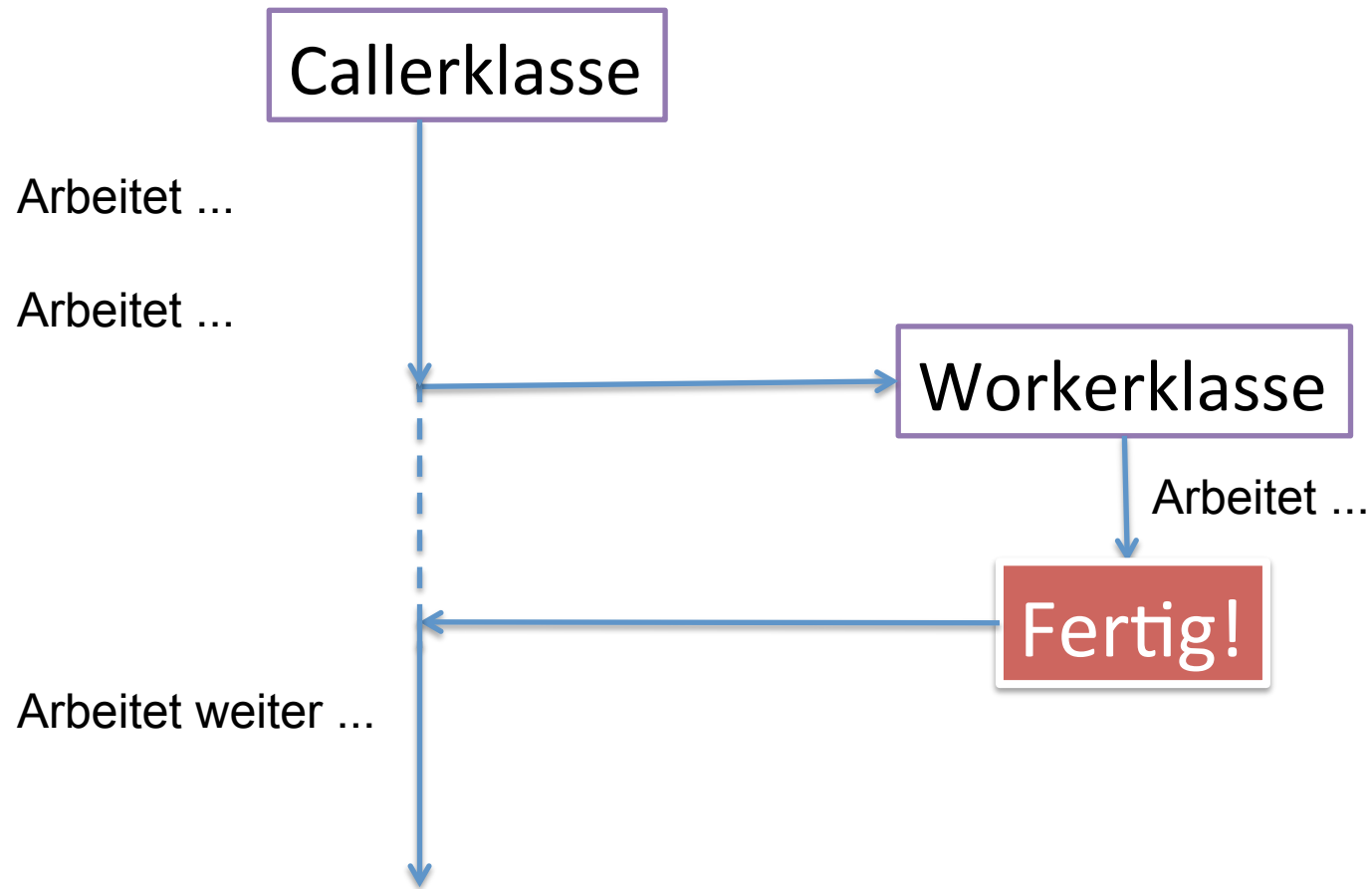
