

12. Java Fehler und Ausnahmen

Fehler, Systemausnahmen, Benutzerausnahmen, Behandeln von Ausnahmen, Spezialfall Ressourcen

Fehler und Ausnahmen in Java

Fehler und Ausnahmen unterbrechen die normale Programmausführung abrupt und stellen ein *nicht geplantes Ereignis* dar.



Ausnahmen sind böse, oder doch nicht?

- Java ermöglicht es, solche Ereignisse abzufangen und zu behandeln (als Alternative zum Programmabsturz).
- Nicht behandelte Fehler und Ausnahmen werden durch den Aufrufstapel hochgereicht.

Fehler (Errors)



Hier ist nichts mehr zu machen

Fehler treten in der virtuellen Maschine von Java auf und sind *nicht reparierbar*.

Beispiele

- Kein Speicher mehr verfügbar
- Zu hoher Aufrufstapel (→ Rekursion)
- Fehlende Programmbibliotheken
- Bug in der virtuellen Maschine
- Computer-Hardwarefehler

Ausnahmen (Exceptions)

Ausnahmen werden von der virtuellen Maschine oder vom Programm selbst ausgelöst und können meist behandelt werden um die *Normalsituation wiederherzustellen*



Aufwischen und neu einschenken

Beispiele

- Dereferenzierung von `null`
- Division durch 0
- Schreib/Lesefehler (Dateien)
- Businesslogik Fehler

Arten von Ausnahmen

Systemausnahmen

(runtime exceptions)

- Können überall auftreten
- *Können* behandelt werden
- Ursache: Bug im Programm

Benutzerausnahmen

(checked exceptions)

- Müssen deklariert werden
- *Müssen* behandelt werden
- Ursache: Unwahrscheinliches, aber prinzipiell mögliches Ereignis

Arten von Ausnahmen

Systemausnahmen

(runtime exceptions)

- Können überall auftreten
- *Können* behandelt werden
- Ursache: Bug im Programm

Benutzerausnahmen

(checked exceptions)

- Müssen deklariert werden
- *Müssen* behandelt werden
- Ursache: Unwahrscheinliches, aber prinzipiell mögliches Ereignis

Beispiel einer Systemausnahme

```
1 import java.util.Scanner;
2 class ReadTest {
3     public static void main(String[] args){
4         int i = readInt("Zahl");
5     }
6     private static int readInt(String prompt){
7         System.out.print(prompt + ": ");
8         Scanner input = new Scanner(System.in);
9         return input.nextInt();
10    }
11 }
```

Eingabe: Zahl: asdf

Nicht behandelte Fehler und Ausnahmen

Das Programm stürzt ab und hinterlässt auf der Konsole eine “Aufrufstapelzurückverfolgung” 😊 (ab jetzt: *Stacktrace*). Darin sehen wir, wo genau das Programm abgebrochen wurde.

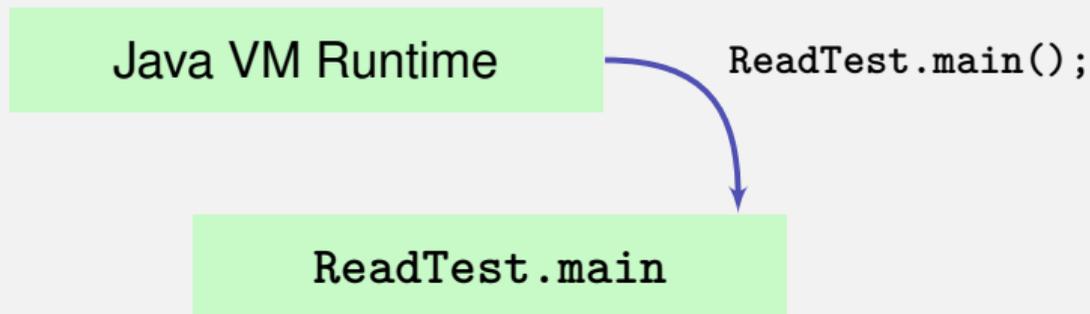
```
Exception in thread "main" java.util.InputMismatchException
    [...]
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at ReadTest.readInt(ReadTest.java:9)
    at ReadTest.main(ReadTest.java:4)
```

⇒ Forensische Nachforschungen mit Hilfe dieser Information.

