Selectionsort – Beispiel

4 7 9 3 1 6

Selectionsort – Beispiel

4 7 9 3 1 6

1 **7** 9 3 4 6

Selectionsort – Beispiel

4	7	9	3	1	6
1	**7**	9	3	4	6
1	3	**9**	7	4	6

Selectionsort – Beispiel

4	7	9	3	1	6
1	**7**	9	3	4	6
1	3	**9**	7	4	6
1	3	4	**7**	9	6

Selectionsort – Beispiel

4	7	9	3	1	6
1	**7**	9	3	4	6
1	3	**9**	7	4	6
1	3	4	**7**	9	6
1	3	4	6	**9**	7

Selectionsort – Beispiel

4	7	9	3	1	6
1	**7**	9	3	4	6
1	3	**9**	7	4	6
1	3	4	**7**	9	6
1	3	4	6	**9**	7
1	3	4	6	7	9

Bubblesort – Beispiel

4 7 9 3 1 6

$\overline{6}$

1

3

9

7

4

Bubblesort – Beispiel

$\overline{6}$	9
1	$\overline{6}$
3	1
9	3
7	7
4	4

Bubblesort – Beispiel

$\overline{6}$	9	9
1	$\overline{6}$	7
3	1	$\overline{6}$
9	3	1
7	7	3
4	4	4

Bubblesort – Beispiel

$\overline{6}$	9	9	9
1	$\overline{6}$	7	7
3	1	$\overline{6}$	6
9	3	1	$\overline{4}$
7	7	3	1
4	4	4	3

Bubblesort – Beispiel

$\overline{6}$	9	9	9	9
1	$\overline{6}$	7	7	7
3	1	$\overline{6}$	6	6
9	3	1	$\overline{4}$	4
7	7	3	1	$\overline{3}$
4	4	4	3	1

Exceptions: Das Konzept

Exceptions sind gewisse Ausnahme-, d.h. Fehlersituationen, die zur Laufzeit ausgelöst werden können.

(*Throwing*)

Beispiel:

```
String str = null; System.out.println(str.length());
```

führt zum *Abbruch* des Programms mit der Fehlermeldung

```
NullPointerException
```

In Java kann eine Reaktion auf das Auftreten von Exceptions programmiert werden, so dass das Programm weiterlaufen kann.

(*Abfangen / Catching* von Exceptions)

Exceptions vs. Errors

Exceptions	Errors
eher leichte Laufzeitfehler	eher schwere Laufzeitfehler
können abgefangen werden	führen zum Programmabbruch
Beispiele:	
`ArithmeticException`	`NoClassDefFoundError`
`ArrayIndexOutOfBoundsException`	`OutOfMemoryError`
`StringIndexOutOfBoundsException`	`Internal Error`
`NumberFormatException`	
`EOFException`	

Exceptions sind Objekte

- Exception-Typ $\longrightarrow$ Klasse (Bsp.: `java.lang.NullPointerException`).
- Tritt ein Laufzeitfehler ein, wird ein Exemplar der jeweiligen Exception-Klasse erzeugt (*Throwing*).
- Alle Exception-Klassen sind Unterklassen von `java.lang.Exception`.
  - hierarchischer Aufbau der Exceptions
  - Verteilung auf verschiedene Pakete
- Sie besitzen daher alle gewisse Methoden, u.a.:
  - `getMessage()`, die bestimmte Informationen über den Fehlerfall liefert

    Beispiel: `java.lang.ArrayIndexOutOfBoundsException: -1`
  - printStackTrace(), die die Fehlermeldung und die dynamische Aufruf-hierarchie auf `stdout` ausgibt

Explizites Abfangen von Exceptions

```
try {
     // Anweisungen, in denen eine XYException
     // ausgelöst werden kann
}

catch (XYException e) {
     // Anweisungen, die beim Auftreten einer
     // XYException ausgeführt werden, z.B.
     System.out.println(e.getMessage());
}
```

mehrere Fehlerarten $\longrightarrow$ mehrere `catch`-Blöcke

Weitergeben von Exceptions

- Alternativ werden Exceptions zur Behandlung weitergegeben, und zwar
  - erst an übergeordnete Programmblöcke derselben Methode,
  - dann (entlang der dynamischen Aufrufhierarchie) an die Aufrufer der Methode,

  in der die Exception ausgelöst wurde.

  Wird sie nirgens explizit abgefangen, so bricht das Programm ab.
- die `throws`-Klausel signalisiert, welche Exceptions die jeweilige Methode nicht selbst behandelt:

  ```
  public void methode() throws IOException { ... }
  ```
- Auf alle Exceptions außer `RuntimeExceptions` muss der Programmierer reagieren!!!

Benutzerdefinierte Exceptions

- für Fehlersituationen im Zusammenhang mit benutzerdefinierten Klassen
- Auslösen mit `throw` <*Exemplar eine Exception-Klasse*>, z.B.:

```
if ( //Fehlersituation )
        throw new RuntimeException("Denominator is zero.");
```

- Jede Exception-Klasse besitzt Konstruktoren

XY`Exception()` und XY`Exception(String message)`

- benutzerdefinierte Exceptions als Unterklasse von einer Exception-Klasse:

`MyException(){super()}` und
`MyException(String msg){super(msg)}`