CSCI-2120 Homework 1. Due date: Wednesday Feb 4, 2004. 100 Points.

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 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 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:

add
subtract
multiply
divide
remainder
invert
set
undo
redo
clear

The first five operations are the usual integer arithmetic operations; the operation invert changes the sign of the current result; the set operation is used to return a value given by the user to initialize the calculator’s result. You can think of those operations as queries; The operation clear sets the current result of the calculator to zero and resets the history of operations to empty. The clear operation cannot be undone. The operation redo redoes the last arithmetic operation carried out having as left operand the current calculator’s result and updates the history. 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 - 10):

```java
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: calResult = requestInput(prompt);
             break;
    case 8: redoOp();
             break;
    case 9: redoOp();
             break;
    case 10: clearOP();
             break;
}
```
The calculator then proceeds to present the answer to the user. This solution can be used to correctly implement the job requested; but this solution will be hard to implement for some cases (such as undo and redo) and is hard for testing and in the future for maintenance and for variability and extensibility. And it is based on codifying the operations with integers (1-10) which eventually will show to be a very poor design.

A desirable solution will make use of interfaces and abstract classes; this is the kind of solution I expect in this homework.

One such solution is based on abstracting an operator via an interface:

```java
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();

    // make an instance of this Operator with a new left operand
    public Operator createWithNewLeft(int left);

    // make an instance of this Operator with a new right operand
    public Operator createWithNewRight(int right);

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

    //performs operation with same operands.
    public int redo();

    //undoes operation returning leftOperand().
    public int undo();

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

This interface is implemented for all the arithmetic operators to be supported, including set. Use abstract classes 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 constructor and it must check for it before creating the operation to be performed.

The calculator will have at least the following command:

```java
// require: operator must be a number between 1..9
public void operate (int operator);
```

These operation will create the appropriate operator; and if the operator is a binary operator, the calculator must request input from the TUI before creating the actual operator object; it then proceeds to request the operator to perform the operation to be used by the calculator to update the running result as well as the history of operations.

The redo operation will redo the last arithmetic operation; using the last operator, create a new instance of it 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. 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. Use JUnit tests.
3. The TUI source code.
4. A script of 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 and the TUI.
   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.

**Appendix**
Here are some test case to consider (which you must complete with input and expected output); use these in your test plan to submit. You should add more cases to this plan.
- perform an arithmetic operation to check for correct answer. Do it for all operators.
- choose a division operation and provide 0 as second operand. Should get an error message.
- perform an arithmetic operation, undo it to come back to old calculator’s result.
- perform two arithmetic operations, and proceed to undo them.
- perform three arithmetic operations and proceed to undo them.
- perform 4 operations. Proceed to undo each. You should be able to undo 3 times only. The forth time the user should get an error message.
- perform an arithmetic operation; proceed to redo it.
- perform an arithmetic operation; proceed to redo it; proceed to undo it to come back to original operation’s answer.
- perform arithmetic operations (as many as you want) proceed to clear the calculator, check that calculator’s result is 0 and that the history is empty by attempting to undo.