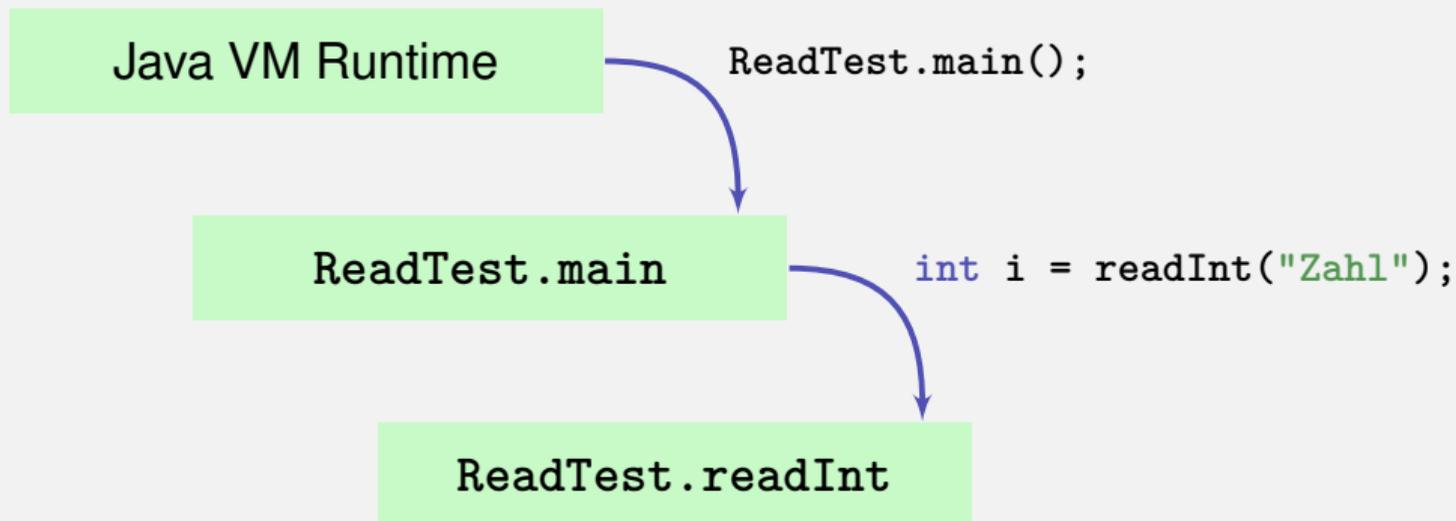
Ausnahme propagiert durch Aufrufstapel

Java VM Runtime

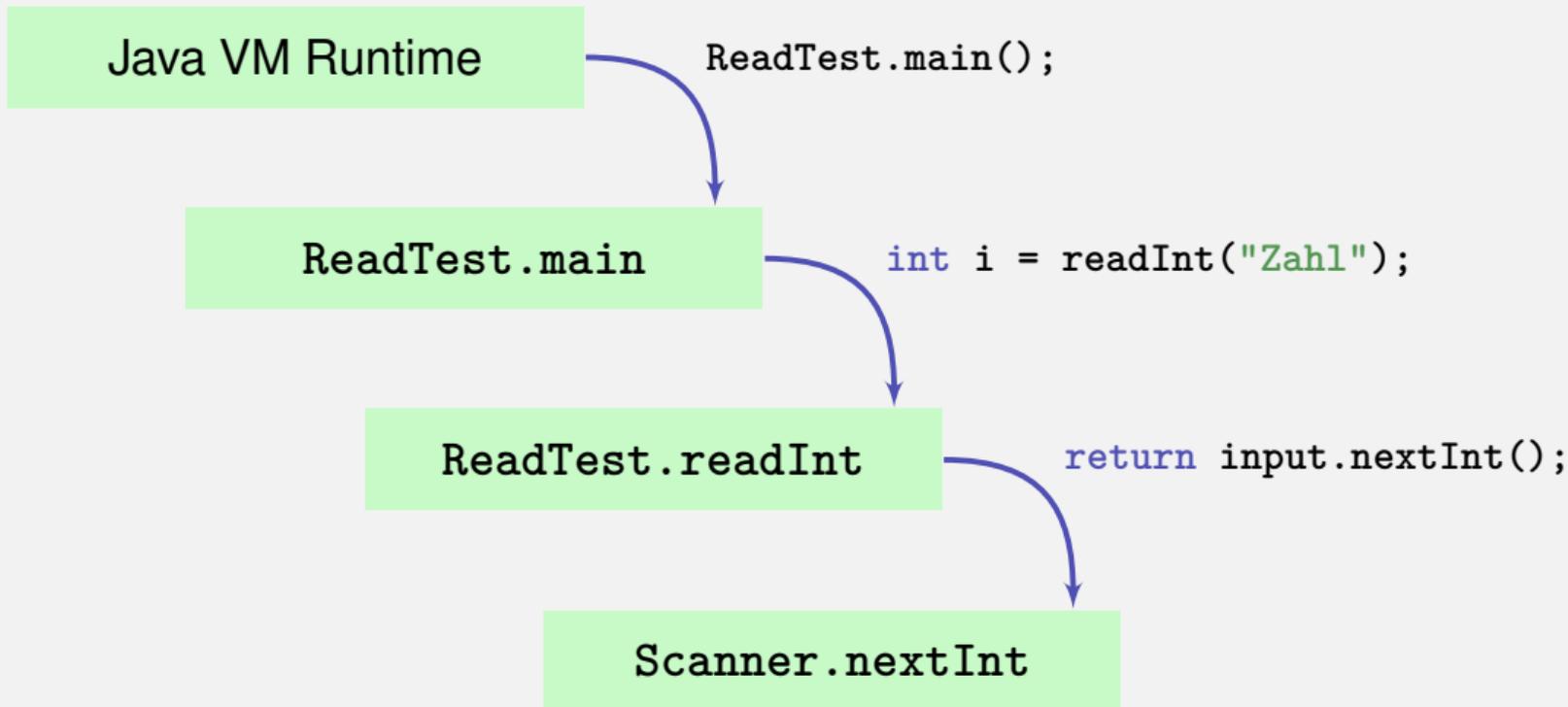
Ausnahme propagiert durch Aufrufstapel



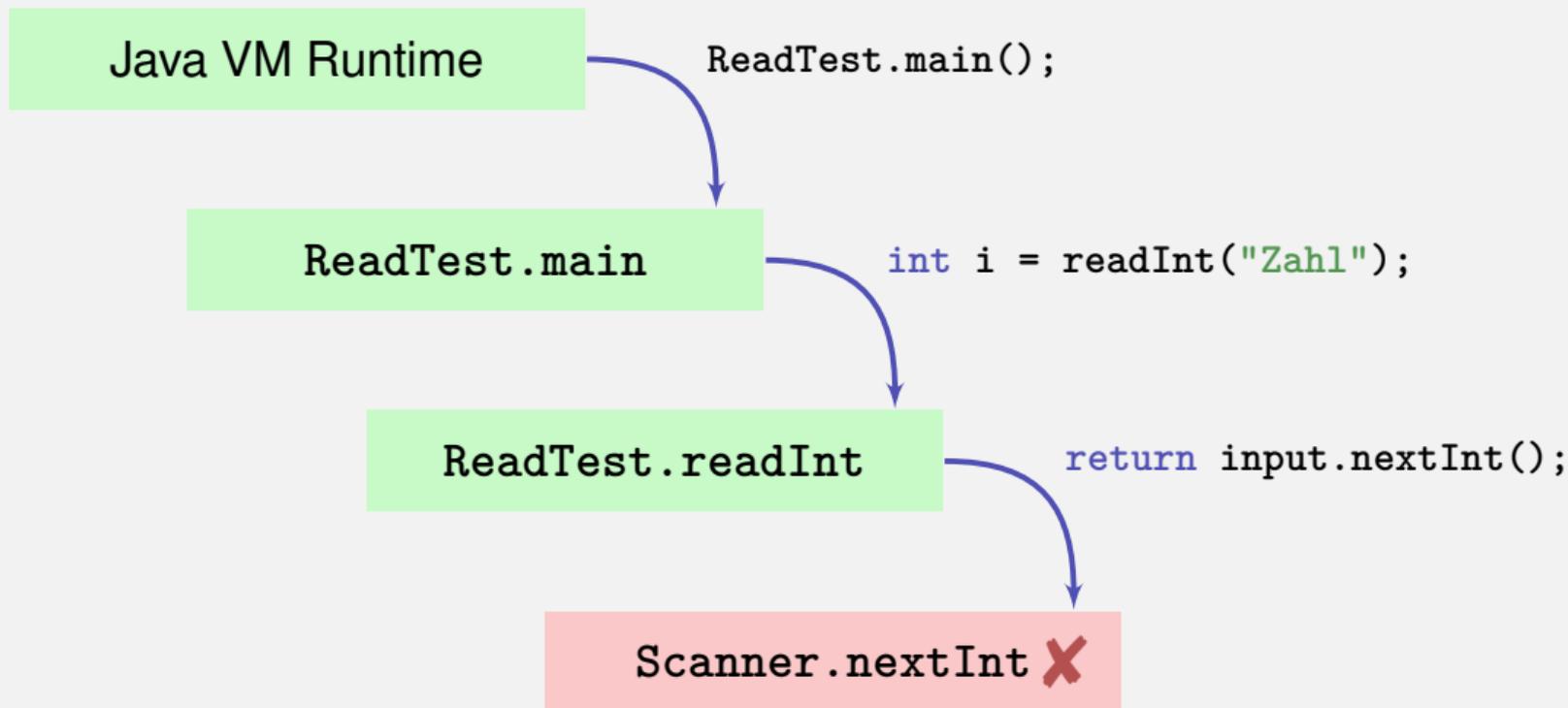
Ausnahme propagiert durch Aufrufstapel



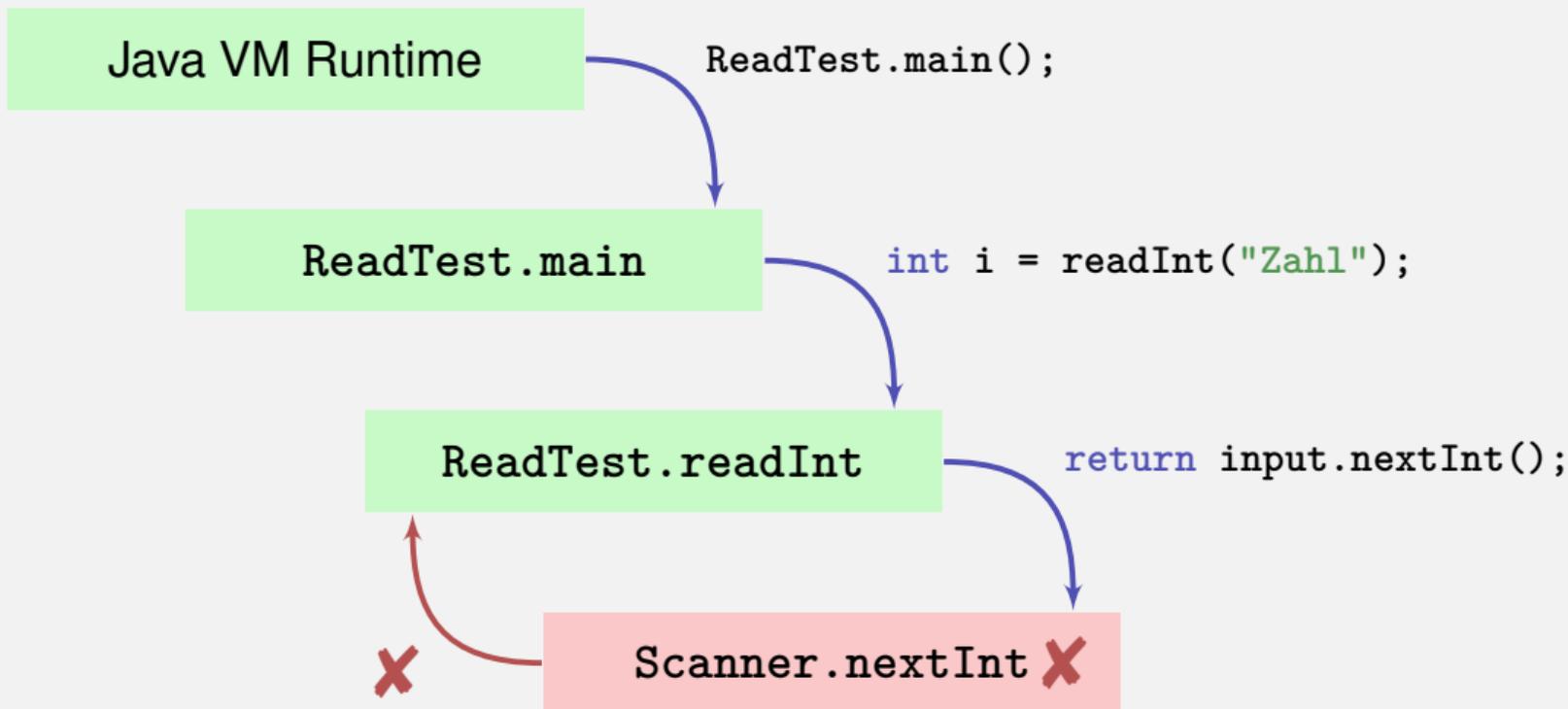
Ausnahme propagiert durch Aufrufstapel



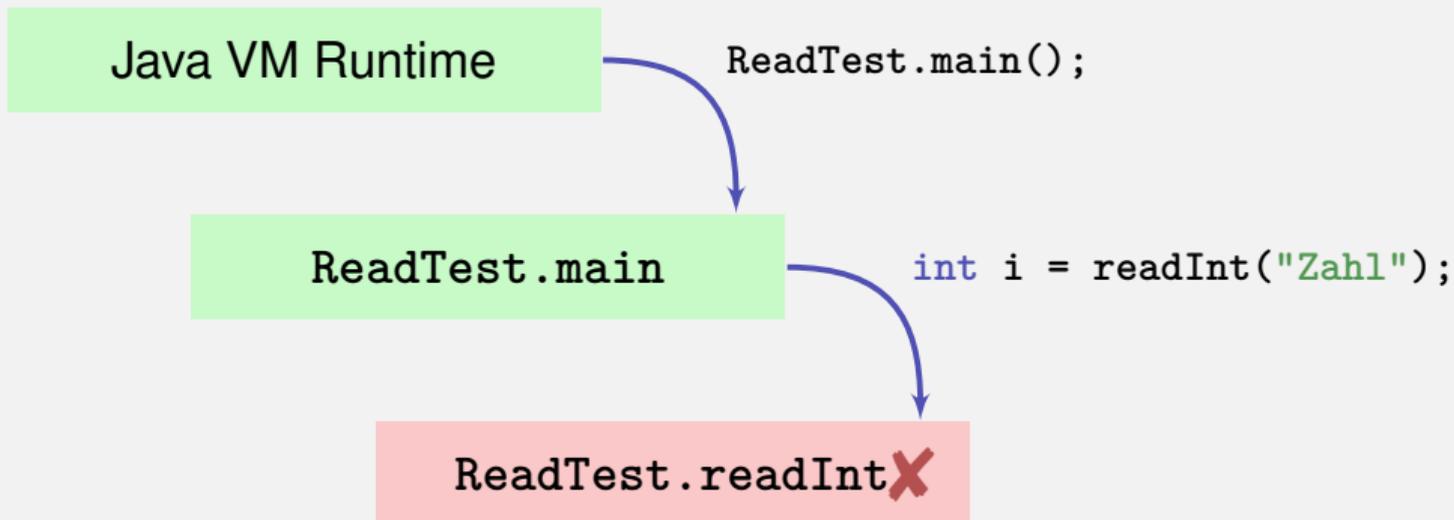
Ausnahme propagiert durch Aufrufstapel



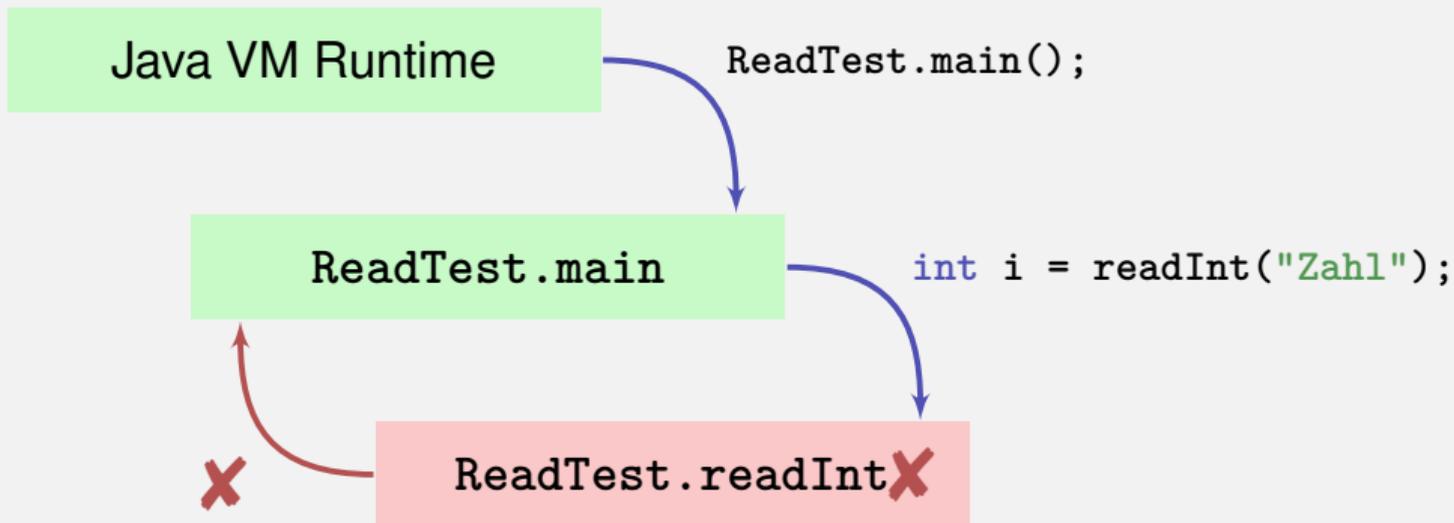
Ausnahme propagiert durch Aufrufstapel



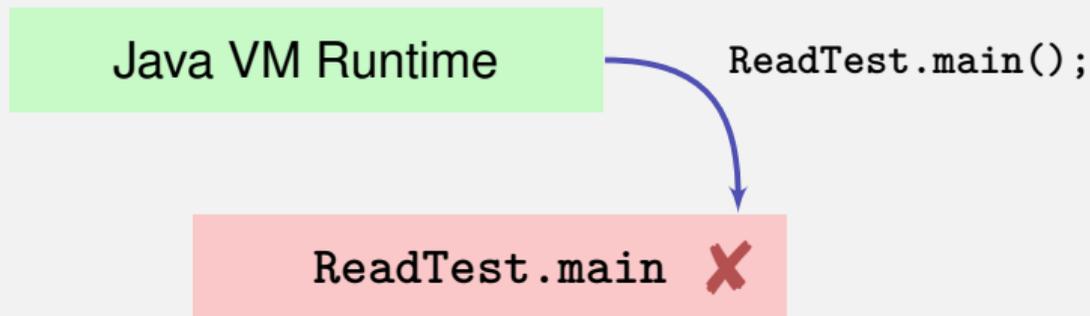
Ausnahme propagiert durch Aufrufstapel



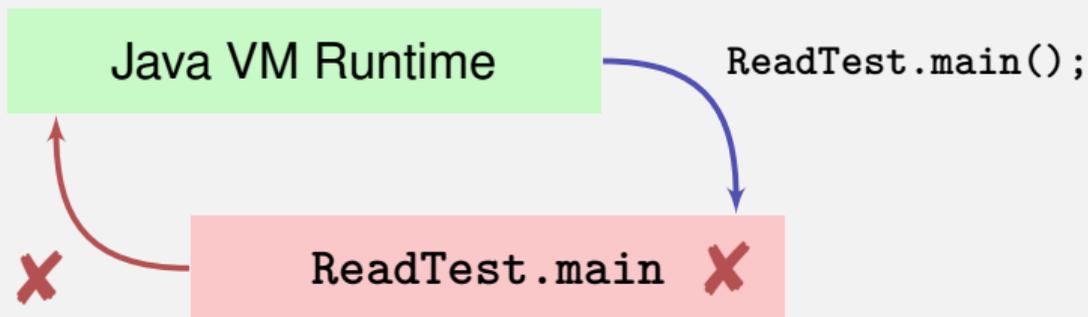
Ausnahme propagiert durch Aufrufstapel



Ausnahme propagiert durch Aufrufstapel



Ausnahme propagiert durch Aufrufstapel



Ausnahme propagiert durch Aufrufstapel

Java VM Runtime †

Stacktraces verstehen

Ausgabe:

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at ReadTest.readInt(ReadTest.java:9)
    at ReadTest.main(ReadTest.java:4)
```

Stacktraces verstehen

Eine unpassende Eingabe ...

Ausgabe:

Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Scanner.java:864)
at java.util.Scanner.next(Scanner.java:1485)
at java.util.Scanner.nextInt(Scanner.java:2117)
at java.util.Scanner.nextInt(Scanner.java:2076)
at ReadTest.readInt(ReadTest.java:9)
at ReadTest.main(ReadTest.java:4)

Stacktraces verstehen

Ausgabe:

```
Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:864)
  at java.util.Scanner.next(Scanner.java:1485)
  at java.util.Scanner.nextInt(Scanner.java:2117)
  at java.util.Scanner.nextInt(Scanner.java:2076)
  at ReadTest.readInt(ReadTest.java:9)
  at ReadTest.main(ReadTest.java:4)
```

Stacktraces verstehen

Ausgabe:

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at ReadTest.readInt(ReadTest.java:9)
    at ReadTest.main(ReadTest.java:4)
```

... in Methode readInt auf Zeile 9 ...

Stacktraces verstehen

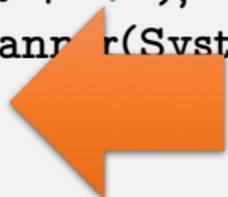
Ausgabe:

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at ReadTest.readInt(ReadTest.java:9)
    at ReadTest.main(ReadTest.java:4)
```

... aufgerufen durch Methode `main` auf Zeile 4.

Stacktraces verstehen

```
1 import java.util.Scanner;
2 class ReadTest {
3     public static void main(String[] args){
4         int i = readInt("Zahl");
5     }
6     private static int readInt(String prompt){
7         System.out.print(prompt + ": ");
8         Scanner input = new Scanner(System.in);
9         return input.nextInt();
10    }
11 }
```



```
at ReadTest.readInt(ReadTest.java:9)
at ReadTest.main(ReadTest.java:4)
```

Stacktraces verstehen

```
1 import java.util.Scanner;
