POLITECHNIKA KRAKOWSKA
WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ

Programowanie wielowersyjne

Systemy odporne na błędy

Sprawozdanie z laboratorium nr 4

Maria Guz
Karol Pątko
Wojciech Maludziński

Wstęp

Programowanie wielowersyjne służy do tworzenia oprogramowania o wysokiej wiarygodności. Technika polega na przygotowaniu kilku (minimum trzech, zawsze nieparzystej liczby) różnych wersji programu realizującego dokładnie to samo zadanie. Celem zwiększenia niezawodności i unikania tych samych błędów projektowych, różne programy powinny być opracowywane przez różne zespoły programistyczne, przy użyciu innych algorytmów, struktur danych, kompilatorów, programów. Wszystkie opracowane wersje programu uruchamiają się dla tych samych danych wejściowych. Najbardziej prawdopodobny wynik ustalany jest w drodze głosowania większościowego.

Silnia - iloczyn wszystkich liczb naturalnych dodatnich, nie większych od n, oznaczana wykrzyknikiem(**!**). Przykład: (4!=1*2*3*4=24; 5!=1*2*3*4*5=120)

Ciąg Fibonacciego - ciąg liczb naturalnych, gdzie każda następna jest sumą dwóch poprzednich(pierwszy wyraz równy 0, drugi 1). Ciekawą wartością jest, że iloraz n oraz n-1 liczby w ciągu wynosi w przybliżeniu 1,618 (tzw. “złota liczba”). Pierwsze liczby ciągu to 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987.

Cel zajęć

Celem zajęć laboratoryjnych nr 4 jest opracowanie 3 różnych wersji: funkcji liczącej dany wyraz ciągu Fibonacciego oraz funkcji liczącej silnię, wszystko w języku Java. Zadanie zostało wykonane przez trzy różne osoby. Przed wykonaniem należało określić założenia wstępne (m. in. rodzaj algorytmu i zwracana wartość). Zdecydowaliśmy się na algorytmy iteracyjne (zwykle bardziej efektywne obliczeniowo od rekurencyjnych), a zwracaną wartością był double, którego maksymalna wartość wynosi w przybliżeniu 1.7*10^308 (int ma maksymalną wartość 2147483647, a więc mniejszą). Po wykonaniu tej części, należało opracować program, który przy wykorzystaniu wszystkich wersji zrealizuje zadanie dla tych samych wejściowych(również wartości graniczne, spoza zakresu), a następnie w drodze głosowania 2/3 wybierze najbardziej prawdopodobny wynik.

Treść zadania

1. *W zespołach 3 – osobowych, przy wykorzystaniu wybranego języka programowania opracować trzy wersje funkcji obliczającej:*
a. wartość wskazanego wyrazu ciągu Fibonacciego;
b. silnię podanej liczby całkowitej.

2. *Każda osoba w zespole opracowuje swoją wersję wybranej funkcji (wszystkie w tym samym języku programowania). Wcześniej należy uzgodnić sposób realizacji algorytmu, np.:*
a. realizację na liczbach całkowitych (int, BigInteger) lub na zmiennoprzecinkowych (dla zwiększenia zakresu liczb);
b. algorytm iteracyjny lub rekurencyjny.

3. *Należy opracować program, który z pomocą trzech wersji funkcji realizuje to samo zadanie dla takich samych danych wejściowych i ustala najbardziej prawdopodobny wynik w drodze głosowania 2/3 (lub zwraca błąd, jeżeli każda wersja funkcji zwraca inne wartości).*

4. *Gotowy program należy przetestować dla danych z całego zakresu liczb, również dla wartości granicznych i wychodzących poza zakres (np. liczby ujemne lub silnia liczby 10000 lub więcej), tak żeby spowodować pojawienie się błędów w jednej lub kilku wersji funkcji (błędy muszą się pojawić!).*

Opracowane funkcje Factorial() i Fibonacci()

