



Fehlersuche (Debugging)

Programmierfehler finden
und beseitigen

Softwareentwicklung

2BI 2018/19

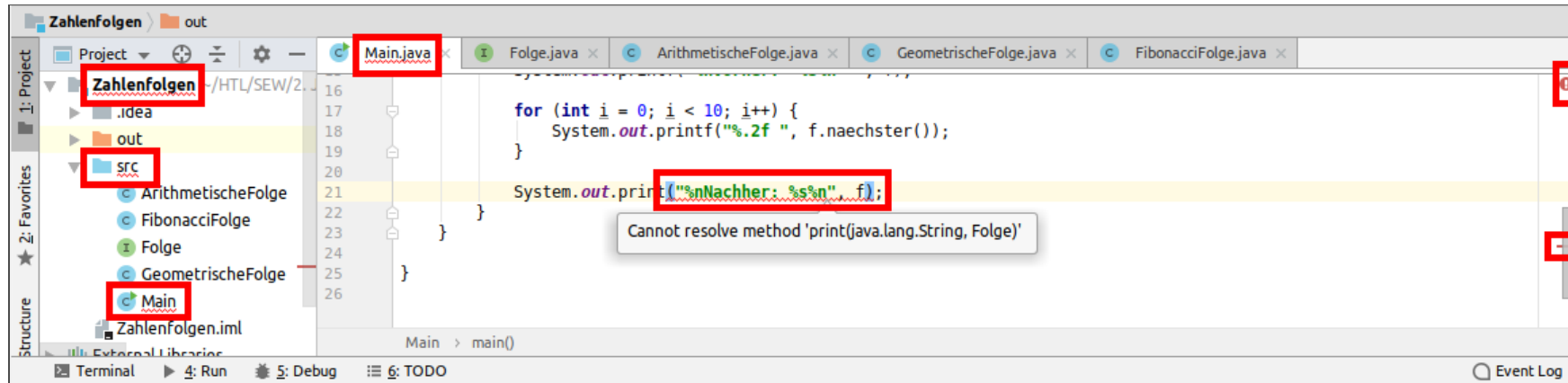
Fehlersuche

▸ Arten von Fehlern

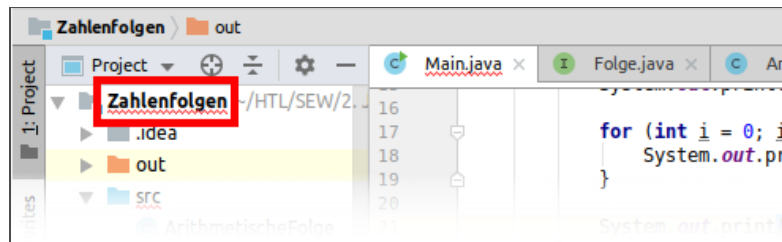
Fehlerart	In Java erkennbar	durch
Syntaxfehler: syntaktisch falsche Anweisungen, z.B. <code>String s = 42;</code>	beim Übersetzen	Syntaxfehler
Semantische Fehler: unzulässige (aber syntaktisch richtige) Anweisungen, z.B. <code>int i;</code> <code>System.out.println(i); // i hat keinen Wert</code>	beim Übersetzen oder zur Laufzeit	Syntaxfehler Exception
Logische Fehler: Anforderungen falsch umgesetzt, z.B. <code>for (int i = 0; i <= arr.length; i++) { ... }</code>	zur Laufzeit	Exception oder falsches Verhalten

