PGdP

JUnit Test

Server

Nebenläufigkeit I

P-Aufgaben

Folien: go.tum.de/904005

Tobias Schamel

JUnit Test

Um einzelne Elemente (Units) eines Programms zu testen, verwendet man das Framework JUnit.

Annotation

```
@Test
public void testSum() {
   int expected = 7;
   int summand1 = 2;
   int summand2 = 5;
   int actual = sum(summand1, summand2);
   assertEquals(expected, actual, "Summe falsch");
}
```

Assertion

JUnit Test

JUnit Annotations

Annotation	Funktion
`@Test`	Identifiziert eine Test Methode
`@Timeout(<time>)` `@Test`	Testmethode, die nach *time* ms fehlschlägt
`@Disabled(<string>)` `@Test`	Ignoriere Test & gebe *string* aus
`@BeforeEach`	Vor jedem Test ausführen
`@AfterEach`	Nach jedem Test ausführen
`@BeforeAll`	Einmal zu Beginn ausführen
`@AfterAll`	Einmal am Ende ausführen

JUnit Test

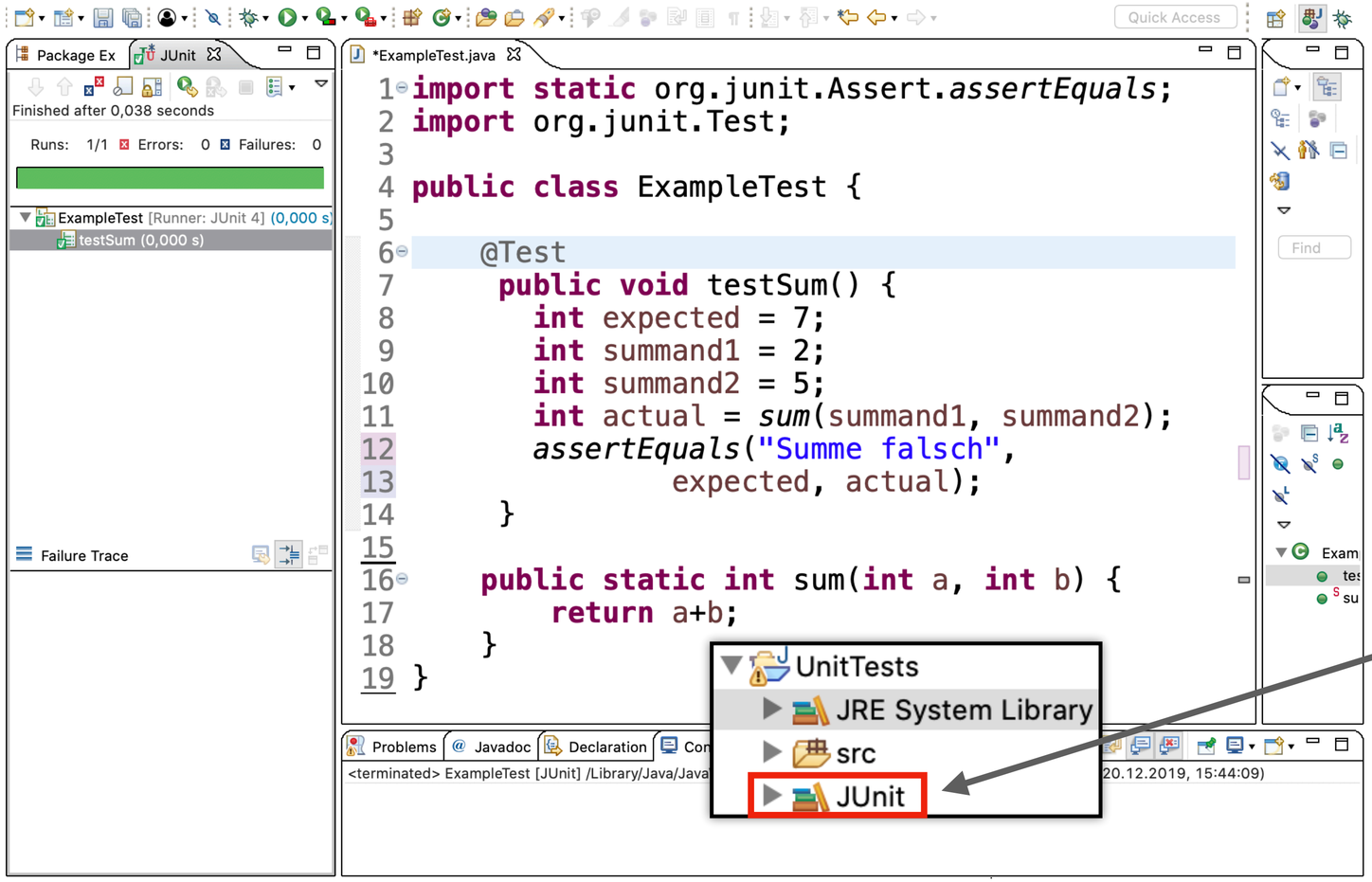
JUnit Assertions

Assertion	Funktion
`assertTrue(pred)`	Prüft, ob *Prädikat* zu true auswertet
`assertFalse(pred)`	Prüft, ob *Prädikat* zu false auswertet
`assertEquals(expected, actual)`	Prüft, ob *Actual* und *Expected* das gleiche sind
`assertSame(expected, actual)`	Prüft, ob *Actual* und *Expected* das selbe sind
`fail(message)`	Lässt den Test mit Fehlermeldung *Message* fehlschlagen
`assertThrows(<exc>.class, function)`	Erwartet beim Ausführen von *function* eine Exception vom Typ *exc.*

Man kann den Assertions immer einen String am Ende übergeben, um die Fehlermeldung zu definieren.

JUnit Test

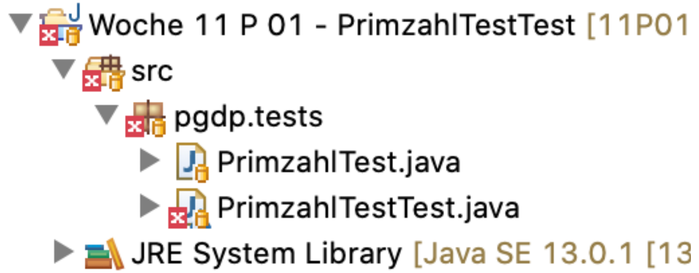
Beispielausführung



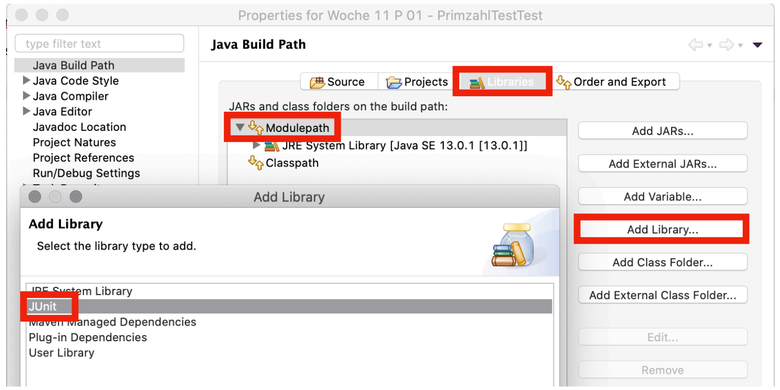
JUnit Framework muss Teil des Projekts sein