Factorial() i Fibonacci() - osoba A

```
public class AlgsA implements IAlgs{

   @Override
   public double Fibonacci(int value) {
       double x1 = 1;
       double x2 = 1;
       double temp;

       if(value == 1) return 1;
       if(value == 2) return 1;
       for (int i = 3; i <= value; ++i) {
           temp = x1 + x2;
           x1 = x2;
           x2 = temp;
       }

       return x2;
   }

   @Override
   public double Factorial(int value) {
       double ret = 1;

       for (int i = 1; i <= value; ++i) {
           ret *= i;
       }

       return ret;
   }
}
```

Factorial() i Fibonacci() - osoba B

```
public class AlgsB implements IAlgs{

   @Override
   public double Fibonacci(int value){
       double i = 0, first=0, second=1, sum=0;

       while(i < value-1){
           sum = first + second;
           first = second;
           second = sum;

           i++;
       }

       return sum;
   }

   @Override
   public double Factorial(int value){
       Integer score = 1, i = 1;

       while(i <= value){
           score = score * i;
           i++;
       }

       return score;
   }
}
```

Factorial() i Fibonacci() - osoba C

```
public class AlgsC implements IAlgs{
   @Override
   public double Fibonacci(int value) {
       int first = 0;
       int var = 1, next;
       for (int i = 2; i <= value; i++) {
           next = first + var;
           first = var;
           var = next;
       }
       return var;
   }

   @Override
   public double Factorial(int value) {
       long var = 1;

       if(value == 0){
           return var;
       }

       for(int i=value; i>1; i--){
           var = var * i;
       }

       return var;
   }
}
```

Wszystkie powyższe klasy implementowały interfejs IAlgs.

Interfejs IAlgs

```
public interface IAlgs {
   double Fibonacci(int value);
   double Factorial(int value); }
```

Kod programu

Poza powyższymi klasami, kod programu został zawarty w trzech klasach: Main, Vote oraz Test.

Main.java

```
public class Main {

   public static void main(String[] args) {

       Vote vote = new Vote();
       //najwiekszy wyraz fib dla doubla to 1476, potem jest infinity


       /*================ZAKRES FIBONACCI==================*/
       for(int i = 1; i<=1476; ++i){
           try {
               Vote.Fibonacci(i);
           } catch (Exception e) {
               //e.printStackTrace(); //nie wypisujemy dla czytelnosci
           }
       }
       System.out.println("\n##########-------FIBONACCI<1;1476>-------##########");
       Test.print();

       //wartości graniczne
       System.out.println("-------Wartości graniczne-------");
       List<Integer> values = new ArrayList<>();

values.add(1);values.add(2);values.add(1473);values.add(1474);values.add(1475);values.add(1476);values.add(1477);v
alues.add(1478);
       for(Integer i : values){
           System.out.println("i -> " +i + " |      :" + (new AlgsA()).Fibonacci(i) + " | " +  (new
AlgsB()).Fibonacci(i) + " | " + (new AlgsC()).Fibonacci(i));
       }

       //kilka różnic
       System.out.println("-------Różnice-------");
       values.clear();
       /*bBad*/values.add(1);
//
/*allGood*/values.add(2);values.add(3);values.add(4);values.add(15);values.add(31);values.add(44);values.add(45);v
alues.add(46);
       /*cBad*/values.add(47);values.add(48);values.add(49);values.add(1477);values.add(1478);
       for(Integer i : values){
           System.out.println("i -> " +i + " |      :" + (new AlgsA()).Fibonacci(i) + " | " +  (new
AlgsB()).Fibonacci(i) + " | " + (new AlgsC()).Fibonacci(i));
       }
       Test.clear();

       /*================UJEMNE FIBONACCI==================*/
       System.out.println("\n##########-------FIBONACCI <-10;0>-------##########");

       for(int i = -10; i<=0; ++i){
           System.out.println("i -> " +i + " |      :" + (new AlgsA()).Fibonacci(i) + " | " +  (new
AlgsB()).Fibonacci(i) + " | " + (new AlgsC()).Fibonacci(i));
           try {
               Vote.Fibonacci(i);
           } catch (Exception e) {
               //e.printStackTrace();
           }
       }


       Test.print();
       Test.clear();


       /*================ZAKRES FACTORIAL==================*/
       //najwiekszy wyraz fac dla doubla to 170, potem jest infinity
       for(int i = 1; i<=170; ++i){
           try {
               Vote.Factorial(i);
```

```
            } catch (Exception e) {
                //e.printStackTrace();
            }
        }

        System.out.println("\n##########-------FACTORIAL<1;170>-------##########");
        Test.print();
        Test.clear();

        //wartości graniczne
        System.out.println("-------Wartości graniczne-------");
        values.clear();

values.add(1);values.add(2);values.add(165);values.add(169);values.add(170);values.add(171);values.add(172);
        for(Integer i : values){
            System.out.println("i -> " +i + " |      :" + (new AlgsA()).Factorial(i) + " | " +  (new
AlgsB()).Factorial(i) + " | " + (new AlgsC()).Factorial(i));
        }

        //kilka różnic
        System.out.println("-------Różnice-------");
        values.clear();
//        /*allGood*/values.add(1);values.add(12);
        /*bBad*/values.add(13);values.add(20);
        /*allDifferent*/values.add(21);values.add(22);values.add(23);values.add(64);values.add(65);
        /*aBad*/values.add(66);values.add(70);

        for(Integer i : values){
            System.out.println("i -> " +i + " |      :" + (new AlgsA()).Factorial(i) + " | " +  (new
AlgsB()).Factorial(i) + " | " + (new AlgsC()).Factorial(i));
        }
//        Test.clear();



        /*================UJEMNE FACTORIAL==================*/

        System.out.println("\n##########-------FACTORIAL<-10;0>-------##########");


        for(int i =-10; i<=0; ++i){
            System.out.println("i -> " +i + " |      :" + (new AlgsA()).Factorial(i) + " | " +  (new
AlgsB()).Factorial(i) + " | " + (new AlgsC()).Factorial(i));
            try {
                Vote.Factorial(i);
            } catch (Exception e) {
                //e.printStackTrace();
            }
        }

        Test.print();
    }
}
```

