```
Script started on 2019-01-29 16:20:37-0800
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# ls -1
\033[0m\033[01;32mAPInt.java\033[0m
\033[01;32mAPRat.java\033[0m
\033[01;32mNoteToGrader.txt\033[0m
\033[01;32mREADME.txt\033[0m
\033[01;32mdemo.java\033[0m
\033[01;32mpa1submissionfile.txt\033[0m
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# \033[2Pexit\033[3Ppw
d\033[1APRat.java\033[C\033[C\033[C\033[C\033[C\033[6PNoteToGrader.txt\033[C\033[C\033[C\033[C\033
[C\033[C
...\CMPS101S18PA1

README.txt- a short file which lists all the files in the directory and describes what they
 are.

NoteToGrader.txt - a short note in which you describe your approach.
                     It also shows a commit log in where I stored
                      my programs in a remote server on gitLab.

APInt.java - The APInt class which represents an arbitary precision integer.

APRat.java - The APRat class which represents an arbitrary precision rational.

demo.java - The test class which prints out both the test cases and 1000!.
           Cases were checked with:
           http://www.javascripter.net/math/calculators/100digitbigintcalculator.htm
                    https://www.quora.com/What-is-the-factorial-of-1000

BigFactorial.txt - A text file with the exact solution to 1000!

\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# cat README.txt\033
\033[2Pexit\033[3Ppwd\033[1APRat.java\033[C\033[C\033[C\033[C\033[C\033[6PAPInt.java\033[6PNote
ToGrader.txt\033[C

Approach:
My intial approach to the problem involved implementing a LinkedList inorder to store the p
ositional digits of my ADTs.
This is due to the fact that, I realized my methods of structuring the ADT and performing o
perations on them would involve
many insertions in both the first and last element. A LinkedList's dynamic structure allowe
d this possibility without having to recreate
and copy the elements like an arraylist would. As a result, this would optimize the running
 time of creating and performing arithmetic operations
on my ADTs and offered more fluidity in terms of how I perform my operations.

In terms developing the methods for my ADTs I used standard arithmetic conventions used by
an anolog. In terms of the division method, '
I used exhaustive subtraction algorithm to determine the positional digits of the quotient.
 I also used Euclid's algorithm for finding the GCD
of my APRat, applying my own modulus and divison methods, in order to reduce my fractions.

Many of the answers were checked with the following online Integer calculator:
http://www.javascripter.net/math/calculators/100digitbigintcalculator.htm


Commit History:
Jeffrey@LAPTOP-52K1L0AJ MINGW64 ~/Desktop/CMPS101S18PA (master)
commit 5f301923cdb65c1e587b888d56bc8506ffb4dc6c (HEAD -> master, origin/master)
Author: jwang358 <jwang358@ucsc.edu>
Date:   Tue Jan 29 12:41:21 2019 -0800

    Commented and debugged everything

commit ce89d71454882c0b83eaa70f5bbceab0889b5430
```

```
Author: jwang358 <jwang358@ucsc.edu>
Date:   Thu Jan 24 18:42:30 2019 -0800

    Got Add, Subtract, and Multiply to work

commit ede338939ea378152ff8b3cca18290757af9f0f7
Author: jwang358 <jwang358@ucsc.edu>
Date:   Thu Jan 24 00:49:16 2019 -0800

    Debugged and compiled APInt and APRat

commit 4858ab1cba9fac11c7957ce46ceb814379fa2174
Author: jwang358 <jwang358@ucsc.edu>
Date:   Wed Jan 23 23:34:56 2019 -0800

    Finished APInt and APRat

commit 4c6bb5ad41cb4d6b5d9c7ffff52ac7227aa4d0b8
Author: jwang358 <jwang358@ucsc.edu>
Date:   Wed Jan 23 01:51:46 2019 -0800

    Finshed raw code for APint

commit 3616121a444362eee559b4bb99e0de5ece510e48
Author: jwang358 <jwang358@ucsc.edu>
Date:   Sat Jan 12 13:53:48 2019 -0800

    Updated LinkedList skeletal structure for ADT and worked on add function

commit b27879c5e68bd10228c0a4fa9d7d066173ecffe3 (testing)
Author: jwang358 <jwang358@ucsc.edu>
Date:   Sat Jan 12 11:46:06 2019 -0800

    Added skeleton Structure For Storing 'Arbitrary Integer Precision Types' inLinkedList

commit 9253c9db5fa8786464cd4a6a96bc4aa519f9d23e (test.idea)
Author: jwang358 <jwang358@ucsc.edu>
Date:   Thu Jan 10 13:30:14 2019 -0800

    Created Initial file structure \n
(END)
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# cat NoteToGrader
EADME.txt\033[C\033[C\033[C\033[C\033[C\033[C\033[2Pext\033[3Pjava\033[1PAPRat.java\033[C\033[C\033[C\033[C\033[C\033
\033[C\033[C\033[C\033[6PAPInt.java\033[C

//Programmer: Jeffrey Wang
//CruzID: 1659820
//Data: 11.29.19
//Class: COMPS-101B (D.Bailey)

/**************************************************************************************
Programming Assignment 1: APInt Class (Abstract Data Type)
An arbitrary precision Integer which has no fixed limit to the size of
the number. It implements a LinkedList where the nodes designates the positional
value of the digits. It contains the following methods

â\200¢a default constructorâ\200¢a constructor which uses a string, made up of optional{+,-
}
followed bya string of characters from{0,1,2,3,4,5,6,7,8,9}as an input argument.
â\200¢a constructor for conversion of ints.â\200¢a constructor for conversion of reals that
 truncates the fractional part.
â\200¢a method for printing.
â\200¢methods for addition, subtraction, multiplication and division.
**************************************************************************************/

public class APInt
{
        private Node<Integer> head = new Node<>(null);  //Represents the head node and inti
alizes at null
        private Node<Integer> tail = new Node<>(null);  //Represents the tail node and inti
alizes at null
        private Node<Integer> current = tail;                   //Default pointer for curre
nt it the tail node
        private int defaultDigits = 1;                                  //Set default size
of each null to one digit
        private int realSign, sign = 1;                         //(+ is a 1, - is a -1)

        /**
        * No-Arg Constructor: Creates an empty APInt type where
        * there is no positional digits, the head and tail of the Linked List
        * points to null.
        */
        public APInt()
        {
                head.next = tail;
                tail.previous = head;
        }

        /**
        * APInt Constructor: Creates an APInt type out of a string where the first
        * character of the string can either denote the sign of an Integer or the
        * first positional digit.
