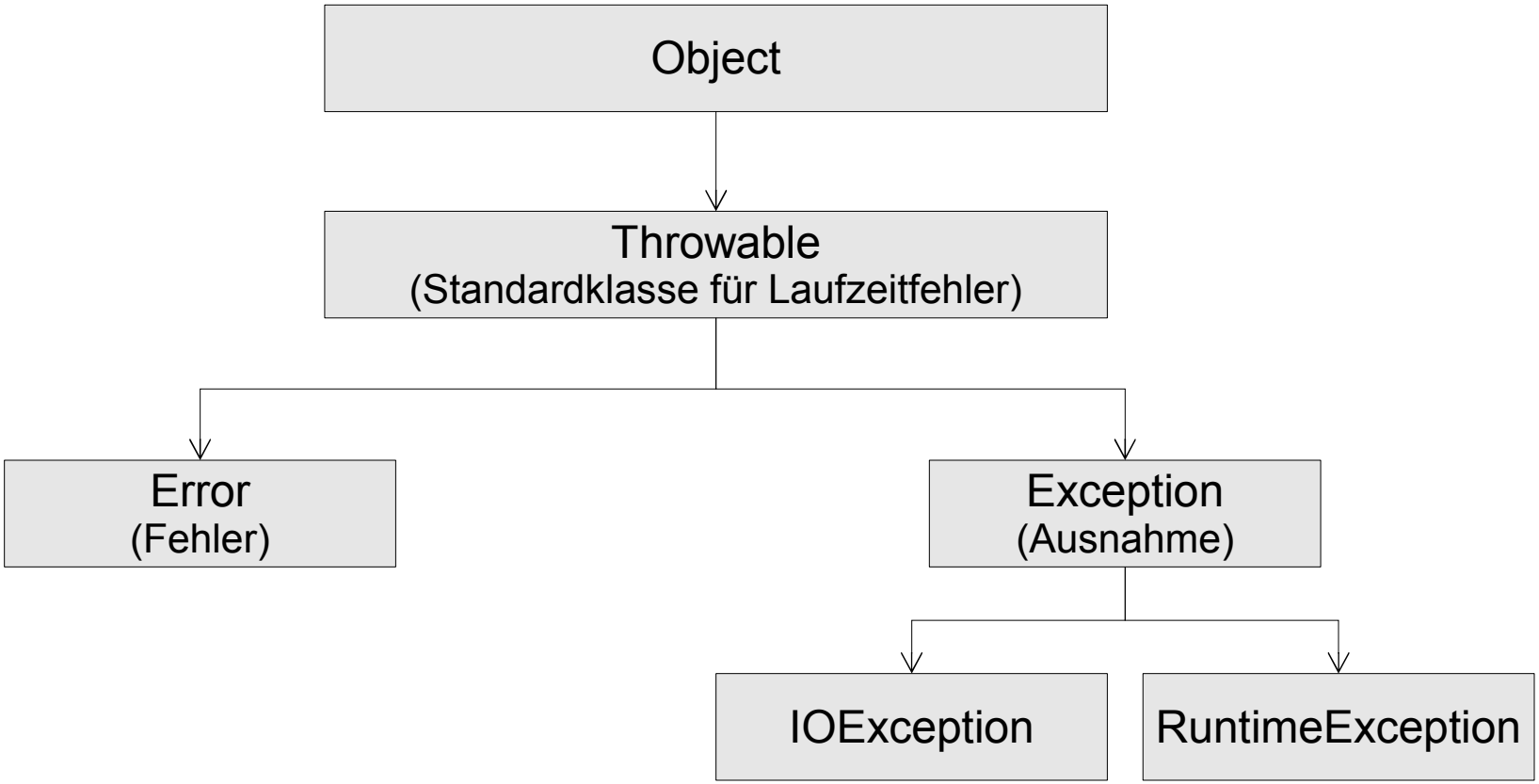
Laufzeitfehler

- Ausdrücke oder Anweisungen werden vom Programm nicht korrekt ausgewertet.
- Fehler, die zur Laufzeit des Programms, ein nicht erwünschtes Verhalten des Programms erzeugen.
- Das Programme stürzt mit einer Fehlermeldung ab.

Beispiele für Laufzeitfehler

- Division durch Null.
- Falsche Verwendung von Operanden und / oder Operatoren.
- Überschreiten der Untergrenze oder Obergrenze von Arrays.
- Endlosschleife
- Ein- und Ausgabefehler.

Laufzeitfehler in Java



Throwable

- Basisklasse für Laufzeitfehler.
- Informationen über die Fehlerursache.
- Abfangen von Fehlern durch die Java Virtual Maschine.

Error

- Schwerwiegender Fehler, der zum Abbruch des Programms führt.
- Systembedingte Fehler.
- Nicht behandelbare Fehler.

Beispiele

- `OutOfMemoryError`. Die Java Virtual Maschine verfügt nicht mehr über genügend Hauptspeicher.
- `StackOverflowError`. Der Fehler kann bei einem rekursiven Aufruf von Methoden auftreten. Überlauf des Stacks.
- `InternalError`. Interner Fehler in der Java Virtual Maschine.

