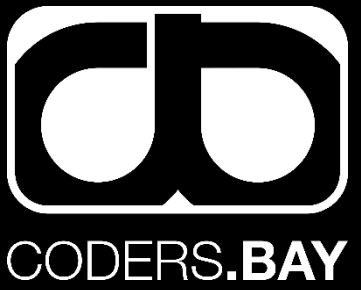
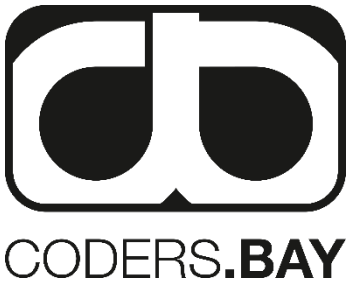


UNITTESTING

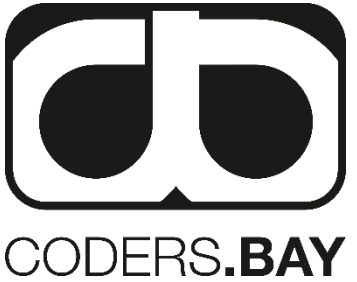
CODERS.BAY / 02.09.2019

INHALT

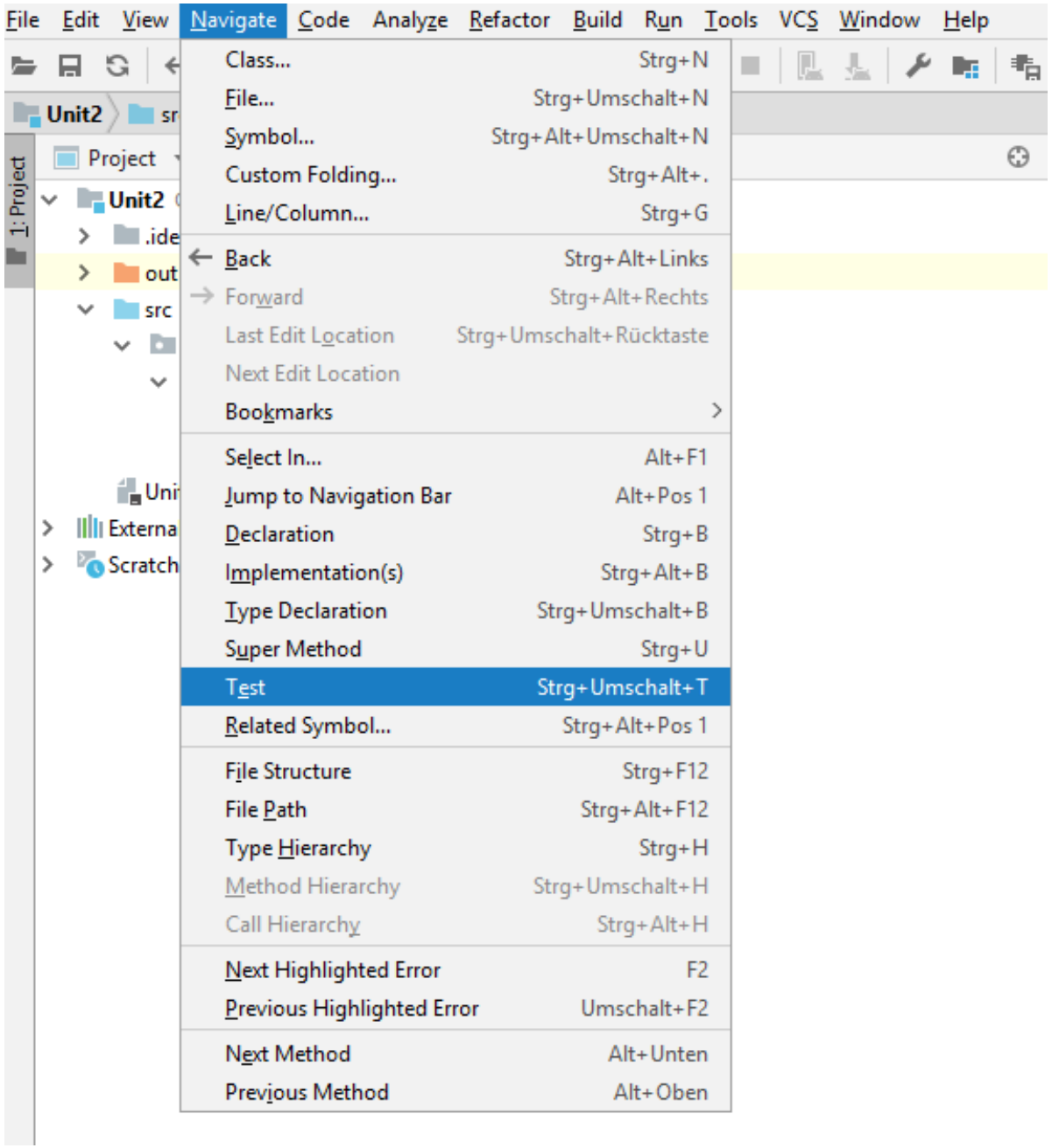


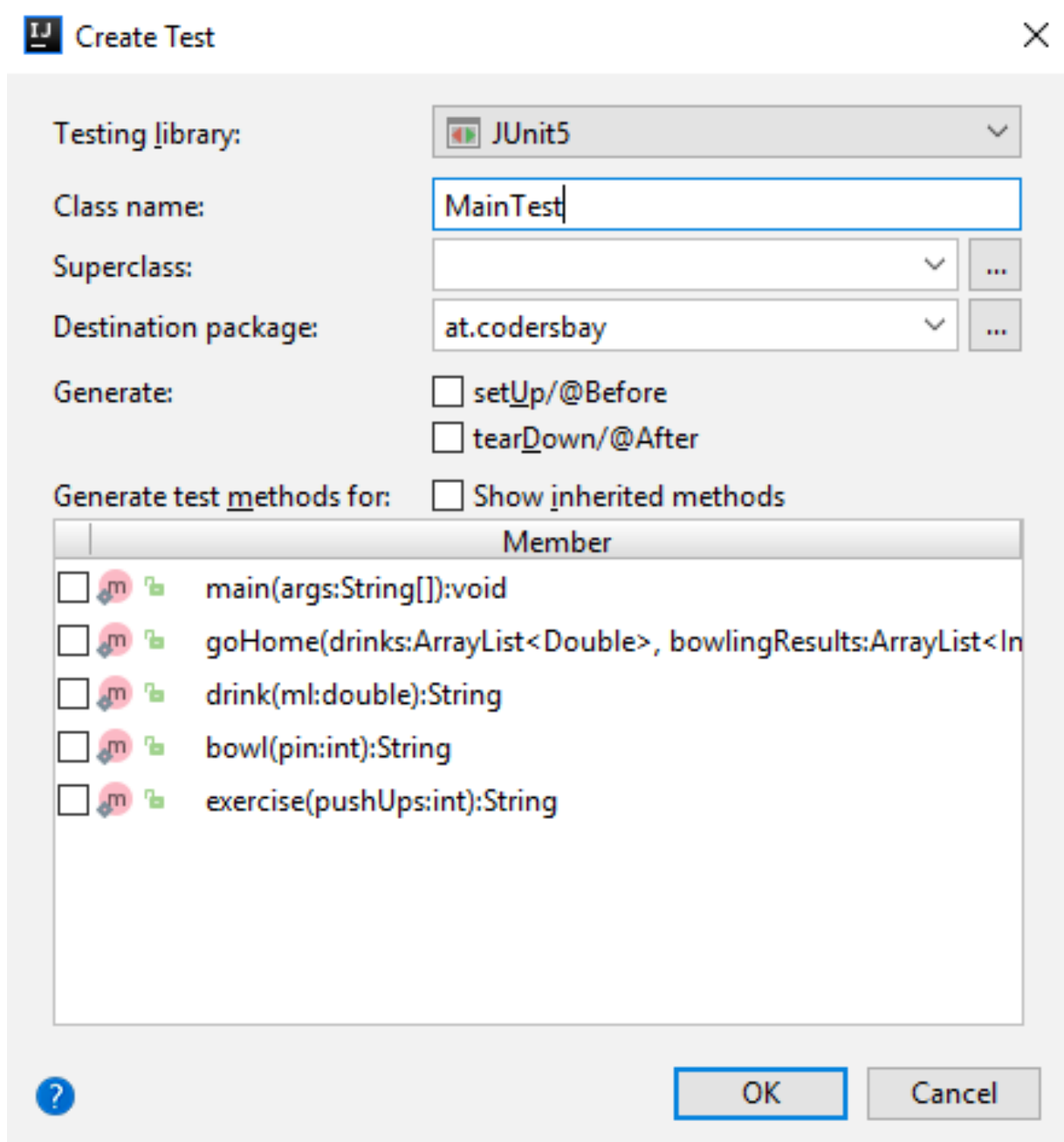
- Überblick
- Annotationen
- Good vs Bad
- Corner Cases
- Testconsole
- Test Driven Development