        * @param number - A string of an optional sign (+/-) and numeric digits (0-9)
        */
        public APInt(String number)             //Needs Revision (Sign is optional)
        {
                //Determines whether the first character contains a sign
                if(number.charAt(0) == '+' || number.charAt(0) == '-')
                {
                        //Checks and changes sign if the first character is a negative (-)
                        if(number.charAt(0)== '-')
                        {
                                sign = -1;
                        }
                        //Take the substring of digits after sign character
                        number = number.substring(1);
                }

                //Intialize a character array preparing to stored
                char[] digits = number.toCharArray();

                //Intialize a Node which stores values from digits array
                Node<Integer> temp;

                //Set pointers for heads and tails
                head.next = tail;
                tail.previous = head;

                //Starting from the first numeric character, store each array element into
LinkedList
                //And set pointers
                for(Character d: digits)
                {
                        temp = tail;
                        tail = new Node<>(Character.getNumericValue(d));
                        temp.next = tail;
                        tail.previous = temp;
                }

                //Remove first two nodes which point to null
                //And add a null to the tail for tranversal condition
                removeFirst();
                removeFirst();
                addLast(null);
        }
        /**
        * APInt Constructor: Creates an APInt type by taking in an int type and extracting
each
        * positional digit of the int.
        * @param integer - An int type
        */
        public APInt(int integer)
        {
                //Change the sign of both the default APInt sign and the integer by mulitip
lying one
                if(integer < 0)
                {
                        changeSign();
                        integer *= -1;
                }

                //Intialize a Node which stores values each positional value of integer
                Node<Integer> temp;

                //Set pointers for heads and tails
                head.next = tail;
                tail.previous = head;

                //Continually tranversal through the positional digits of integer by taking
 the modulus of
                //Integer by 10 until temp reachs the last digit
                do
                {
                        temp = head;
                        head = new Node<>(integer % 10);
                        temp.previous = head;
                        head.next = temp;
                        integer /= 10;
                }
                while(integer != 0);

                //Remove additional null
                removeLast();
        }

        /**
        * APInt Constructor: Creates an APInt type by taking in an double type and extracti
ng each
        * positional digit of the int. Note that the precision digits are removed
        * @param realNum - An double type
        */
        public APInt(double realNum)
        {
                int altInt = (int) realNum;     //Remove precision digits

                //Change the sign of both the default APInt sign and the double by mulitipl
ying one
                if (altInt < 0)
                {
                        changeSign();
                        altInt *= -1;
                }

                //Intialize a Node which stores values each positional value of realNum
                Node<Integer> temp;

                //Set pointers for heads and tails
                head.next = tail;
                tail.previous = head;

                //Continually tranversal through the positional digits of realNum by taking
 the modulus of
```

```
                //RealNum by 10 until temp reachs the last digit
                do
                {
                        temp = head;
                        head = new Node<>(altInt % 10);
                        head.next = temp;
                        temp.previous = head;
                        altInt /= 10;
                }
                while(altInt != 0);

                //Remove additional null
                removeLast();
        }

        /**
        * APInt Constructor: Creates a deep copy of APInt through copying the value and eac
h node of the original
        * APInt type.
        * @param original - An APInt type that needs to be copied.
        */
        public APInt(APInt original)
        {
                //Changes the default sign of APInt if original's sign is negative
                if(original.getSign() == -1)
                        changeSign();

                //Set's original current node to head for transveral.

                original.setCurrent(0);

                //Intialize a Node which stores values each positional value of original
                Node<Integer> temp;

                //Set pointers for heads and tails
                head.next = tail;
                tail.previous = head;

                //Traverse through original and update tail of this APInt
                do
                {
                        temp = tail;
                        tail = new Node<>(original.getCurrent());
                        temp.next = tail;
                        tail.previous = temp;
                        original.nextCurrent();         //Updates original's current node
                }
                while(original.getCurrent() != null);

                //Remove first two nodes which point to null
                //And add a null to the tail for tranversal condition
                removeFirst();
                removeFirst();
                addLast(null);

        }
        /**
        * getFirst: returns the value of the head node
        * @return head.value -- An int of the first positional value
        */
        public Integer getFirst()
        {
                return head.value;
        }

        /**
        * getLast: returns the value of the tail node
        * @return tail.value -- An int of the last positional value
        */
        public Integer getLast()
        {
                return tail.value;
        }

        /**
        * addFirst: Adds a value to the new head node, and updates pointer to
        * have the odd head node be the next node.
        * @param digit -- An integer value
        */
        public void addFirst(Integer digit)
        {
                Node<Integer> temp = head;                //Temporary node to hold the addr
ess for the old head.
                head = new Node<Integer>(digit);
                head.next = temp;
                temp.previous = head;
        }

        /**
        * addFirst: Adds a value to the new tail node, and updates pointer to
        * have the odd tail node be the previous node.
        * @param digit -- An integer value
        */
        public void addLast(Integer digit)
        {

                Node<Integer> temp = tail;               //Temporary node to hold the addre
ss for the old teil.
                tail = new Node<Integer>(digit);
                tail.previous = temp;
                temp.next = tail;
        }

        /**
        * removeFirst: Removes the head node and
        * reassigns the next node to be the new head
        */
        public void removeFirst()
        {
                head = head.next;
                head.previous = null;
        }

        /**
        * removeLast: Removes the tail node and
        * reassigns the previous node to be the new tail
        */
        public void removeLast()
        {
                tail = tail.previous;
                tail.next = null;
        }

        /**
        * setCurrent: Sets the current node to a specific index on the LinkedList
        * @param index - An int that designates which node should be set to current
        */
        public void setCurrent(int index)
        {
                //set current to head if index = 0
                if(index == 0)
                        current = head;

                //set current to tail if index = -1
                else if(index == -1)
                        current = tail;
                //Find designated node and set it to currrent
                else
                {
                        Node<Integer> temp = head;
                        for(int i = 1; i <= index; i++)
                                temp = temp.next;
                        current = temp;
                }
        }

        /**
        * getCurrent: returns the value of the current node
        * @return current.value: The int value of current node
        * @return null: Returns null if current value is null
        */
        public Integer getCurrent()
        {
                if(current.value != null)
                        return current.value;
                return null;
        }

        /**prevCurrent: Assigns current to the previous node*/
        public void prevCurrent()
        {
                current = current.previous;
        }

        /**nextCurrent: Assigns current to the next node*/
        public void nextCurrent()
        {
                current = current.next;
        }

        /**
        * Set’s sign to a specific int
        * @param i - An integer
        */
        private void setSign(int i)
        {
                sign = i;
        }

        /**changeSign: changes sign by mulitiplying 1*/
        public void changeSign()
        {
                sign *= -1;
        }

        /**
        * getSign: Returns sign
        * @return sign - An integer designating the sign of APInt
        */
        public int getSign()
        {
                return sign;
        }

        /**
        * add: Adds the positional values of APInt types. It also
        * takes into account carryover and special cases such as the sign
        * @param addend - An APInt type which is the addend of the add method
        * @return sum - An APInt type that is representative of the sum
        */
        public APInt add(APInt addend)
        {
                //Performs an additional if addend’s sign matches to this APInt
                if (sign == addend.getSign())
                {
                        //Intialize sum as an APInt type
                        APInt sum = new APInt();

                        //Intialize a pointer to larger or smaller APInt
                        APInt biggerAddend;
                        APInt smallerAddend;

                        //Assigns addend to the biggerAddend if its bigger than this APInt
(v.v.)
