

Woche 05

Debugging, Arrays und Sortierung

Arrays

- grundlegenden Datenstrukturen
- Elemente nebeneinander stehen
- einfach, effizient und unverzichtbar
- Werte/Objekte speichern, organisieren und bearbeiten - alles an einem Ort.
- Eigenschaften

- Fester Typ

Jedes Array hat einen festgelegten Typ.

- Feste Größe

Die Größe eines Arrays wird beim Erstellen festgelegt.

- Zugriff mit Index

0 indiziert: Index fängt mit 0 an

⇒ Index Out of Bounds

Exception tritt in IDEA

Arrays

- Arrays sind ein Container-Objekt, das eine feste Anzahl von Objekten/Werten eines einzigen Typs speichert
- Arrays sind 0-indiziert

```
1 public static void main(String[] args) {  
2     int size = 5;  
3  
4     int[] array1 = new int[size];  
5     int[] array2 = {1, 2, 3, 4, 5};  
6  
7     array1[1] = 7;  
8  
9     array2[5] = 2 // BUG!  
10 }
```

a. erstellen leeres Array mit „new“

z.B. `int arr1[] = new int[5];`

`int[] arr1 = new int[5];`

b. erstellen Array mit vorgegebenen Werten in „{ }“

z.B. `int[] arr1 = {1, 2, 3, 4, 5}`

`int[] arr1 = new int[] {1, 2, 3, 4, 5};`

Ausgabe:

```
1 ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
```

Arrays

- Die Klasse Arrays enthält viele praktische Hilfsmethoden für Arrays

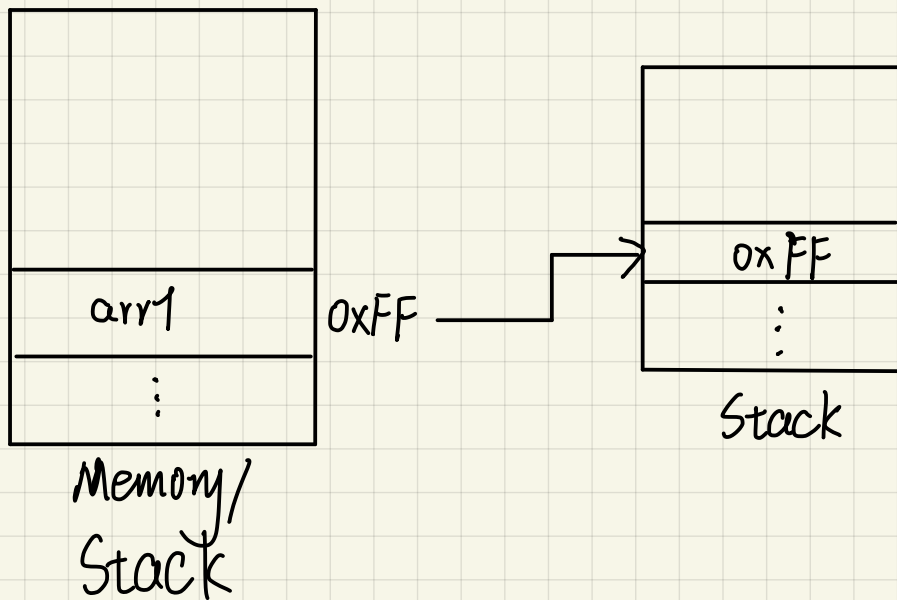
```
1 public static void main(String[] args) {  
2     int size = 5;  
3     int[] array1 = new int[size];  
4  
5     array1[1] = 7;  
6  
7     System.out.println(Arrays.toString(array1));  
8 }
```

Ausgabe:

```
1 [0, 7, 0, 0, 0]
```

Array - Call by Reference

- bei Methodencalls grundsätzlich Call by Value
- bei Objekten : Call by Reference.
- Arrays sind Objekte in Java



Arrays

- Achtung bei Call-By-Reference!

