Die Bernoulli-Zahlen B_n Ac 2022

Bernoulli-Zahlen werden z.B. gebraucht bei der **Stirling-Formel**, der **Taylor-Formel für tan(x)**, etc.

Definition der $\mathbf{B_n}$ (rekursiv) :

$$B_0 = 1 \; ; \quad B_n = \frac{-1}{n+1} \cdot \sum_{k=0}^{n-1} \binom{n+1}{k} \cdot B_k \quad \text{für } n \geq 1$$

$$\text{d.h. } B_0 = 1 \; ; \quad B_n = \frac{-1}{n+1} \cdot [\binom{n+1}{0} \cdot B_0 + \binom{n+1}{1} \cdot B_1 + \binom{n+1}{2} \cdot B_2 + \ldots + \binom{n+1}{n-1} \cdot B_{n-1}]$$

B1 = -1/2 * (2 über 0)*B0 = **-1/2**
B2 = -1/3 * [(3 über 0)*B0 + (3 über 1)*B1] = -1/3 * [1 + 3*(-1/2)] = -1/3 * (-1/2) = **1/6**
B3 = -1/4 * [(4 über 0)*B0 + (4 über 1)*B1 + (4 über 2)*B2] = -1/4 * [1 + 4*(-1/2) + 6*1/6] = **0**
B4 = -1/5 * [(5 über 0)*B0 + (5 über 1)*B1 + (5 über 2)*B2 + (5 über 3)*B3] =
-1/5 * [1 + 5*(-1/2) + 10*1/6 + 10*0] = -1/5 * 1/6 = **1/30**

Achtung : Für **ungerade** Indizes $k \geq 3$ gilt: $\mathbf{B_k = 0}$

Ein Programm zur **schnellen** Berechnung der Bernoulli-Zahlen ist nur sinnvoll, wenn man Arrays B[i] verwendet ! Die in Programmiersprachen wie Java eingebaute Rekursion kostet bei höheren n-Werten enorm viel Zeit !

Darüberhinaus muss mit höheren Datentypen wie **long** oder besser **BigInteger** in Bruchform gerechnet werden, um die größer werdenden Brüche fehlerfrei verarbeiten zu können.

Der Datentyp long liefert Bernoulli-Zahlen bis zu B35, bevor es zum Überlauf kommt.
Mit BigInteger lässt sich natürlich unbeschränkt rechnen.

Beide Methoden (iterativ) seien hier vorgestellt, wobei jeweils eine Bruch-Klasse verwendet wird:

```
static BruchLong[] bernoulliZahlIter(int n) {
   // Bn als exakter Bruch (iterativ berechnet mittels long)
   // B[n] = -1/(n+1) *
   //       [ 1*B[0] + (n+1)*B[1] + ((n+1) über 2)*B[2] + ... ((n+1) über (n-1))*B[n-1] ]
   // funktioniert bis n = 35;  danach Long-Überlauf !
   BruchLong[] brB = new BruchLong[n+1]; // Feld für die Bi von 0 bis n
   brB[0] = BruchLong.EINS;
   System.out.println("B0 = " + brB[0].toString());

   BruchLong brnUeberk;
   BruchLong brProd = null;
   BruchLong brSum;
   for (int i = 1; i <= n; i++) {
      brSum = BruchLong.NULL; // Summe = 0
      for (int k = 0; k < i; k++) {
         brnUeberk = new BruchLong(nUeberKLong(1 + i, k), 1L);
         brProd = brnUeberk.prodBr(brB[k]);
         brSum = brSum.sumBr(brProd);
      }
      brB[i] = new BruchLong(-1L, 1L + i).prodBr(brSum);
      System.out.println("B" + i + " = " + brB[i].toString());
   }

   return brB;
}
```

```
static BigBruch[] bernoulliZahlBigIter(int n) {
   // Bn als exakter Bruch (iterativ berechnet mittels BigInteger)
   // B[n] = -1/(n+1) *
   //       [ 1*B[0] + (n+1)*B[1] + ((n+1) über 2)*B[2] + ... ((n+1) über (n-1))*B[n-1] ]
   BigBruch[] brB = new BigBruch[n+1]; // Feld für die Bi von 0 bis n
   brB[0] = BigBruch.EINS;
   System.out.println("B0 = " + brB[0].toString());

   BigBruch brnUeberk;
   BigBruch brProd = null;
   BigBruch brSum;
   for (int i = 1; i <= n; i++) {
      brSum = BigBruch.NULL; // Summe = 0
      for (int k = 0; k < i; k++) {
         brnUeberk = new BigBruch(BigIntTools.nUeberKBig(1 + i, k), BigInteger.ONE);
         brProd = brnUeberk.brMul(brB[k]);
         brSum = brSum.brAdd(brProd);
      }
      brB[i] = new BigBruch(BigInteger.ONE.negate(),
                            BigInteger.ONE.add(BigInteger.valueOf(i))).brMul(brSum);
      System.out.println("B" + i + " = " + brB[i].toString());
   }

   return brB;
}
```