                        if(compareTo(addend) < 0)
                        {
                                biggerAddend = addend;
                                smallerAddend = this;
                        }
                        else
                        {
                                biggerAddend = this;
                                smallerAddend = addend;
                        }

                        //Set the bigger and smaller addend to previous node of the tail.
                        //Since the tail's value is null
                        biggerAddend.setCurrent(-1);
                        smallerAddend.setCurrent(-1);
                        biggerAddend.prevCurrent();
                        smallerAddend.prevCurrent();


                        int temp = 0;                           //Set addition carry over t
o 0
                        int limit = (int) Math.pow(10,defaultDigits);   //Set the size of e
ach node to be one digit

                        //Add a null value to the head of both addends to signal
                        //When the addition of each positional digit is no longer needed.
                        smallerAddend.addFirst(null);
                        biggerAddend.addFirst(null);

                        //Adds each positional digit from smallerAddend to bigger addend
                        while(smallerAddend.getCurrent() != null)
                        {
                                //Takes the sum of each positional column
                                int total = biggerAddend.getCurrent() + smallerAddend.getCu
rrent() + temp;

                                temp = 0;       //Reassigns carryover to zero

                                //If total contains more than 1 digit than take the leading
 digits and assign it to
                                //Temp as carryover.
                                if(total >= limit)
                                {
                                        temp = total / limit;
                                        total %= limit;
                                }

                                //Add total to the sum and update both addends
                                sum.addFirst(total);
                                biggerAddend.prevCurrent();
                                smallerAddend.prevCurrent();
                        }

                        //Remove null value from head
                        smallerAddend.removeFirst();

                        //Input the remaining digits including the carryover
                        while(biggerAddend.getCurrent() != null)
                        {
                                sum.addFirst(biggerAddend.getCurrent() + temp);
                                temp = 0;
                                biggerAddend.prevCurrent();
                        }

                        //Input last carryover if it is not 0
                        if(temp != 0)
                                sum.addFirst(temp);

                        //Remove null value from head
                        biggerAddend.removeFirst();

                        //Set the sign of the sum as the sign of this addend
                        sum.setSign(sign);
                        return sum;
                }
                else
                {
                        //Apply a subtract method by changing the sign of a copy of
                        //Addend, since the signs are different.
                        APInt new_addend = new APInt(addend);
                        new_addend.changeSign();
                        return subtract(new_addend);
                }

        }

        /**
        * subtract: Subtract the positional values of APInt types. It also
        * takes into account carryover and special cases such as the sign
        * @param subtractor - An APInt type which is the subtrahend of the subtrahend metho
d
        * @return difference - An APInt type that is representative of the difference.
        */
        public APInt subtract(APInt subtractor)
        {
                if (sign == subtractor.getSign())
                {
                        //Intialize difference as an APInt type
                        APInt diff = new APInt();
                        //Create a pointer to the minuend end and subtrahend.
                        APInt minuend, subtrahend;

                        //Assign subtractor to minuend is bigger and reverse the sign of mi
nuend
                        //Which will later be used to set diff's sign.
                        if(compareTo(subtractor) < 0)
                        {
                                minuend = new APInt(subtractor);
                                subtrahend = new APInt(this);
                                minuend.changeSign();
                        }
                        else
                        {
                                minuend = new APInt(this);
                                subtrahend = new APInt(subtractor);
                        }

                        //Set the minuend and subtrahend to previous node of the tail.
                        //Since the tail's value is null
                        minuend.setCurrent(-1);
                        subtrahend.setCurrent(-1);
                        minuend.prevCurrent();
                        subtrahend.prevCurrent();

                        int temp = 0;           //Set the subtraction carry over to 0
                        int limit = (int) Math.pow(10, defaultDigits); //Set the size of ea
ch node to be one digit

                        //Add a null value to the head of both the minuhead and subtrahend
to signal
                        //When the subtraction of each positional digit is no longer needed
.
                        subtrahend.addFirst(null);
                        minuend.addFirst(null);

                        //Subtracts each positional digit from minuend to subtrahend
                        while(subtrahend.getCurrent() != null)
                        {
                                //Takes the difference of each positional column
                                int difference = (minuend.getCurrent() - temp) - subtrahend
.getCurrent();

                                temp = 0;       //Reasigns the carryover to zero

                                //Increments carryover if the difference is less zero
                                if (difference < 0)
                                {
                                        temp++;
                                        //Finds the positive difference by adding the 10
                                        difference += limit;
                                }

                                //Adds the positional differences and updates both the minu
end and
                                //subtrahend
                                diff.addFirst(difference);
                                minuend.prevCurrent();
                                subtrahend.prevCurrent();
                        }

                        //Remove null value from head
                        subtrahend.removeFirst();

                        //Input the remaining digits including the carryover
                        while(minuend.getCurrent() != null)
                        {
                                diff.addFirst(minuend.getCurrent() - temp);
                                temp = 0;
                                minuend.prevCurrent();
                        }

                        //Remove null value from head
                        minuend.removeFirst();

                        diff.setCurrent(1);
                        //Remove's First digit if it is 0 followed by non-zero digits
                        while(diff.getFirst() == 0 && diff.getCurrent() != null)
                        {
                                diff.removeFirst();
                                diff.nextCurrent();
                        }

                        //Set the sign of the difference as minuend's sign
                        diff.setSign(minuend.getSign());
                        return diff;

                }
                else
                {
                        //Apply the add method by changing the sign of a copy of
                        //subtractor, since signs are different.