2 class ReadTest {
3     public static void main(String[] args){
4         int i = readInt("Zahl");
5     }
6     private static int readInt(String prompt){
7         System.out.print(prompt + ": ");
8         Scanner input = new Scanner(System.in);
9         return input.nextInt();
10    }
11 }
```



```
at ReadTest.readInt(ReadTest.java:9)
at ReadTest.main(ReadTest.java:4)
```

Systemausnahme: Bug im Programm?!

Wo ist der Fehler?

```
private static int readInt(String prompt){  
    System.out.print(prompt + ": ");  
    Scanner input = new Scanner(System.in);  
    return input.nextInt();  
}
```

Systemausnahme: Bug im Programm?!

Wo ist der Fehler?

```
private static int readInt(String prompt){  
    System.out.print(prompt + ": ");  
    Scanner input = new Scanner(System.in);  
    return input.nextInt();  
}
```



Nicht garantiert, dass als nächstes ein `int` anliegt.

Systemausnahme: Bug im Programm?!

Wo ist der Fehler?

```
private static int readInt(String prompt){  
    System.out.print(prompt + ": ");  
    Scanner input = new Scanner(System.in);  
    return input.nextInt();  
}
```



Nicht garantiert, dass als nächstes ein `int` anliegt.

- ⇒ Die Scanner Klasse bietet ein Test dafür an
- ⇒ Optionale Datentypen ermöglichen guten Stil

Erste Erkenntnis: Oft keine Ausnahmesituation

Oft sind die “Sonderfälle” gar kein besonderes Ereignis, sondern absehbar. Hier sollten *keine* Ausnahmen verwendet werden!



Kinder kippen Becher um.
Man gewöhnt sich daran.

Beispiele

- Falsche Credentials beim Einloggen
- Leere Pflichtfelder in Eingabemasken
- Nicht verfügbare Internet-Ressourcen
- Timeouts

Zweite Erkenntnis: Ausnahmen verhindern



Problem gelöst.

Statt eine Systemausnahme abzuwarten *aktiv verhindern*, dass diese überhaupt auftreten kann.

Beispiele

- Usereingaben frühzeitig prüfen
- Optionale Typen verwenden
- Timeout Situationen voraussehen
- Plan B für nicht verfügbare Ressourcen

Arten von Ausnahmen

Systemausnahmen

(runtime exceptions)

- Können überall auftreten
- *Können* behandelt werden
- Ursache: Bug im Programm

Benutzerausnahmen

(checked exceptions)

- Müssen deklariert werden
- *Müssen* behandelt werden
- Ursache: Unwahrscheinliches, aber prinzipiell mögliches Ereignis

Arten von Ausnahmen

Systemausnahmen

(runtime exceptions)

- Können überall auftreten
- *Können* behandelt werden
- Ursache: Bug im Programm

Benutzerausnahmen

(checked exceptions)

- Müssen deklariert werden
- *Müssen* behandelt werden
- Ursache: Unwahrscheinliches, aber prinzipiell mögliches Ereignis

Beispiel einer Benutzerausnahme

