PRESTIDIGITATION

BIGINTEGER

Consider the Digits interface shown below. This class represents a single *non-negative integer number*. The meaning of each method is shown below the interface.

```
public interface Digits extends Comparable<Digits> {
   public Digits add(Digits other) throws IllegalArgumentException;
   public Digits mul(Digits other) throws IllegalArgumentException;
   public Digits pow(Digits other) throws IllegalArgumentException;
   public Iterator<Integer> getDigits( boolean leastFirst );
   public int numberOfDigits();
   public String toString();
   public boolean equals(Object other);
   public boolean isZero();
   public boolean isOne();
   public Digits half();
   public boolean isEven();
}
```

- **Digits add(Digits other):** This method constructs a Digits which is the sum of the active Digits and the other. The method must not modify either of its operands (the active Digits or the other). The method throws IllegalArgumentException if the other is null.
- **Digits mul(Digits other):** This method constructs a Digits which is the product of the active Digits and the other. The method must not modify either of its operands (the active Digits or the other). In addition, note that this method must be efficient in the sense that you must not simply call the "add" method repeatedly. This method throws an IllegalArgumentException if the other is null;
- **Digits pow(Digits other):** This method constructs a Digits that corresponds to the active Digits raised to the power of the other. The method must not modify either of its operands (the active Digits or the other). This method throws an IllegalArgumentException if the other is null. This method throws an UndefinedValueException if both Digits are zero.
- **int numberOfDigits():** This method returns the number of digits in the Digits object.
- **String toString():** This method returns the textual representation of the Digits object as a sequence of one-or more decimal digits. The string must not contain leading zeros unless the Digits object represents the number zero.
- **boolean equals(Object other):** This method returns true if the other is a Digits object representing the same integer number as the active Digits.
- **boolean isZero():** This method returns true if the Digits object represents the integer zero.
- **boolean isOne():** This method returns true if the Digits object represents the integer one.
- **boolean isEven():** This method returns true if the Digits object represents an even integer.
- **Digits half():** This method constructs a Digits object that represents half of the active object (using integer division). This must must not alter the active Digits object in any way.
- **Iterator<Integer> getDigits(boolean leastFirst):** Returns an iterator over the digits beginning with the least significant if leastFirst is true and beginning with the most significant if leastFirst is false.

PRESTIDIGITATION

You must implement the Digits interface by writing a class named Prestidigitation. Your implementation must have the structure shown in the code below and conform to the following constraints.

1. You must have a private List<Integer> object named 'digits' that contains the data related to the Prestidigitation object. The "digits" list contains a sequence of base-10 digits (each in the range 0-9) with the least significant digit occurring first.
2. There must not be leading zeros in your textual representation (except in the case of representing the value zero itself!).
3. **You must not have any instance variables besides 'digits'**.
4. **Prestidigitation must be immutable**. Once created, they will never change their internal state.
5. **You must not use Strings anywhere except for the toString method and the String constructor**.
6. **You must not use arrays anywhere**.
7. **You must not use any Java class other than List, Iterator, and ArrayList**.

```
public class Prestidigitation implements Digits {
   private List<Integer> digits; // NO OTHER VARIABLES

   public Prestidigitation (int num) throws IllegalArgumentException {…}
   public Prestidigitation (String num) throws NumberFormatException {…}
   public Prestidigitation (long num) throws IllegalArgumentException {…}
   public Prestidigitation (Digits num) throw IllegalArgumentException {…}
   // other methods not shown
}
```

- **Prestidigitation(String num):** This method is a constructor that takes a textual representation of an integer value and creates a Prestidigitation. The textual representation of a Prestidigitation is a sequence of digits 0-9. The sequence will never start with zero unless the sequence is of length 1. This method must throw a NumberFormatException if num is not the textual representation of some Prestidigitation.
- **Prestidigitation (int num):** This method is a constructor that takes an int value and converts it to a Prestidigitation. This method must throw an IllegalArgumentException if num is negative.
- **Prestidigitation (long num):** This method is a constructor that takes a long value and converts it to a Prestidigitation. This method must throw an IllegalArgumentException if num is negative.
- **Prestidigitation (Digits num):** This method is a constructor that takes a Digits and converts it to a Prestidigitaion. This method must throw an IllegalArgumentException if num is null.

NUMBER CRUNCHER

You must write a program named **Cruncher**. Your program will read a 'crunchy' file that contains numeric computations on positive integers. Your program will compute the result of those computations and print the result to the screen.

CRUNCHY FILE FORMAT

'Crunched' files are text file that have exactly one element on each line. Each line will contain either a number or an operator as described below.

- **Number**. A sequence of one-or-more base-10 digits. There is no limit on how many digits may be on a single line. The list of digits will not start with zero unless it is the only digit on the line. The most significant digit will occur first.
- **Operator**. One of the following symbols: +, *, ^

In addition, the lines are structured such that the first two lines are numbers and the third line is an operator. The meaning is that the operator on the third line should be applied to the numbers on the first two lines to produce a result. Each crunchy file will have at least three lines. Zero-or-more line-pairs follow the first three lines. A line-pair is a pair of lines such that the first line of the pair is a number and the second line of the pair is an operator. The meaning is to take the result of the previously completed operation and apply the specified operator to that result and the 'new' number. For example, consider the following crunchy file.

```
6
2
^
10
+
```

This file corresponds to the computational expression ((6 ^ 2) + 10) such that the result is 46.

TESTING

When testing your code, you may also choose to try the following interesting test cases (ones that are likely to illustrate bugs in your code) below.

- `1 + 9999`
- `9999 + 1`
- `0 + 9999`
- `0 * 9999`
- `9999^0`
- `0^9999`
- `0^0`
- `1^9999`
- `99 * 9`

EXAMPLE FILE

Here is another crunchy file; and the expected output is given below the example.

```
53535510
23
^
```

```
57380551577016827745381974259972143729317415293749484100418102719653271044139783118046554680732769737548346303268898512819581519461025244523110090595439151000000000000000000000000
```