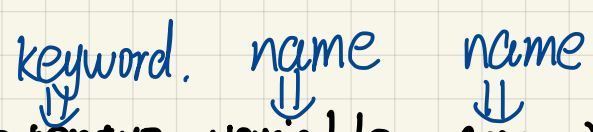
```
1 public static void main(String[] args) {  
2     int[] array1 = {1, 2, 3, 4, 5}  
3  
4     foo(array1);  
5  
6     System.out.println(Arrays.toString(array1));  
7 }  
8  
9 public void foo(int[] arr) {  
10     arr[2] = 1;  
11 }
```

Ausgabe:

```
1 [1, 2, 1, 4, 5]
```

for - Each Loop

- alle Elemente einer Datenstruktur zu iterieren.
 - ohne expliziter Index

- Syntax:

for (Datentyp variable : array) {
 // Codeblock
}

- Vorteile
 - Einfachheit
 - Lesbarkeit
 - Fehlersicherheit

W05P02 - Grundlegende Array-Funktionen

Bearbeite nun die Aufgabe W05P02 - Grundlegende Array-Funktionen

Mehrdimensionale Arrays

- Man kann Arrays ineinander verschachteln für mehrdimensionale Arrays

```
1 public static void main(String[] args) {  
2     int[][][] array = new int[3][2][2];    a. Normale Erstellen  
3  
4     int[][] matrix = {                    b. Direkte Erstellen  
5         {1, 2},  
6         {3, 4},  
7         {5, 6}  
8     };  
9  
10    matrix[0][1] = 7;  
11  
12    System.out.println(Arrays.deepToString(array))  
13    System.out.println(Arrays.deepToString(matrix));  
14 }
```

Ausgabe:

```
1  [[[0, 0], [0, 0]], [[0, 0], [0, 0]], [[0, 0], [0, 0]]]  
2  [[1, 7], [3, 4], [5, 6]]
```

W05P04 - Matrixmultiplikation

Bearbeite nun die Aufgabe W05P04 - Matrixmultiplikation

Matrix A: 2×3

a_{11} a_{12} a_{13}

a_{21} a_{22} a_{23}

Matrix B: 3×2

b_{11} b_{12}

b_{21} b_{22}

b_{31} b_{32}

$$\left\{ \begin{array}{l} a_{11} \cdot b_{11} + \textcircled{1} a_{12} \cdot b_{21} + a_{13} \cdot b_{31} \\ a_{21} \cdot b_{11} + \textcircled{3} a_{22} \cdot b_{21} + a_{23} \cdot b_{31} \end{array} \right.$$

$$\left. \begin{array}{l} a_{11} \cdot b_{12} + \textcircled{2} a_{12} \cdot b_{22} + a_{13} \cdot b_{32} \\ a_{21} \cdot b_{12} + \textcircled{4} a_{22} \cdot b_{22} + a_{23} \cdot b_{32} \end{array} \right\}$$

Bubblesort

1. Wir gehen mehrmals durch das Feld und vergleichen jedes benachbarte Paar von Elementen.
2. Wenn das linke Element größer ist als das rechte, vertauschen wir sie.
3. Dieser Vorgang wiederholt sich so oft, bis das Feld vollständig sortiert ist.

Wir sortieren das Array in mehreren Durchgängen:

- Nach dem ersten Durchgang : größte Element ans Ende
- Nach dem zweiten Durchgang : das zweitgrößte Element an die vorletzte Position usw.
- In jedem Durchgang : die Elemente, die noch nicht an die korrekte Position
- Nach k Durchgängen : die größten k Elemente an die letzten k Positionen

W05P03 - Bubblesort

Bearbeite nun die Aufgabe W05P03 - Bubblesort

[6, 1, 6, 8, 9, 1, 9, 1, 5]