```
private static String[] readFile(String filename){  
    FileReader fr = new FileReader(filename);  
    BufferedReader bufr = new BufferedReader(fr);  
    ...  
    line = bufr.readLine();  
    ...  
}
```

Beispiel einer Benutzerausnahme

```
private static String[] readFile(String filename){  
    FileReader fr = new FileReader(filename);  
    BufferedReader bufr = new BufferedReader(fr);  
    ...  
    line = bufr.readLine();  
    ...  
}
```

Compiler Fehler:

```
./Root/Main.java:9: error: unreported exception FileNotFoundException; must be caught or declared to be  
    FileReader fr = new FileReader(filename);  
                    ^
```

```
./Root/Main.java:11: error: unreported exception IOException; must be caught or declared to be thrown  
    String line = bufr.readLine();  
                    ^
```

2 errors

Kurzer Blick in die Javadoc

`readLine`

```
public String readLine()  
                throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

`IOException` - If an I/O error occurs

See Also:

`Files.readAllLines(java.nio.file.Path, java.nio.charset.Charset)`

Warum eine Benutzerausnahme?

Situation rechtfertigt die Benutzerausnahme:

- Fehlerfall ist *unwahrscheinlich aber prinzipiell möglich* – und kann durch geeignete Massnahmen zur Laufzeit behoben werden können.

Warum eine Benutzerausnahme?

Situation rechtfertigt die Benutzerausnahme:

- Fehlerfall ist *unwahrscheinlich aber prinzipiell möglich* – und kann durch geeignete Massnahmen zur Laufzeit behoben werden können.

Der Aufrufer einer Methode mit einer deklarierten Benutzerausnahme wird gezwungen, sich damit zu beschäftigen – behandeln oder weiterreichen.

Behandeln von Ausnahmen

```
private static String[] readFile(String filename){  
    try{  
        FileReader fr = new FileReader(filename);  
        BufferedReader bufr = new BufferedReader(fr);  
        ...  
        line = bufr.readLine();  
        ...  
    } catch (IOException e){  
        // do some recovery handling  
    } finally {  
        // close resources  
    }  
}
```

*Geschützter
Bereich*

Behandeln von Ausnahmen

```
private static String[] readFile(String filename){
    try{
        FileReader fr = new FileReader(filename);
        BufferedReader bufr = new BufferedReader(fr);
        ...
        line = bufr.readLine();
        ...
    } catch (IOException e){
        // do some recovery handling
    } finally {
        // close resources
    }
}
```

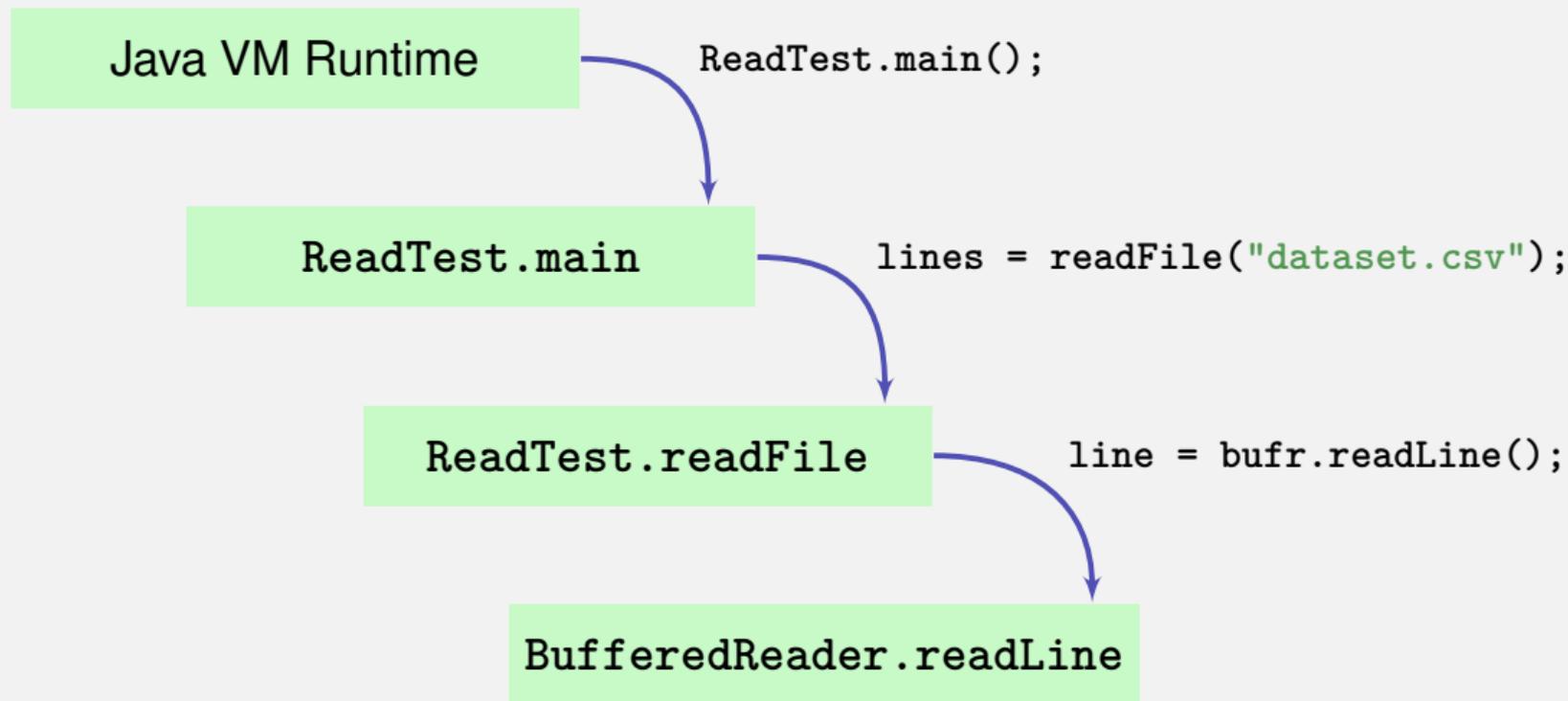
Massnahmen zur Wiederherstellung der Normalsituation

Behandeln von Ausnahmen

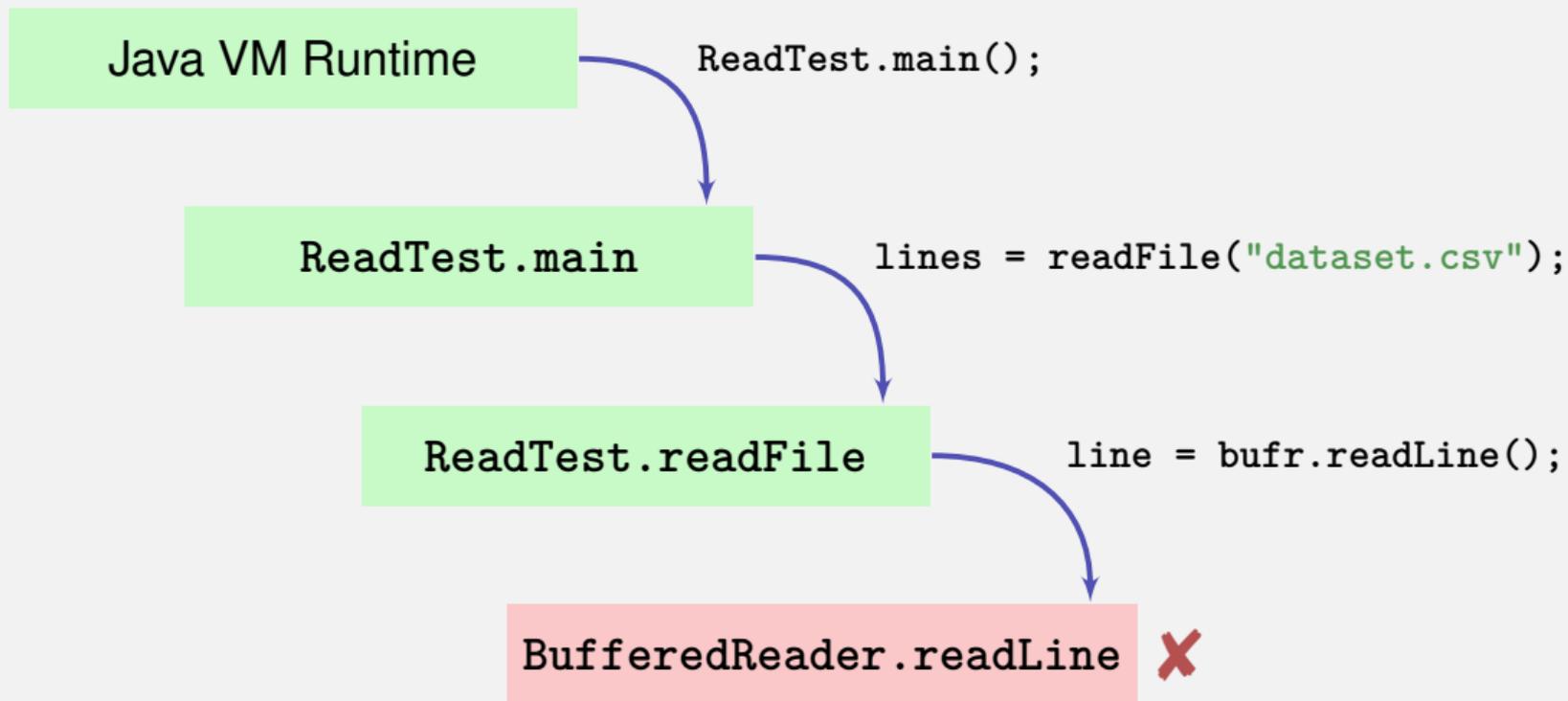
```
private static String[] readFile(String filename){
    try{
        FileReader fr = new FileReader(filename);
        BufferedReader bufr = new BufferedReader(fr);
        ...
        line = bufr.readLine();
        ...
    } catch (IOException e){
        // do some recovery handling
    } finally {
        // close resources
    }
}
```

← Wird in jedem Fall am Schluss ausgeführt, immer!

