



## Aufgabe 1: Testen mit JUnit (8 Punkte)

Gegeben ist eine Java-Funktion zur Berechnung der Fakultät. Implementieren Sie für diese Funktion eine JUnit-Testmethode. Achten Sie darauf, eine vollständige Testabdeckung zu erreichen.

```
static long fact(int number) {
    if (number < 0)
        return -1;
    if (number == 0)
        return 1;
    long result = 1;
    for (int i = 2; i <= number; i++) {
        long tmp = result;
        result *= i;
        if (tmp > result) { // overflow
            return -1;
        }
    }
    return result;
}
```

Abzugeben ist ein Java-Programm, welches die Methode *static long fact(int number)* sowie die zugehörige JUnit-Testmethode *public void testFact()* enthält.

## Aufgabe 2: Hyperbolischer Sinus (16 Punkte)

Hyperbelfunktionen finden in den verschiedensten Bereichen der Mathematik, Physik und des Ingenieurwesens Anwendung. So können beispielsweise beim Entwurf von Hängebrücken die von Seilen gebildeten Bögen mithilfe hyperbolischer Funktionen wie dem Sinus Hyperbolicus modelliert werden.

Schreiben Sie eine Methode *static double sinh(double x)* zur Berechnung des Sinus Hyperbolicus. Dieser kann mit folgender Reihe entwickelt werden:

$$\sinh(x) = x + \underbrace{\frac{x^3}{3!}}_{\text{Term}} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} + \dots = \sum_{n=1}^{\infty} \frac{x^{2n-1}}{(2n-1)!}$$

Ihre Methode soll die Reihe bis zu einer Genauigkeit  $\varepsilon < 10^{-6}$  entwickeln. Der erste Term der Reihe  $< 10^{-6}$  soll noch berücksichtigt werden, anschließend soll die Berechnung abbrechen.

Schreiben Sie ein Programm, das Ihre Methode verwendet. Berechnen Sie den Sinus Hyperbolicus für die Werte  $x = 0.0, 0.1, 0.2$  bis  $1.0$  und vergleichen Sie das Ergebnis mit der Java-Funktion *Math.sinh(double x)*. Geben Sie jeweils  $x$ , den durch Ihre Funktion *sinh(double*

$x$ ) berechneten Wert, den Wert von *Math.sinh(double x)* sowie die Differenz der beiden Ergebnisse in der Methode *main* als Tabelle formatiert aus.

Die Konsolenausgabe sollte wie folgt aussehen:

x	sinh(x)	Math.sinh(x)	Difference
0,0	0,0000000000	0,0000000000	0,0000000000
0,1	0,1001667500	0,1001667500	0,0000000000
0,2	0,2013360025	0,2013360025	0,0000000000
0,3	0,3045202934	0,3045202934	0,0000000001
0,4	0,4107523251	0,4107523258	0,0000000007
0,5	0,5210953055	0,5210953055	0,0000000000
0,6	0,6366535821	0,6366535821	0,0000000001
0,7	0,7585837013	0,7585837018	0,0000000005
0,8	0,8881059800	0,8881059822	0,0000000022
0,9	1,0265167257	1,0265167257	0,0000000000
1,0	1,1752011935	1,1752011936	0,0000000002

### Hinweis:

In der Klasse *Math* gibt es eine Reihe von Methoden für numerische Berechnungen, unter anderem eine Funktion *Math.pow* für das Potenzieren. Die Verwendung dieser Methoden ist jedoch nicht erlaubt. Entwickeln Sie stattdessen Ihre Zwischenergebnisse in jedem Schleifendurchlauf durch Multiplikation selbst weiter. Erlaubt ist nur die Funktion *Math.sinh* für den tabellarischen Vergleich. Zur Formatierung der Ausgabe können Sie folgende Methode verwenden, z. B. um die Ausgabe von  $x$  auf 10 Kommastellen zu begrenzen:

```
Out.print(String.format("%.10f", x))
```