Es existieren auch **iterative** Formeln zur Berechnung der Bernoulli-Zahlen, z.B. die folgende von Louis Saalschütz (1893):

$$B_n = \sum_{k=0}^{n} \sum_{j=0}^{k} (-1)^j \cdot \binom{k}{j} \cdot \frac{j^n}{k+1} \quad \text{für } n \geq 1$$

Beispiele:

B1: k=0: j=0: ∑ = 1*1*0 = 0
k=1: j=0: ∑ = 1*1*0 = 0
k=1: j=1: ∑ = -1*1*1/2 = -1/2
also **B1** = 0 + 0 - 1/2 = **-1/2**

B2: k=0: j=0: ∑ = 1*1*0 = 0
k=1: j=0: ∑ = 1*1*0 = 0
k=1: j=1: ∑ = -1*1*1/2 = -1/2
k=2: j=0: ∑ = 1*1*0 = 0
k=2: j=1: ∑ = -1*2*1/3 = -2/3
k=2: j=2: ∑ = 1*1*4/3 = 4/3
also **B2** = 0 + 0 – 1/2 + 0 – 2/3 + 4/3 = **1/6**

Algorithmus (iterativ; für ein vorgegebenes n):

```
Bn = 0
für k von 0 bis n
     für j von 0 bis k
          Bn = Bn + (-1)^j * nCr(k, j) * j^n / (k+1)
```

Die ersten Bernoulli-Zahlen :

B0 = 1
B1 = -1 / 2
B2 = 1 / 6
B4 = -1 / 30
B6 = 1 / 42
B8 = -1 / 30
B10 = 5 / 66
B12 = -691 / 2730
B14 = 7 / 6
B16 = -3617 / 510
B18 = 43867 / 798
B20 = -174611 / 330
B22 = 854513 / 138
B24 = -236364091 / 2730
B26 = 8553103 / 6
B28 = -23749461029 / 870
B30 = 8615841276005 / 14322
B32 = -7709321041217 / 510
B34 = 2577687858367 / 6
B36 = -26315271553053477373 / 1919190
B38 = 2929993913841559 / 6
B40 = -261082718496449122051 / 13530
B42 = 1520097643918070802691 / 1806
B44 = -27833269579301024235023 / 690
B46 = 596451111593912163277961 / 282
B48 = -560940336899781768624912754 7 / 46410
B50 = 495057205241079648212477525 / 66
B52 = -8011657181354899573479249918 53 / 1590
B54 = 291499636348848624214181238126 91 / 798
B56 = -247939292931322675368541573966 3229 / 870
B58 = 844836133488800418620467759940360 21 / 354
B60 = -1215233140483755572040304994079820246041491 / 56786730
B62 = 1230058543408685854195303985740338615 1 / 6
B64 = -106783830147866529886385444979142647942017 / 510
B66 = 1472600022126335654051619428551932342241899101 / 64722
B68 = -787731308587187281419091492084746062443470 01 / 30
B70 = 1505381347333367003803076567377857208511438160235 / 4686
B72 = -582795496166994411043827724464106736528248830184426 0429 / 140100870
B74 = 3415241728922116801433007373147263518668830778308 7 / 6
B76 = -246550888259353727076871960405851999043652678288658 01 / 30
B78 = 414846365575400828295179035549542073492199375372400483487 / 3318
B80 = -460378429947945764693557496901904684979425787275128891965686 7 / 230010
B82 = 167701414918514583682315450978626990020773602757025341488161 3 / 498
B84 = -202457619593529036023113116011173100998991739119809087728108393247 7 / 3404310
B86 = 660714619417678653573847847426261496277830686653388931761996983 / 6
B88 = -131142648867401750799551142401931184334575027557202864429691989057404 7 / 61410
B90 = 1179057279021082799884123351249215083775254949669647116231545215727922535 / 272118
B92 = -1295585948207537527989427828538576749659341483719435143023316326829946247 / 1410
B94 = 1220813806579744469607301679413201203958508415202696621436215105284649447 / 6
B96 = -2116004495972665130975977281098242336730439543890602341506387334200506683499872 59 /
4501770
B98 = 67908260672905495624051117546403605607342195728504487509073961249992947058239 / 6
B100 = -945980378191221252952274330694937218727028415330669361333856962043113954151972477 11 /
33330
B102 = 3204019410860907078243020782116241775491817197152717450679002501086861530836678158791 /
4326
B104 = -319533631363830011287103352796174274671189606078272738327103470162849568365549721224 0
53 / 1590
B106 = 3637390317261741440815182015159342716923129864058169003893081637828187987338620234657 2
901 / 642
B108 = -346934224784782878955208865932385254139976678576049114687000589137150126631972489759 23
06597338057 / 209191710
B110 = 7645992940484742892248134246724347500528752413412307906683593870759797606269585779977 93
0217515 / 1518