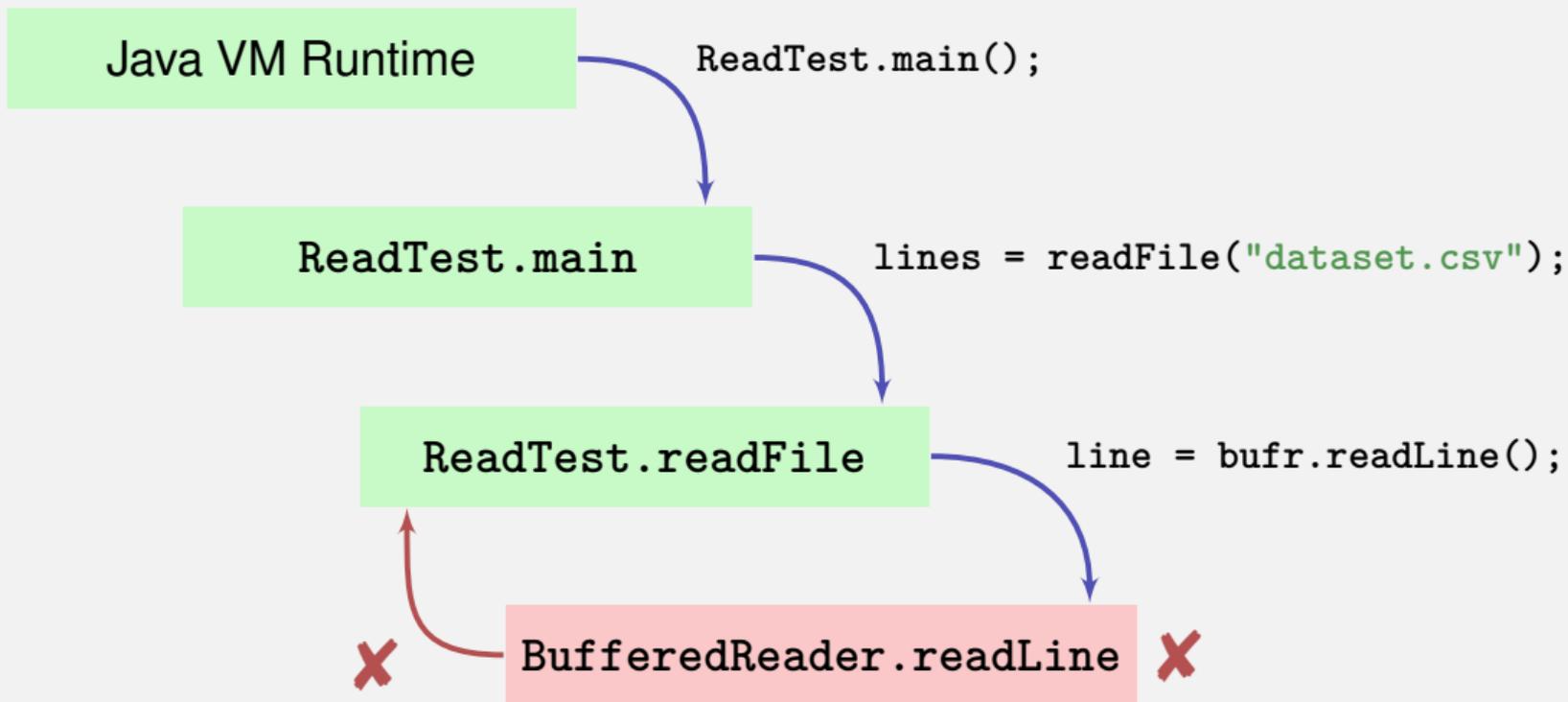
Behandlung von Ausnahmen: Propagieren stoppen!



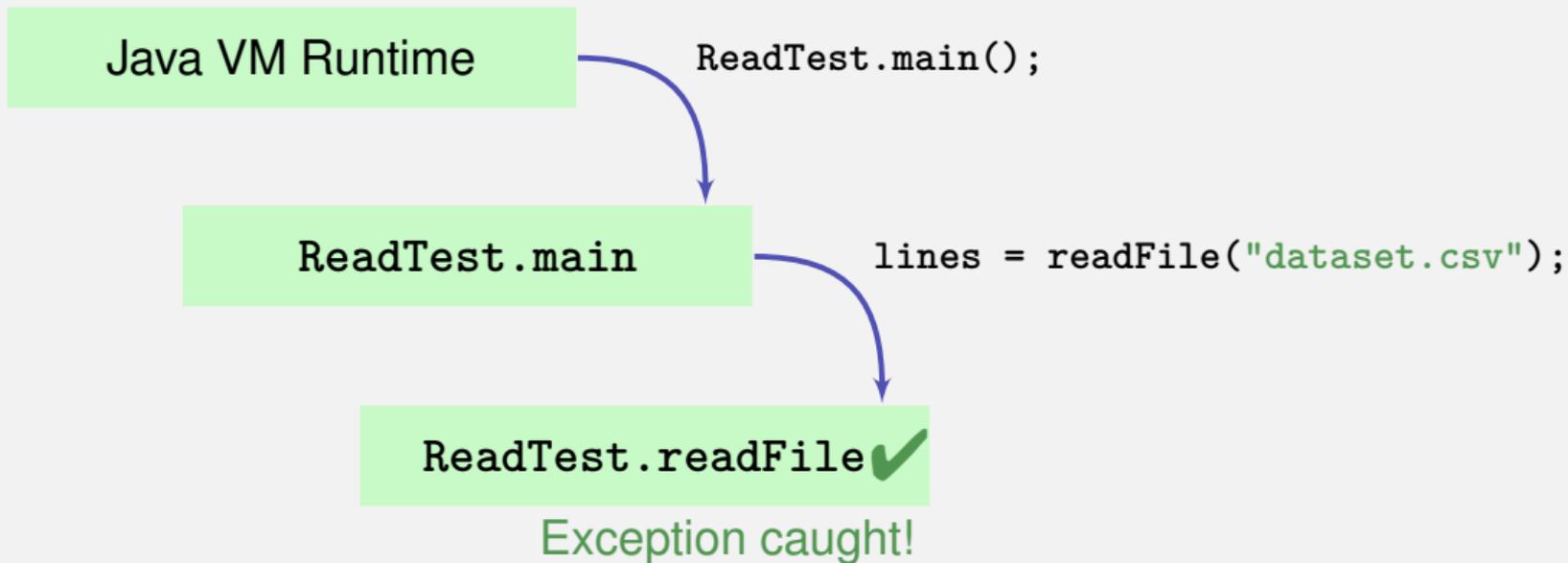
Behandlung von Ausnahmen: Propagieren stoppen!



Behandlung von Ausnahmen: Propagieren stoppen!



Behandlung von Ausnahmen: Propagieren stoppen!



Finally: Ressourcen schliessen!

In Java müssen *Ressourcen* unbedingt geschlossen werden nach Gebrauch. Ansonsten wird Speicher nicht freigegeben.

Ressourcen:

- Dateien
- Datenströme
- GUI Elemente
- ...



Try-with-resources Anweisung

Spezifische Syntax an, um Ressourcen *automatisch* zu schliessen:

```
private static String[] readFile(String filename){
    try ( FileReader fr = new FileReader(filename);
         BufferedReader bufr = new BufferedReader(fr)) {
        ...
        line = bufr.readLine();
        ...
    } catch (IOException e){
        // do some recovery handling
    }
}
```

Try-with-resources Anweisung

Spezifische Syntax an, um Ressourcen *automatisch* zu schliessen:

```
private static String[] readFile(String filename){  
    try (  
        FileReader fr = new FileReader(filename);  
        BufferedReader bufr = new BufferedReader(fr)) {  
        ...  
        line = bufr.readLine();  
        ...  
    } catch (IOException e){  
        // do some recovery handling  
    }  
}
```

Ressourcen werden hier geöffnet

Try-with-resources Anweisung

Spezifische Syntax an, um Ressourcen *automatisch* zu schliessen:

```
private static String[] readFile(String filename){
    try ( FileReader fr = new FileReader(filename);
         BufferedReader bufr = new BufferedReader(fr)) {
        ...
        line = bufr.readLine();
        ...
    } catch (IOException e){
        // do some recovery handling
    } ← Ressourcen werden hier automatisch geschlossen
}
```

13. Sortieren

Einfache Sortierverfahren, Quicksort

Problemstellung

Eingabe: Ein Array $A = (A[1], \dots, A[n])$ der Länge n .

Ausgabe: Eine Permutation A' von A , die sortiert ist: $A'[i] \leq A'[j]$ für alle $1 \leq i \leq j \leq n$.

Sortieren durch Auswahl

5 6 2 8 4 1 ($i = 1$)
↑

- Iteratives Vorgehen wie bei Bubblesort.