► Syntaxfehler und semantische Fehler

- Werden auf Anweisungs-, Methoden-, Klassen- und Verzeichnisebene markiert

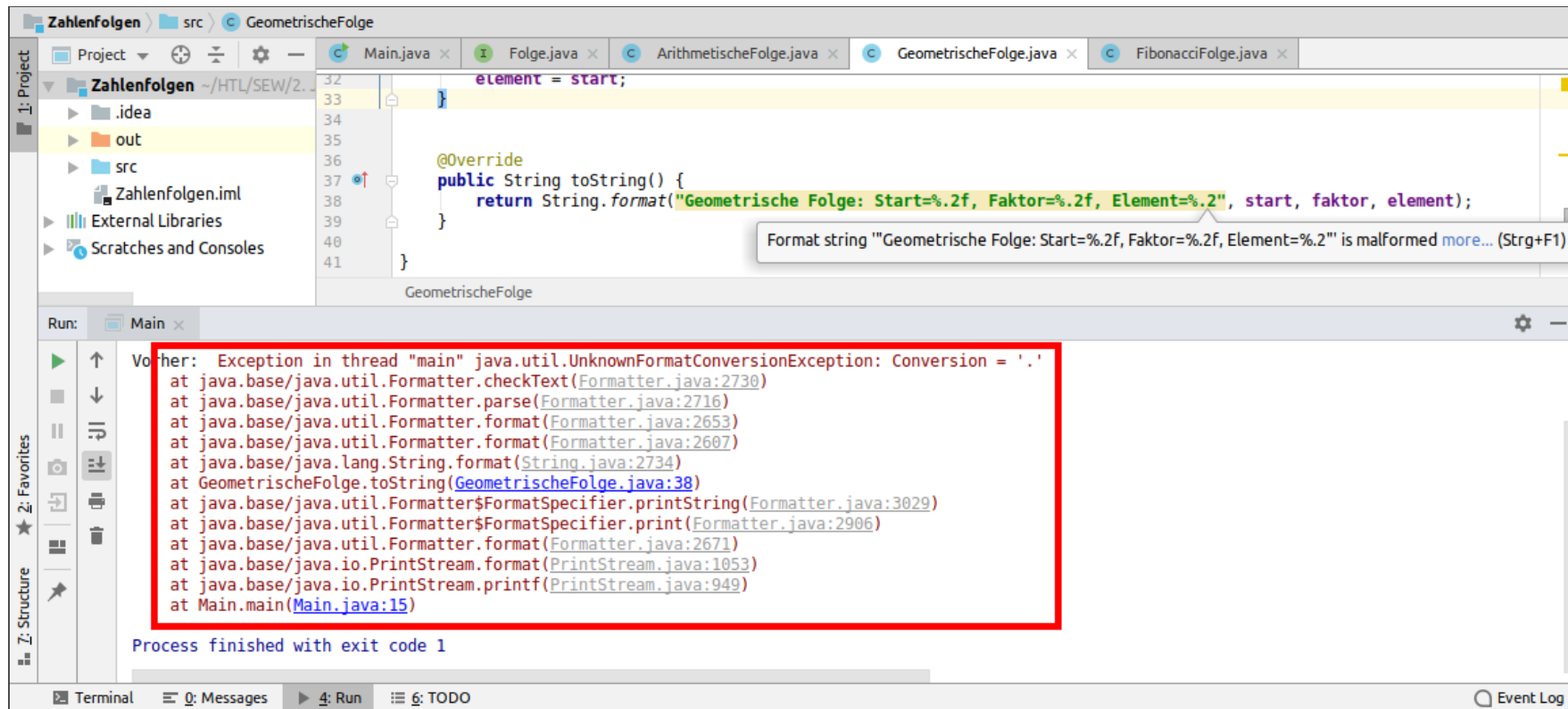


- Optionen zur Behebung → Cursor auf Wellenlinie und **ALT+Enter**
- Vor dem ersten Programmstart
 - *Alle* Syntaxfehler beheben → Kontrolle in Projektansicht



► Nicht behandelte Exceptions


- Programm (bzw. Thread) wird beendet, Exception in Konsole (Run-Ansicht) sichtbar
 - Art der Exception, z.B. **UnknownFormatConversionException**
 - Methodenaufrufe, die zur Exception geführt haben → Aufrufstapel (call stack)
 - Auch Methodenaufrufe innerhalb des Java-API werden gezeigt