Klasa Main wypisuje wyniki działań na funkcjach factorial oraz fibonacci. Działania te są przeprowadzane na dużym zbiorze testowym, są zawarte przypadki graniczne, liczby ze zbioru ujemnego, tak aby przetestować działanie funkcji w każdych warunkach.

Vote.java

```
public class Vote {
   public static double Fibonacci(int value) throws Exception{

        IAlgs algsA = new AlgsA();
        IAlgs algsB = new AlgsB();
        IAlgs algsC = new AlgsC();

        double fib1, fib2, fib3;

        fib1 = algsA.Fibonacci(value);
        fib2 = algsB.Fibonacci(value);
        fib3 = algsC.Fibonacci(value);

        if(fib1 == fib2 && fib1 == fib3){
            //wszystkie zgodne - najlepszy przypadek;
            Test.incrementAllGood(value);
```

```
            return fib1;
        }
        if(fib1 == fib2 && fib1 != fib3){
            //2 zgodne - sredni przypadek
            Test.incrementCBad(value);
            return fib1;
        }
        if(fib1 == fib3 && fib1 != fib2){
            //2 zgodne - sredni przypadek
            Test.incrementBBad(value);
            return fib1;
        }
        if(fib2 == fib3 && fib2 != fib1){
            //2 zgodne - sredni przypadek
            Test.incrementABad(value);
            return fib2;
        }

        Test.incrementAllDifferent(value);
        throw new Exception("All return values are different.");
    }

    public static double Factorial(int value) throws Exception{

        IAlgs algsA = new AlgsA();
        IAlgs algsB = new AlgsB();
        IAlgs algsC = new AlgsC();

        double fac1, fac2, fac3;

        fac1 = algsA.Factorial(value);
        fac2 = algsB.Factorial(value);
        fac3 = algsC.Factorial(value);

        if(fac1 == fac2 && fac1 == fac3){
            //wszystkie zgodne - najlepszy przypadek;
            Test.incrementAllGood(value);
            return fac1;
        }
        if(fac1 == fac2 && fac1 != fac3){
            //2 zgodne - sredni przypadek
            Test.incrementCBad(value);
            return fac1;
        }
        if(fac1 == fac3 && fac1 != fac2){
            //2 zgodne - sredni przypadek
            Test.incrementBBad(value);
            return fac1;
        }
        if(fac2 == fac3 && fac2 != fac1){
            //2 zgodne - sredni przypadek
            Test.incrementABad(value);
            return fac2;
        }

        Test.incrementAllDifferent(value);
        throw new Exception("All return values are different.");
    }
}
```

Klasa Vote wykonuje wszystkie warianty funkcji dla danej wartości, a także sprawdza ich zgodność.

