The purpose of this homework is to write an implementation of a simple integer arithmetic calculator. The calculator must support the following binary operations: +, -, *, /, % as well as the unary operation -, which we shall call \textit{invert} below. The left operand of any operation always comes from the current result the calculator has; the right operand must be supplied by the user. The calculator carries a history of the last 3 arithmetic operations performed; it also allows us to \textit{undo} them starting from the last one and working backwards. In the undoing process, the result of the calculator should be the value it had before the operation being undone was carried out.

The UI of the calculator presents the user with the current result (which is zero at the start), and provides the user with a menu of operations to carry out:

\begin{verbatim}
add
subtract
multiply
divide
remainder
invert
undo
redo
clear
\end{verbatim}

The first five operations are the usual integer arithmetic operations; the operation \textit{invert} changes the sign of the current result; you can think of those operations as queries; The operation \textit{clear} sets the current result of the calculator to zero and resets the history of operations to empty. The \textit{clear} operation cannot be undone. The operation \textit{redo} redoes the last arithmetic operation carried out having as left operand the current calculator's result. You can think of the last 3 operations as commands. For any of the first 5 operations, the user is prompted for input which will be used as the second operand of the operation requested.

A solution to this problem can be readily provided, by encoding the operations with numbers (1-7); after the user chooses one of those operations, the calculator uses code such as the following to proceed (assuming operation is an int with values 1 - 8):

\begin{verbatim}
switch (operation) {
    case 1: secondOP = requestInput(prompt);
             calResult = calResult + secondOP;
             break;
    case 2: secondOP = requestInput(prompt);
             calResult = calResult - secondOP;
             break;
    case 3: secondOP = requestInput(prompt);
             calResult = calResult * secondOP;
             break;
    case 4: secondOP = requestInput(prompt);
             calResult = calResult / secondOP;
             break;
    case 5: secondOP = requestInput(prompt);
             calResult = calResult % secondOP;
             break;
    case 6: calResult = (-1) * calResult;
             break;+
    case 7: undoOp();
             break;
    case 8: redoOp();
             break;
    case 9: clearOP();
             break;
}\end{verbatim}
The calculator then proceeds to present the answer to the user. This solution can be used to correctly implement the job requested using a case statement; but this solution will be hard to implement for some cases (such as redo) and is hard for testing and in the future for maintenance and for variability and extensibility.

You want to provide a better solution; one such solution is based on abstracting an operator via an interface:

```java
public interface Operator {
    // returns true if operation is binary.
    public boolean isBinary();

    // returns left operand for operation.
    public int leftOperand();

    // returns right operand for operation.
    // require: isBinary () == true.
    public int rightOperand();

    // performs operation with operands.
    public int operate();

    // this is the identity element for the operation.
    public int identity();
}
```

This interface is implemented for all the arithmetic operators to be supported. Use an abstract class to reduce repeatability of code. A concrete class of `Operator` will use its constructor having as its left operand the current result of the calculator, with the right operator (if any) provided by the user. Note that you need to handle the case when the operation is “/” or “%” and the user provides 0 as its right operand. The actual concrete class must put a precondition to the operation and the TUI must check for it before requesting the operation to be performed.

The calculator will have at least the following commands:

```java
// for invert, clear, undo, redo operations
public void operate (int operator);

// for the binary operators.
public void operate (int operator, int rightOperand);
```

These operations will create the appropriate operator (for binary operators via a case statement); and proceeds to request the operator to perform the operation to update the running result appropriately as well as the history of operations as needed.

The `redo` operation will redo the last arithmetic operation; using `instanceof` on the last operator, it will determine which operator to instantiate having as left operand the current calculator’s result and as right operand the last operator’s right operand. Assume below that this represents the calculator object and lastOp represents the last arithmetic operation done:

```java
int left = this.result();
int right = lastOp.rightOperand();
if (lastOp instanceof Addition)
    newOp = new Addition (left, right);
else if (lastOp instanceof Subtraction)
    newOp = new Subtraction(left, right);
```
else if ....

Once the operation is redone against the current calculator’s result, you add it to the history.

What to submit:
0. A test plan to use in the testing of the calculator. This test most likely must be broken into several parts; each part must state what is to be tested along with input and expected output.
1. All the classes source code formatted according to the coding standards.
2. A tester for each of the model concrete classes implemented.
3. The TUI source code.
4. A run of the test plan.
5. A report stating the following:
   a. what you completely implemented and tested successfully;
   b. what you implemented but failed the tests completely or partially.
   c. what you did not implement along with the reason for not having done it.

Grading: Assuming that the code you submit follows the coding standards and is tested,
   a. For a grade of C: successfully implement all the arithmetic operations.
   b. For a grade of B: successfully implement a. plus undo and clear.
   c. For a grade of A: successfully implement b. plus redo
   d. For bonus points do all plus further operations. 10 points per new operation implemented.

Note: You can submit this homework up to 5 days late; grading rules for late work will apply.