Exceptions

- Reaktion auf Laufzeitfehler
- Definierte Fehlerbehandlung
- Sicheres Fortführen des Programmes

Exceptions

- **RuntimeException** (z. B.: `ArithmeticException`, `IndexOutOfBoundsException`)
- **IOException** (u. a.: `EOFException`, `FileNotFoundException`)
- …

Abfangen der Exceptions

```
try {
      . . .
//  Anweisungen, die möglicherweise
//  Ausnahme(n) verursachen

} catch (ExceptionTyp1 e1) {
      . . . //  Ausnahmenbehandlung 1
} catch (ExceptionTyp2 e2) {
      . . . //  Ausnahmenbehandlung 2
} catch     . . .

} finally {
      . . . //  Aufruf erfolgt immer
}
```

Exception- Beispiel

```
class ExceptionDivision {
public static void main(String args[]) {
int zaehler = 13; int nenner = 0; int bruch = 6;
    try { bruch = zaehler/nenner; }
        catch (ArithmeticException e) {
        System.out.println("Division durch 0 ");
        }
        finally {System.out.println(„Diese
Ausgabe erfolgt immer");}
    }
}
```

Exception- Beispiel - allgemeiner

```
class ExceptionDivision {
public static void main(String args[]) {
 int zaehler = 13; int nenner = 0; int bruch;
    try { bruch = zaehler/nenner; }
        catch (Exception e) {
        System.out.println("Fehler !!");
        System.out.println(e.toString());
        }
        finally {System.out.println(„Diese
Ausgabe erfolgt immer");}
    }
}
```

Exception- Behandlung in der aufrufenden Ebene

```
class ExceptionDivision {
public static void main(String args[]) {
    int zaehler = 13; int nenner = 0; int bruch = 6;
    try { bruch = divi(zaehler,nenner); }
        catch (ArithmeticException e) {
        System.out.println(e.getMessage());
        System.out.println("Division durch 0 ");
        }

    }
static int divi(int dividend, int divisor)
   {
    if (divisor == 0)
        throw new ArithmeticException ("Fehler in
Division");
            else return dividend / divisor;
    }
}
```

Aufgabe

Exceptions

Einlesen von Daten über die Tastatur

Klasse Keyin

Zur Verfügung gestellte Methoden

- `inString()`
- `inChar()`
- `inInt()`
- `inDouble()`
- `inputFlush()` – **löscht Eingangsbuffer**

Aufgabe

Einlesen von der Tastatur mit

- **Keyin und**
- **StringTokenizer**

Files – einfache Klassen

Verwendete Klassen (über import java.io.*;)

- `BufferedReader`
- `BufferedWriter`
- `FileReader`
- `FileWriter`

Zur Verfügung gestellte Methoden

- `write(), read()`
- `readInt(), readDouble(), writeInt(), writeDouble()`
- `close()`

Files – komplexe Klassen

Verwendete Klassen (über import java.io.*;)

- `DataInputStream`
- `DataOutputStream`
- `FileInputStream`
- `FileOutputStream`

Zur Verfügung gestellte Methoden

- `readInt(), readDouble(), writeInt(), writeDouble()`
- `close()`

Zur Abspeicherung von Instanzen ist Serialisierung erforderlich:

Entsprechende Klasse enthält Vererbung mittels „implements Serializable“

- `readObject(), writeObject()`

Aufgabe

- **Files**
- **Files mit Serialisierung**