

5.6 Funktionen und Prozeduren

Oft möchte man:

- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

Beispiel

```
public static int[] readArray(int number) {  
    // number = Anzahl zu lesender Elemente  
    int[] result = new int[number]; // Feld anlegen  
    for (int i = 0; i < number; ++i) {  
        result[i] = read();  
    }  
    return result;  
}
```

Einlesen eines Feldes

5.6 Funktionen und Prozeduren

Oft möchte man:

- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

Beispiel

Type des Rückgabewertes

```
public static int[] readArray(int number) {  
    // number = Anzahl zu lesender Elemente  
    int[] result = new int[number]; // Feld anlegen  
    for (int i = 0; i < number; ++i) {  
        result[i] = read();  
    }  
    return result;  
}
```

Einlesen eines Feldes

5.6 Funktionen und Prozeduren

Oft möchte man:

- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

Beispiel

Funktionsname

Type des Rückgabewertes

```
public static int[] readArray(int number) {  
    // number = Anzahl zu lesender Elemente  
    int[] result = new int[number]; // Feld anlegen  
    for (int i = 0; i < number; ++i) {  
        result[i] = read();  
    }  
    return result;  
}
```

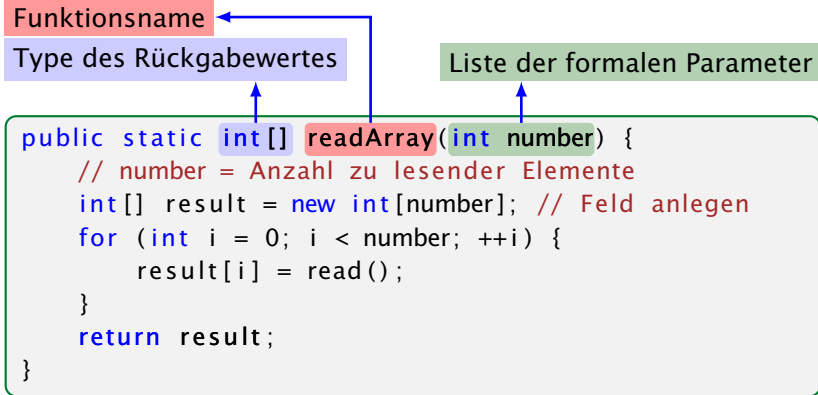
Einlesen eines Feldes

5.6 Funktionen und Prozeduren

Oft möchte man:

- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

Beispiel



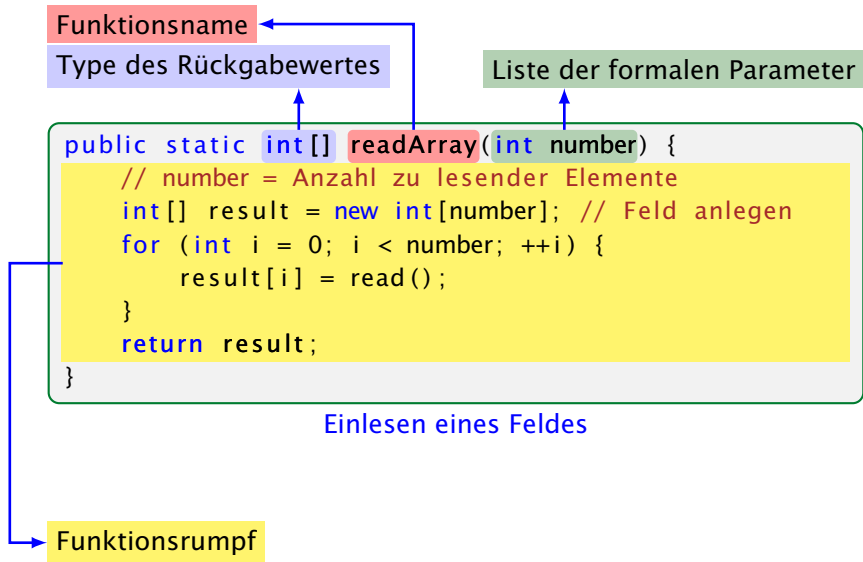
Einlesen eines Feldes

5.6 Funktionen und Prozeduren

Oft möchte man:

- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

Beispiel

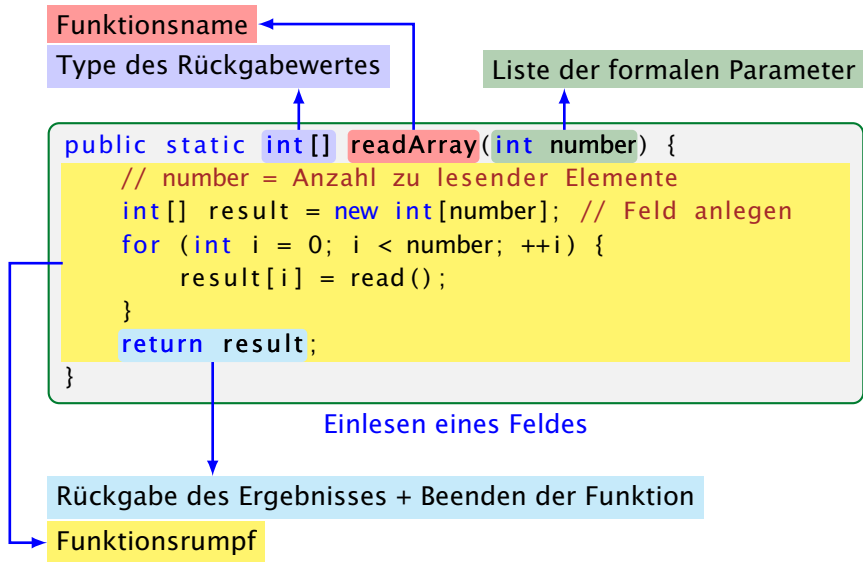


5.6 Funktionen und Prozeduren

Oft möchte man:

- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

Beispiel



5.6 Funktionen und Prozeduren

Oft möchte man:

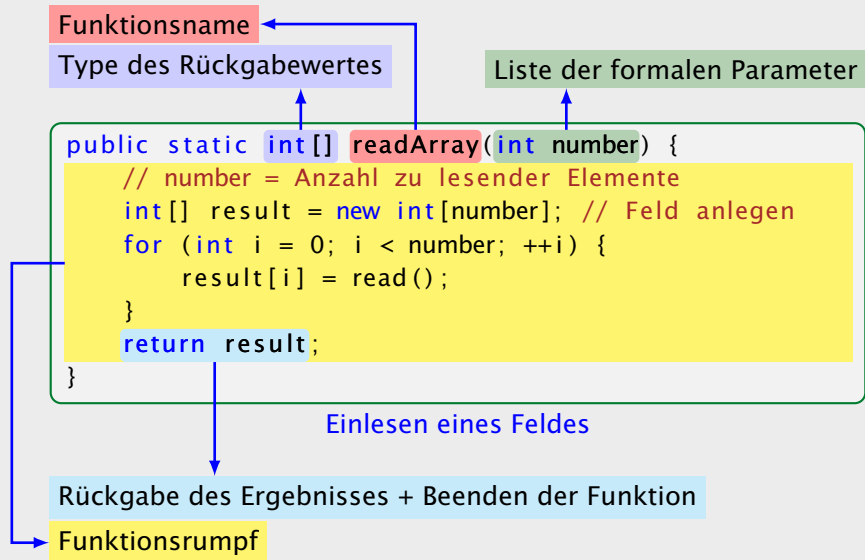
- ▶ Teilprobleme **separat** lösen; und dann
- ▶ die Lösung **mehrfach** verwenden.

5.6 Funktionen und Prozeduren

Erläuterungen:

- ▶ Die erste Zeile ist der **Header** der Funktion.
- ▶ `public`, und `static` kommen später
- ▶ `int[]` gibt den Typ des Rückgabe-Werts an.
- ▶ `readArray` ist der Name, mit dem die Funktion aufgerufen wird.
- ▶ Dann folgt (in runden Klammern und komma-separiert) die Liste der **formalen Parameter**, hier: `(int number)`.
- ▶ Der Rumpf der Funktion steht in geschweiften Klammern.
- ▶ `return expr`; beendet die Ausführung der Funktion und liefert den Wert von `expr` zurück.