ÜBERBLICK

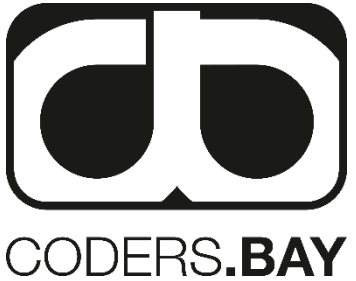


- Warum?
  - Strukturiertes überprüfen von bestimmten Outputs mit bestimmten Inputs
- Was?
  - JUnit = Java Version des SUnit-Frameworks von Kent Beck zur Unterstützung automatisierter Tests
- Wie?
  - In der Klasse über Navigate → Test oder Ctrl+Shift+T (Shift+⌘+T bei Mac)

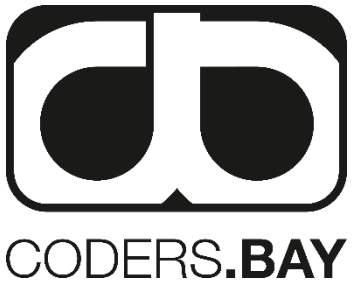

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Class... Strg+N
File... Strg+Umschalt+N
Symbol... Strg+Alt+Umschalt+N
Custom Folding... Strg+Alt+.
Line/Column... Strg+G
Back Strg+Alt+Links
Forward Strg+Alt+Rechts
Last Edit Location Strg+Umschalt+Rücktaste
Next Edit Location
Bookmarks
Select In... Alt+F1
Jump to Navigation Bar Alt+Pos 1
Declaration Strg+B
Implementation(s) Strg+Alt+B
Type Declaration Strg+Umschalt+B
Super Method Strg+U
Test Strg+Umschalt+T
Related Symbol... Strg+Alt+Pos 1
File Structure Strg+F12
File Path Strg+Alt+F12
Type Hierarchy Strg+H
Method Hierarchy Strg+Umschalt+H
Call Hierarchy Strg+Alt+H
Next Highlighted Error F2
Previous Highlighted Error Umschalt+F2
Next Method Alt+Unten
Previous Method Alt+Oben
Unit2
Project
1: Project
Externa
Scratch


Create Test
Testing library: JUnit5
Class name: MainTest
Superclass:
Destination package: at.codersbay
Generate: setUp/@Before
tearDown/@After
Generate test methods for: Show inherited methods
Member
main(args:String[]):void
goHome(drinks:ArrayList<Double>, bowlingResults:ArrayList<In
drink(ml:double):String
bowl(pin:int):String
exercise(pushUps:int):String
OK
Cancel

ANNOTATIONEN

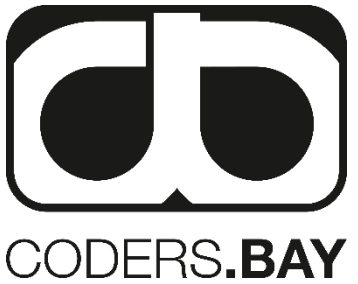


- @Test
  - Markiert einen Testcase
- @BeforeEach / @AfterEach
  - Wird vor, bzw. nach jedem einzelnen Test ausgeführt
- @BeforeAll / @AfterAll
  - Wird vor, bzw. nach allen Tests einmal ausgeführt
  - Müssen *static* sein
- @Displayname
  - Ermöglicht es den Anzeigenamen des Tests für die Console zu ändern




```java
@Test
@DisplayName("Test drink with 250ml")
void drinkFullGlass() {
}
```

GOOD VS BAD

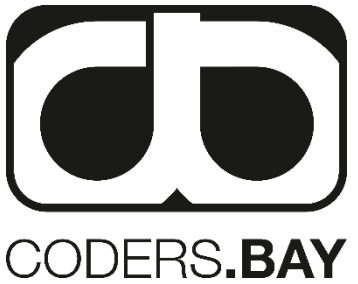


- Testcases sollen Ergebnisse überprüfen

```
@Test
@DisplayName("Test drink with 250ml")
void drinkFullGlass() {
    String result = Main.drink(250);
    System.out.println(result);
}
```

  - Hier wird das Ergebnis nur ausgegeben und nicht überprüft der Entwickler muss eingreifen