                        APInt new_subtractor = new APInt(subtractor);
                        new_subtractor.changeSign();
                        return add(new_subtractor);
                }
        }

        /**
        * multiply: Multiplies the positional values of APInt types. It also
        * takes into account carryover and special cases such as the sign.
        * @param factor - An APInt type which is the factor of the multiply method
        * @return product - An APInt type that is representative of the product
        */
        public APInt multiply(APInt factor)
        {
                //Intialize the product as an APInt type of value zero
                APInt product = new APInt(0);
                //Set the size of each node to be one digit
                int position = (int) Math.pow(10, defaultDigits);

                //Set’s this APInt’s and factor’s current node to the previous
                //Node of tail since the tail’s value is null
                setCurrent(-1);
                factor.setCurrent(-1);
                prevCurrent();
                factor.prevCurrent();

                int carryOver;          //Intialize an int type carryover.
                //Represents a APInt of zero to correct place each tempPlaceHolder
                APInt placeHolder = new APInt();

                //Add a null value to the head of both t this APInt and factor to signal
                //When the multiply method of each positional digit is no longer needed.
                addFirst(null);
                factor.addFirst(null);

                //Adds up all the tempPlaceholder APInts that represent standard
                //multi-digit multiplication.
                while(getCurrent() != null)
                {
                        //Intialize an APInt representative of a poistional multiplying ano
ther APInt
                        APInt tempPlaceHolder = new APInt();

                        //Reset factor’s current node to the previous node of its tail
                        factor.setCurrent(-1);
                        factor.prevCurrent();

                        //Reset the carryover to 0
                        carryOver = 0;

                        //Creates a tempPlaceHolder where a positional digit from this APIn
t
                        //Is multiplied with a positional digit from factor
                        while(factor.getCurrent() != null)
                        {
                                //Takes the product of two positional digits and adds the c
arryover
                                int dig = (getCurrent() * factor.getCurrent()) + carryOver;
                                carryOver = 0;   //Reset the carryover to 0

                                //Adds digs to tempPlaceHolder and updates carryover if
                                //digs is greater than 10
                                if(dig > position)
                                {
                                        tempPlaceHolder.addFirst(dig % position);
                                        carryOver = dig/position;
                                }
                                else
                                        tempPlaceHolder.addFirst(dig);
                                //Updates factor
                                factor.prevCurrent();;
                        }
                        //Adds carryover to tempHolder if it is nonzero
                        if(carryOver != 0)
                                tempPlaceHolder.addFirst(carryOver);
                        tempPlaceHolder.removeLast();

                        //Sets placeholder to the head
                        placeHolder.setCurrent(0);

                        //Removes null from the tail of tempPlaceHolder for input
                        tempPlaceHolder.removeLast();

                        //Adds neccessary zeros before addition of tempPlaceHoldders
                        while(placeHolder.getCurrent() != null)
                        {
                                tempPlaceHolder.addLast(0);
                                placeHolder.nextCurrent();
                        }
                        //Reassigns tail to null
                        tempPlaceHolder.addLast(null);

                        //Adds tempPlaceHolder to product
                        product = product.add(tempPlaceHolder);
                        product.removeLast();

                        //Increase the amount of 0s in placeholder by 1
                        placeHolder.addFirst(0);

                        placeHolder.addLast(0);
                        //Updates this APInt’s current
                        prevCurrent();
                }
                //Remove null value from head from both APInts
                factor.removeFirst();
                removeFirst();

                //Change the sign of product to negative if signs don’t match.
                if(getSign() != factor.getSign())
                        product.changeSign();

                return product;
        }

        /**
        * divide: Divides the positional values of APInt types. It also
        * takes into account carryover and special cases such as the sign.
        * @param divisor - An APInt type which is the divisor of the divide method
        * @return quotient - An APInt type that is representative of the quotient
        */
        public APInt divide(APInt divisor)
        {
                //Intialize the quotient
                APInt quotient = new APInt();
                APInt identity = new APInt(1);

                //If divisor is 0 return a null quotient (undefined)
                if(divisor.getFirst() == 0)
                        return quotient;
                //If divisor is 1 return copy of dividend
                else if (divisor.compareTo(identity) == 0)
                        return new APInt(this);
                //Returns quotient as zero if divisor is greater than this int
                else if (divisor.compareTo(this) > 0)
                {
                        quotient.addFirst(0);
                        return quotient;
                }
                //Returns a quotient as 1 if divisor is equal to this APInt
                else if(divisor.compareTo(this) == 0)
                        return new APInt(1);

                //Intialize a copy of this APInt and divisor
                APInt dynamicDividend = new APInt(this);
                APInt dynamicDivisor = new APInt(divisor);

                //Set dividend’s and divisor’s current node as the head.
                dynamicDividend.setCurrent(0);
                dynamicDivisor.setCurrent(0);
                //Intialize a partition of the dividend the same size of divisor
                APInt dynamic = new APInt();

                //Add the positional digits from dividend to dynamic
                while(dynamicDivisor.getCurrent() != null)
                {
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamicDividend.nextCurrent();
                        dynamicDivisor.nextCurrent();

                }

                //Add an additional null value as tail.
                dynamicDividend.addLast(null);
                //Add an additional digit from dividend to dynamic if its still
                //less than divisor
                if(dynamic.compareTo(dynamicDivisor) < 0)
                {
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamicDividend.nextCurrent();
                }

                //Remove null values from head and add null value to tail
                dynamic.removeFirst();
                dynamic.removeFirst();
                dynamic.addLast(null);

                //change sign of dynamic if it is different from divisor
                if(dynamicDivisor.getSign() != dynamic.getSign())
                        dynamic.changeSign();

                //Set the current node of dynamic to head.
                dynamic.setCurrent(0);


                //Continously subtract divisor from dynamic and count the number of
                //Times it can be subtracted from dynamic till it becomes greater.
                //The result is represented as the quotient
                while(dynamicDividend.getCurrent() != null)
                {
                        //Remove’s First digit if it is 0 followed by non-zero digits
                        dynamic.setCurrent(1);
                        while(dynamic.getFirst() == 0 && dynamic.getCurrent() != null)
                        {
                                dynamic.removeFirst();
                                dynamic.nextCurrent();
                        }

                        int count = 0; //Set quotient of dynamic and divisor to zero
                        //Update count till dynamic is less than divisor
                        while(dynamic.compareTo(divisor) >= 0)
                        {
                                dynamic = dynamic.subtract(dynamicDivisor);
                                count++;
                        }

                        //Remove null tail of dynamic if count is not zero.