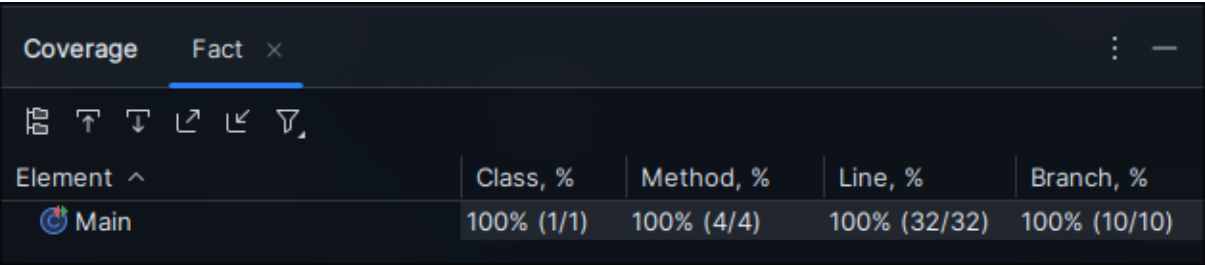
Testen Sie Ihr Programm durch Ausgabe obiger Tabelle, Benutzereingaben über die Konsole sind nicht erforderlich. Geben Sie (i) ein Ablaufdiagramm für Ihre *sinh*-Funktion, (ii) das Java-Programm sowie (iii) eine JUnit-Testmethode für Ihre *sinh*-Funktion ab.

## Aufgabe 1 - Testen mit JUnit

### Aufgabe 1/1: Java Program

Der Quellcode des Programms kann den beigefügten Dateien im Ordner "fact" entnommen werden..

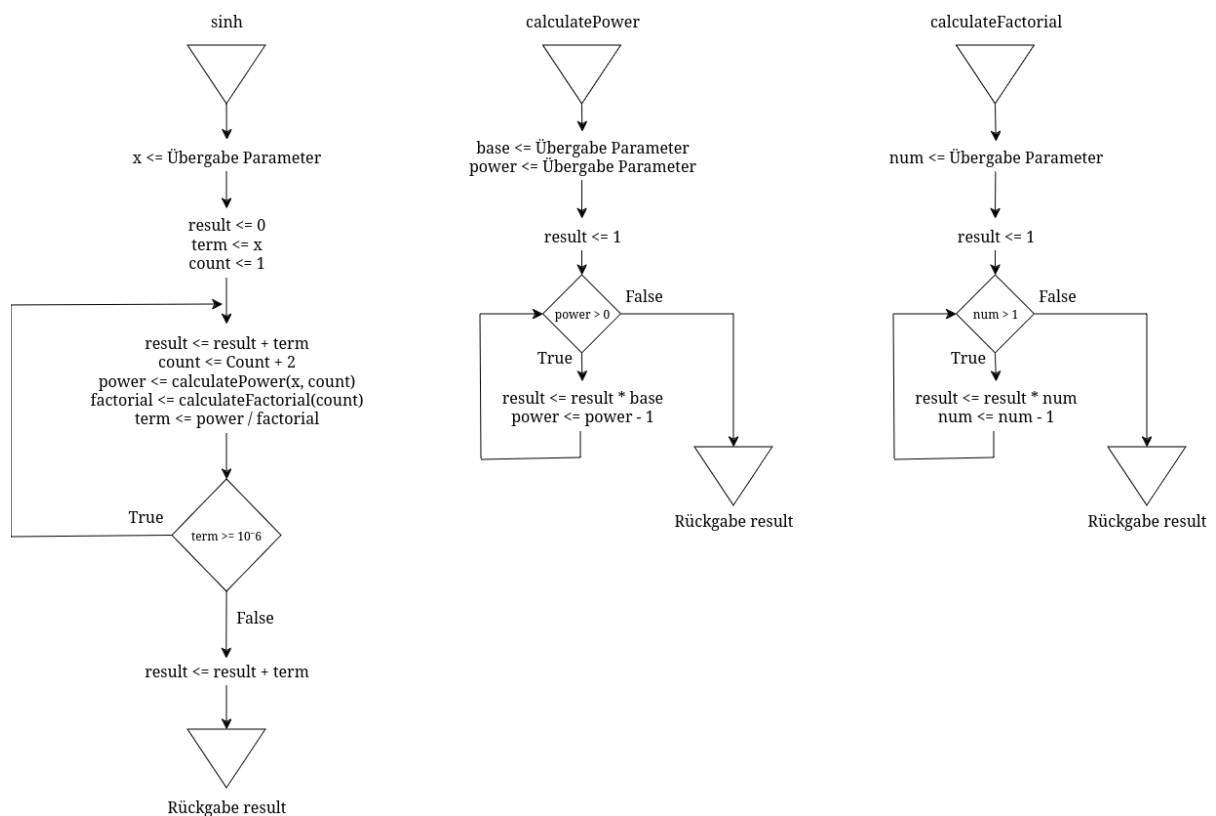
Die Testabdeckung wurde zu 100% erreicht, siehe Screenshot:



Element ^	Class, %	Method, %	Line, %	Branch, %
Main	100% (1/1)	100% (4/4)	100% (32/32)	100% (10/10)

## Aufgabe 2 - Hyperbolischer Sinus

### Aufgabe 2/1: Ablaufdiagramm



## Aufgabe 2/2: Java Program

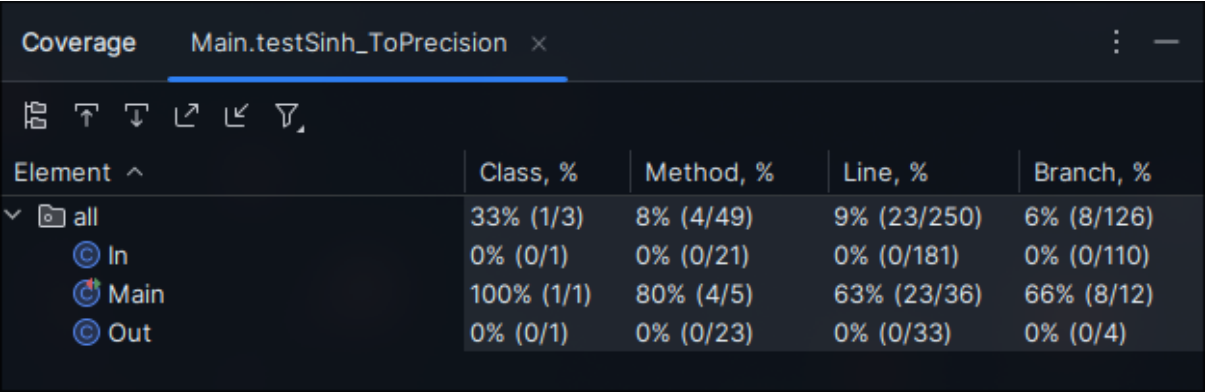
Der Quellcode des Programms kann den beigefügten Dateien im Ordner “sinh” entnommen werden.

## Aufgabe 2/3: JUnit Testmethode

Der Quellcode des Programms kann den beigefügten Dateien im Ordner “sinh” entnommen werden.

Ergebnis der Testabdeckung:

Diese Testabdeckung macht Sinn, da weder die In und Out Klassen noch die main Methode in unserem Hauptprogramm getestet wurden und wir daher auch nicht die 100% Abdeckung erwarten können.



Coverage Main.testSinh_ToPrecision ×				
Element ^				
	Class, %	Method, %	Line, %	Branch, %
all	33% (1/3)	8% (4/49)	9% (23/250)	6% (8/126)
In	0% (0/1)	0% (0/21)	0% (0/181)	0% (0/110)
Main	100% (1/1)	80% (4/5)	63% (23/36)	66% (8/12)
Out	0% (0/1)	0% (0/23)	0% (0/33)	0% (0/4)