GOOD VS BAD



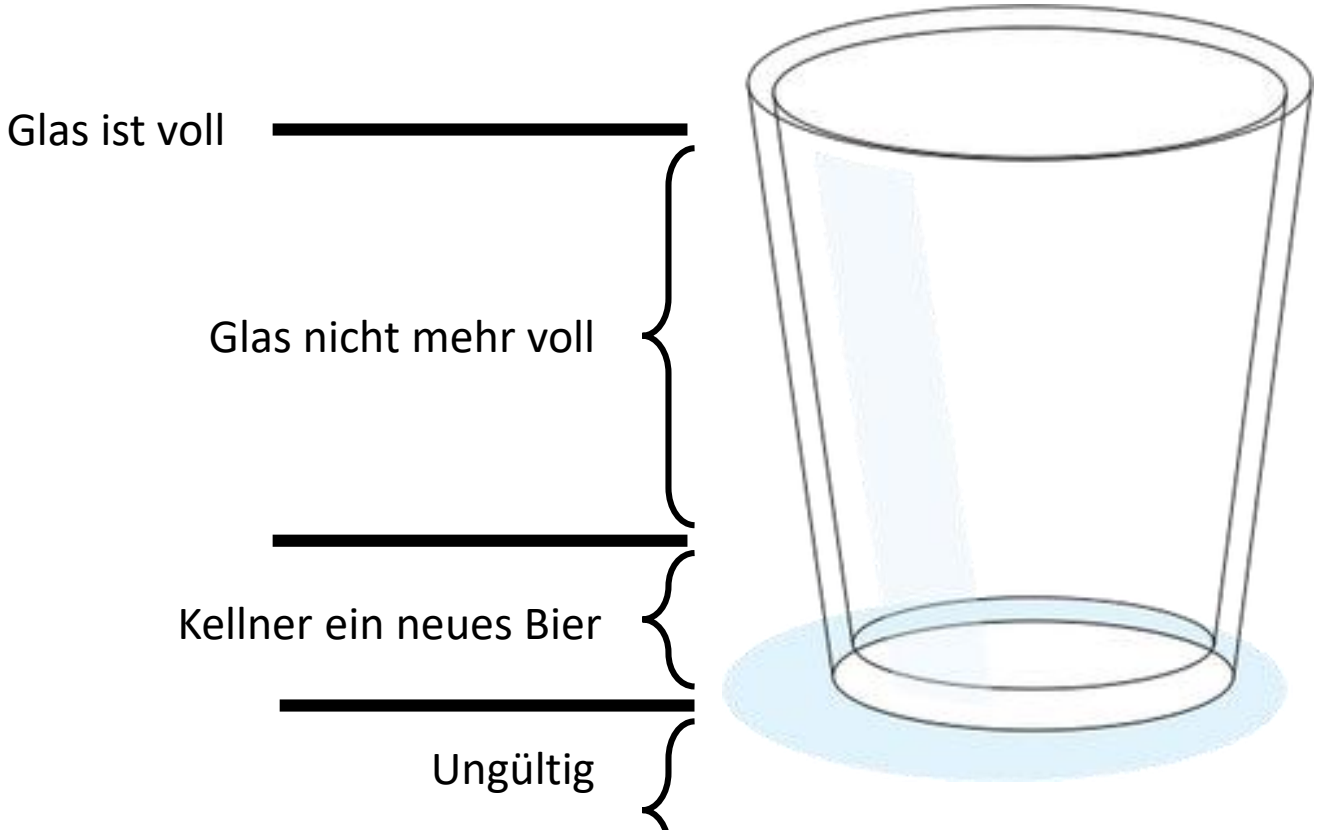

```
@Test
@DisplayName("Test drink with 250ml")
void drinkFullGlass() {
    String result = Main.drink(250);
    assertEquals("Das Glas ist voll", result);
}
```

- Durch die Funktion assertEquals wird die Variable result mit dem String „Das Glas ist voll“ auf Gleichheit überprüft
- Wenn result == „Das Glas ist voll → Testcase bestanden
- Wenn result != „Das Glas ist voll → Testcase fehlgeschlagen