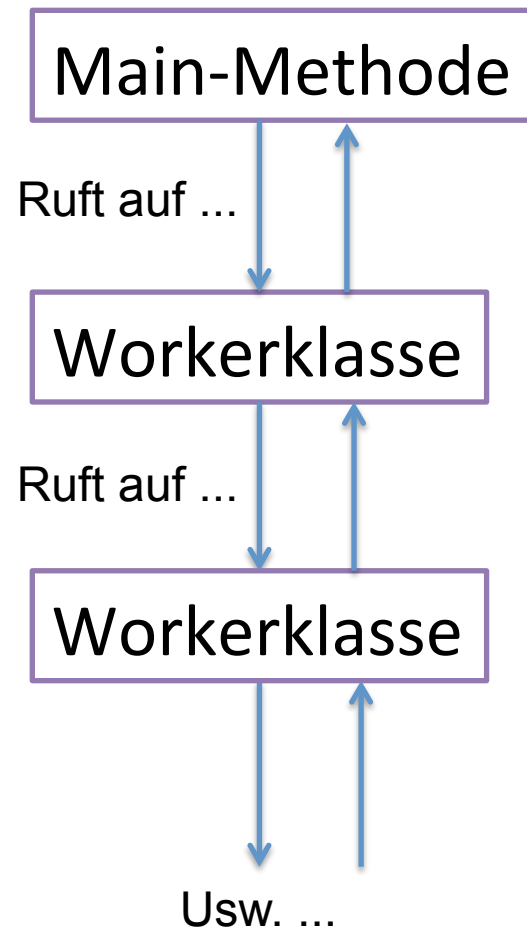


