X: Ausnahmebehandlung

- Ausnahmen “werfen” (auslösen)
- Eigene Ausnahmen definieren
- Ausnahmen abfangen
- Ausnahmen definieren
- Sinnvolle Verwendung von Ausnahmen

Beispiel

```
public static int fakt(int x) {
    if (x < 0)
        throw new IllegalArgumentException
                    ("fakt: Negatives Argument " + x);
    else { ... }
}
```

Die Ausführung von `System.out.println(fakt(-2))` führt zu

```
Exception in thread "main" \
java.lang.IllegalArgumentException: \
fakt: Negatives Argument -2
        at Fakt.fakt(Fakt.java:7)
        at Fakt.main(Fakt.java:3)

Process Fakt exited abnormally with code 1
```

Was passiert hier?

- Durch das `throw` statement wird eine Ausnahme, hier `IllegalArgumentException` *geworfen*.
- Die Ausnahme ist ein spezielles Objekt, das wie üblich mit `new` erzeugt werden kann.
- Einer der Konstruktoren der verwendeten Ausnahme hat ein Argument vom Typ `String`.
- Das Werfen der Ausnahme bricht die Programmabarbeitung sofort ab.
- Es kommt zu einer Fehlermeldung aus der die Art der geworfenen Ausnahme, ihr `String`-Parameter, sowie der Ort (Klasse, Methode, Programmzeile) ihres Auftretens hervorgeht.

Vordefinierte Ausnahmen

- Alle Ausnahmen gehören zur (vordefinierten) Klasse `Exception` oder einer Unterklasse.
- `Exception` selbst ist Unterklasse von `Throwable`. Das “Argument” der `throw`-Anweisung muss `Throwable` sein.
- Unterklassen von `Exception` sind `IOException`, `ClassNotFoundException`, `CloneNotSupportedException`,`RuntimeException`, u.v.a.m.
- Unterklassen von `IOException` sind `EOFException` und `FileNotFoundException` u.v.a.m.
- Unterklassen von `RuntimeException` sind `IllegalArgumentException`, `IndexOutOfBoundsException` u.v.a.m.
- Unterklasse von `IllegalArgumentException` ist z.B.: `NumberFormatException`

Geprüfte Ausnahmen

Manche Ausnahmen sind *überprüft* (*checked*), andere nicht (*unchecked*).

Überprüfte Ausnahmen müssen entweder aufgefangen werden (kommt später) oder explizit als möglich deklariert werden:

```
import java.io.*;
...
    public void m() {
        throw new IOException("");
    }
```

Kompilieren führt zu folgender Fehlermeldung:

```
/home/mhofmann/work/teaching/InfoII/Fakt.java:19:\
 unreported exception java.io.IOException; must \
be caught or declared to be thrown
        throw new IOException("");
        ^
```

Deklaration überprüfter Ausnahmen

Kann in einer Methode eine überprüfte Ausnahme auftreten und wird sie nicht aufgefangen, so muss sie mit `throws` deklariert werden:

```
import java.io.*;
...
    public void m() throws IOException{
        throw new IOException("");
    }
```

Die “Philosophie” ist, dass Ausnahmen, deren Auftreten auf einen Programmierfehler hindeutet, nicht überprüft werden.

Ausnahmen, deren Auftreten von Zeit zu Zeit unvermeidlich ist (`FileNotFoundException`, sind dagegen überprüft.

`RuntimeException` und ihre Unterklassen sind nicht überprüft; alle anderen sind überprüft.

Deklaration eigener Ausnahmen

```
    public static int fakt(int x) {
        if (x < 0)
            throw new FakultaetAusnahme
                        ("fakt: Negatives Argument " + x);
        else { ... }
    }
 public class FakultaetAusnahme extends RuntimeException {
       public FakultaetAusnahme() {}
       public FakultaetAusnahme(String grund)
           {
              super(grund);
           }
}
```

Ausnahmen werden also genauso wie Klassen deklariert und sind ganz normale Objekte.