1. 6, 1, 6, 8, 9, 1, 9, 1, 5

⇒ 1, 6, 6, —————

⇒ 1, 6, 6, 8, —————

⇒ 1, 6, 6, 8, 9, - - -

⇒ 1, 6, 6, 8, 9, 1, - - -

⇒ 1, 6, 6, 8, 1, 9, 9, - -

⇒ 1, 6, 6, 8, 1, 9, 9, 1, -

⇒ 1, 6, 6, 8, 1, 9, 1, 9, 5

⇒ 1, 6, 6, 8, 1, 9, 1, 5, 9

2. 1, 6, 6, 8, 1, 9, 1, 5, 9

⇒ 1, 6, 6, —————

⇒ 1, 6, 6, 8, —————

⇒ 1, 6, 6, 8, 1, —————

⇒ 1, 6, 6, 1, 8, 9, —

⇒ 1, 6, 6, 1, 8, 9, 1, —

⇒ 1, 6, 6, 1, 8, 1, 9, 5, -

⇒ 1, 6, 6, 1, 8, 1, 5, 9, 9

Debugging

- Rubber Ducky Debugging
- mit Main-Methode
- IntelliJ - Debugger

Rubber Ducky Debugging

1. Finde eine Gummiente (oder eine andere Person)
2. Erkläre ihr so kleinschrittig wie möglich den Code mit dem du Probleme hast
3. Irgendwann wirst du realisieren, dass der Code den du beim erklären durchgehst nicht das macht was du möchtest

W05P01 - Rubber Penguin Debugging

Bearbeite nun die Aufgabe W05P01 - Rubber Penguin Debugging

Testen mit der `main`-Methode

- Diese spezielle Methode ist der Startpunkt eines jeden Java Programms
- Man kann hier Code schreiben und direkt in IntelliJ mit dem grünen Startknopf ausführen

```
1 class Main {  
2     public static void main(String[] args) {  
3         Object someObject = new Object();  
4  
5         System.out.println(someObject);  
6     }  
7 }
```


Die toString Methode

- Wenn man Objekte auf der Konsole ausgibt, dann ist die Ausgabe meist unverständlich
- Die toString-Methode diktiert, wie ein Objekt als String dargestellt wird und dementsprechend ausgegeben wird
- IntelliJ kann diese Methode automatisch für eine Klasse generieren

```
1  class Penguin {  
2  
3      private String name;  
4  
5      // ... Constructors and other fun stuff  
6  
7      public String toString() {  
8          return "I am the mighty Penguin " + name;  
9      }  
10 }
```

Die toString Methode

```
1 class Penguin {  
2     // ... Constructors, attributes, methods and other fun stuff  
3  
4     public static void main(String[] args) {  
5         Penguin penguin = new Penguin("Caro");  
6         System.out.println(penguin);  
7     }  
8 }
```