JUnit Test

Beispielausführung

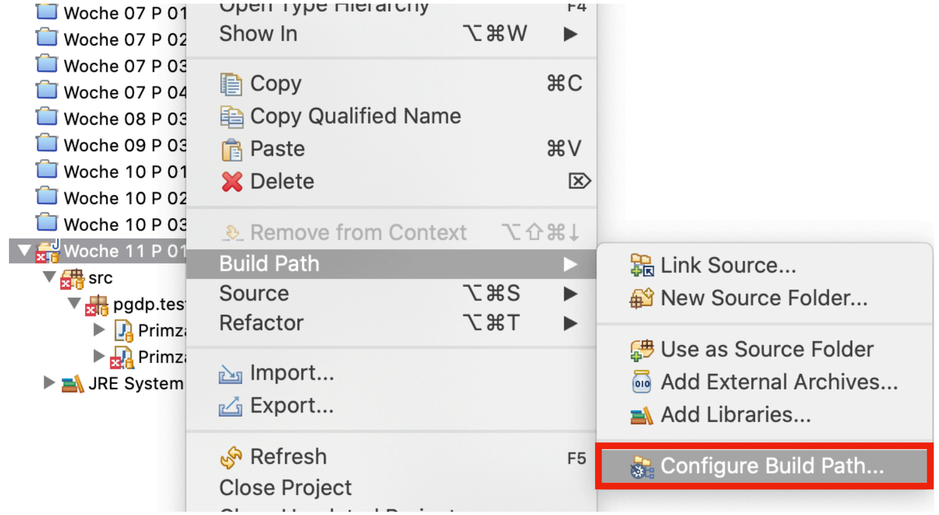


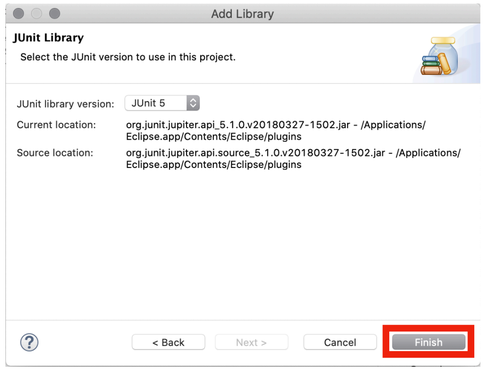
1: JUnit Framework fehlt



3: JUnit auswählen

2: Configure Build Path

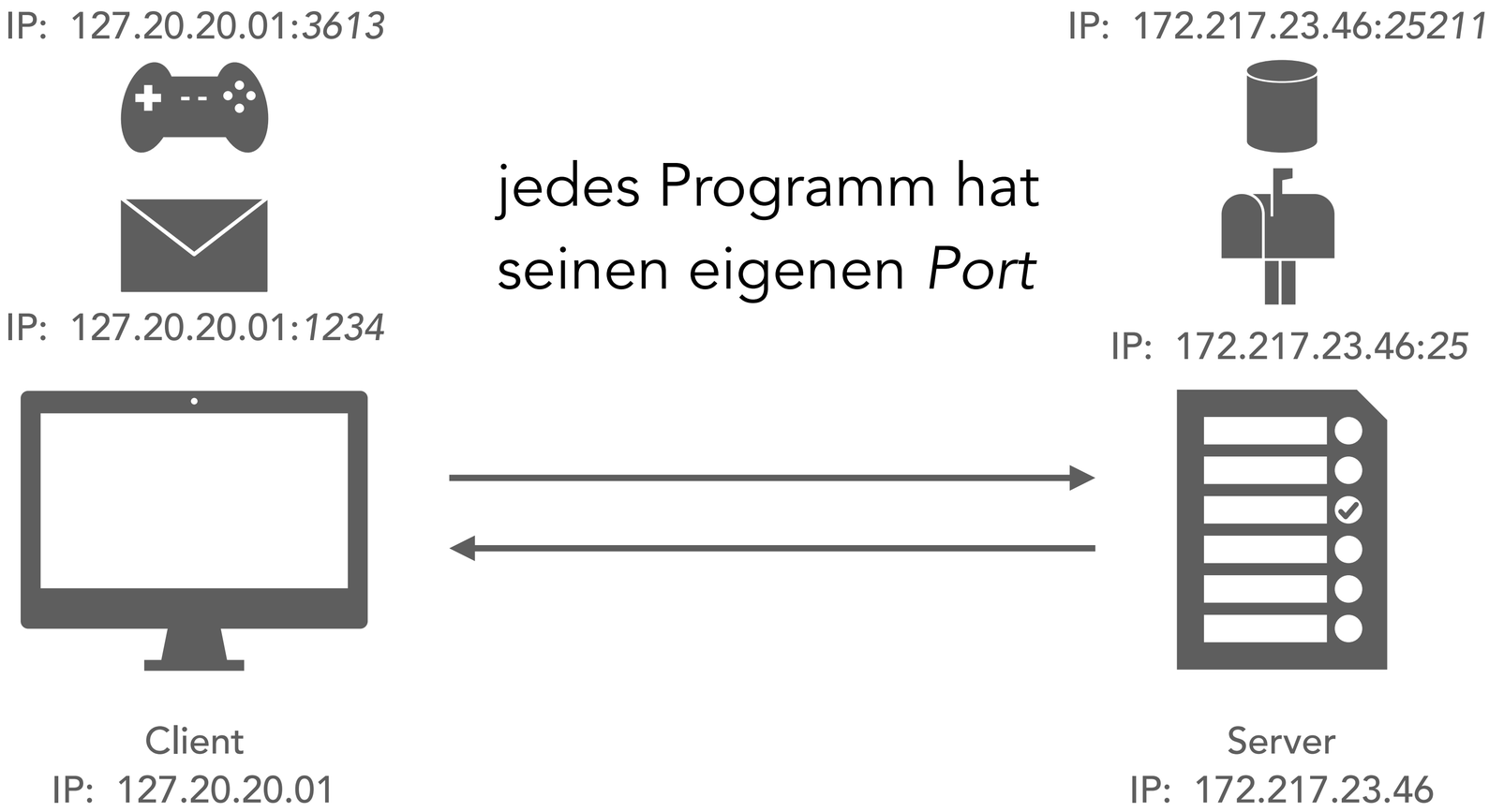




4: Finish 🙂

Server

Client verbindet sich zum Server & kommuniziert mit ihm.



Server

Server nimmt einen Client an und kommuniziert mit ihm

- Server erstellen

```