Test.java

```
public class Test {
    static private int allGood = 0; static private List<Integer> allGoodList = new ArrayList<>();
    static private int aBad = 0; static private List<Integer> aBadList = new ArrayList<>();
    static private int bBad = 0; static private List<Integer> bBadList = new ArrayList<>();
    static private int cBad = 0; static private List<Integer> cBadList = new ArrayList<>();
    static private int allDifferent = 0; static private List<Integer> allDifferentList = new ArrayList<>();

    public static void incrementAllGood(int i){
        ++allGood;
        allGoodList.add(i);
    }

    public static void incrementABad(int i){
```

```
        ++aBad;
        aBadList.add(i);
    }

    public static void incrementBBad(int i){
        ++bBad;
        bBadList.add(i);
    }

    public static void incrementCBad(int i){
        ++cBad;
        cBadList.add(i);
    }

    public static void incrementAllDifferent(int i){
        ++allDifferent;
        allDifferentList.add(i);
    }

    public static void clear(){
        allGood = 0;
        aBad = 0;
        bBad = 0;
        cBad = 0;
        allDifferent = 0;
        allGoodList.clear();
        aBadList.clear();
        bBadList.clear();
        cBadList.clear();
        allDifferentList.clear();
    }

    public static void print(){
        System.out.println("allGood: "+allGood+", "+allGoodList);
        System.out.println("aBad: "+aBad+", "+aBadList);
        System.out.println("bBad: "+bBad+", "+bBadList);
        System.out.println("cBad: "+cBad+", "+cBadList);
        System.out.println("allDifferent: "+allDifferent+", "+allDifferentList);}}
```

Klasa Test zbiera dane o tym, ile razy które funkcje były zgodne co do wyniku, ile razy wynik jednej różnił się od pozostałych, a także wyświetla te dane.

Działanie programu

Funkcja main wywołuje metody liczące ciąg Fibonacciego oraz silnię dla różnych liczb (w tym spoza zakresu oraz liczby ujemne). Jako wyjście wypisywane są proste statystyki pokazujące dla jakiej liczby jaka wartość została wybrana co pozwala później odnaleźć źródło błędu. Po wypisaniu informacji o wyborze wartości sprawdzane są wartości graniczne, przekraczające zakres oraz wartości ujemne.

Output

```
##########-------FIBONACCI<1;1476>-------##########
allGood: 45, [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46]
aBad: 0, []
bBad: 1, [1]
cBad: 1430, [47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162,
163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216,
217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297,
```

298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378,
379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405,
406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432,
433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459,
460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486,
487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513,
514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540,
541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567,
568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594,
595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621,
622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648,
649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675,
676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702,
703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729,
730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756,
757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783,
784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810,
811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837,
838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864,
865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891,
892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918,
919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945,
946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972,
973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999,
1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021,
1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043,
1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065,
1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087,
1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109,
1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131,
1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153,
1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175,
1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197,
1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219,
1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241,
1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263,
1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285,
1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307,
1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329,
1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351,
1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373,
1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395,
1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417,
1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439,
1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461,
1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476]
allDifferent: 0, []
-------Wartości graniczne-------
i -> 1 | :1.0 | 0.0 | 1.0
i -> 2 | :1.0 | 1.0 | 1.0
i -> 1473 | :3.085383026678456E307 | 3.085383026678456E307 | 2.31838946E8
i -> 1474 | :4.992254605477766E307 | 4.992254605477766E307 | -2.085564377E9
i -> 1475 | :8.077637632156222E307 | 8.077637632156222E307 | -1.853725431E9
i -> 1476 | :1.3069892237633987E308 | 1.3069892237633987E308 | 3.55677488E8
i -> 1477 | :Infinity | Infinity | -1.498047943E9
i -> 1478 | :Infinity | Infinity | -1.142370455E9
-------Różnice-------
i -> 1 | :1.0 | 0.0 | 1.0
i -> 47 | :2.971215073E9 | 2.971215073E9 | -1.323752223E9
i -> 48 | :4.807526976E9 | 4.807526976E9 | 5.1255968E8
i -> 49 | :7.778742049E9 | 7.778742049E9 | -8.11192543E8
i -> 1477 | :Infinity | Infinity | -1.498047943E9
i -> 1478 | :Infinity | Infinity | -1.142370455E9