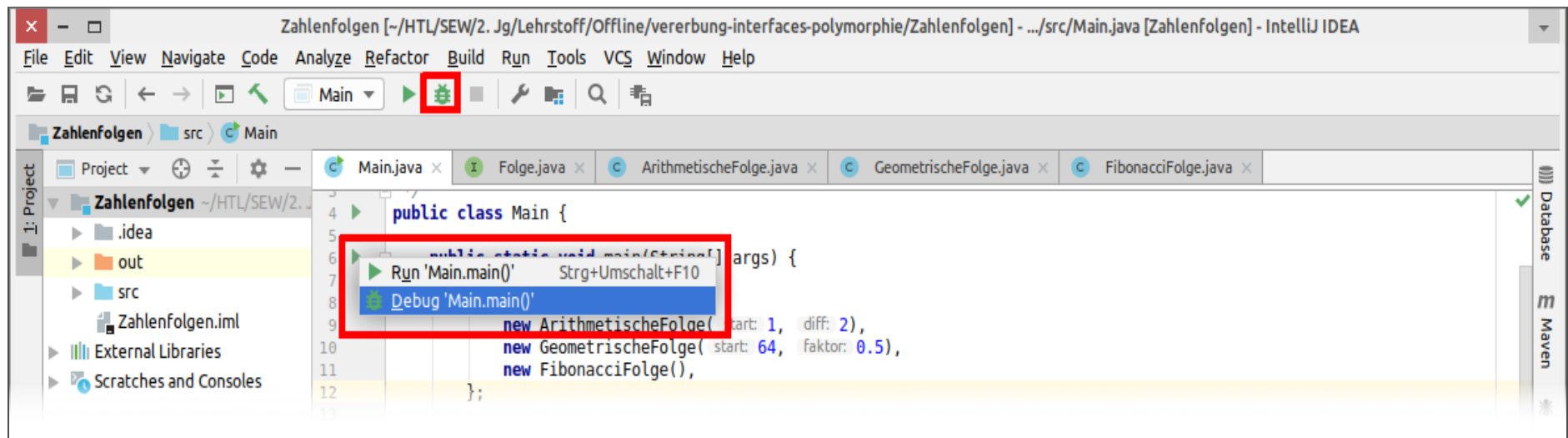


► Debugging

- [Warum man Programmfehler „Bugs“ nennt](#)
- De-Bugging → Programmfehler suchen und entfernen
 1. Welches falsche Verhalten/Ergebnis verursacht der Bug?
 2. Wie kann man das wiederholbar hervorrufen?
 3. Welche Programmstellen könnten dafür verantwortlich sein?
 4. Mit dem [Debugger](#) dort [Haltepunkte setzen](#)
 5. Programm im [Debug-Modus ausführen](#)
 6. An den Haltepunkten auf erwartetes [Verhalten überprüfen](#)
 7. [Sprich zur Quietsch-Ente!](#) (Wird in den Übungen erklärt)
 8. Keine Erleuchtung → weiter bei 3.
 9. Fehler im Quellcode beheben
 10. Testen!!! Denn: [99 little bugs in the code ...](#)