                        if(count != 0)
                                dynamic.removeLast();

                        //Update the previous node of tail to the next digit
                        //Of next dynamic digit
                        dynamic.removeLast();
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamic.addLast(null);

                        //Add count to quotient
                        quotient.addLast(count);

                        //Update
                        dynamicDividend.nextCurrent();
                        count = 0;
                }

                //Add and additional zero if dynamic contains a leading zero,
                //And the compareTo method states that it is still bigger.
                if(dynamic.compareTo(divisor) > 0 && dynamic.getFirst() == 0)
                        quotient.addLast(0);

                //Update the last count if dynamic is still greater than or equal to
                //divisor
                else if(dynamic.compareTo(divisor) >= 0)
                {
                        int count = 0;
                        while(dynamic.compareTo(divisor) >= 0)
                        {
                                dynamic = dynamic.subtract(dynamicDivisor);
                                count++;
                        }
                        dynamic.removeLast();
                        dynamic.removeLast();
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamic.addLast(null);
                        if(count != 0)
                                quotient.addLast(count);
                }

                //Remove null values from head and add null value to the tail
                quotient.removeFirst();
                quotient.removeFirst();
                quotient.addLast(null);

                //Change the sign of quotient if the dividend and divisor signs are
                //Different
                if(sign != divisor.getSign())
                        quotient.changeSign();
                return quotient;
        }

        /**
        * getRemainder: This is modulus method in which the remainder is returned. This is
        * similar to the divide method except the remainder (dynamic is returned).
        * @param divisor - An APInt type which is the divisor of the getRemainder method
        * @return dynamic - An APInt type that is representative of the remainder
        */
        public APInt getRemainder(APInt divisor)
        {
                APInt remainder = new APInt();

                //Return's Remainder 0 if modulus is 1
                if(divisor.compareTo(new APInt(1)) == 0)
                        return new APInt(0);

                //Returns remainder as 0 if divisor is 0
                if(divisor.compareTo(new APInt(0)) == 0)
                {
                        return new APInt(0);
                }

                //Returns the remainder as 1 if the divisor is equal to this APInt
                if (divisor.compareTo(this) == 0)
                {
                        remainder.addFirst(0);
                        return remainder;
                }

                //Intialize a copy of this APInt and divisor
                APInt dynamicDividend = new APInt(this);
                APInt dynamicDivisor = new APInt(divisor);

                //Set dividend’s and divisor’s current node as the head.
                dynamicDividend.setCurrent(0);
                dynamicDivisor.setCurrent(0);
                //Intialize a partition of the dividend the same size of divisor
                APInt dynamic = new APInt();

                //Add the positional digits from dividend to dynamic
                while(dynamicDivisor.getCurrent() != null)
                {
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamicDividend.nextCurrent();
                        dynamicDivisor.nextCurrent();
                }

                dynamicDividend.addLast(null);
                if(dynamic.compareTo(dynamicDivisor) < 0)
                {
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamicDividend.nextCurrent();
                }
                //Add an additional null value as tail.
                dynamic.removeFirst();
                dynamic.removeFirst();
                dynamic.addLast(null);

                //Add an additional digit from dividend to dynamic if its still
                //less than divisor
                if(dynamicDivisor.getSign() != dynamic.getSign())
                        dynamic.changeSign();

                //Set the current node of dynamic to head.
                dynamic.setCurrent(0);
                int count = 0; //Set quotient of dynamic and divisor to zero

                //Continously subtract divisor from dynamic and count the number of
                //Times it can be subtracted from dynamic till it becomes greater.
                //The result is represented as the quotient
                while(dynamicDividend.getCurrent() != null)
                {
                        //Remove’s First digit if it is 0 followed by non-zero digits
                        dynamic.setCurrent(1);
                        while(dynamic.getFirst() == 0 && dynamic.getCurrent() != null)
                        {
                                dynamic.removeFirst();
                                dynamic.nextCurrent();
                        }

                        //Update count till dynamic is less than divisor
                        while(dynamic.compareTo(divisor) > 0)
                        {
                                dynamic = dynamic.subtract(dynamicDivisor);
                                count++;
                        }
                        //Remove null tail of dynamic if count is not zero.
                        if(count != 0)
                                dynamic.removeLast();

                        //Update the previous node of tail to the next digit
                        //Of next dynamic digit
                        dynamic.removeLast();
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamic.addLast(null);

                        //Update
                        dynamicDividend.nextCurrent();
                        count = 0;
                }

                //Update the last count if dynamic is still greater than or equal to
                //divisor
                if(dynamic.compareTo(divisor) >= 0)
                {
                        while(dynamic.compareTo(divisor) >= 0)
                        {
                                dynamic = dynamic.subtract(dynamicDivisor);
                        }
                        dynamic.removeLast();
                        dynamic.removeLast();
                        dynamic.addLast(dynamicDividend.getCurrent());
                        dynamic.addLast(null);
                }

                //Returns dynamic which represents the remainder
                return dynamic;
        }

        /**
        * compareTo: Displays 1 if this APInt is greater,
        * 0 if it is equal to
        * -1 if it is less than
        * @param logic - The APInt that is being compared to
        @ @return An Integer representing the state of the comparsion
        */
        public int compareTo(APInt logic)
        {
                //Sets the current node of both comparisions to head
                setCurrent(0);
                logic.setCurrent(0);

                //Compares the positional digits, if one has more returns the states.