##########-------FIBONACCI <-10;0>-------##########
i -> -10 | :1.0 | 0.0 | 1.0
i -> -9 | :1.0 | 0.0 | 1.0
i -> -8 | :1.0 | 0.0 | 1.0
i -> -7 | :1.0 | 0.0 | 1.0
i -> -6 | :1.0 | 0.0 | 1.0
i -> -5 | :1.0 | 0.0 | 1.0
i -> -4 | :1.0 | 0.0 | 1.0
i -> -3 | :1.0 | 0.0 | 1.0
i -> -2 | :1.0 | 0.0 | 1.0
i -> -1 | :1.0 | 0.0 | 1.0
i -> 0 | :1.0 | 0.0 | 1.0
allGood: 0, []
aBad: 0, []
bBad: 11, [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0]
cBad: 0, []
allDifferent: 0, []

##########-------FACTORIAL<1;170>-------##########
allGood: 12, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
aBad: 105, [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96,
97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124,
125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151,
152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170]
bBad: 8, [13, 14, 15, 16, 17, 18, 19, 20]
cBad: 0, []
allDifferent: 45, [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,

```
50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]
-------Wartości graniczne-------
i -> 1 |      :1.0 | 1.0 | 1.0
i -> 2 |      :2.0 | 2.0 | 2.0
i -> 165 |      :5.4239106661315 86E295 | 0.0 | 0.0
i -> 169 |      :4.2690680090047027E304 | 0.0 | 0.0
i -> 170 |      :7.257415615307994E306 | 0.0 | 0.0
i -> 171 |      :Infinity | 0.0 | 0.0
i -> 172 |      :Infinity | 0.0 | 0.0
-------Różnice-------
i -> 13 |      :6.2270208E9 | 1.932053504E9 | 6.2270208E9
i -> 20 |      :2.43290200817664E18 | -2.102132736E9 | 2.43290200817664E18
i -> 21 |      :5.109094217170944E19 | -1.195114496E9 | -4.2492900494192148E18
i -> 22 |      :1.1240007277776077E21 | -5.22715136E8 | -1.250660718674968 58E18
i -> 23 |      :2.585201673888498E22 | 8.6245376E8 | 8.1282916178948 26E18
i -> 64 |      :1.2688693218588417E89 | 0.0 | -9.223372036854776E18
i -> 65 |      :8.247650592082472E90 | 0.0 | -9.223372036854776E18
i -> 66 |      :5.443449390774431E92 | 0.0 | 0.0
i -> 70 |      :1.1978571669969 89E100 | 0.0 | 0.0

##########-------FACTORIAL<-10;0>-------##########
i -> -10 |      :1.0 | 1.0 | 1.0
i -> -9 |      :1.0 | 1.0 | 1.0
i -> -8 |      :1.0 | 1.0 | 1.0
i -> -7 |      :1.0 | 1.0 | 1.0
i -> -6 |      :1.0 | 1.0 | 1.0
i -> -5 |      :1.0 | 1.0 | 1.0
i -> -4 |      :1.0 | 1.0 | 1.0
i -> -3 |      :1.0 | 1.0 | 1.0
i -> -2 |      :1.0 | 1.0 | 1.0
i -> -1 |      :1.0 | 1.0 | 1.0
i -> 0 |      :1.0 | 1.0 | 1.0
allGood: 11, [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0]
aBad: 0, []
bBad: 0, []
cBad: 0, []
allDifferent: 0, []

Process finished with exit code 0
```

Omówienie działania

Zakres < 1 ; 1476 >

Dla 2 - 46 wyrazu ciągu wszystkie algorytmy zwracają ten sam wynik. Dla pierwszego wyrazu ciągu algorytm B zwraca wartość inną niż pozostałe algorytmy, ze względu na błędną implementację. Od 47 wyrazu ciągu algorytm C daje inny wynik, ponieważ działa na intach zamiast na doublach.