ServerSocket server = new ServerSocket(port);
```

- Client annehmen

```
Socket client = server.accept();
```

Client kann ein zweites Programm sein oder man selbst über das Terminal.

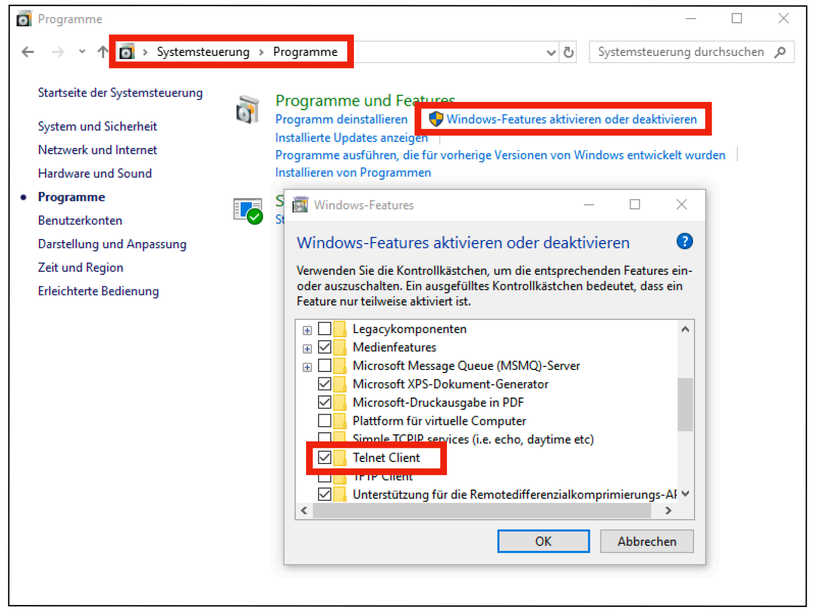
Server

Client

- Client erstellen

```
Socket client = new Socket(adresse, port);
```

- Client über Terminal
  `telnet` *<adresse> <port>*
  > Telnet muss bei Windows aktiviert werden