Ausgabe:

```
1 I am the mighty Penguin Caro
```

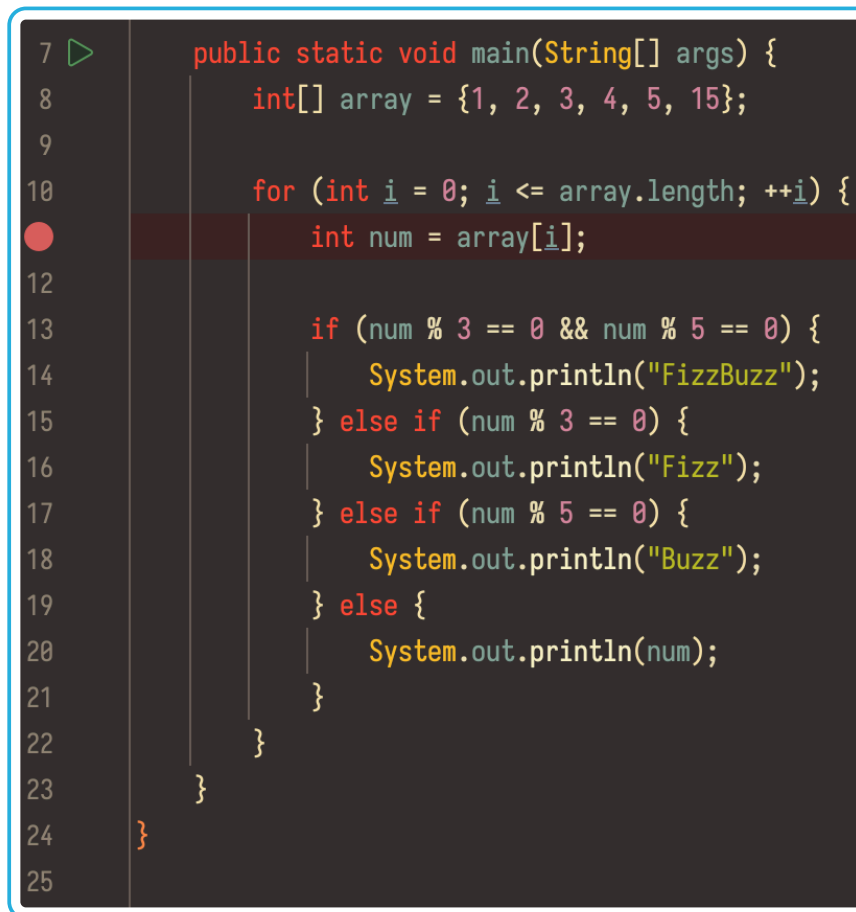
Der IntelliJ-Debugger

- Dieser Code enthält einen Fehler den wir herausfinden wollen:

```
1 public static void main(String[] args) {
2     int[] array = {1, 2, 3, 4, 5, 15};
3
4     for (int i = 0; i <= array.length; ++i) {
5         int num = array[i];
6
7         if (num % 3 == 0 && num % 5 == 0) {
8             System.out.println("FizzBuzz");
9         } else if (num % 3 == 0) {
10             System.out.println("Fizz");
11         } else if (num % 5 == 0) {
12             System.out.println("Buzz");
13         } else {
14             System.out.println(num);
15         }
16     }
17 }
```

Der IntelliJ-Debugger

- Mit dem IntelliJ Debugger kannst du mit einem Klick auf die Zeilenzahl einen Breakpoint setzen an dem das Programm anhält, falls es das erreicht