Sortieren durch Auswahl

5 6 2 8 4 1 ($i = 1$)
↑

- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl

5 6 2 8 4 1 ($i = 1$)



1 6 2 8 4 5 ($i = 2$)



- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl

5 6 2 8 4 1 ($i = 1$)

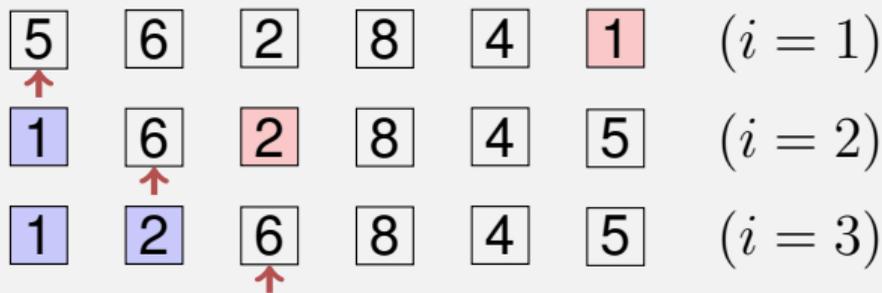


1 6 2 8 4 5 ($i = 2$)



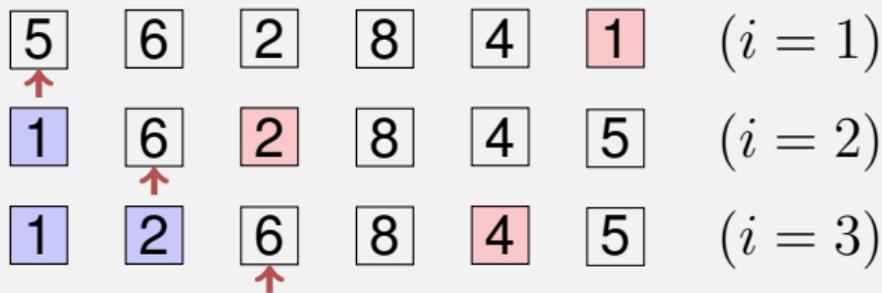
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



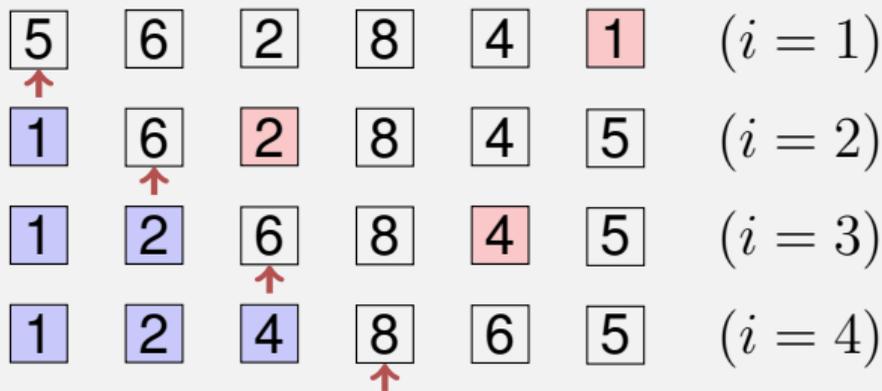
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



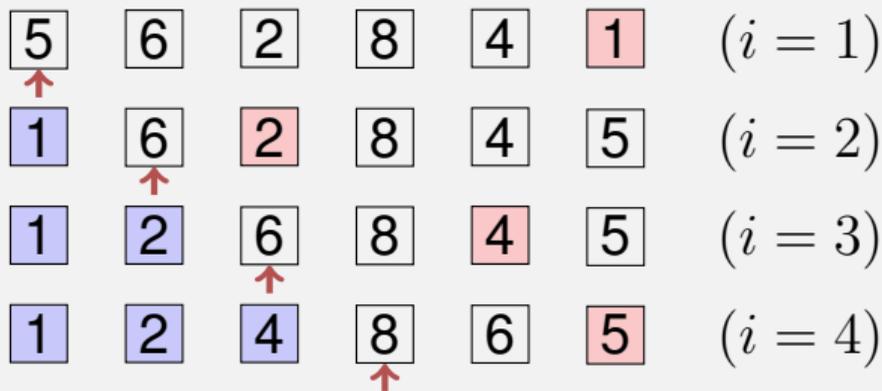
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



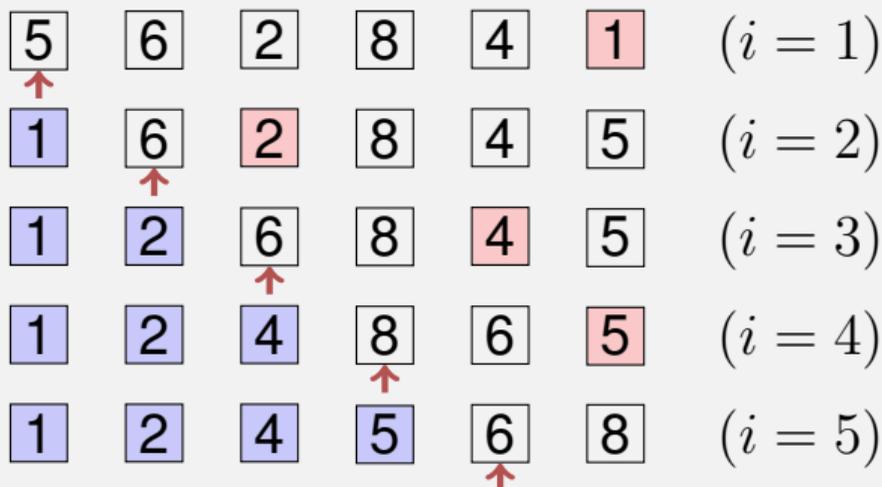
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



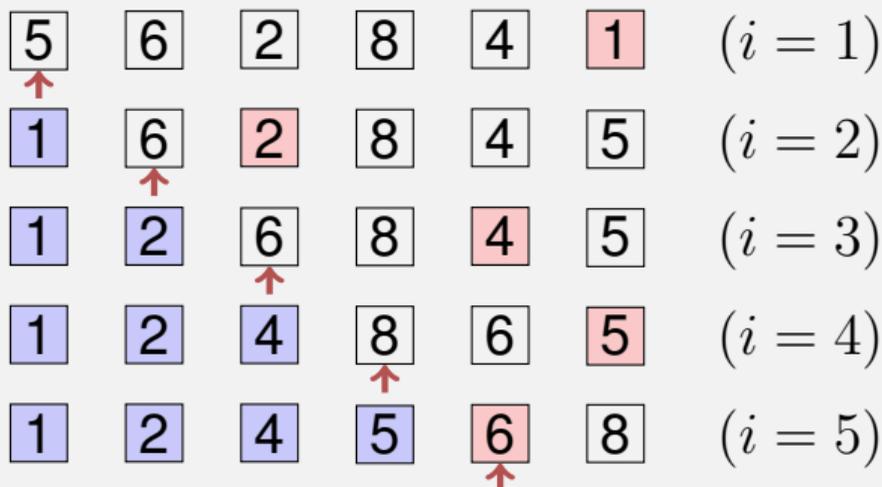
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



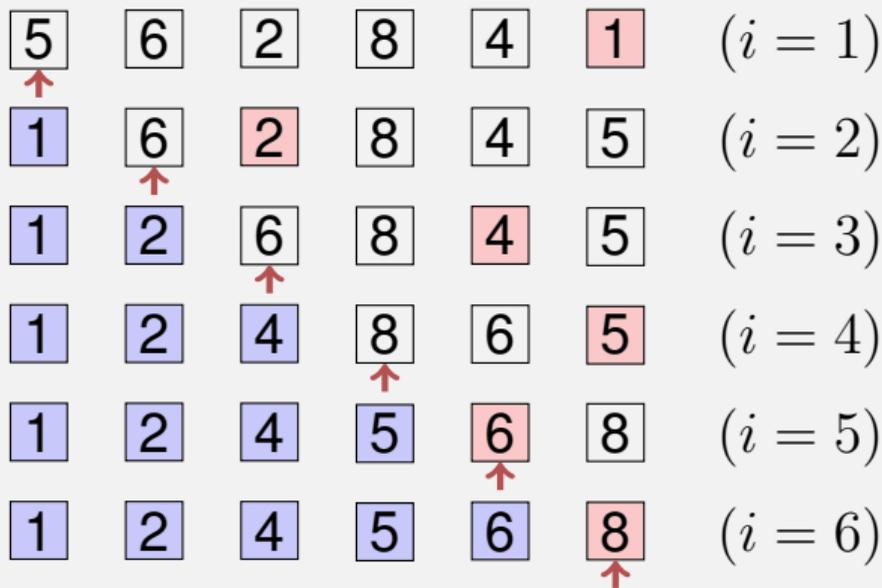
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



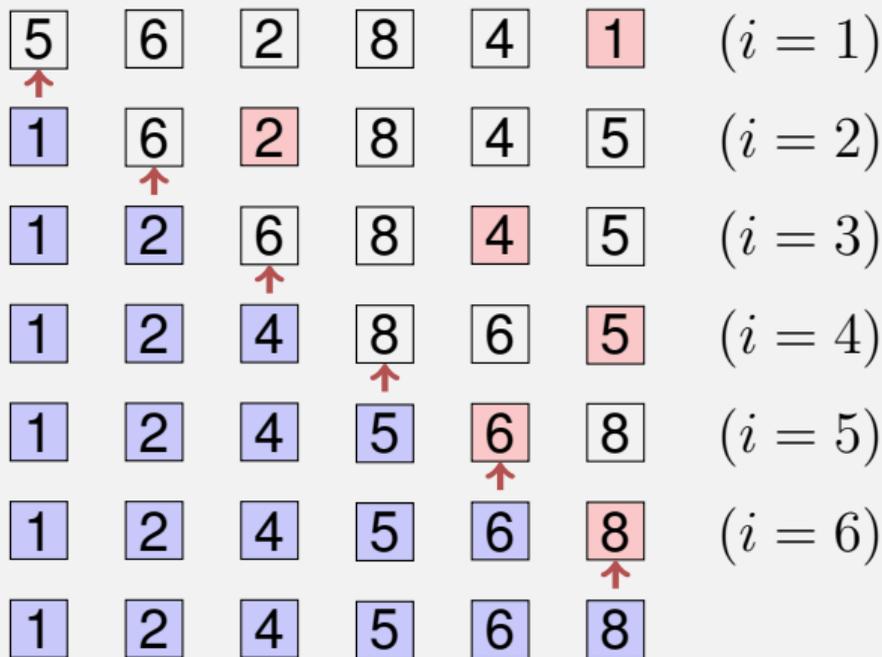
- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Sortieren durch Auswahl



- Iteratives Vorgehen wie bei Bubblesort.
- Auswahl des kleinsten (oder grössten) Elementes durch direkte Suche.