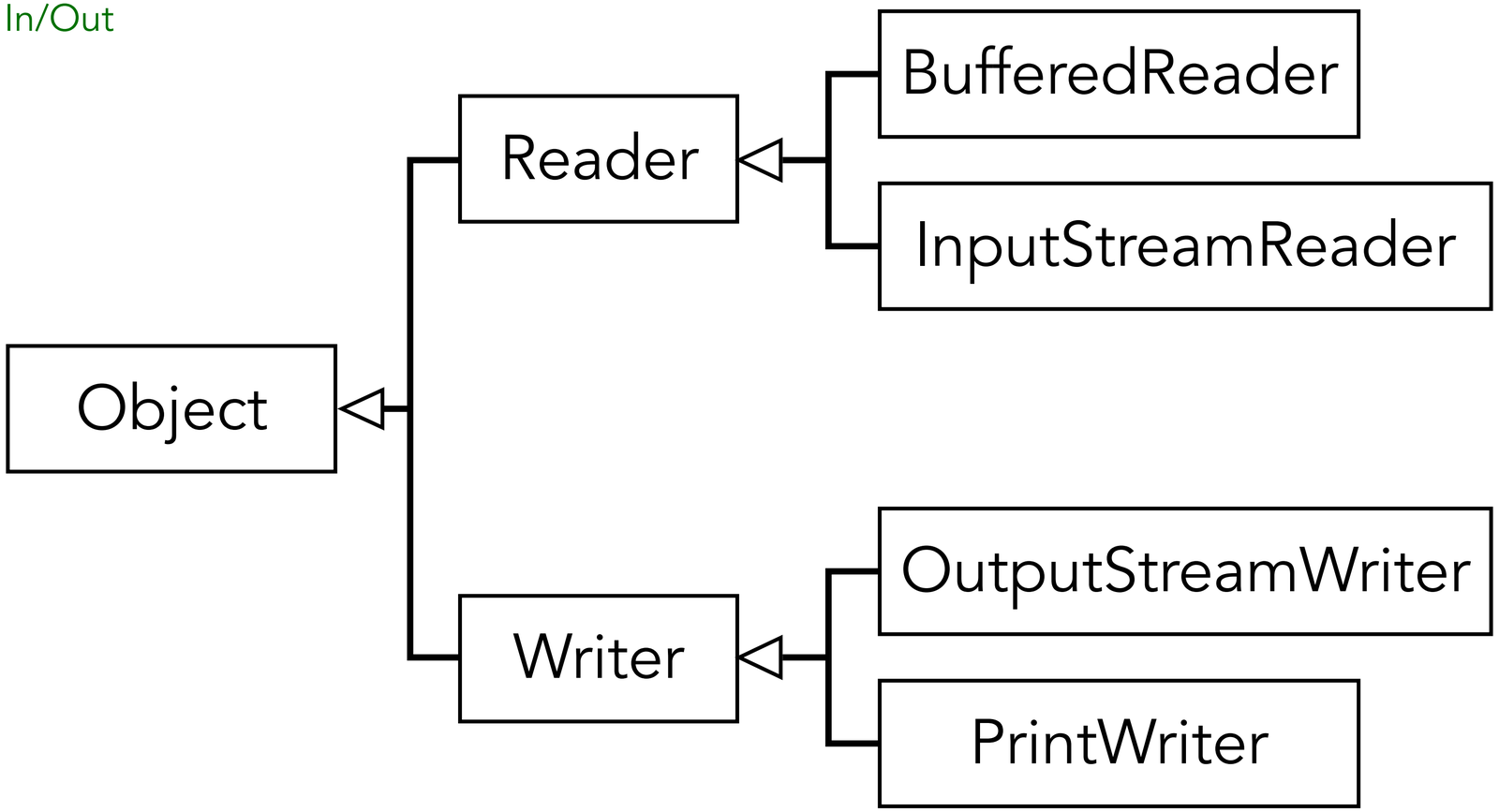


Lokaler Server ist immer *127.0.0.1* bzw. *localhost*

Server

I/O Streams zur Kommunikation

In/Out



Server

I/O Streams zur Kommunikation, Standardsetup

```
Socket client = server.accept();


PrintWriter out = new PrintWriter(
 new OutputStreamWriter(
 client.getOutputStream()));


BufferedReader in  = new BufferedReader(
 new InputStreamReader(
 client.getInputStream()));


client.close();
```

Server

I/O Streams zur Kommunikation, Standardsetup

```
// an Client senden
out.println("Hello on " + server.getPort());
out.flush();


// vom Client empfangen
String received = in.readLine();


// am Ende Streams schließen
in.close();
out.close();
```

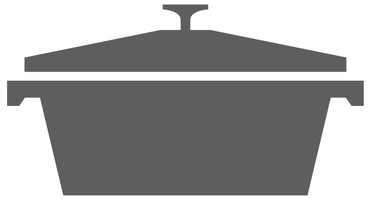
Server

Kleiner Echo Server:

```
public static void main(String[] args) throws
 IOException {
  ServerSocket server = new ServerSocket(80);
  while(true) {
    Socket client = server.accept();
    PrintWriter out = new PrintWriter(new
     OutputStreamWriter(client.getOutputStream()));
    BufferedReader in  = new BufferedReader(new
     InputStreamReader(client.getInputStream()));
    out.print("> "); out.flush;
    out.println("echo: " + in.readLine());
    out.flush(); out.close(); in.close();
    client.close();
  }
}
```