Abfangen von Ausnahmen

Mit `try catch` werden Ausnahmen abgefangen.

```
    public static void main(String[] args) {
        Scanner  konsole = new Scanner(System.in);
        try {
            System.out.println("Bitte die Zahl");
            String input = konsole.nextLine();
            int zahl = Integer.parseInt(input);
            System.out.println("Fakt(" + zahl + ") = " + fakt(zahl
        }
        catch (NumberFormatException ausnahme) {
            System.out.println("Falsche Eingabe.");
        }
        catch (IllegalArgumentException e) {
            System.out.print(e.getMessage());
        }
    }
```

Abfangen von Ausnahmen

- Wird im `try` Block eine Ausnahme geworfen, so werden der Reihe nach alle `catch` Blöcke (*handler*) durchgegangen.
- Der erste Handler, der zur geworfenen Ausnahme passt, kommt zur Anwendung. Sein formaler Parameter wird an das geworfene Ausnahmeobjekt gebunden und der entsprechende Code wird ausgeführt.
- Man sollte darauf achten, dass Handler die Ausnahmen sinnvoll bearbeiten.

  ```
  catch (Exception e) {}
  ```

  ist nicht sinnvoll.
- Jede Ausnahme versteht die Methode `printStackTrace()`. Dies gibt die Folge der Methodenaufrufe bis zu ihrer Auslösung auf der Konsole aus.

Die `finally`-Klausel

Es kann passieren, dass bestimmte Anweisungen auf jeden Fall durchgeführt werden müssen, auch wenn eine Ausnahme geworfen wird.

Man könnte dann die Ausnahme fangen, die bestimmten Anweisungen ausführen und dann die Ausnahme gleich wieder werfen.

Alternativ gibt es das `finally`-Konstrukt.

Beispiel:

```
try {
   /* Alle Kombinationen der Reihe nach
      durchprobieren. */
}
finally {
   /* Abhauen */
}
```

Im [Horstmann] stehen geringfügig sinnvollere Beispiele.

Das Singleton Pattern

Manchmal möchte man von einer Klasse nur eine einzige Instanz erzeugen.

Beispiel: Ausnahmen bei der Chipkartenprogrammierung.

Man kann dann zu Beginn ein Objekt erzeugen mit einer Methode, die die gewünschte Instanz liefert. Ist noch keine erzeugt, so wird eine erzeugt, ansonsten die bereits erzeugte zurückgeliefert.

Chipkarten (JavaCard) haben keine Garbage Collection; man muss daher vermeiden, zuviele Ausnahmeobjekte zu erzeugen.

Durchführung

```
class Fehler extends Exception {
    private String grund;
    private static Fehler instanz = null;;

    private Fehler() {}

    public String getGrund() {
        return grund;
    }
    private void setGrund(String s) {
        this.grund = s;
    }
    public static Fehler getInstanz(String s) {
        if (instanz == null) instanz = new Fehler();
        instanz.setGrund(s);
        return instanz;
    }}
```

Durchführung

```
import javax.swing.*;
public class Anwendung {
    public static void main(String[] args) {
        try {
            String input = JOptionPane.showInputDialog("Bitte die
            int n = Integer.parseInt(input);
            if (n < 0)
                throw Fehler.getInstanz("Negative Zahl " + n);
            System.out.println(n);
            System.exit(0);
        }
        catch (Fehler f) {
            System.out.println(f.getGrund());main(args);
        }
    }
}
```

Zusammenfassung

- Mit `throw` werden Ausnahmen geworfen.
- Eine nicht aufgefangene Ausnahme fürt zum Programmabbruch.
- Ausnahmen sind Objekte von Unterklassen von `Exception`.
- Es gibt vordefinierte Ausnahmen und selbstdefinierte.
- Mit `catch` werden Ausnahmen abgefangen.
- Ein `finally` Block wird immer ausgeführt, auch wenn der vorangegangene `try`-Block eine Ausnahme wirft.
- Das Singleton Entwurfsmuster besteht darin, von einem Objekt nur einmal eine Instanz zu erzeugen.
- Bei der Chipkartenprogrammierung wird das Singleton Entwurfsmuster für Ausnahmen verwendet, da keine Garbage Collection vorhanden ist.