Algorithmus: Sortieren durch Auswahl

Input : Array $A = (A[1], \dots, A[n])$, $n \geq 0$.

Output : Sortiertes Array A

for $i \leftarrow 1$ **to** $n - 1$ **do**

$p \leftarrow i$

for $j \leftarrow i + 1$ **to** n **do**

if $A[j] < A[p]$ **then**

$p \leftarrow j$;

 swap($A[i], A[p]$)

Analyse

Anzahl Vergleiche im schlechtesten Fall:

Analyse

Anzahl Vergleiche im schlechtesten Fall: $\Theta(n^2)$.

Anzahl Vertauschungen im schlechtesten Fall:

Analyse

Anzahl Vergleiche im schlechtesten Fall: $\Theta(n^2)$.

Anzahl Vertauschungen im schlechtesten Fall: $n - 1 = \Theta(n)$

Anzahl Vergleiche im besten Fall:

Analyse

Anzahl Vergleiche im schlechtesten Fall: $\Theta(n^2)$.

Anzahl Vertauschungen im schlechtesten Fall: $n - 1 = \Theta(n)$

Anzahl Vergleiche im besten Fall: $\Theta(n^2)$.

Sortieren durch Einfügen

5 | 6 2 8 4 1 ($i = 1$)

Sortieren durch Einfügen

↑ 5 | 6 2 8 4 1 ($i = 1$)

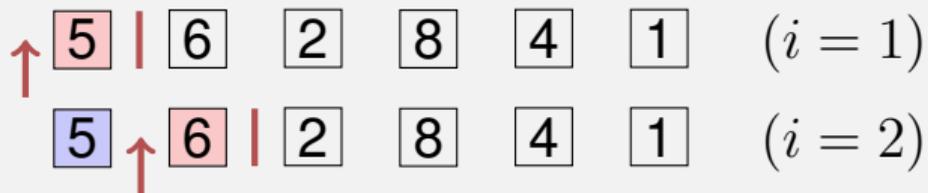
- Iteratives Vorgehen:
 $i = 1 \dots n$

Sortieren durch Einfügen



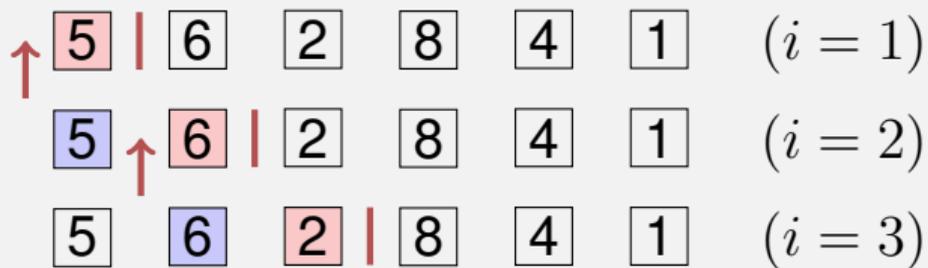
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.

Sortieren durch Einfügen



- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen,

Sortieren durch Einfügen



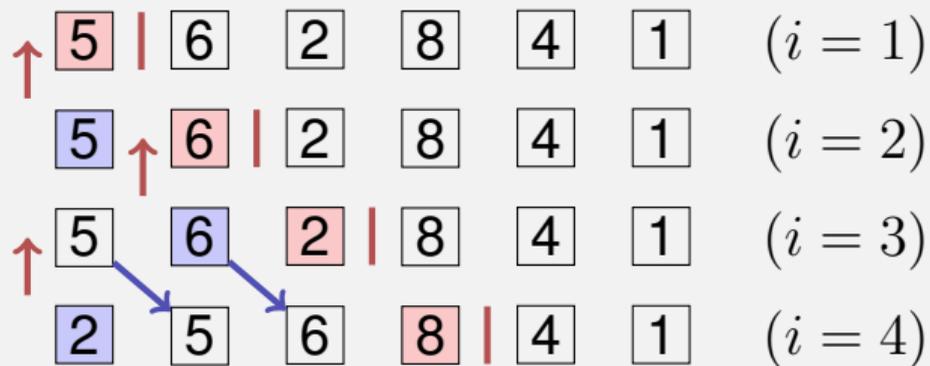
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen,

Sortieren durch Einfügen



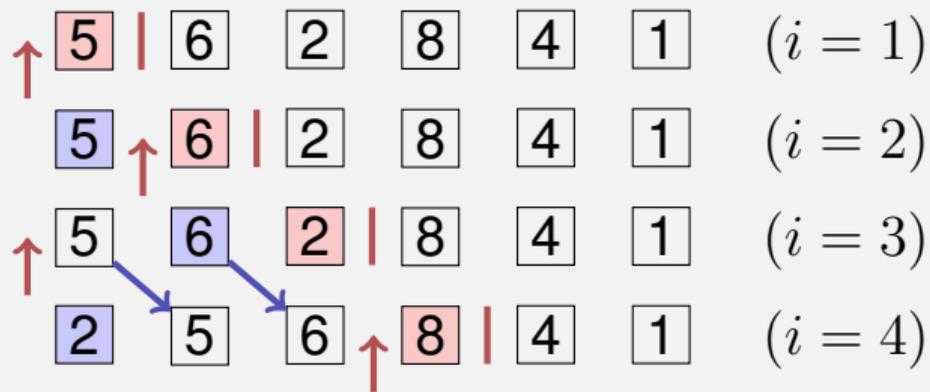
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen,

Sortieren durch Einfügen



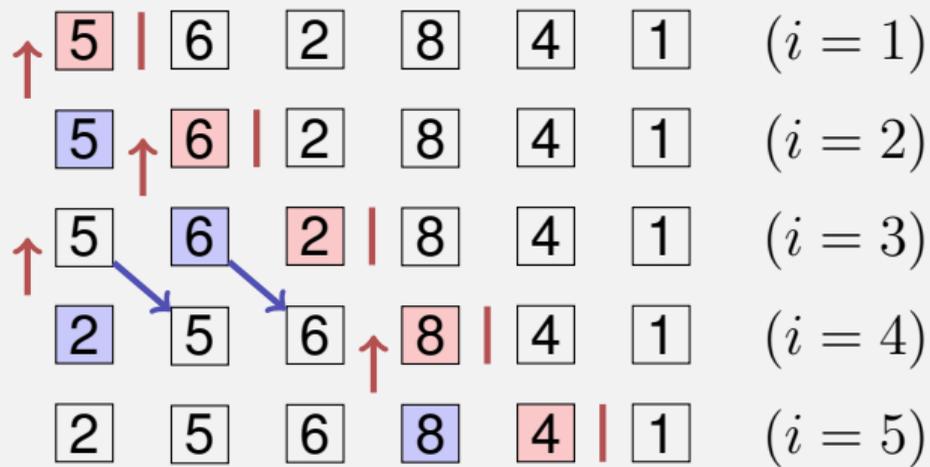
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



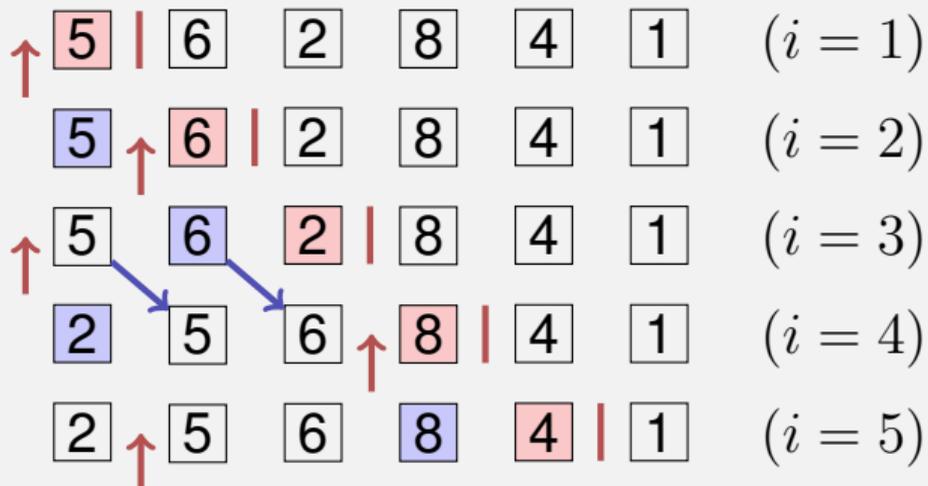
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



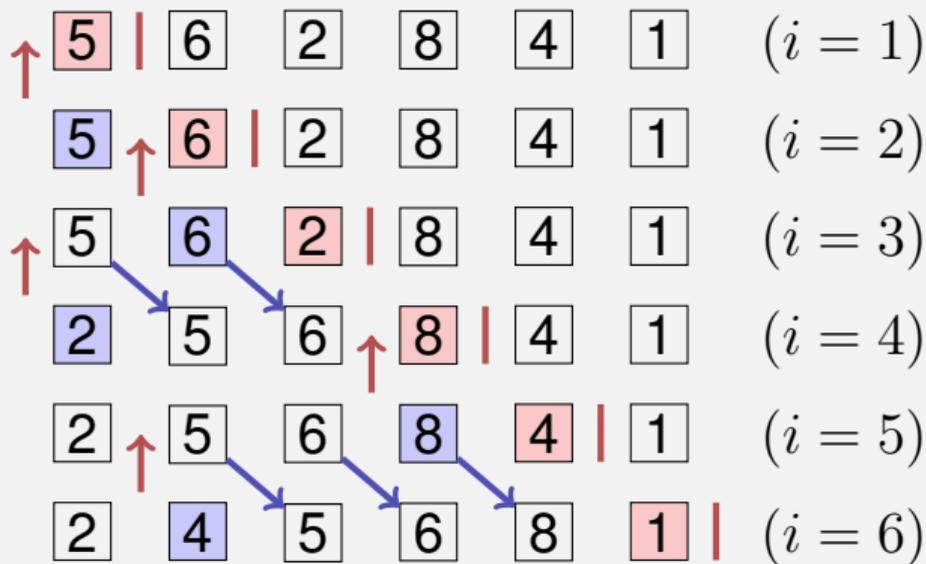
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