                while(this.getCurrent() != null || logic.getCurrent() != null)
                {
                        if(this.getCurrent() == null && logic.getCurrent() != null)
                                return -1;
                        else if(this.getCurrent() != null && logic.getCurrent() == null)
                                return 1;
                        nextCurrent();
                        logic.nextCurrent();
                }

                //Sets the current node of both comparisions to head
                setCurrent(0);
                logic.setCurrent(0);

                //Compares the first node if they both have the same positional digits
                while(getCurrent() != null)
                {
                        if(getCurrent() > logic.getCurrent())
                                return 1;
                        else if(getCurrent() < logic.getCurrent())
```

```
                                return -1;
                        nextCurrent();
                        logic.nextCurrent();
                }
                return 0;
        }

        /** toString: Print method for APInt Class
        * @Override toString
        * @return: A string representative of an Integer
        */
        public String toString()
        {
                StringBuilder number = new StringBuilder();
                setCurrent(0);
                if(sign == 1)
                        number.append('+');
                else
                        number.append('-');
                while(getCurrent() != null)
                {
                        number.append(Integer.toString(getCurrent()));
                        nextCurrent();
                }
                return number.toString();
        }

        /**Node<Integer> Represents and Integer Node for LinkedList*/
        private static class Node<Integer>
        {
                Node<Integer> next;                     //Points to the next Node
                Node<Integer> previous;         //Points to the previous Node
                Integer value;                          //Represents the value

                /**Constructor which assigns Integer value to value*/
                public Node(Integer value)
                {
                        this.value = value;
                }
        }
}
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# cat APInt.java
\033[6PREADME\033[C\033[C\033[C\033[9B[C\033[C\033[2P\033[3P\033[1PRat.java\033[C\033[C\033
[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C


//Programmer: Jeffrey Wang
//CruzID: 1659820
//Data: 11.29.19
//Class: COMPS-101B (D.Bailey)

/*******************************************************************************
Programming Assignment 1: APRat Class (Abstract Data Type)
An arbitrary precision Rational which has no fixed limit to the size of
the number. It implements a LinkedList where the nodes designates the positional
value of the digits. It contains the following methods

â\200¢a defaultconstructor
â\200¢a constructor for usingapints to represent the numerator and denomina-tor.
â\200¢a constructor for conversion of a pair of ints.â\200¢a constructor for conversion of
reals to a specified precision.
â\200¢a method for printing.
â\200¢methods for addition, subtraction, multiplication and division.
â\200¢normalize the result of every operation, i.e., reduce the fraction tolowestterms.
*******************************************************************************/
public class APRat
{
        private APInt numerator;        //An APInt numerator
        private APInt denominator;      //An APInt denominator

        /**
        * No-Arg Constructor for APRat
        * Assigns the numerator with an APInt of value 0
        * Assigns the denominator with an APInt of value 1.
        */
        public APRat()
        {
                numerator = new APInt(0);
                denominator = new APInt(1);
        }

        /**
        * APRat Constructor: Intializes an APRAt with APInt inputs
        * @param numerator - An APInt representative of the numerator
        * @param denominator - An APInt representative of the denominator
        */
        public APRat(APInt numerator, APInt denominator)
        {
                this.numerator = numerator;
                this.denominator = denominator;
        }

        /**
        * APRat Constructor: Intializes an APRAt with int inputs
        * @param numerator - An int representative of the numerator
        * @param denominator - An int representative of the denominator
        */
        public APRat(int numerator, int denominator)
        {
                this.numerator = new APInt(numerator);
                this.denominator = new APInt(denominator);
        }

        /**
        * APRat Constructor: Intializes an APRAt with double inputs with a
        * specific precision
        * @param numerator - A double representative of the numerator
        * @param denominator - An double representative of the denominator
        * @param pos - The level of precision of APRat
        */
        public APRat(double numerator, double denominator, int pos)
        {
                this.numerator = new APInt((numerator * Math.pow(10,pos)));
                this.denominator = new APInt(denominator * Math.pow(10,pos));

        }

        /**
        * getNumerator: Returns the APInt numerator
        * @return numerator - The numerator
        */
        public APInt getNumerator()
        {
                return new APInt(numerator);
        }

        /**
        * getNumerator: Returns the APInt denominator
        * @return denominator - The denominator
        */
        public APInt getDenominator()
        {
                return new APInt(denominator);
        }
        /**
        * add: Performs and fractional addition
        * This is done by multiplying each numerators by its opposit denominators.
        * And adding the numerators
        * @param fracAdd - The fractional addend
        * @return An APRat that represents the sum
        */
        public APRat add(APRat fracAdd)
        {
                APInt num2 = fracAdd.getNumerator();
                APInt dem2 = fracAdd.getDenominator();
                APInt newNum = (numerator.multiply(dem2)).add(num2.multiply(denominator));
                APInt newDem = (denominator.multiply(dem2));
                return new APRat(newNum, newDem);
        }

        /**
        * subtract: Performs and fractional subtraction
        * This is done by multiplying each numerators by its opposit denominators.
        * And subtracting the numerators
        * @param fracSubtr - The fractional subtrahend
        * @return An APRat that represents the difference
        */
        public APRat subtract(APRat fracSubtr)
        {
                APInt num2 = fracSubtr.getNumerator();
                APInt dem2 = fracSubtr.getDenominator();
                APInt newNum = (numerator.multiply(dem2)).subtract(num2.multiply(denominato
r));
                APInt newDem = (denominator.multiply(dem2));
                return new APRat(newNum, newDem);

        }

        /**
        * multiply: Performs and fractional multiplication
        * This is done through multiplying both the numerators and denominators
        * @param fracFac - The fractional factor
        * @return An APRat that represents the product
        */
        public APRat multiply(APRat fracFac)
        {
                APInt num = numerator.multiply(fracFac.getNumerator());
                APInt dem = denominator.multiply(fracFac.getDenominator());
                return new APRat(num, dem);
        }

        /**
        * divide: Performs and fractional division
        * This is done by multiplying the reciprocal
        * @param fracDiv - The fractional divisor
        * @return An APRat that represents the quotient
        */
        public APRat divide(APRat fracDiv)
        {
                APInt num = numerator.multiply(fracDiv.getDenominator());
                APInt dem = denominator.multiply(fracDiv.getNumerator());
                return new APRat(num, dem);
        }

        /**
        * normalize: Reduce the fraction to its simpliest form by finding the
        * Greatest Common Multiple(GCM) and dividing it from the numerator and denominator.
        * This is done through using Euclid’s method.