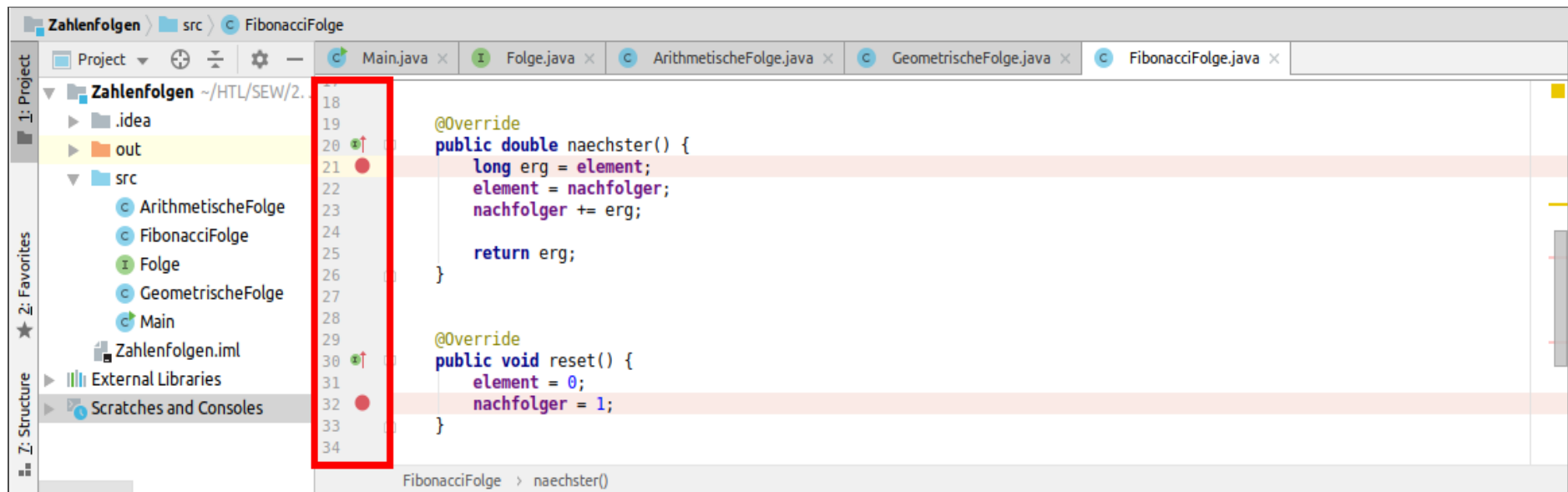
► Debugger

- Softwarewerkzeug, um zur Laufzeit [logische Programmfehler](#) zu finden
 - An beliebigen Programmstellen die [Variablenwerte ansehen/ändern](#)