Beispiel



5.6 Funktionen und Prozeduren

Erläuterungen:

- ▶ Die Variablen, die innerhalb eines Blocks angelegt werden, d.h. innerhalb von '{' und '}', sind nur innerhalb dieses Blocks **sichtbar**, d.h. benutzbar (**lokale Variablen**).
- ▶ Der Rumpf einer Funktion ist ein Block.
- ▶ Die formalen Parameter können auch als lokale Variablen aufgefasst werden.
- ▶ Bei dem Aufruf `readArray(7)` erhält der formale Parameter `number` den Wert `7` (**aktueller Parameter**).

5.6 Funktionen und Prozeduren

Erläuterungen:

- ▶ Die erste Zeile ist der **Header** der Funktion.
- ▶ `public`, und `static` kommen später
- ▶ `int[]` gibt den Typ des Rückgabe-Werts an.
- ▶ `readArray` ist der Name, mit dem die Funktion aufgerufen wird.
- ▶ Dann folgt (in runden Klammern und komma-separiert) die Liste der **formalen Parameter**, hier: `(int number)`.
- ▶ Der Rumpf der Funktion steht in geschweiften Klammern.
- ▶ `return expr;` beendet die Ausführung der Funktion und liefert den Wert von `expr` zurück.

```
public static int min(int[] b) {  
    int result = b[0];  
    for (int i = 1; i < b.length; ++i) {  
        if (b[i] < result)  
            result = b[i];  
    }  
    return result;  
}
```

Bestimmung des Minimums

Erläuterungen:

- ▶ Die Variablen, die innerhalb eines Blocks angelegt werden, d.h. innerhalb von '{' und '}', sind nur innerhalb dieses Blocks **sichtbar**, d.h. benutzbar (**lokale Variablen**).
- ▶ Der Rumpf einer Funktion ist ein Block.
- ▶ Die formalen Parameter können auch als lokale Variablen aufgefasst werden.
- ▶ Bei dem Aufruf `readArray(7)` erhält der formale Parameter `number` den Wert `7` (**aktueller Parameter**).

Beispiel

```
public class Min extends MiniJava {
    public static int[] readArray(int number) { ... }
    public static int min(int[] b) { ... }
    // Jetzt kommt das Hauptprogramm
    public static void main(String[] args) {
        int n = read();
        int[] a = readArray(n);
        int result = min(a);
        write(result);
    } // end of main()
} // end of class Min
```

Programm zur Minimumsberechnung

Beispiel

```
public static int min(int[] b) {
    int result = b[0];
    for (int i = 1; i < b.length; ++i) {
        if (b[i] < result)
            result = b[i];
    }
    return result;
}
```

Bestimmung des Minimums

Beispiel

Erläuterungen:

- ▶ Manche Funktionen, deren Ergebnistyp `void` ist, geben gar keine Werte zurück – im Beispiel: `write()` und `main()`. Diese Funktionen heißen **Prozeduren**.
- ▶ Das Hauptprogramm hat immer als Parameter ein Feld `args` von `String`-Elementen.
- ▶ In diesem Argument-Feld werden dem Programm Kommandozeilen-Argumente verfügbar gemacht.

```
public class Test extends MiniJava {  
    public static void main (String [] args) {  
        write (args [0]+args [1]);  
    }  
} // end of class Test
```

Beispiel

```
public class Min extends MiniJava {  
    public static int [] readArray (int number) { ... }  
    public static int min (int [] b) { ... }  
    // Jetzt kommt das Hauptprogramm  
    public static void main (String [] args) {  
        int n = read ();  
        int [] a = readArray (n);  
        int result = min (a);  
        write (result);  
    } // end of main ()  
} // end of class Min
```

Programm zur Minimumsberechnung

Beispiel

Der Aufruf

```
java Test "He1" "lo World!"
```

liefert: Hello World!