Nebenläufigkeit I

Beispiel, Nudeln mit Soße kochen

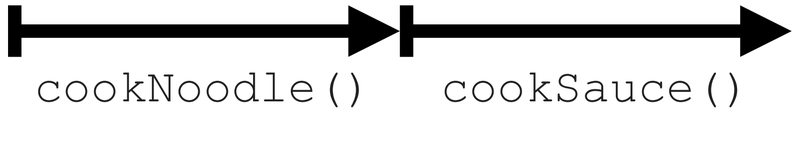


Nudeln

Soße

Bisher:

```
cookNoodle();
cookSauce();
```

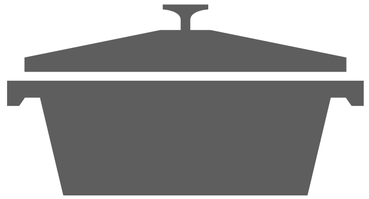


Nudeln schon kalt

Dauert lange

Nebenläufigkeit I

Beispiel, Nudeln mit Soße kochen



Nudeln

Soße

Bisher:

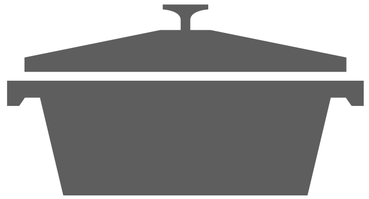
```
cookNoodle();
cookSauce();
```

Nudeln schon

Zwei unabhängige Prozesse können parallel laufen.

Nebenläufigkeit I

Beispiel, Nudeln mit Soße kochen

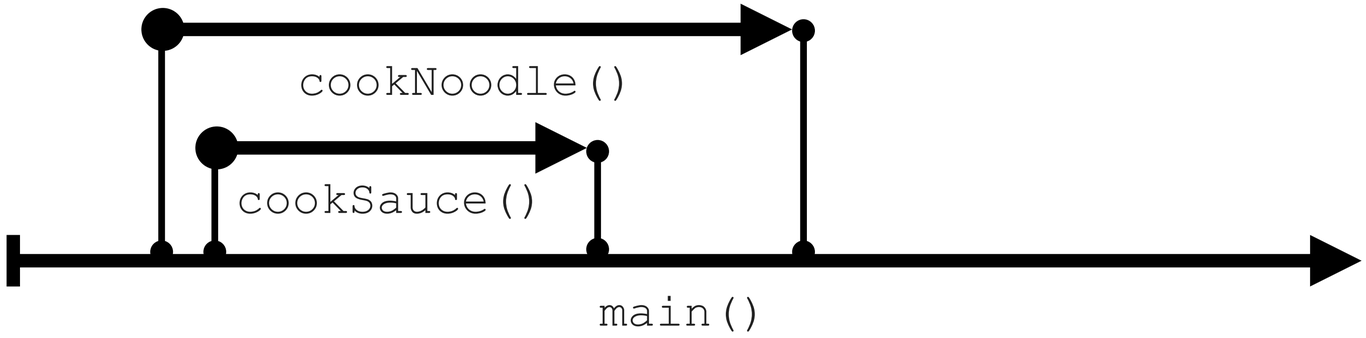


Nudeln

Soße

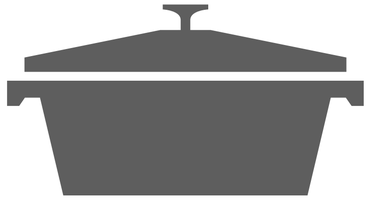
Mit Nebenläufigkeit:

```
(new Thread(Kitchen::cookNoodle)).start();
(new Thread(Kitchen::cookSauce)).start();
```