 - Das Programm [schrittweise ausführen](#) und die Auswirkungen beobachten
- Nur wirksam im Debug-Modus → Programm mit -Symbol starten



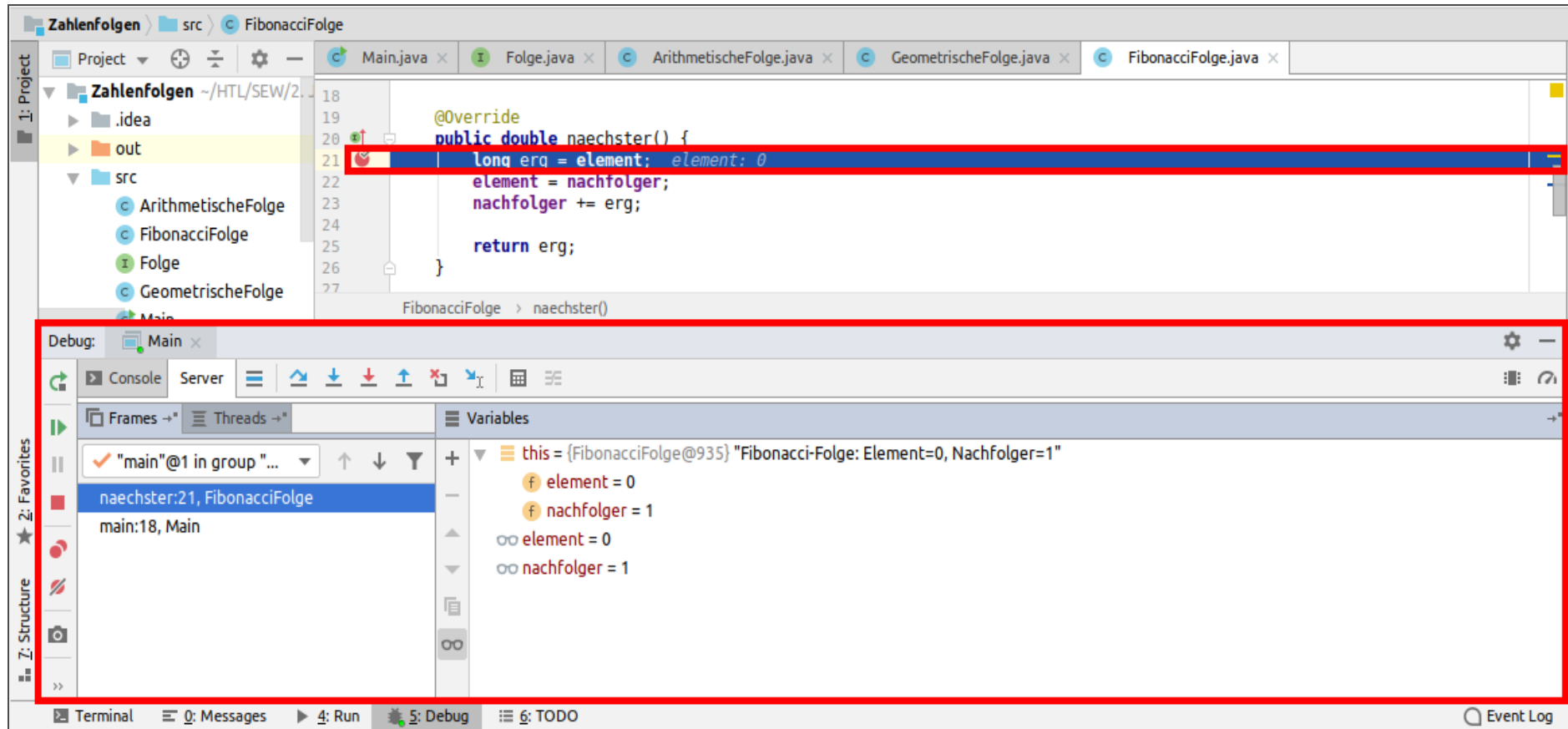
► Haltepunkte (Breakpoints)