B112 = -265087960215509971335259721468516201444315149919250989645178842768096675651487551536678120355260010 9 / 1671270
B114 = 21737832319369163333310761086652991475721156679090831360806110114933605484234593650904188618562649 / 42
B116 = -309553916571842976912513458033841416869004128064329844245504045721008957524571968271388199595754752259 / 1770
B118 = 3669631199697131115349471515855850066846063610806992043010594406764144850458064618893717763545170957 99 / 6
B120 = -515074865350791090618439968578499832740951703532626752130928691671992974749229853588113293670776826778032820701 31 / 2328255930
B122 = 4963366607926258191253263747599075743872279031106013977030931179315068321410043132903311367809803796856443 1 / 6
B124 = -958767753342471287507749031075424446205788300132973368195535127293585933544359444136319436102684726890946090 01 / 30
B126 = 5556330281949274850616324408918951380525567307126747246796782304333594286400508981287241419934529638692081513802696639 / 4357878
B128 = -267754707742548082886954405585282394779291459592551740629978686063357792734863530145362663093519862048495908453718017 / 510
B130 = 19282151751361309156452995222715964353076110101647284587837330205285486224035040785951744116938938827393347351425624180 15 / 8646
B132 = -41095194584699337820902048652357193812325807787047750243346974796265007075470486381264639280186368669410680574733537031 2946831 / 4206930
B134 = 26459017187071772563363573724887901515125452559316868841191855484066776559169054072798731639125243434866469463939484190167 / 6
B136 = -84290226343367405131287578060366193649336612397547435767189206912230442242628212786558235455817749737691517685781164837036649737 / 4110
B138 = 26948665489908809360438516837241130408490784946642824838621508930604785015595462434236333756933257577957094383259071549735902881364 29 / 274386
B140 = -32894909864358988039306995488518840068805374769311309813074670851625048029736180966938595981252747416041814678266511443938746966019460 49 / 679470
B142 = 14731853280888589565870080442453214239804217023990642676194878997407546061581643106569966189211748270209483494554402556608073385149191 / 6
B144 = -30502446983736075650351558369017263574050071042565667618841918524348510337447612763926956693296268559651835032957935174115260562444310246126404 93 / 2381714790
B146 = 41205700262801148715261133159078640261655456088085411539738176800347902626835242848558100086219052382902401434814030229870372716839898248 63 / 6
B148 = -16917371456140189798655610951121661896076828521473014008164806759169578711786484332848214936063612359733465846673361817939379503448285578983471 49 / 4470
B150 = 46336557938916274144328442581180626498223372542529579985229980732537931550157230576003059476968829630837519391378770370769301022410161390422797906627 5 / 2162622