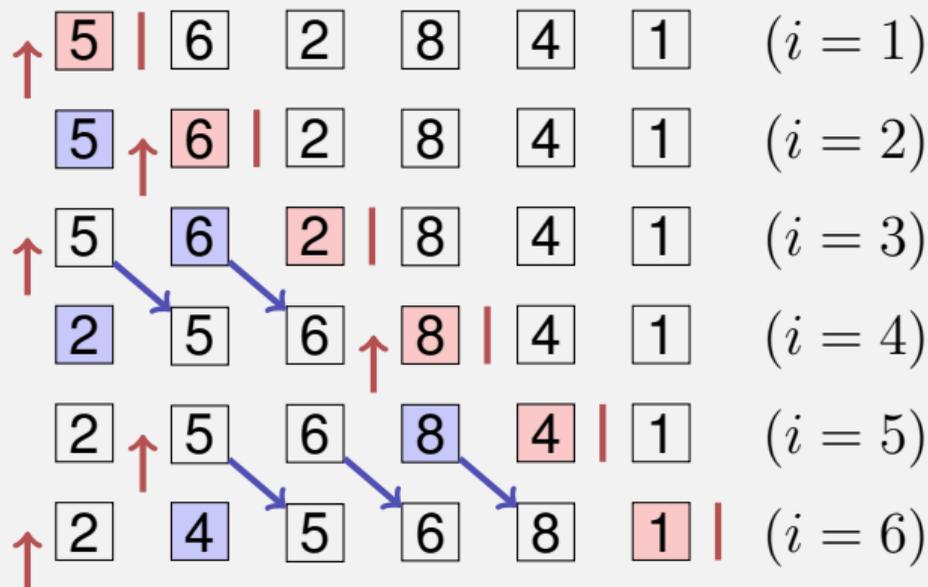
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



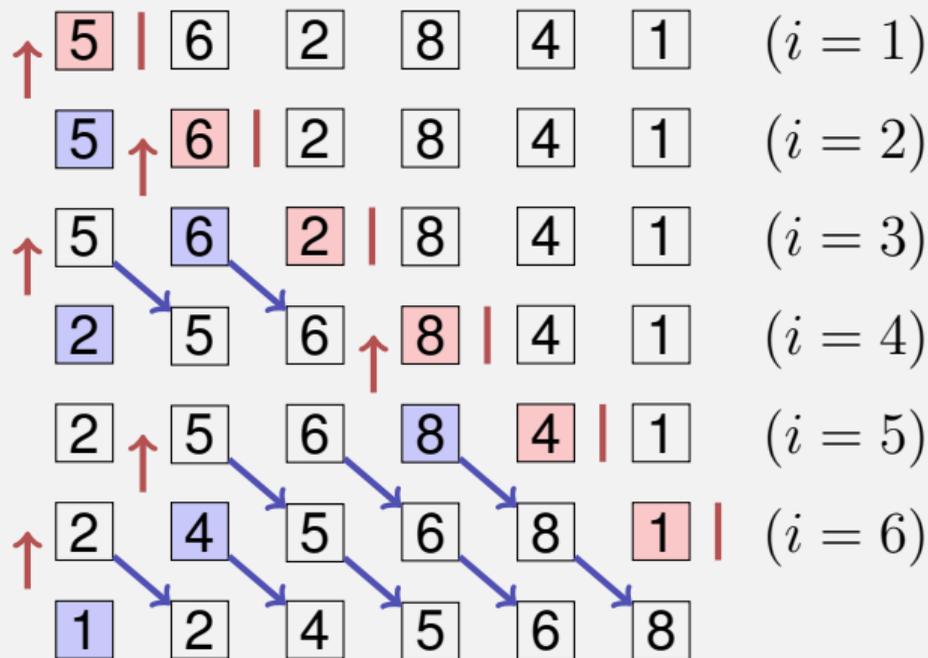
- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen



- Iteratives Vorgehen:
 $i = 1 \dots n$
- Einfügeposition für Element i bestimmen.
- Element i einfügen, ggfs. Verschiebung nötig.

Sortieren durch Einfügen

② Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

Sortieren durch Einfügen

② Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

⚠ Im schlechtesten Fall viele Elementverschiebungen.

② Welchen Vorteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

Sortieren durch Einfügen

❓ Welchen Nachteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

❗ Im schlechtesten Fall viele Elementverschiebungen.

❓ Welchen Vorteil hat der Algorithmus im Vergleich zum Sortieren durch Auswahl?

❗ Der Suchbereich (Einfügebereich) ist bereits sortiert.
Konsequenz: binäre Suche möglich.

Algorithmus: Sortieren durch Einfügen

Input : Array $A = (A[1], \dots, A[n])$, $n \geq 0$.

Output : Sortiertes Array A

for $i \leftarrow 2$ **to** n **do**

$x \leftarrow A[i]$

$p \leftarrow \text{BinarySearch}(A[1..i-1], x)$; // Kleinstes $p \in [1, i]$ mit $A[p] \geq x$

for $j \leftarrow i - 1$ **downto** p **do**

$A[j+1] \leftarrow A[j]$

$A[p] \leftarrow x$

Analyse

Anzahl Vergleiche im schlechtesten Fall:

⁷Mit leichter Anpassung der Funktion BinarySearch für das Minimum / Maximum: $\Theta(n)$

Analyse

Anzahl Vergleiche im schlechtesten Fall:

$$\sum_{k=1}^{n-1} a \cdot \log k = a \log((n-1)!) \in \mathcal{O}(n \log n).$$

Anzahl Vergleiche im besten Fall:

⁷Mit leichter Anpassung der Funktion BinarySearch für das Minimum / Maximum: $\Theta(n)$

Analyse

Anzahl Vergleiche im schlechtesten Fall:

$$\sum_{k=1}^{n-1} a \cdot \log k = a \log((n-1)!) \in \mathcal{O}(n \log n).$$

Anzahl Vergleiche im besten Fall: $\Theta(n \log n)$.⁷

Anzahl Vertauschungen im schlechtesten Fall:

⁷Mit leichter Anpassung der Funktion BinarySearch für das Minimum / Maximum: $\Theta(n)$

Analyse

Anzahl Vergleiche im schlechtesten Fall:

$$\sum_{k=1}^{n-1} a \cdot \log k = a \log((n-1)!) \in \mathcal{O}(n \log n).$$

Anzahl Vergleiche im besten Fall: $\Theta(n \log n)$.⁷

Anzahl Vertauschungen im schlechtesten Fall: $\sum_{k=2}^n (k-1) \in \Theta(n^2)$

⁷Mit leichter Anpassung der Funktion BinarySearch für das Minimum / Maximum: $\Theta(n)$

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

1 2 3 4 5 8 7 9 6

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

1 2 3 4 5 8 7 9 6

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

1 2 3 4 5 8 7 9 6

1 2 3 4 5 6 7 9 8

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

1 2 3 4 5 8 7 9 6

1 2 3 4 5 6 7 9 8

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

1 2 3 4 5 8 7 9 6

1 2 3 4 5 6 7 9 8

1 2 3 4 5 6 7 8 9

Quicksort (willkürlicher Pivot)

2 4 5 6 8 3 7 9 1

2 1 3 6 8 5 7 9 4

1 2 3 4 5 8 7 9 6

1 2 3 4 5 6 7 9 8

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9

Algorithmus Quicksort($A[l, \dots, r]$)

Input : Array A der Länge n . $1 \leq l \leq r \leq n$.

Output : Array A , sortiert zwischen l und r .

if $l < r$ **then**

 Wähle Pivot $p \in A[l, \dots, r]$

$k \leftarrow \text{Partition}(A[l, \dots, r], p)$

 Quicksort($A[l, \dots, k - 1]$)

 Quicksort($A[k + 1, \dots, r]$)

Analyse: Anzahl Vergleiche

Bester Fall.

Analyse: Anzahl Vergleiche

Bester Fall. Pivotelement = Median; Anzahl Vergleiche:

$$T(n) = 2T(n/2) + c \cdot n, T(1) = 0 \quad \Rightarrow \quad T(n) \in \mathcal{O}(n \log n)$$

Schlechtester Fall.

Analyse: Anzahl Vergleiche

Bester Fall. Pivotelement = Median; Anzahl Vergleiche:

$$T(n) = 2T(n/2) + c \cdot n, T(1) = 0 \quad \Rightarrow \quad T(n) \in \mathcal{O}(n \log n)$$

Schlechtester Fall. Pivotelement = Minimum oder Maximum; Anzahl Vergleiche:

$$T(n) = T(n - 1) + c \cdot n, T(1) = 0 \quad \Rightarrow \quad T(n) \in \Theta(n^2)$$

Analyse (Randomisiertes Quicksort)

Theorem

Im Mittel benötigt randomisiertes Quicksort $\mathcal{O}(n \cdot \log n)$ Vergleiche.

Praktische Anmerkungen

Rekursionstiefe im schlechtesten Fall: $n - 1^8$. Dann auch Speicherplatzbedarf $\mathcal{O}(n)$.

Kann vermieden werden: Rekursion nur auf dem kleineren Teil. Dann garantiert $\mathcal{O}(\log n)$ Rekursionstiefe und Speicherplatzbedarf.

⁸Stack-Overflow möglich!

Praktische Anmerkungen

Für den Pivot wird in der Praxis oft der Median von drei Elementen genommen. Beispiel: $\text{Median3}(A[l], A[r], A[\lfloor l + r/2 \rfloor])$.