        */
        public void normalize()
        {
                //The following three APInt’s represents the three values in Euclid’s Algor
```

```
ithm
                APInt remainder;
                APInt modulus_a;
                APInt modulus_b;
                APInt dividend;
                APInt empty = new APInt(1);

                // Only normalize if the numerator isn't zero
                if(numerator.getFirst() != 0)
                {
                        //Assign the largest part of the fraction to the dividend
                        if(numerator.compareTo(denominator) > 0)
                        {
                                dividend = numerator;
                                remainder = new APInt(denominator);
                        }
                        else
                        {
                                dividend = denominator;
                                remainder = new APInt(numerator);
                        }

                        //Euclid's method
                        modulus_a = new APInt(remainder);
                        modulus_b = dividend.getRemainder(remainder);
                        remainder = modulus_a.getRemainder(modulus_b);

                        //Continue till remainder is zero
                        while(remainder.getFirst() != 0 )
                        {
                                remainder = modulus_a.getRemainder(modulus_b);
                                modulus_a = new APInt(modulus_b);
                                modulus_b = new APInt(remainder);
                        }

                        //modulus_a represents the Greatest common multiple
                        numerator = numerator.divide(modulus_a);
                        denominator = denominator.divide(modulus_a);
                }
        }

        /** toString: Print method for APInt Class
        * @Override toString
        * @return: A string representative of an Integer
        */
        public String toString()
        {
                normalize();
                return "Numerator: " + numerator.toString() +
                "\nDenominator: " + denominator.toString();
        }
}
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# cat APRat.java
\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[1P
javac APRat.java
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# javac APInt.java
In\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[2Pex
\033[3Pjavac APRat.java
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# ls -1
\033[0m\033[01;32m'APInt$Node.class'\033[0m
\033[01;32mAPInt.class\033[0m
\033[01;32mAPInt.java\033[0m
\033[01;32mAPRat.class\033[0m
\033[01;32mAPRat.java\033[0m
\033[01;32mNoteToGrader.txt\033[0m
```

```
\033[01;32mREADME.txt\033[0m
\033[01;32mdemo.java\033[0m
\033[01;32mpa1submissionfile.txt\033[0m
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# cat demo.java
import java.io.*; //For PrintWriter class

//Programmer: Jeffrey Wang
//CruzID: 1659820
//Data: 11.29.19
//Class: COMPS-101B (D.Bailey)

/*************************************************************************************
Programming Assignment 1: Demo -- Insures that the ADT's are working
correctly is working correctly
It also writes 1000! in a file called BigFactorial.txt
*************************************************************************************/

public class demo
{
        public static void main(String[] args) throws IOException
        {

                APInt num0 = new APInt();
                APInt num1 = new APInt(3141592.65897);
                APInt num2 = new APInt("-2718281828459045235360287471352");
                APInt num2_5 = new APInt("1414213562373095048801688724209698078569671875");
                APInt num3 = new APInt(-123456);


                System.out.println("Representation of No-Arg Constructor for APInt: ");
                System.out.println("num0: " + num0);
                System.out.println();

                System.out.println("Representation of Floating Point Number conversion to A
PInt: ");
                System.out.println("num1:" + num1);
                System.out.println();

                System.out.println("Representation of String Conversion (with sign) to APIn
t: ");
                System.out.println("num2:" + num2);
                System.out.println();

                System.out.println("Representation of String Conversion (without sign) to A
PInt: ");
                System.out.println("num2_5:" + num2_5);
                System.out.println();

                System.out.println("Representation of Integer Conversion: ");
                System.out.println("num3:" + num3);
                System.out.println();

                System.out.println("num1 + num2_5: ");
                System.out.println("Sum: " + num1.add(num2_5));
                System.out.println();

                System.out.println("num1 + num2: ");
                System.out.println("Sum: " + num1.add(num2));
                System.out.println();

                System.out.println("num2 + num2_5: ");
                System.out.println("Sum: " + num2.add(num2_5));
                System.out.println();

                System.out.println("num2 + num3: ");
                System.out.println("Sum: " + num2.add(num3));
                System.out.println();
                System.out.println("num1 - num2_5: ");
                System.out.println("Difference: " + num1.subtract(num2_5));
                System.out.println();

                System.out.println("num1 - num2: ");
                System.out.println("Difference: " + num1.subtract(num2));
                System.out.println();

                System.out.println("num2 - num2_5: ");
                System.out.println("Difference: " + num2.subtract(num2_5));
                System.out.println();

                System.out.println("num2 - num3: ");
                System.out.println("Difference: " + num2.subtract(num3));
                System.out.println();

                System.out.println("num1 * num2_5: ");
                System.out.println("Product: " + num1.multiply(num2_5));
                System.out.println();

                System.out.println("num1 * num2: ");
                System.out.println("Product: " + num1.multiply(num2));
                System.out.println();

                System.out.println("num2 * num2_5: ");
                System.out.println("Product: " + num2.multiply(num2_5));
                System.out.println();

                System.out.println("num2 * num3: ");
                System.out.println("Product: " + num2.multiply(num3));
                System.out.println();

                System.out.println("num2_5 / num1: ");
                System.out.println("Quotient: " + num2_5.divide(num1));
                System.out.println();

                System.out.println("num2 / num1: ");
                System.out.println("Quotient: " + num2.divide(num1));
                System.out.println();

                System.out.println("num2_5 / num2: ");
                System.out.println("Quotient: " + num2_5.divide(num2));
                System.out.println();

                System.out.println("num2 / num3: ");
                System.out.println("Quotient: " + num2.divide(num3));
                System.out.println();

                APRat frac0 = new APRat();
                APRat frac1 = new APRat(100, 50);
                APRat frac2 = new APRat(-3.141592, -1.0, 2);
                APRat frac3 = new APRat(num2, num2_5);


                System.out.println("Representation of No-Arg Constructor for APRat: ");
                System.out.println("frac0:\n  " + frac0);
                System.out.println();