Java :
Fehlerbehandlung

Prinzip „Callerklassen“ / „Workerklassen“

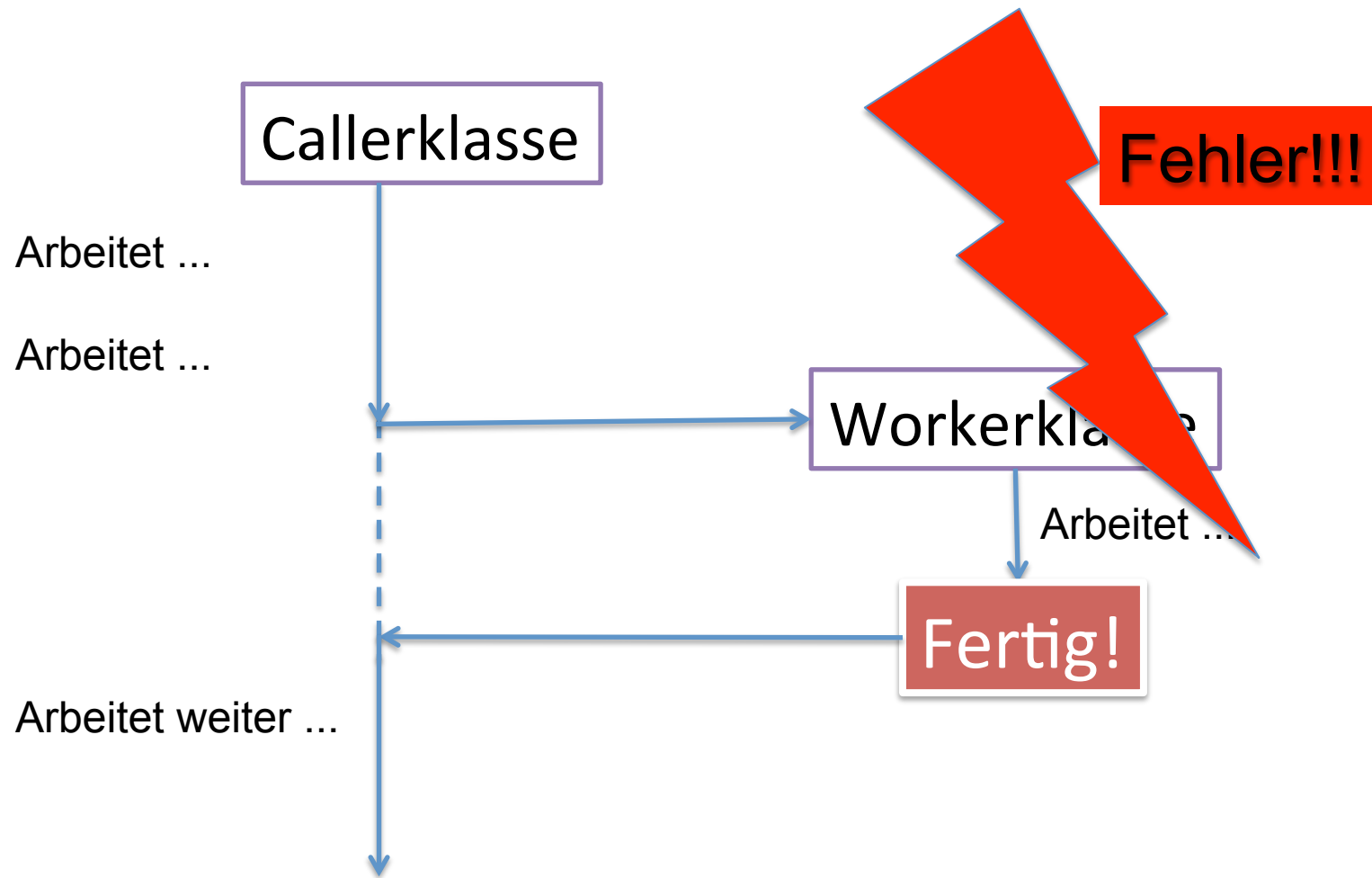


Prinzip „Callerklassen“ / „Workerklassen“



„call-stack“
(„Aufrufer-Stapel“)

Prinzip „Callerklassen“ / „Workerklassen“



```
1 package pkg140107_test_fehlerbehandlung;
2
3 public class EineCallerKlasse {
4     public static void main(String[] args) {
5         EineWorkerKlasse z = new EineWorkerKlasse();
6         z.etwasTun();
7     }
```

EineWorkerKlasse.java

Source

History

```
1 package pkg140107_test_fehlerbehandlung;
2
3 public class EineWorkerKlasse {
4     private int[] arraychen = new int[3];
5
6     public void etwasTun() {
7         arraychen[4] = 12;
```

```
1 package pkg140107_test_fehlerbehandlung;
2
3 public class EineCallerKlasse {
4     public static void main(String[] args) {
5         EineWorkerKlasse z = new EineWorkerKlasse();
6         z.etwasTun();
7     }
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at pkg140107_test_fehlerbehandlung.EineWorkerKlasse.etwasTun(EineWorkerKlasse.java:19)
    at pkg140107_test_fehlerbehandlung.EineCallerKlasse.main(EineCallerKlasse.java:22)
Java Result: 1
```

```
1 package pkg140107_test_fehlerbehandlung;
2
3 public class EineWorkerKlasse {
4     private int[] arraychen = new int[3];
5
6     public void etwasTun() {
7         arraychen[4] = 12;
```

Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException: 4

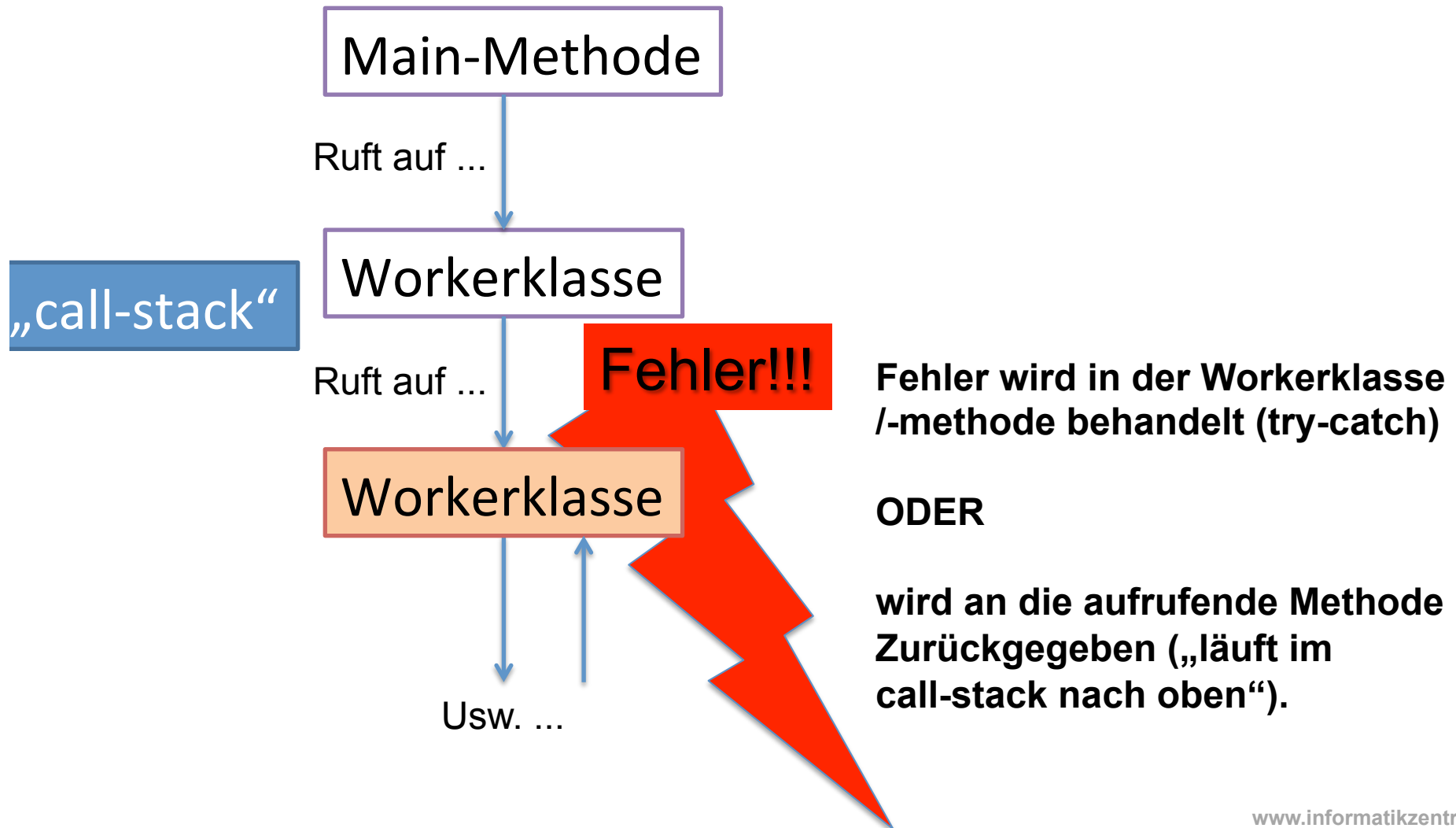
at

pkg...EineWorkerKlasse.etwasTun(EineWorkerKlasse.java:19)

at

pkg...EineCallerKlasse.main(EineCallerKlasse.java:22)

Prinzip „Callerklassen“ / „Workerklassen“



```
1 package pkg140107_test_fehlerbehandlung;
2
3 public class EineCallerKlasse {
4     public static void main(String[] args) {
5         EineWorkerKlasse z = new EineWorkerKlasse();
6         z.etwasTun();
7     }
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at pkg140107_test_fehlerbehandlung.EineWorkerKlasse.etwasTun(EineWorkerKlasse.java:19)
    at pkg140107_test_fehlerbehandlung.EineCallerKlasse.main(EineCallerKlasse.java:22)
```

Java F

Frage: Wie kann ich vermeiden, dass ein Fehler das Programm zum Absturz bringt?