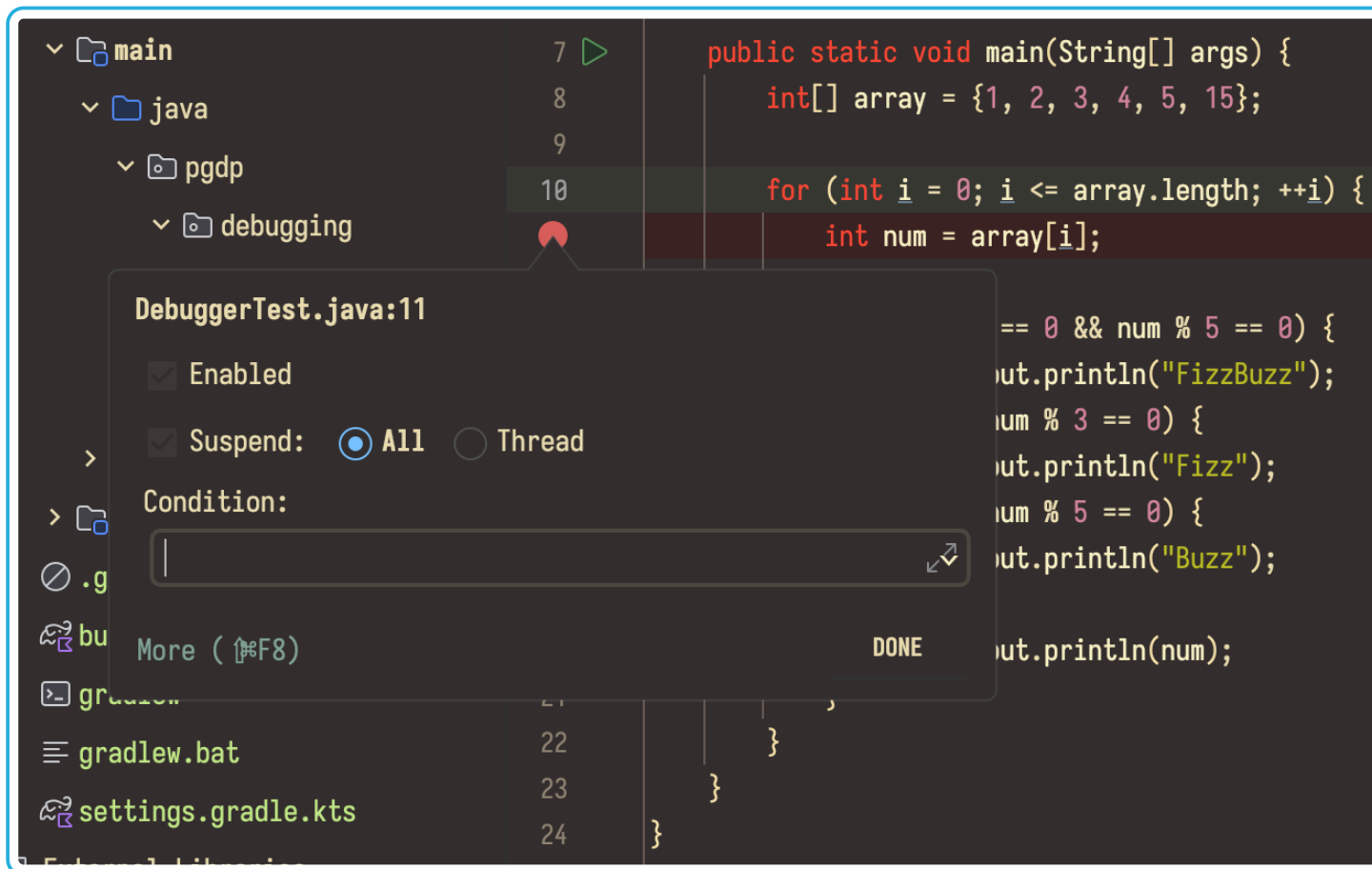


```
7  ▶ public static void main(String[] args) {  
8      int[] array = {1, 2, 3, 4, 5, 15};  
9  
10     for (int i = 0; i <= array.length; ++i) {  
11         int num = array[i];  
12  
13         if (num % 3 == 0 && num % 5 == 0) {  
14             System.out.println("FizzBuzz");  
15         } else if (num % 3 == 0) {  
16             System.out.println("Fizz");  
17         } else if (num % 5 == 0) {  
18             System.out.println("Buzz");  
19         } else {  
20             System.out.println(num);  
21         }  
22     }  
23 }  
24 }  
25
```

The screenshot shows the IntelliJ IDE with a Java file open. A breakpoint, represented by a red circle, is set on line 11 at the start of the line `int num = array[i];`. The code is a `main` method that iterates over an array `{1, 2, 3, 4, 5, 15}` and prints "FizzBuzz", "Fizz", "Buzz", or the number itself based on divisibility rules. The line numbers 7 through 25 are visible on the left margin.