Kod

```
for(int i = 1; i<=1476; ++i){
           try {
               Vote.Fibonacci(i);
           } catch (Exception e) {
               //e.printStackTrace();
           }
       }
       System.out.println("\n##########-------FIBONACCI<1;1476>-------##########");
       Test.print();

       //wartości graniczne
       System.out.println("-------Wartości graniczne-------");
       List<Integer> values = new ArrayList<>();

values.add(1);values.add(2);values.add(1473);values.add(1474);values.add(1475);values.add(1476);values.add(1477);v
alues.add(1478);
       for(Integer i : values){
           System.out.println("i -> " +i + " |      :" + (new AlgsA()).Fibonacci(i) + " | " +  (new
AlgsB()).Fibonacci(i) + " | " + (new AlgsC()).Fibonacci(i));
       }

       //kilka różnic
```

```
        System.out.println("-------Różnice-------");
        values.clear();
        /*bBad*/values.add(1);
//
/*allGood*/values.add(2);values.add(3);values.add(4);values.add(15);values.add(31);values.add(44);values.add(45);v
alues.add(46);
        /*cBad*/values.add(47);values.add(48);values.add(49);values.add(1477);values.add(1478);
        for(Integer i : values){
            System.out.println("i -> " +i + " |      :" + (new AlgsA()).Fibonacci(i) + " | " +  (new
AlgsB()).Fibonacci(i) + " | " + (new AlgsC()).Fibonacci(i));
        }
        Test.clear();
```

Output

```
allGood: 45, [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46]
aBad: 0, []
bBad: 1, [1]
cBad: 1430, [47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77,
78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162,
163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189,
190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216,
217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297,
298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378,
379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405,
406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432,
433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459,
460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486,
487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513,
514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540,
541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567,
568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594,
595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621,
622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648,
649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675,
676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702,
703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729,
730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756,
757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783,
784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810,
811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837,
838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864,
865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891,
892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918,
919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945,
946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972,
973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999,
1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021,
1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043,
1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065,
1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087,
1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109,
1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131,
1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153,
1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175,
1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197,
1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219,
1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241,
1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263,
1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285,
1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307,
1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329,
1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351,
1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373,
1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395,
1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417,
1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439,
1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461,
1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476]
allDifferent: 0, []
```

Wartości graniczne, różnice

Dla wizualizacji wyników zostały wypisane wartości graniczne oraz różnice.

```
-------Wartości graniczne-------
i -> 1 |      :1.0 | 0.0 | 1.0
i -> 2 |      :1.0 | 1.0 | 1.0
i -> 1473 |      :3.085383026678456E307 | 3.085383026678456E307 | 2.31838946E8
i -> 1474 |      :4.992254605477766E307 | 4.992254605477766E307 | -2.085564377E9
i -> 1475 |      :8.077637632156222E307 | 8.077637632156222E307 | -1.853725431E9
i -> 1476 |      :1.3069892237633987E308 | 1.3069892237633987E308 | 3.55677488E8
i -> 1477 |      :Infinity | Infinity | -1.498047943E9
i -> 1478 |      :Infinity | Infinity | -1.142370455E9
-------Różnice-------
i -> 1 |      :1.0 | 0.0 | 1.0
i -> 47 |      :2.971215073E9 | 2.971215073E9 | -1.323752223E9
i -> 48 |      :4.807526976E9 | 4.807526976E9 | 5.1255968E8
i -> 49 |      :7.778742049E9 | 7.778742049E9 | -8.11192543E8
i -> 1477 |      :Infinity | Infinity | -1.498047943E9
i -> 1478 |      :Infinity | Infinity | -1.142370455E9
```

Zakres < -10 ; 0 >

Dla wartości ujemnych żaden program nie działa poprawnie, aczkolwiek ze względu na to, że A i C zwracają 1, a B przyjmuje 0, to wyniki zwracane przez większość algorytmów zostały uznane za najbardziej prawdopodobne.