Ex

java.lang.ArrayIndexOutOfBoundsException: 4

at

pkg...EineWorkerKlasse.etwasTun(EineWorkerKlasse.java:19)

at

pkg...EineCallerKlasse.main(EineCallerKlasse.java:22)

Klasse "Exception"

java.lang

Class Exception

[java.lang.Object](#)

└ [java.lang.Throwable](#)

└ [java.lang.Exception](#)

All Implemented Interfaces:

[Serializable](#)

Direct Known Subclasses:

[AclNotFoundException](#), [ActivationException](#), [AlreadyBoundException](#), [ApplicationException](#),
[AWTException](#), [BackingStoreException](#), [BadLocationException](#), [CertificateException](#),
[ClassNotFoundException](#), [CloneNotSupportedException](#), [DataFormatException](#), [DestroyFailedException](#),
[ExpandVetoException](#), [FontFormatException](#), [GeneralSecurityException](#), [GSSEException](#),
[IllegalAccessException](#), [InstantiationException](#), [InterruptedException](#), [IntrospectionException](#),
[InvalidMidiDataException](#), [InvalidPreferencesFormatException](#), [InvocationTargetException](#), [IOException](#),
[LastOwnerException](#), [LineUnavailableException](#), [MidiUnavailableException](#), [MimeTypeParseException](#),
[NamingException](#), [NoninvertibleTransformException](#), [NoSuchFieldException](#), [NoSuchMethodException](#),
[NotBoundException](#), [NotOwnerException](#), [ParseException](#), [ParserConfigurationException](#), [PrinterException](#),
[PrintException](#), [PrivilegedActionException](#), [PropertyVetoException](#), [RefreshFailedException](#),
[RemarshalException](#), [RuntimeException](#), [SAXException](#), [ServerNotActiveException](#), [SQLException](#),
[TooManyListenersException](#), [TransformerException](#), [UnsupportedAudioFormatException](#),
[UnsupportedCallbackException](#), [UnsupportedFlavorException](#), [UnsupportedLookAndFeelException](#),
[URISyntaxException](#), [UserException](#), [XAException](#)

Kindklassen unterscheiden sich eigentlich nur im Namen.

Alle haben weitere Kindklassen

```
public class Exception  
extends Throwable
```

"checked" vs. "unchecked" Exceptions

Der Compiler prüft („checkt“) beim Kompiliervorgang das Programm auf mögliche Fehlertypen.

"checked" vs. "unchecked" Exceptions

unchecked Exceptions:

meist Programmierfehler ("Bugs"), z.B. falsche Parameter (z.B. ungültiger Array-Index), ungeprüfte Benutzereingaben

checked Exceptions:

Fehler, die man beim Programmieren nicht verhindern kann (z.B. Hardwarefehler, Benutzereingaben) - Programm soll Fehler korrigieren und MUSS weiterarbeiten (z.B. bei Benutzereingaben: Fehlermeldung ausgeben, Korrektur ermöglichen)

MUSS geworfen (= behandelt) werden → Voraussetzung für Kompilierung

"checked" vs. "unchecked" Exceptions

3 Haupttypen der Klasse Throwable:

1) Error

i.d.R. nicht behebbarer externer Fehler (z.B. nicht genug Speicher).

Lösung: Möglicher Fehler z.B. durch Kontrollstrukturen abfangen.

unchecked

2) Exception

Schon VOR Start des Programmes behebbar; Compiler erwartet, dass Programmierer sich drum kümmert.

checked

3) RuntimeException [Unterklasse v. Exception, Ausnahme, dass unchecked - ansonsten alle Unterklassen von Exception = checked]

Programmierfehler - Programmierer muss überprüfen, ob bspw. falsche Indexwerte an ein Array übergeben werden.

Lösung: Code verbessern

unchecked

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at pkg140107_test_fehlerbehandlung.EineWorkerKlasse.etwasTun(EineWorkerKlasse.java:19)
    at pkg140107_test_fehlerbehandlung.EineCallerKlasse.main(EineCallerKlasse.java:22)
Java Result: 1
```

java.lang

Class ArrayIndexOutOfBoundsException

[java.lang.Object](#)

└ [java.lang.Throwable](#)

└ [java.lang.Exception](#)

└ [java.lang.RuntimeException](#)

└ [java.lang.IndexOutOfBoundsException](#)

└ **java.lang.ArrayIndexOutOfBoundsException**

Beispiel: Error

```
ArrayList listchen = new ArrayList();
    while(true) {
        String irgendwas = "bla bla bla";
        listchen.add(irgendwas); // Zeile 12
        if(listchen.size() % 1000000 == 0) {
            System.out.println("Liste hat jetzt "
+ listchen.size()/1000000 + " Millionen
Einträge.");
        }
    }
}
```

Beispiel: Error