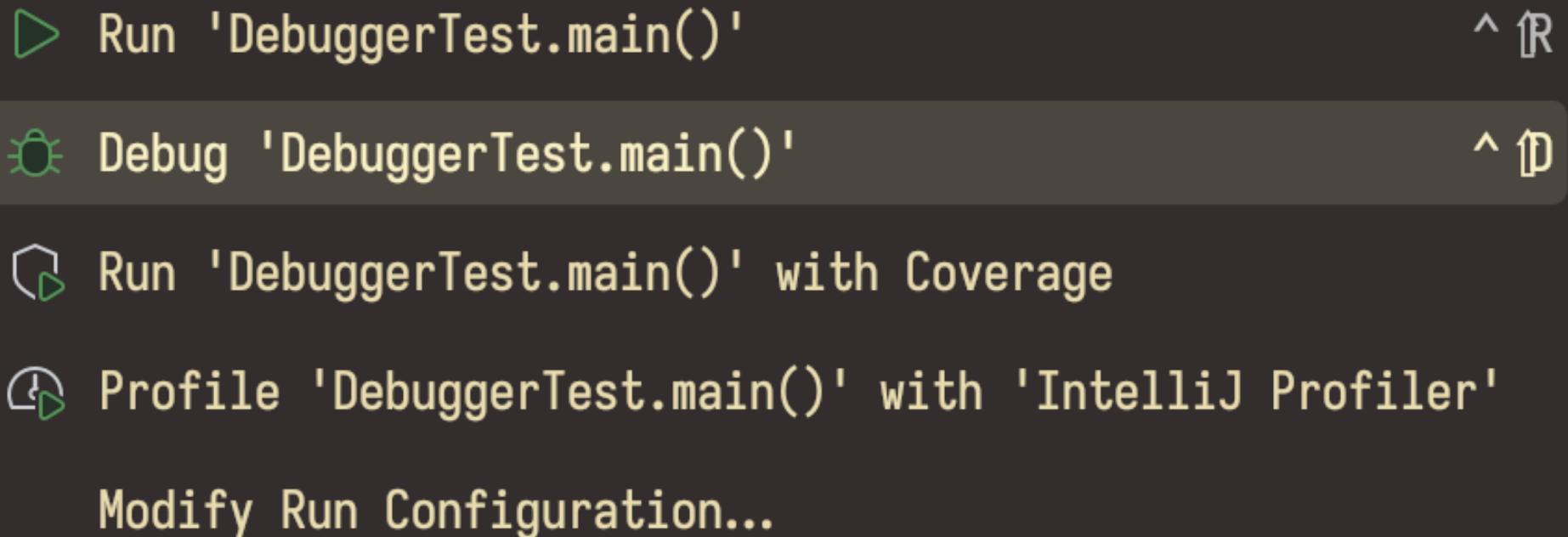
Der IntelliJ-Debugger

- Mit einem Rechtsklick auf den roten Kreis, kannst du die Breakpoints auch abhängig von einer Bedingung machen



Der IntelliJ-Debugger

- Jetzt musst du beim ausführen nurnoch anstatt “Run” die “Debug” Option auswählen

A screenshot of the IntelliJ IDE's Run/Debug menu. The menu is dark-themed with a list of options. The 'Debug' option is highlighted with a light gray background. Each option has a green icon on the left and keyboard shortcuts on the right.