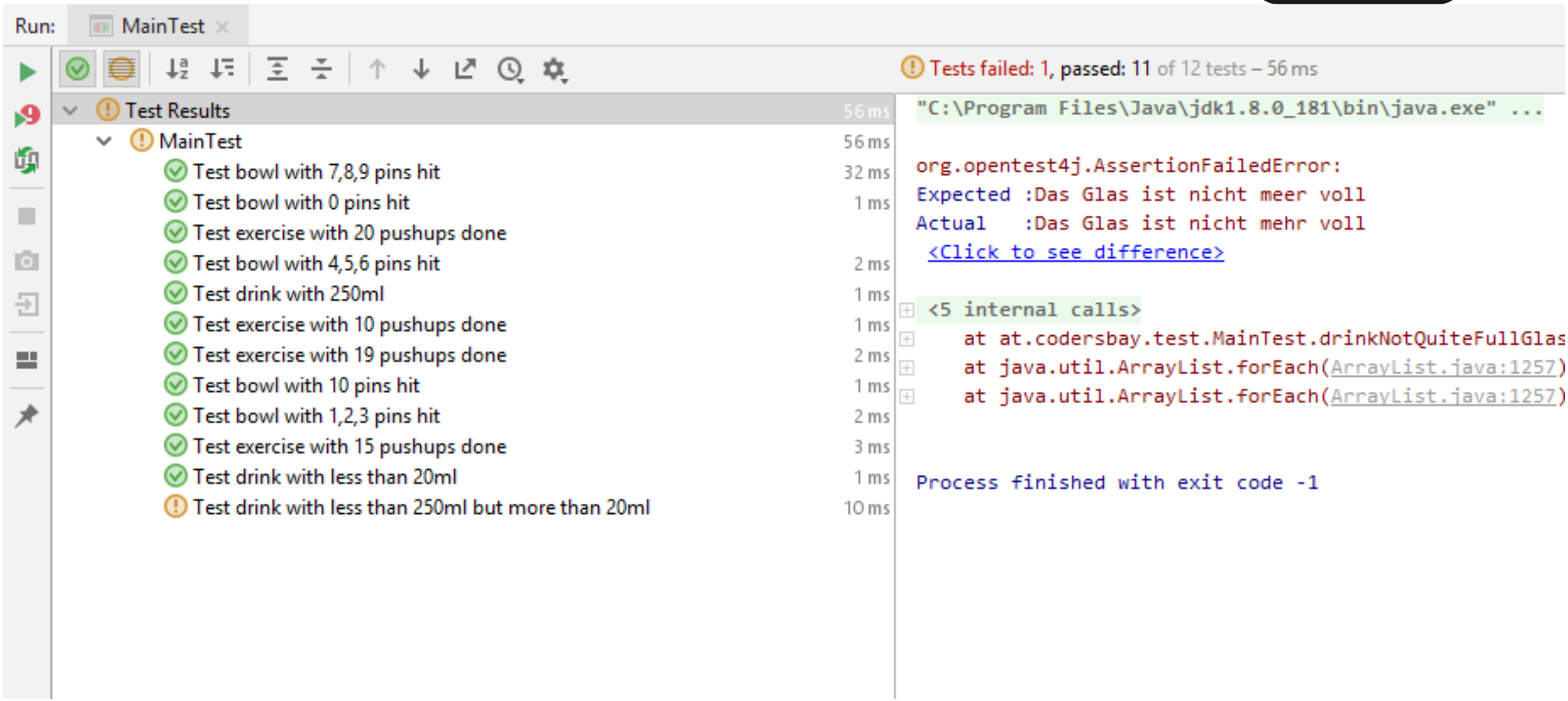
CORNER CASES

CODERS.BAY

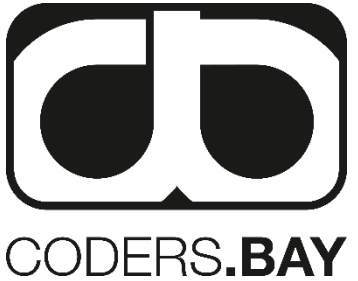
- Corner Cases = Grenzwerte, mögliche Problemfälle
  - Bsp.: Grenzwerte, Negativwerte, Nullwerte



TESTCONSOLE

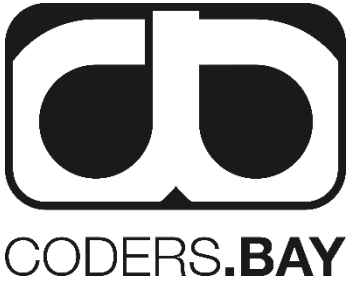


TEST DRIVEN DEVELOPMENT



- Zuerst Testfälle entwerfen
- Dann Logik programmieren
- Verhindert Betriebsblindheit (nicht testen von möglichen Fehlern)
- Erhöht Testbarkeit des Systems

TDD NACH KENT BECK



- Red
  - Schreibe Test der neue Funktion testen soll
- Green
  - Programmiere Funktion mit möglichst geringem Aufwand
- Refactoring
  - Räume im Code auf (Codeverdoppelungen entfernen Codequality verbessern)
  - Keine neuen Funktionen ergänzen

VIELEN DANK FÜR EURE AUFMERKSAMKEIT !