```
ArrayList listchen = new ArrayList();
while(true){
    String irgendwas = "bla bla bla";
    listchen.add(irgendwas);
    if(listchen.size() % 1000000 == 0){
        System.out.println("Liste hat jetzt " +
listchen.size()/1000000 + " Millionen Einträge.");
    }
}
```

Liste hat jetzt 530 Millionen Einträge.

Liste hat jetzt 531 Millionen Einträge.

Liste hat jetzt 532 Millionen Einträge.

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
  at java.util.Arrays.copyOf(Arrays.java:2245)
  at java.util.Arrays.copyOf(Arrays.java:2219)
  at java.util.ArrayList.grow(ArrayList.java:242)
  at java.util.ArrayList.ensureExplicitCapacity(ArrayList.java:216)
  at java.util.ArrayList.ensureCapacityInternal(ArrayList.java:208)
  at java.util.ArrayList.add(ArrayList.java:440)
  at pkg140107_test_fehlerbehandlung.Startklasse.main(Startklasse.java:12)
```

Java Result: 1

```
OUTER SUCCESSFUL (4-4-1 time: 1 minute 50 seconds)
```

Beispiel: NullPointerException

```
Object einObjekt = null;  
einObjekt.toString();
```

Programm ausführen

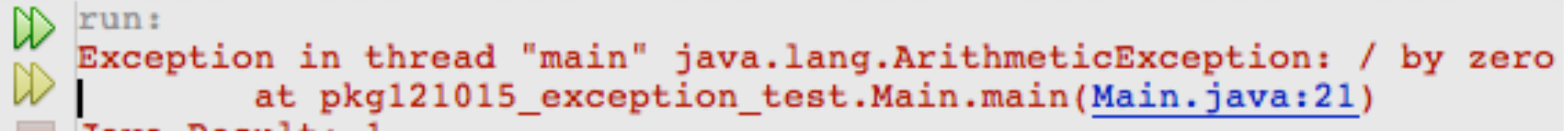
```
run:  
Exception in thread "main" java.lang.NullPointerException  
    at pkg121015_exception_test.Main.main(Main.java:20)
```

"**StackTrace**" = "Stapel" von Prozeduren, die
bisher aufgerufen wurden
→ erleichtert Lokalisierung von Fehlern

Beispiel: Division by zero

```
int a = 10; int b = 0;  
int c = a/b;
```

Programm ausführen

A screenshot of a Java IDE's output window. It shows a green play button icon followed by the text 'run:'. Below this, a red error message is displayed: 'Exception in thread "main" java.lang.ArithmeticException: / by zero'. The next line shows the stack trace: 'at pkg121015_exception_test.Main.main(Main.java:21)'. The file name 'Main.java' is underlined in blue. A yellow play button icon is visible to the left of the stack trace line.

```
run:  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
| at pkg121015_exception_test.Main.main(Main.java:21)
```

try - catch

Fehler "abfangen"

Exception wird "geworfen"
und von uns "(ab)gefangen"

try - catch

Fehler "abfangen"

Exception wird "geworfen" und von uns "(ab)gefangen"

```
try
{
    Object o = null;
    o.toString();
}
catch (NullPointerException e)
{
    System.out.println("Fehler!");
}
```

```
19     public static void main(String[] args)
20     {
21         try
22         {
23             Object o = null;
24             o.toString();
25         }
26         catch (NullPointerException e)
27         {
28             System.out.println("Fehler!");
29         }
30     }
```

kein Stacktrace mehr!

```
Ausgabe - 121015_exception_test (run)
run:
Fehler!
ERSTELLEN ERFOLGREICH (Gesamtzeit: 0 Minuten 0 Sekunden)
```

try - catch

```
try
{
    Object o = null;
    o.toString();
    System.out.println("Das hier sieht keiner");
}
catch (NullPointerException e)
{
    System.out.println("Fehler!");
}
// ... bla bla weiter geht's im Programm
```

**Nach Fehlerbehandlung fährt Programm
NACH dem catch-Block fort (nicht an der Stelle,
an der der Fehler verursacht wurde!)**