Kod

```
System.out.println("\n##########-------FIBONACCI <-10;0>-------##########");

for(int i = -10; i<=0; ++i){
   System.out.println("i -> " +i + " |      :" + (new AlgsA()).Fibonacci(i) + " | " +  (new AlgsB()).Fibonacci(i)
+ " | " + (new AlgsC()).Fibonacci(i));
   try {
       Vote.Fibonacci(i);
   } catch (Exception e) {
       //e.printStackTrace();
   }
}

Test.print();
Test.clear();
```

Output

```
##########-------FIBONACCI <-10;0>-------##########
i -> -10 |      :1.0 | 0.0 | 1.0
i -> -9 |      :1.0 | 0.0 | 1.0
i -> -8 |      :1.0 | 0.0 | 1.0
i -> -7 |      :1.0 | 0.0 | 1.0
i -> -6 |      :1.0 | 0.0 | 1.0
i -> -5 |      :1.0 | 0.0 | 1.0
i -> -4 |      :1.0 | 0.0 | 1.0
i -> -3 |      :1.0 | 0.0 | 1.0
i -> -2 |      :1.0 | 0.0 | 1.0
i -> -1 |      :1.0 | 0.0 | 1.0
i -> 0 |      :1.0 | 0.0 | 1.0
allGood: 0, []
aBad: 0, []
bBad: 11, [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0]
cBad: 0, []
allDifferent: 0, []
```

Przykłady Silnia(factorial)

Zakres < 1 ; 170 >

Dla zakresu liczb 1-12 wszystkie wyniki są takie same. Dla zakresu 13-20 algorytm B daje inne wyniki(ponieważ nie mieści się w zakresie). Dla zakresu 21-65 wszystkie algorytmy dają inne wyniki(C kończy się mieścić w zakresie). Dla zakresu 66-170 algorytm A daje inne wyniki. Co ciekawe, są one poprawne, a algorytmy B oraz C oba zwracają 0, i stanowią większość.

Kod

```
        for(int i = 1; i<=170; ++i){
            try {
                Vote.Factorial(i);
            } catch (Exception e) {
                //e.printStackTrace();
            }
        }

        System.out.println("\n##########-------FACTORIAL<1;170>-------##########");
        Test.print();
        Test.clear();
        //wartości graniczne
        System.out.println("-------Wartości graniczne-------");
        values.clear();

values.add(1);values.add(2);values.add(165);values.add(169);values.add(170);values.add(171);values.add(172);
        for(Integer i : values){
            System.out.println("i -> " +i + " |      :" + (new AlgsA()).Factorial(i) + " | " +  (new
AlgsB()).Factorial(i) + " | " + (new AlgsC()).Factorial(i));
        }

        //kilka różnic
        System.out.println("-------Różnice-------");
        values.clear();
//         /*allGood*/values.add(1);values.add(12);
        /*bBad*/values.add(13);values.add(20);
        /*allDifferent*/values.add(21);values.add(22);values.add(23);values.add(64);values.add(65);
        /*aBad*/values.add(66);values.add(70);

        for(Integer i : values){
            System.out.println("i -> " +i + " |      :" + (new AlgsA()).Factorial(i) + " | " +  (new
AlgsB()).Factorial(i) + " | " + (new AlgsC()).Factorial(i));
        }
```

Output

```
##########-------FACTORIAL<1;170>-------##########
allGood: 12, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
aBad: 105, [66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162,
163, 164, 165, 166, 167, 168, 169, 170]
bBad: 8, [13, 14, 15, 16, 17, 18, 19, 20]
cBad: 0, []
allDifferent: 45, [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65]
```