                System.out.println("Representation of Integer conversion to APRat (Normaliz
e 100/50): ");
                System.out.println("frac1:\n " + frac1);
                System.out.println();

                System.out.println("Representation of Floating Point Conversion (precision
2) to APRat: ");
                System.out.println("frac2:\n " + frac2);
                System.out.println();
                System.out.println("Representation of APInt Conversion to APRat: ");
                System.out.println("frac3:\n " + frac3);
                System.out.println();

                APInt newNum = new APInt("16598201897434");
                APInt newDem = new APInt("180028489283563922160785l");
                APRat frac4 = new APRat(newNum, newDem);

                System.out.println("Representation of frac4:");
                System.out.println("frac4:\n " + frac4);
                System.out.println();

                System.out.println("frac3 + frac4");
                System.out.println("Sum:\n" + frac3.add(frac4));
                System.out.println();

                System.out.println("frac3 - frac4");
                System.out.println("Difference:\n" + frac3.subtract(frac4));
                System.out.println();

                System.out.println("frac3 * frac4");
                System.out.println("Product:\n" + frac3.multiply(frac4));
                System.out.println();

                System.out.println("frac3 / frac4");
                System.out.println("Quotient:\n" + frac3.divide(frac4));
                System.out.println();

                PrintWriter bigfactorial = new PrintWriter("BigFactorial.txt");
                //ExtraCredit Problem:
                APInt factorial = new APInt(1);
                for(int i = 2; i <= 1000; i++)
                {
                        factorial = factorial.multiply(new APInt(i));
                }

                //AutoWrap factorial by length of 20 characters
                char[] digits = factorial.toString().toCharArray();
                StringBuilder write = new StringBuilder();
                write.append("This is 1000!:\n");
                int count = 0;
                for(char dig: digits)
                {
                        write.append(dig);
                        count++;
                        if(count > 20)
                        {
                                write.append("\n");
                                count = 0;
                        }
                }
                bigfactorial.println(write.toString());
                bigfactorial.close();
        }
}
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# javac test\033[K
\033[K\033[K\033[Kdemo.java
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# t\033[Kca\033[K\033
[Klst\033[K -1
\033[0m\033[01;32m'APInt$Node.class'\033[0m
\033[01;32mAPInt.class\033[0m
\033[01;32mAPInt.java\033[0m
\033[01;32mAPRat.class\033[0m
\033[01;32mAPRat.java\033[0m
\033[01;32mNoteToGrader.txt\033[0m
```

```
\033[01;32mREADME.txt\033[0m
\033[01;32mdemo.class\033[0m
\033[01;32mdemo.java\033[0m
\033[01;32mpa1submissionfile.txt\033[0m
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# javade\033[K\033[K
 demo
Representation of No-Arg Constructor for APInt:
num0: +

Representation of Floating Point Number conversion to APInt:
num1:+3141592

Representation of String Conversion (with sign) to APInt:
num2:-2718281828459045235360287471352

Representation of String Conversion (without sign) to APInt:
num2_5:+14142135623730950488016887242096980785696718 75

Representation of Integer Conversion:
num3:-123456

num1 + num2_5:
Sum: +1414213562373095048801688724209698078572813467

num1 + num2:
Sum: -2718281828459045235360284329760

num2 + num2_5:
Sum: +1414213562373092330519860265164462718282200523

num2 + num3:
Sum: -2718281828459045235360287594808

num1 - num2_5:
Difference: -1414213562373095048801688724209698078566530283

num1 - num2:
Difference: +2718281828459045235360290612944

num2 - num2_5:
Difference: -1414213562373097767083517183254933438857143227

num2 - num3:
Difference: -2718281828459045235360287347896

num1 * num2_5:
Product: +4442882013842816420554994882467393806049852605125000

num1 * num2:
Product: -8539732446032308839045996237699672384

num2 * num2_5:
Product: -3844231028159116824863671637425339964674857801644174015055505725800102625000

num2 * num3:
Product: +335588201414239888576639650063232512

num2_5 / num1:
Quotient: +450158251731318086117385301531738710363

num2 / num1:
Quotient: -865256159443697728845848

num2_5 / num2:
Quotient: -520260095022888

num2 / num3:
```

```
Quotient: +22018223727150120167187398

Representation of No-Arg Constructor for APRat:
frac0:
  Numerator: +0
Denominator: +1

Representation of Integer conversion to APRat (Normalize 100/50):
frac1:
 Numerator: +2
Denominator: +1

Representation of Floating Point Conversion (precision 2) to APRat:
frac2:
 Numerator: +157
Denominator: +50

Representation of APInt Conversion to APRat:
frac3:
 Numerator: +209098602189157325796945190104
Denominator: -108785658644084234523206824939207544505359375

Representation of frac4:
frac4:
 Numerator: +16598201897434
Denominator: +1800284892835639221607851

frac3 + frac4
Sum:
Numerator: -1805269888665212176089400970610062353847934776707326437246
Denominator: -195845177814119615698954007717163264478664652436966083585174076453125

frac3 - frac4
Difference:
Numerator: +1806022762774480554195432969414684319970433181506794250254
Denominator: -195845177814119615698954007717163264478664652436966083585174076453125

frac3 * frac4
Product:
Numerator: +1156886938535622757074798990195037546597710
Denominator: -65281725938039871899651335905721088159554884145655361195058025484375

frac3 / frac4
Quotient:
Numerator: +188218527317094526507999701155491530624601199866953250
Denominator: -902823162859923182571208485006186668454591989553530171875

\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# java\033[K\033[K
\033[K\033[Kls -1 \033[K\033[K
\033[0m\033[01;32m'APInt$Node.class'\033[0m
\033[01;32mAPInt.class\033[0m
\033[01;32mAPInt.java\033[0m
\033[01;32mAPRat.class\033[0m
\033[01;32mAPRat.java\033[0m
\033[01;32mBigFactorial.txt\033[0m
\033[01;32mNoteToGrader.txt\033[0m
\033[01;32mREADME.txt\033[0m
\033[01;32mdemo.class\033[0m
\033[01;32mdemo.java\033[0m
\033[01;32mpa1submissionfile.txt\033[0m
\033[0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# cat big\033[K\033[
K\033[Kgi\033[K\033[Kbi\007\033[K\033[KBigFactorial.txt
This is 1000!:
+40238726007709377354
370243392300398571937
486421071463254379991
```

```
042993851239862902059
```

```
243846565724501440282
188525247093519062092
902313649327349756551
395872055965422874977
401141334696271542284
586237738753823048386
568897646192738381490
014076731044664025989
949022222176590433990
188601856652648506179
970235619389701786004
081188972991831102117
122984590164192106888
438712185564612496079
872290851929681937238
864261483965738229112
312502418664935314397
013742853192664987533
721894069428143411852
015801412334482801505
139969429015348307764
456909907315243327828
826986460278986432113
908350621709500259738
986355427719674282224
875758676575234422020
757363056949882508796
892816275384886339690
995982628095612145099
487170124451646126037
902930912088908694202
851064018215439945715
680594187274899809425
474217358240106367740
459574178516082923013
535808184009699637252
423056085590370062427
124341690900415369010
593398383577793941097
002775347200000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
000000000000000000000
0000000
\033]0;root@LAPTOP-52K1L0AJ: /mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1\007roo
t@LAPTOP-52K1L0AJ:/mnt/c/Users/Jeffrey/Desktop/CMPS101S18PA/CMPS101S18PA1# exit\033[K
exit

Script done on 2019-01-29 16:23:37-0800
```