Nebenläufigkeit I

Beispiel, Nudeln mit Soße kochen

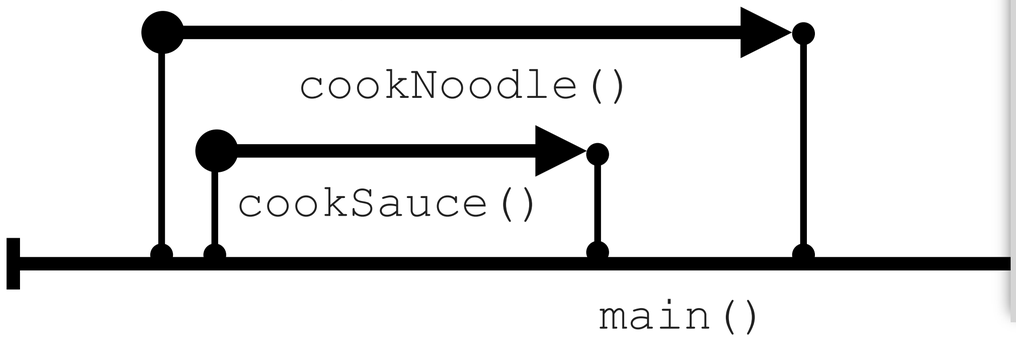


Nudeln

Soße

Mit Nebenläufigkeit:

```
(new Thread(Kitchen::cookNoodl
(new Thread(Kitchen::cookSauce
```



Zwei unabhängige Prozesse können parallel laufen.

Nebenläufigkeit I

Die Oberklasse Thread

```
public class Websession extends Thread {
  private Socket client;
  public Websession(Socket client) {
    this.client = client;
  }
  public void run() {
    answerRequest(this.client);
  }
  //...
}
```

Nebenläufigkeit I

Die Oberklasse Thread (in Action)

```
public static void main(String[] args) {
  //...
  while(true) {
    Socket client = server.accept();
    Websession session = new W
    session.start();
    // session.run(); FALSCH
  }
}
```

.start() → neuer Thread

.run() → normale Ausführung

run() ≡ Main-Methode des Threads

P11.01

PrimzahlTestTest

- Implementieren einer Klasse `PrimzahlTestTest` mit JUnit Tests, die `PrimzahlTest.isPrime(int x)` testen
- Tests müssen nur Eingaben $x > 0$ testen

Was sind sinnvolle Testfälle? Welche Edge-Cases gibt es?

P11.02

Stacktest

- Implementieren einer Klasse `StackTest` mit JUnit Tests, die Methoden von `TestStack` testen

Was sind sinnvolle Testfälle? Welche Edge-Cases gibt es?

P11.03

Chat

Die Kommunikation hat folgenden Ablauf:

1. Das Programm fragt den Nutzer nach einer Eingabe (bereits im Template). Diese Abfrage wird so lange wiederholt, bis eine Verbindung zustandekommt oder der Nutzer `exit` eingibt.
    1. Gibt der Nutzer etwas ein, was keinen `:` enthält, wird die Eingabe als Port interpretiert und das Programm versucht, auf diesem Port einen `ServerSocket` zu starten und auf Verbindungen zu warten. Der Server erwartet genau eine Verbindung.
    2. Gibt der Nutzer etwas mit `:` ein, wird die Eingabe als `<host>:<port>` interpretiert und das Programm versucht, sich mit dem gegebenen host am gegebenen Port zu verbinden.
2. Beide Teilnehmer tauschen Nachrichten aus. **Der Server beginnt mit senden.** Das Programm wartet also abwechselnd auf Nachrichten vom Nutzer und vom Socket.
3. Einer der Teilnehmer gibt `exit` ein. Beide Programme beenden sich dadurch.

P11.04

Geschäftspartner

Die beiden Pinguine Peter und Paul haben beschlossen, im Fischhandel zusammenzuarbeiten, indem jeder die Hälfte seines Einkommens durch Fischverkäufe dem anderen gibt.

Ergänzen Sie daher die Methode `sellFish` in `BusinessPenguin`, die jeweils dem Pinguin selbst und seinem Partner den halben Preis gutschreibt. Gehen Sie davon aus, dass der Preis gerade ist.

Jeder der beiden Pinguine hat nun einen Stammkunden, der bei ihm einkauft. Beide Kunden kaufen parallel ein. Ergänzen Sie die Klasse Customer so, dass ein Customer nebenläufig arbeitet. Die Kunden kaufen jeweils 5000 Fische für je 2 ein.

Vervollständigen Sie zuletzt die `main`-Methode in `Main`, sodass Sie den Pinguinen jeweils einen Stammkunden zuordnet, diese einkaufen lässt und, sobald beide Kunden fertig sind, ausgibt, wie viel Geld die beiden Pinguine jeweils besitzen.

Führen Sie das Programm mehrmals aus. Klären Sie, wie es zu diesem Ergebnis kommen konnte. Diskutieren Sie, wie man das Problem lösen könnte.