Wartości graniczne, różnice

Dla wizualizacji wyników zostały wypisane wartości graniczne oraz różnice.

```
-------Wartości graniczne-------
i -> 1 |      :1.0 | 1.0 | 1.0
i -> 2 |      :2.0 | 2.0 | 2.0
i -> 165 |      :5.4239106661315 86E295 | 0.0 | 0.0
i -> 169 |      :4.2690680900470 27E304 | 0.0 | 0.0
i -> 170 |      :7.2574156153079 94E306 | 0.0 | 0.0
i -> 171 |      :Infinity | 0.0 | 0.0
i -> 172 |      :Infinity | 0.0 | 0.0
-------Różnice-------
i -> 13 |      :6.2270208E9 | 1.932053504E9 | 6.2270208E9
i -> 20 |      :2.43290200817664E18 | -2.102132736E9 | 2.43290200817664E18
i -> 21 |      :5.109094217170944E19 | -1.195114496E9 | -4.2492900494192148E18
i -> 22 |      :1.1240007277776077E21 | -5.22715136E8 | -1.25066071867496858E18
i -> 23 |      :2.585201673888498E22 | 8.6245376E8 | 8.128291617894826E18
i -> 64 |      :1.2688693218588417E89 | 0.0 | -9.223372036854776E18
i -> 65 |      :8.247650592082472E90 | 0.0 | -9.223372036854776E18
i -> 66 |      :5.443449390774431E92 | 0.0 | 0.0
i -> 70 |      :1.197857166996989E100 | 0.0 | 0.0
```

Zakres < -10 ; 0 >

Żaden z algorytmów nie został wyposażony w mechanizm odrzucający niepoprawne dane wejściowe, tj. np. liczby ujemne. W związku z tym, wszystkie algorytmy zwracają w wyniku wartość 1.0, która została uznana przez program jako najbardziej prawdopodobna.

Kod

```
System.out.println("\n##########-------FACTORIAL<-10;0>-------##########");

for(int i =-10; i<=0; ++i){
System.out.println("i -> " +i + " |      :" + (new AlgsA()).Factorial(i) + " | " +  (new AlgsB()).Factorial(i) + "
| " + (new AlgsC()).Factorial(i));
   try {
       Vote.Factorial(i);
   } catch (Exception e) {
       //e.printStackTrace();
   }
}

Test.print()
```

Output

```
##########-------FACTORIAL<-10;0>-------##########
i -> -10 |      :1.0 | 1.0 | 1.0
i -> -9 |      :1.0 | 1.0 | 1.0
i -> -8 |      :1.0 | 1.0 | 1.0
i -> -7 |      :1.0 | 1.0 | 1.0
i -> -6 |      :1.0 | 1.0 | 1.0
i -> -5 |      :1.0 | 1.0 | 1.0
i -> -4 |      :1.0 | 1.0 | 1.0
i -> -3 |      :1.0 | 1.0 | 1.0
i -> -2 |      :1.0 | 1.0 | 1.0
i -> -1 |      :1.0 | 1.0 | 1.0
i -> 0 |      :1.0 | 1.0 | 1.0
allGood: 11, [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0]
aBad: 0, []
bBad: 0, []
cBad: 0, []
allDifferent: 0, []
```

Wnioski

Programowanie wielowersyjne jest zdecydowanie bardziej kosztownym podejściem, ponieważ trzeba opłacić kilka zespołów programistycznych, aby robiły to samo zadanie. Może być z powodzeniem stosowane w branżach, które wymagają mniejszej tolerancji na błędy, jak medycyna(od programu zależy ludzkie życie) i astronautyka(wysokie koszty misji kosmicznych). Nie zeruje możliwości błędu, ponieważ w pewnych przypadkach, dwie osoby mogą zrealizować program mniej optymalnie niż trzecia. Jak można zauważyć na wcześniej opisanych przykładach, zdarzały się sytuacje, gdzie dwa algorytmy stanowiące większość zwracały niepoprawny wynik w przeciwieństwie do pozostałego. Głosowanie porównuje ze sobą wyniki wszystkich algorytmów oraz uznaje za najbardziej prawdopodobne te, których jest najwięcej.