Welche Exception?

```
try
{
    Object o = null;
    o.toString();
    System.out.println("Das hier sieht keiner");
}
catch (NullPointerException e)
// oder "catch (Exception e)" → Elternklasse
von NullPointerException
{
    System.out.println("Fehler!");
}
```

**catch-Block muss zum Fehlertyp passen,
sonst Programmabsturz (als gäbe es keinen catch-Block)**

checked Exception

Programm kompiliert nicht, wenn Exception nicht abgefangen wird

```
18 unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown
19 ----
20 (Alt-Enter zeigt Hinweise)
21
22 FileReader fr = new FileReader("C:/michgibtsnicht/hallo.txt");
23
```

Konstruktor von FileReader wirft eine checked (!) Exception.
Deshalb try-catch verwenden.

NetBeans: Linksklick auf Fehlersymbol

```
20 unreported exception java.io.FileNotFoundException; must be caught or declared to
21
22 FileReader fr = new FileReader("C:/michgibtsnicht/hallo.txt");
23
24 "throws"-Klausel für java.io.FileNotFoundException hinzufügen
25 Anweisung mit "try-catch" umgeben
26
```

```
try
{
    FileReader fr = new FileReader("C:/michgibtsnicht/hallo.txt");
}
catch (FileNotFoundException ex)
{
    // Logger.getLogger(Main.class.getName()).log(Level.SEVERE, nu
    ex.printStackTrace();
}
```


Mehrere catch-Blöcke

Wenn auf verschiedene Fehler unterschiedlich reagiert werden soll

```
try
{
    Object einObjekt = null;
    einObjekt.toString();
}
catch (NullPointerException e)
{
    System.out.println("Nullpointer-Exception!");
}
catch (Exception e)
{
    System.out.println("Ein anderer Fehler ...");
}
```

Mehrere catch-Blöcke

```
try
{
    Object einObjekt = null;
    einObjekt.toString();
}
catch (Exception e)
{
    System.out.println("Ein anderer Fehler ...");
}
catch (NullPointerException e)
{
    System.out.println("Nullpointer-Exception!");
}
```



Falsche Reihenfolge!

Falls Fehler auftritt, wird er durch Exception (Elternklasse!) auf jeden Fall gefangen, d.h. NullPointerException wird nie gefangen werden.


```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at pkg140107_test_fehlerbehandlung.EineWorkerKlasse.etwasTun(EineWorkerKlasse.java:19)
    at pkg140107_test_fehlerbehandlung.EineCallerKlasse.main(EineCallerKlasse.java:22)
Java Result: 1
```

java.lang

Class ArrayIndexOutOfBoundsException

[java.lang.Object](#)

└ [java.lang.Throwable](#)

└ [java.lang.Exception](#)

└ [java.lang.RuntimeException](#)

└ [java.lang.IndexOutOfBoundsException](#)

└ **java.lang.ArrayIndexOutOfBoundsException**

throw

Fehler "werfen"


Wir "werfen" eine Exception
selbst

d.h.: Wir programmieren eine Methode so, dass sie sich nicht selbst um den Fehler kümmert, sondern ihn an die Caller-Methode zurückgibt.

throw

wenn *Exception checked*
→ *in der Signatur deklarieren*

wenn *Exception unchecked*
→ *einfach "throwen"*



```
public static void checkedEx() throws FileNotFoundException
{
    FileReader fr = new FileReader("C:/gibtsnicht.dat");
    throw new FileNotFoundException();
}
```

throw

Wenn Fehler auftritt, muss er nicht behandelt werden, sondern kann auch "weggeworfen" werden.

Aufrufer muss diese Exception dann aber behandeln (oder selbst weiterwerfen!)

```
public static void main(String[] args)
{
    try {
        checkedEx();
        // MUSS hier mit try-catch gefangen werden!
    } catch (FileNotFoundException e) {
        System.out.println("Filenotfound-Ex!");
    }
}

public static void checkedEx() throws FileNotFoundException
{
    FileReader fr = new FileReader("C:/gibtsnicht.dat");
    throw new FileNotFoundException();
}
```