Beispiel

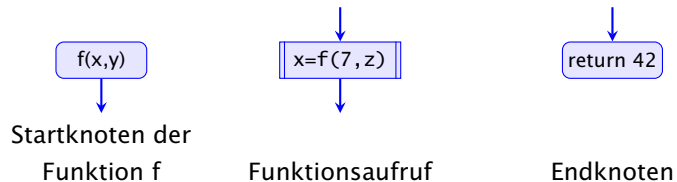
Erläuterungen:

- ▶ Manche Funktionen, deren Ergebnistyp `void` ist, geben gar keine Werte zurück – im Beispiel: `write()` und `main()`. Diese Funktionen heißen **Prozeduren**.
- ▶ Das Hauptprogramm hat immer als Parameter ein Feld `args` von `String`-Elementen.
- ▶ In diesem Argument-Feld werden dem Programm Kommandozeilen-Argumente verfügbar gemacht.

```
public class Test extends MiniJava {  
    public static void main (String [] args) {  
        write (args [0]+args [1]);  
    }  
} // end of class Test
```

5.6 Funktionen und Prozeduren

Um die Arbeitsweise von Funktionen zu veranschaulichen erweitern/modifizieren wir die Kontrollflussdiagramme



- ▶ Für jede Funktion wird ein eigenes Teildiagramm erstellt.
- ▶ Ein Aufrufknoten repräsentiert eine Teilberechnung der aufgerufenen Funktion.

Beispiel

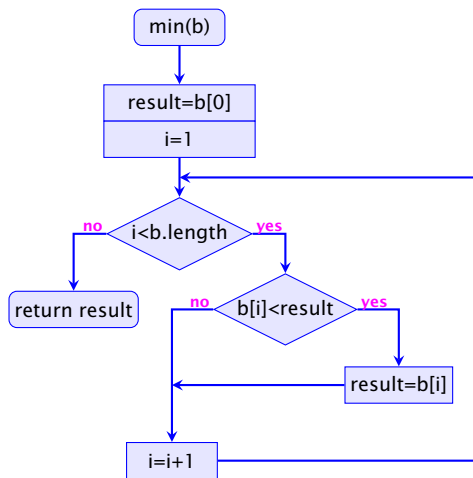
Der Aufruf

```
java Test "He1" "1o Wor1d!"
```

liefert: He11o Wor1d!

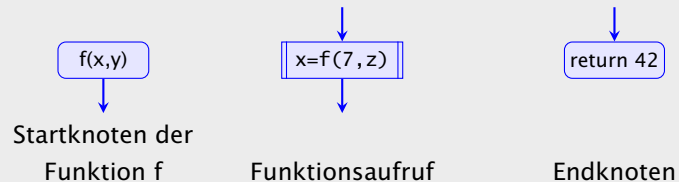
5.6 Funktionen und Prozeduren

Teildiagramm der Funktion `min()`:



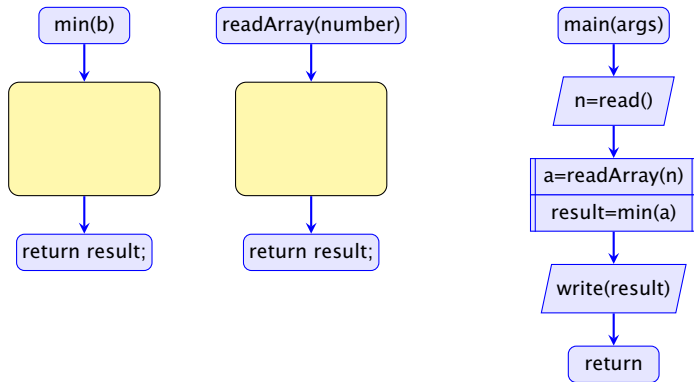
5.6 Funktionen und Prozeduren

Um die Arbeitsweise von Funktionen zu veranschaulichen erweitern/modifizieren wir die Kontrollflussdiagramme