- Stellen im Quelltext, an denen das Programm *im Debug-Modus* angehalten wird
- Beliebige viele Breakpoints in beliebigen Klassen
- Nur auf *ausführbaren* Anweisungen platzierbar
- Breakpoint setzen/entfernen → Klick in die Seitenleiste

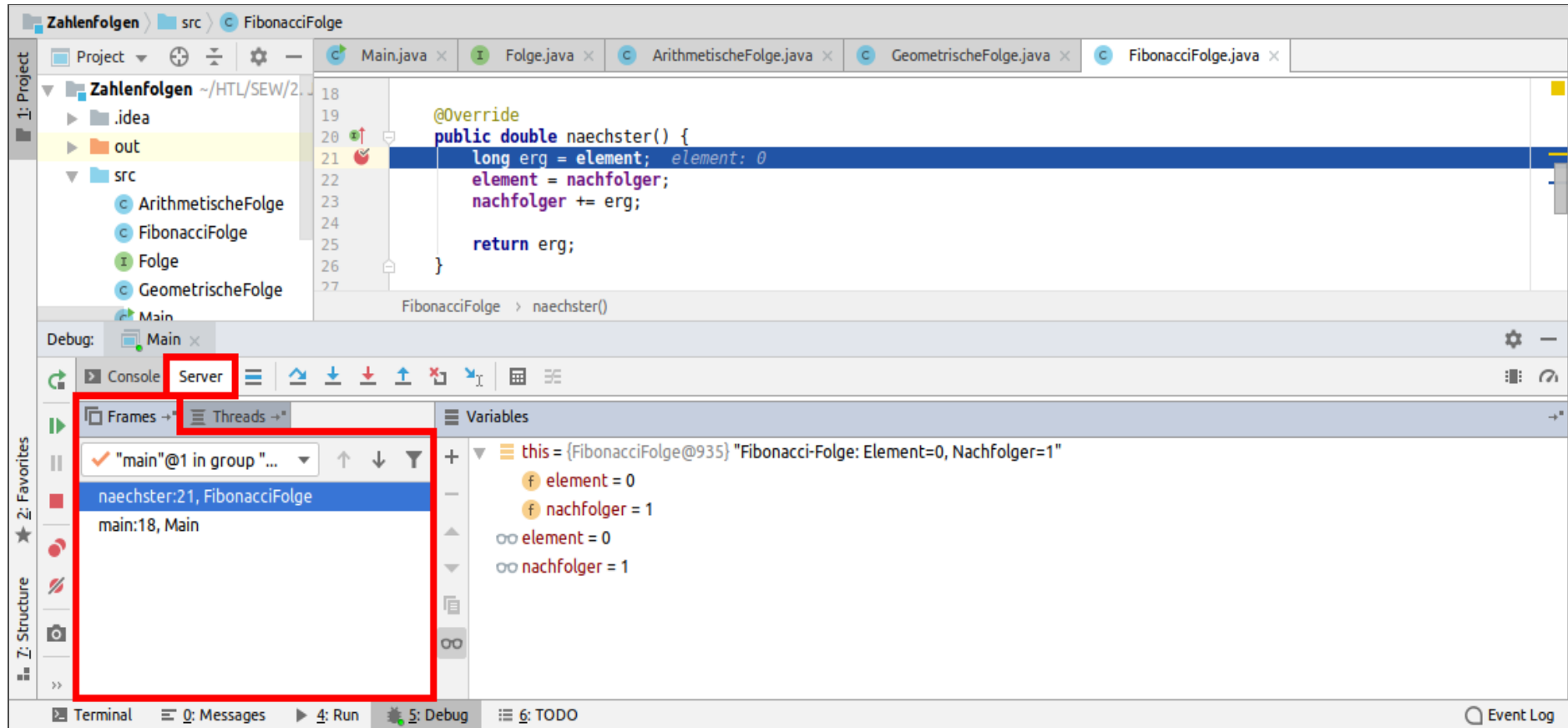


► Breakpoint erreicht

- Programm wird angehalten *vor* Ausführung der Anweisung
- **Debug**-Ansicht erscheint

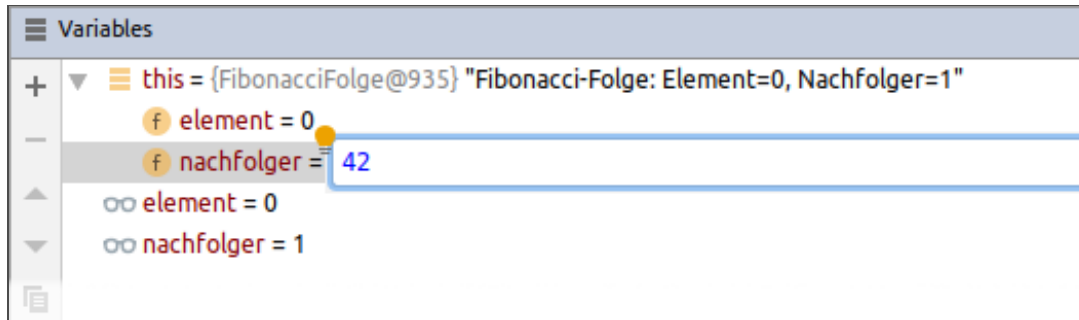


- ▶ Wie gelangte das Skript zum Breakpoint?
- Aus Aufrufstapel (call stack) ersichtlich (analog zu [Exception](#))