Exception

- Behebbare Fehler, die spätestens von der Java Virtual Maschine abgefangen werden.
- Fehler, die durch eine defensive Programmierung vermieden werden können.
- Fehler, die im Programm abgefangen und gelöst werden können.
- Ausnahmen, die von NetBeans in dem Fenster Output angezeigt werden.

Informationen zu Exceptions

- Tutorial:
  https://docs.oracle.com/javase/tutorial/essential/exceptions/
- Dokumentation:
  http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

Ungeprüfte Exceptions

- Unchecked Exceptions.
- Die Ausnahmen können abgefangen werden, müssen aber nicht.
- Ausnahmen der Klasse `RuntimeException`.
- Mit Hilfe einer defensiven Programmierung sind diese Fehler zu vermeiden.

RuntimeException (Laufzeitfehler)

- `ArrayIndexOutOfBoundsException`. Indexgrenzen eines Arrays werden verletzt.
- `ArithmeticException`. Ganzzahlige Division durch Null.
- `IllegalArgumentException`. Falsche Argumentübergabe bei dem Aufruf einer Methode. Die Kindklasse `NumberFormatException` wird durch die verschiedenen parse-Methoden ausgelöst.
- `NullPointerException` wird zum Beispiel bei ein String der Länge null geworfen.

Problem: Division durch Null

```
double dblDividend;
double dblDivisor;
double ergebnis;
Scanner input;

input = new Scanner(System.in);
System.out.println("Dividend: ");
dblDividend = input.nextDouble();
System.out.println("Divisor: ");
dblDivisor = input.nextDouble();
input.close();

ergebnis = dblDividend / dblDivisor;
```

Defensive Programmierung

```
while(true){
      System.out.println("Divisor ");
      dblDivisor = input.nextDouble();

      if (dblDivisor != 0.0){
         ergebnis = dblDivident / dblDivisor;
         System.out.println("Ergebnis:" + ergebnis);
         break;
      }
      else{
         System.out.println("Fehlermeldung");
      }
}
```

Weitere Möglichkeit

```
if (Double.isInfinite(dblDividend / dblDivisor))
{
   System.out.println("Division durch Null.");
}
else
{
  ergebnis = dblDividend / dblDivisor;
  System.out.println("Ergebnis:" + ergebnis);
}
```

Methoden der Wrapper-Klasse Float und Double

- `.isInfinite()`. Die Gleitkommazahl ist unendlich.
- `.isNaN()`. Die Gleitkommazahl kann nicht als Zahl interpretiert werden.

Geprüfte Exceptions

- Checked Exception.
- Fehler, die nicht in der Klasse `RuntimeException` deklariert sind.
- Ausnahmen dieser Kategorie müssen behandelt werden.

IOException (Ein- und Ausgabefehler)

- Die Fehler werden durch die Klasse `import java.io.*` erzeugt.
- `EOFException`. Es wird versucht, Daten nach dem Dateiende zu lesen.
- `FileNotFoundException`. Die gewünschte Datei ist nicht vorhanden.

Behandlung eines Fehlers

- Ignorieren. Der Fehler wird durch das gesamte Programm getragen und kann weitere Fehler nach sich ziehen.
- Der Fehler wird direkt im Programm „geheilt“.
- Ausgabe einer Meldung. Der Nutzer wird über den Fehler und einer eventuellen Behebung informiert.

Fehler abfangen

```
try
{
   ergebnis = intDividend / intDivisor;
   System.out.println("Ergebnis:" + ergebnis);
}
catch(Exception e)
{
  System.out.println("Fehler: Division durch Null.");
}
```

Versuche folgende Anweisungen auszuführen...

```
try
{
   ergebnis = intDividend / intDivisor;
   System.out.println("Ergebnis:" + ergebnis);
}
```

- Zusammenfassung von allen Anweisungen, bei denen eine Exception abgefangen werden soll.
- Versuche die Anweisungen ohne Fehler auszuführen …
- try-Blöcke können verschachtelt werden.

Fange die Ausnahme ab

```
catch(Exception e)
{
  System.out.println("Fehler: Division durch Null.");
}
```

- Mit Hilfe des Schlüsselwortes `catch` werden Exceptions (Ausnahmen) eingefangen.
- Der einzufangende Fehler wird dem Schlüsselwort in den runden Klammern übergeben.
- Zu einem try-Block gehört mindestens eine catch-Anweisung.

Fange alle Ausnahmen ab

```
catch(Exception e)
{
  System.out.println("Fehler: Division durch Null.");
}
```

- Die Variable `e` kann Instanzen der Klasse `Exception` speichern.
- Die Klasse `Exception` ist die Eltern-Klasse aller Ausnahmen.
- Alle Exceptions werden durch die Angabe `Exception e` angefangen.
- Die Ausnahmen werden allgemein abgefangen.