▶ Run 'DebuggerTest.main()' ^ ⌘

🐛 Debug 'DebuggerTest.main()' ^ ⌘

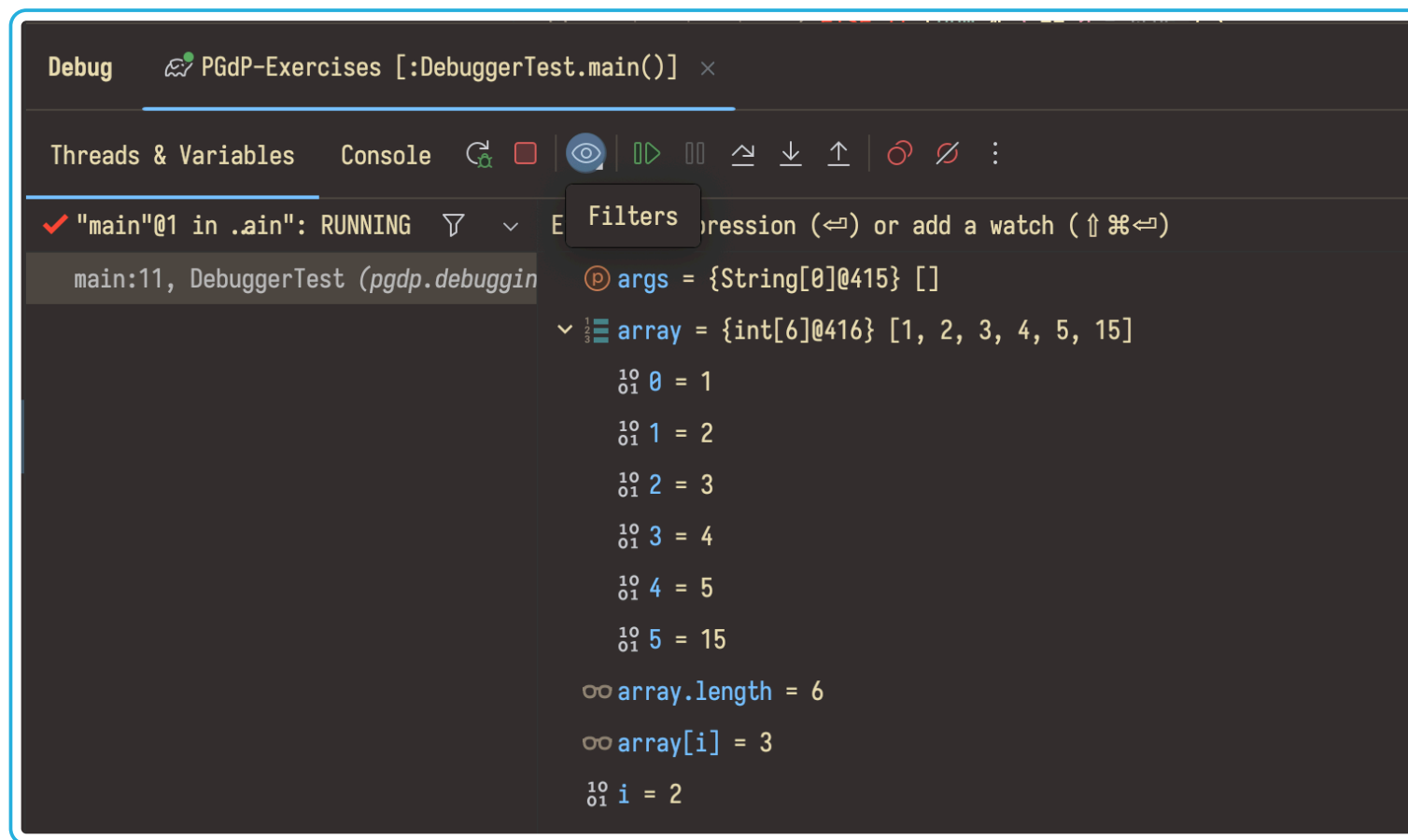
📊 Run 'DebuggerTest.main()' with Coverage

📈 Profile 'DebuggerTest.main()' with 'IntelliJ Profiler'

Modify Run Configuration...

Der IntelliJ-Debugger

- Sobald das Programm den Breakpoint erreicht, stoppt es und lässt dich das Programm inspizieren
- Mittels der verschiedenen Kontrollen kannst du durch das Programm hüpfen



Der IntelliJ-Debugger

- Probiere nun den Debugger an einer der Aufgaben selber aus!
- [Hier klicken für mehr Infos zu dem Debugger](#)

Extra: Variadic arguments

- Manchmal wollen wir eine flexible, beliebige Anzahl von Argumenten übergeben
- Die variadic arguments verhalten sich wie ein Array Objekt

```
1 public class VarArgs {  
2     public static void foo(int... args) {  
3         for (int arg : args) {  
4             System.out.print(arg + " ");  
5         }  
6         System.out.println();  
7     }  
8 }
```

Extra: Variadic arguments

```
1 public class VarArgs {  
2  
3     public static void main(String... args) {  
4  
5         // Alles valide Aufrufe:  
6         foo();  
7         foo(1, 2);  
8  
9         int[] arr = new int[10];  
10        foo(arr)  
11    }  
12 }
```

Ausgabe:

```
1  
2 1 2  
3 0 0 0 0 0 0 0 0 0 0
```

Slide und Code von Aufgaben

GitHub : Code aller P-Aufgaben

- <https://github.com/SikiaLi/PGDP.git>