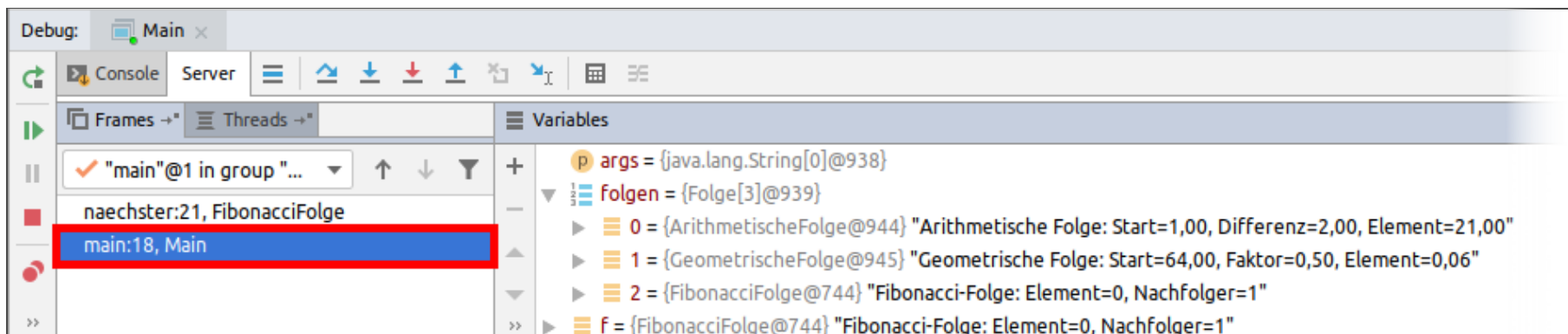


▸ Variablenwerte ansehen und ändern

- In einem Breakpoint behalten alle Variablen ihre Werte

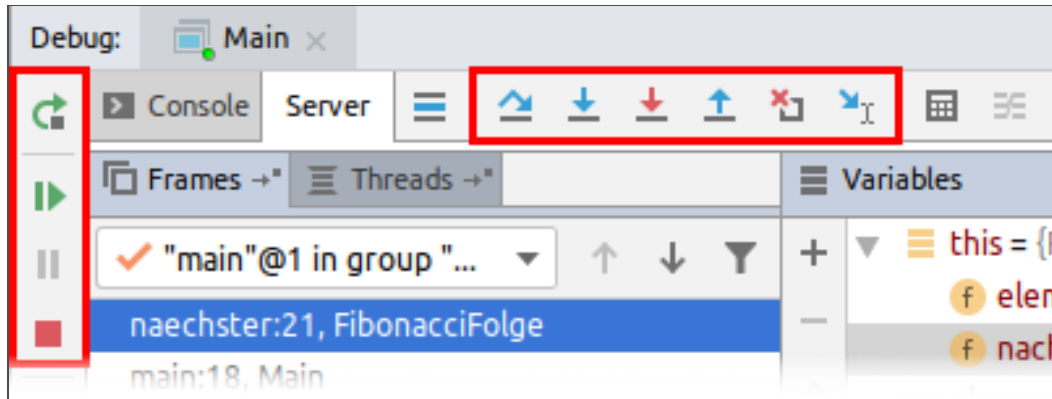


- **this** → Objekt, in dem sich die Methode befindet
 - Variablenwert ändern → **Set Value...** im Kontextmenü
- Um Variablen einer anderen Methode anzusehen → Methode im call stack auswählen



▸ Programm fortsetzen

- Weiterer Programmablauf → diese Buttons:



- Anweisungen einzeln ausführen

 Einzelschritt über Methodenaufrufe hinweg,  Einzelschritt in Methode

- Mehrere Anweisungen ausführen

 bis zum nächsten **return**,  bis zum Cursor im Quelltext

- Debugging-Sitzung steuern

 Programm fortsetzen bis Breakpoint oder Ende,  Programm abbrechen,

 Programm neu starten

Inhaltsverzeichnis

Fehlersuche (Debugging)	1
Programmierfehler finden und beseitigen	1
Softwareentwicklung 2BI 2018/19	1
Fehlersuche	2
Arten von Fehlern	2
Syntaxfehler und semantische Fehler	3
Nicht behandelte Exceptions	4
Debugging	5
Debugger	6
Haltepunkte (Breakpoints)	7
Breakpoint erreicht	8
Wie gelangte das Skript zum Breakpoint?	9
Variablenwerte ansehen und ändern	10
Programm fortsetzen	11
Inhaltsverzeichnis	12