Abfangen des Fehlers bei Ganzzahlen

```
try{
   ergebnis = intDividend / intDivisor;
}
catch(java.util.InputMismatchException e){
    System.out.println("Falsche Eingabe.");
}
catch(ArithmeticException e){
    System.out.println("Fehler in der Division.");
}
catch(Exception e){
     System.out.println("Unbekannter Fehler.");
}
```

Verarbeitung einer Ausnahme

- Die catch-Anweisungen werden von oben nach unten durchlaufen.
- Die catch-Anweisungen werden von oben nach unten immer allgemeiner.
- Sobald eine passende Ausnahme gefunden wird, wird der dazugehörige Block abgearbeitet. Alle nachfolgenden catch-Anweisungen werden nicht beachtet.

Objekt-Variable

(	`ArithmeticException`	`e`	)
(	`java.util.InputMismatchException`	`e`	)
(	`Ausnahme`	`e`	)

- Instanz einer Fehlerklasse. Die Fehlerklasse muss von der Basisklasse `Throwable` abgeleitet sein.
- Mit Hilfe des Punktoperators wird der „Pfad“ zu der Klasse dargestellt.
- Jede Instanz hat einen eindeutigen Namen. Häufig wird `e` genutzt.

Informationen zur Ausnahme

- `e.getMessage()`. Liefert die, bei der Erzeugung mit new() angegebene, Meldung.
- `e.toString()`. Der Typ und die Meldung werden angezeigt.
- `e.getCause()`. Bei verketteten Exception wird die auslösende Exception zurückgegeben.
- `e.printStackTrace()`. Beschreibung der Ausnahme. Der Ablauf der Aufrufe wird dargestellt.

Aufräumarbeiten

```
try
{
   ergebnis = intDividend / intDivisor;
   strMessage = "Ergebnis:" + ergebnis;
}
catch(Exception e)
{
  strMessage = e.toString();
}
finally
{
  System.out.println(strMessage);
}
```

Erläuterung

- Der finally-Block ist optional.
- Der finally-Block folgt der letzten catch-Anweisung.
- Falls der Block vorhanden ist, wird dieser immer ausgeführt.
- In diesem Block werden Aufräumarbeiten wie zum Beispiel das Schließen des Scanners oder eine Datei ausgeführt.

„Werfen von Fehlern“

```
try
{
    input = new Scanner(System.in);
    System.out.println("Ganzzahl: ");
    intWert = input.nextInt();
    return intWert;
}
catch(Exception e)
{
  throw e;
}
```

Erläuterung

```
throw e;
```

- Das Schlüsselwort `throw` wirft eine Ausnahme von Klassen, die von der Basisklasse `Throwable()` abgeleitet sind oder von der Basisklasse selber.
- Wenn Ausnahmen geworfen werden, sollte in einer finally-Anweisung aufgeräumt werden.
- Falls der Fehler in einer Methode geworfen wird, wird der Fehler an den Aufrufer der Methode zurückgegeben. Dieser kann den Fehler weitergeben. Spätestens in der main-Methode sollten alle Fehler abgefangen werden.

Weitere Möglichkeit

```
public static double dividiere(
            int divident, int divisor) throws Exception
{
   double ergebnis;

   ergebnis = divident / divisor;
   return (ergebnis);
}
```

Erläuterung

- Die Methode wirft alle Fehler an den Aufrufer zurück.
- Mit Hilfe des Schlüsselwortes `throw` am Ende des Methodenkopfes werden alle Ausnahmen der angegebenen Klasse an die nächst höhere Stelle weitergegeben.
- In diesem Beispiel werden alle Fehler der Klasse `Exception` an den Aufrufer weitergegeben.

Dokumentationskommentare

- Beschreibung von Klassen, Methoden, Schnittstellen, Aufzählungen etc.
- Dokumentation direkt im Programm.
- JavaDoc erstellt aus diesen Kommentaren Beschreibungsdateien (meist im Format HTML).

Aufbau

```
/**
 *
 * @author Teilnehmer A
 * Arbeiten mit einem Kreis
 */
```

- Beginn: Schrägstrich und zwei Sternchen
- Ende: Sternchen und Schrägstrich
- Zeilen zwischen Beginn und Ende: Sternchen
- Mit Hilfe des Add-Zeichens werden vordefinierte Tags gekennzeichnet.

Tags in Dokumentationskommentaren

- @*autor name*. Der Name des Software-Entwicklers.
- @*param variable*. Die Parameter, die an eine Methode übergeben werden.
- @*return variable*. Welcher Wert wird zurückgegeben?
- @*version version*. Version der Klasse.
- @*since jdk-version*. Seit wann ist die Funktionalität implementiert?

Beispiel: Methode ohne Rückgabewert

```
 /**
* Radius eines Kreises setzen
* @param radius
* @return null
*/
```

Beispiel: Methode mit Rückgabewert

```
 /**
* Fläche eines Kreises berechnen
* @param radius
* @return flaeche
*/
```