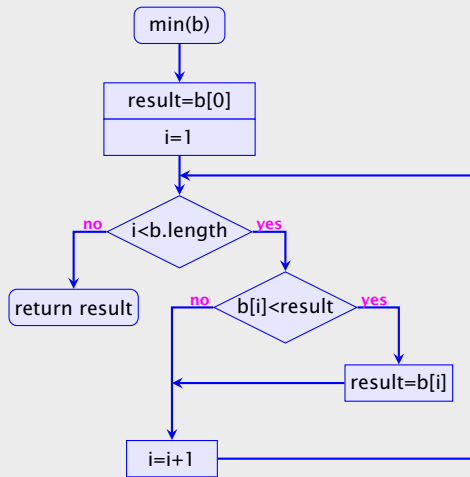
- ▶ Für jede Funktion wird ein eigenes Teildiagramm erstellt.
- ▶ Ein Aufrufknoten repräsentiert eine Teilberechnung der aufgerufenen Funktion.

5.6 Funktionen und Prozeduren

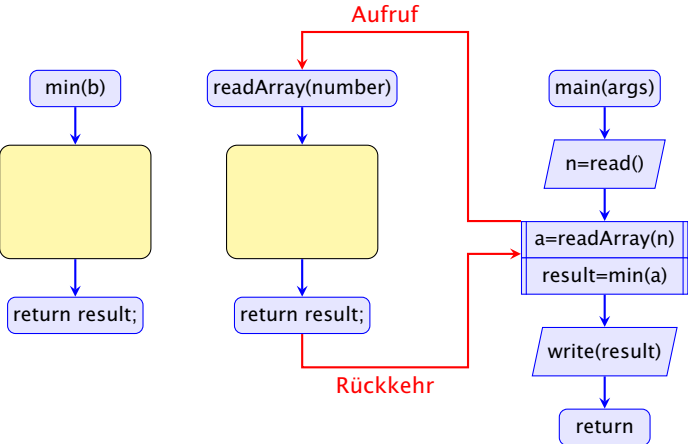


5.6 Funktionen und Prozeduren

Teildiagramm der Funktion `min()`:

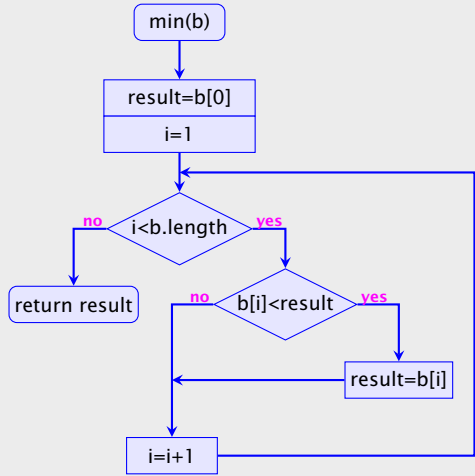


5.6 Funktionen und Prozeduren

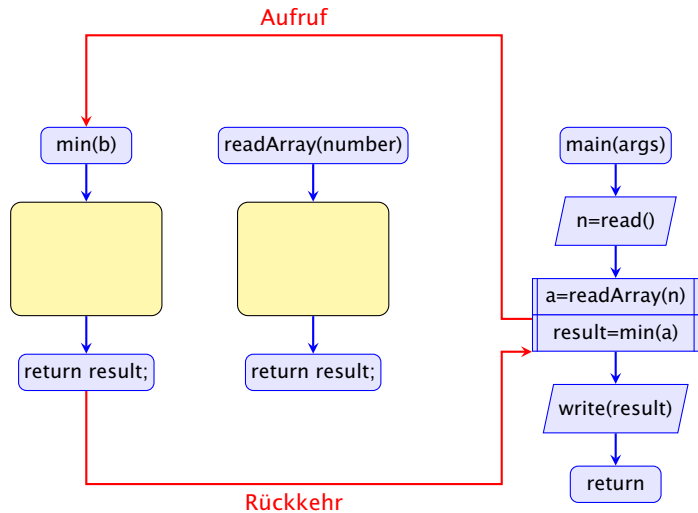


5.6 Funktionen und Prozeduren

Teildiagramm der Funktion min():



5.6 Funktionen und Prozeduren



5.6 Funktionen und Prozeduren

Teildiagramm der Funktion `min()`:

