UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFORMATICS 1 - OBJECT-ORIENTED PROGRAMMING

Monday 16th August 2010

14:30 to 16:30

Convener: J Bradfield
External Examiner: A Preece

INSTRUCTIONS TO CANDIDATES

1. **Note that ALL QUESTIONS ARE COMPULSORY.**
2. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS. Take note of this in allocating time to questions.**

1. In each of parts (a)–(c) below, you will be asked to supply the body to a method inside a class. There will be a separate class for each part, named `OneA`, `OneB` and `OneC` respectively. You will be given a skeleton file for each of these classes, and the skeleton will contain the appropriate method declaration. You should add your definitions of the methods at the points marked as follows:

   `// ADD CODE HERE`

   (a) Implement the method `int product(int[] a)` in the class `OneA` (skeleton file supplied). Given an array of `int`s, `a`, this method should return the product of all of the elements, that is, the value of every element multiplied together.

   Expected behaviour:

```
product(new int[] {1, 2, 3, 4, 5}) -> 120
product(new int[] {2, 3, 7}) -> 42
product(new int[] {16, 8, 4, 2}) -> 1024
```

   [*15 marks*]

   (b) Implement the method `int[] sumDiffs(int[] a, int[] b)` in the class `OneB`. Given two arrays of `int`s, `a` and `b`, this should return a new array. For each position i in the arrays `a` and `b`, if the elements at i differ, then the element at i in the new array should be the sum of the two, otherwise it should be the value shared between the arrays. You can assume the two arrays are the same length.

   Expected behaviour:

```
sumDiffs(new int[] {1, 2, 3, 4, 5}, new int[] {1, 2, 3, 4, 5})
                                             -> {1, 2, 3, 4, 5}
sumDiffs(new int[] {1, 2, 3, 4, 5}, new int[] {1, 6, 7, 4, 5})
                                            -> {1, 8, 10, 4, 5}
sumDiffs(new int[] {5, 13, 3, 9}, new int[] {9, 3, 13, 5})
                                            -> {14, 16, 16, 14}
sumDiffs(new int[] {1, 2, 3, 5, 8}, new int[] {1, 2, 4, 8, 8})
                                           -> {1, 2, 7, 13, 8}
```

   [*15 marks*]

   (c) Implement the method `int[] stretch(int[] a)` in the class `OneC`. Given an array of `int`s, `a`, this method should return a new array twice as large as the original, replacing every integer from the original array with two integers, each half the original. If a number in the original array is odd, then the first of the two new numbers should be one higher than the second so that the sum equals the original number.

   Expected behaviour:

```
sumDiffs(new int[] {1, 2, 3, 4, 5})
                                -> {1, 0, 1, 1, 2, 1, 2, 2, 3, 2}
sumDiffs(new int[] {5, 13, 3, 9}) -> {3, 2, 7, 6, 2, 1, 5, 4}
sumDiffs(new int[] {1, 2, 4, 8, 16, 32})
                        -> {1, 0, 1, 1, 2, 2, 4, 4, 8, 8, 16, 16}
```

[*14 marks*]

(d) The class QuestionOneLauncher (skeleton file supplied) has a single main() method. Inside main(), add calls to each of the methods you have defined in parts (a)–(c) above to test that your implementations produce the correct results. You can write your tests in any way you think appropriate, but you should have at least two tests for each method. [*6 marks*]

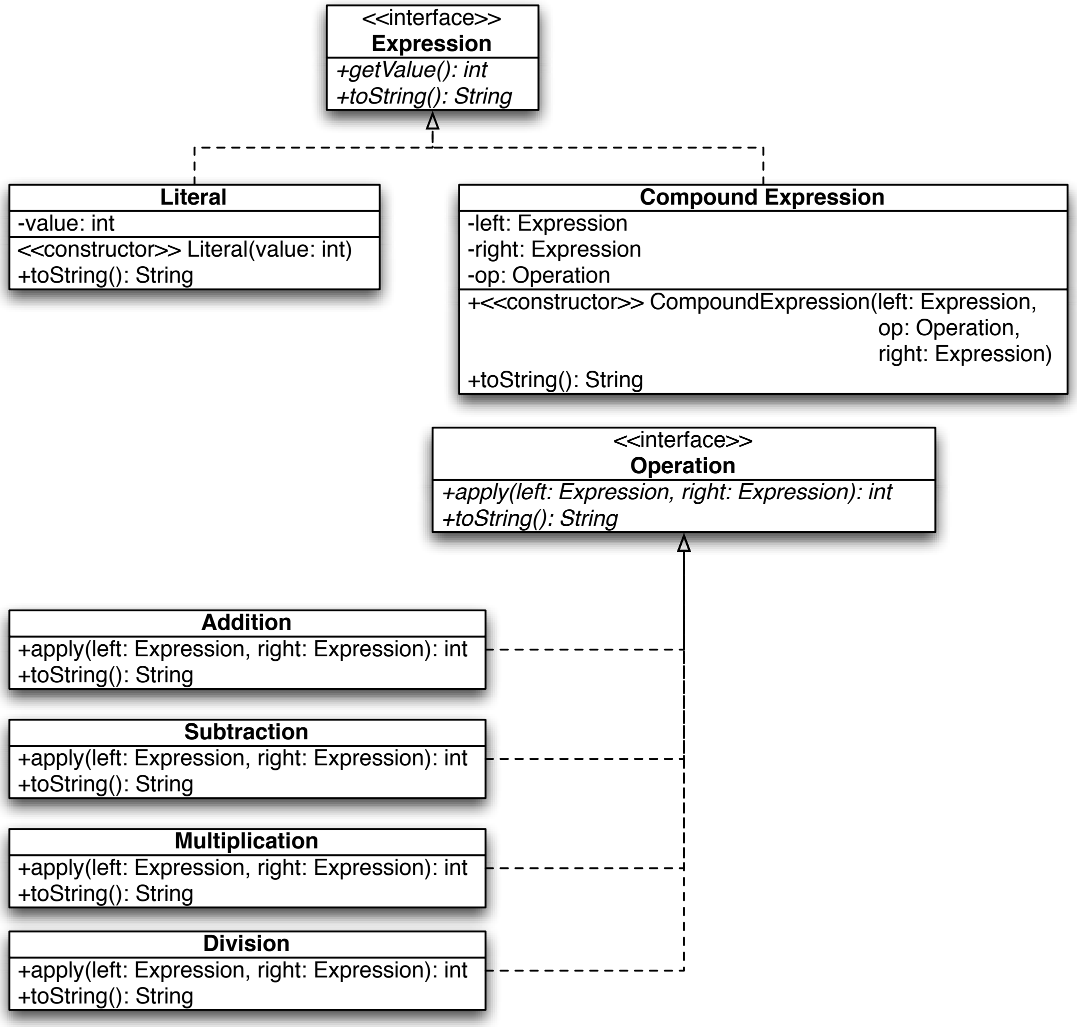
The files that you must submit for this question are the following:

(a) OneA.java

(b) OneB.java

(c) OneC.java

(d) QuestionOneLauncher.java



<<interface>>
Expression
+getValue(): int
+toString(): String
Literal
-value: int
<<constructor>> Literal(value: int)
+toString(): String
Compound Expression
-left: Expression
-right: Expression
-op: Operation
+<<constructor>> CompoundExpression(left: Expression, op: Operation, right: Expression)
+toString(): String
<<interface>>
Operation
+apply(left: Expression, right: Expression): int
+toString(): String
Addition
+apply(left: Expression, right: Expression): int
+toString(): String
Subtraction
+apply(left: Expression, right: Expression): int
+toString(): String
Multiplication
+apply(left: Expression, right: Expression): int
+toString(): String
Division
+apply(left: Expression, right: Expression): int
+toString(): String

Figure 1: UML Diagram of the Calculator System

2. This question involves the construction of a calculator program that stores and evaluates expressions. In the design, the system has a generic `Expression` interface upon which all expressions are built. Implementing this interface are two classes representing two different types of expression. The `Literal` class stores an expression as an integer literal. For example, the Java statement

```
  Literal lit = new Literal(-3);
```

creates a `Literal` object which stores the numeral -3. The `CompoundExpression` class stores an expression made up of two other `Expressions` and an `Operation`. The design of the calculator is shown in Figure 1.

You have been provided with the `Operation` interface, and the `Literal`, `Addition` and `Subtraction` classes.

You are required to write an interface and four new classes: `Expression`, `Division`, `Multiplication`, `CompoundExpression` and `CalculatorLauncher`. The task is broken down into more detail below.

(a) Write the interface `Expression`. You should be able to infer its methods from Figure 1. [*5 marks*]

(b) Write the classes `Division` and `Multiplication`. These classes should implement the `Operation` interface. Their methods should work like those of the `Addition` and `Subtraction` classes. Note that `Division` should just use integer division internally, since the return value of `apply()` is of type `int`. Use the symbol `*` as the string representation of multiplication. [*5 marks*]

(c) Write the class `CompoundExpression` which implements `Expression`. This task is split into more detail below:

(i) Create instance variables to store the `Operation` and two component `Expression`s which are used to build up the the `Compoundexpression`.

(ii) Write the constructor for this class as specified in Figure 1 in order to initialise the relevant instance variables.

(iii) Write the method `getValue()` which evaluates the result of applying the `Operation` provided to the two component `Expression`s.

(iv) Provide a body for the `toString()` method so as to return a `String` representation of the `CompoundExpression`. For example, if the first and second component `Expression`s were the `Literal`s 3 and 4 respectively, and the operation was `Addition`, this method should output `(3 + 4)`. [*20 marks*]

(d) The class `CalculatorLauncher` (skeleton provided) contains a single `main()` method. Extend this method to both represent **and** evaluate the arithmetic expressions listed below, using the classes and methods provided plus those resulting from tasks (a)–(c) earlier in this question. In each case you should provide code to print out an equation whose lefthand side is the `Expression` in question and whose righthand side is the value of the `Expression`. For example, in the case of (ii) below, your code should print out lines of the form

```
(1 - 2) = -1
(12 / 2) = 6
```

(It is acceptable if your code omits parentheses where there is no ambiguity, but you will find it easier to include the parentheses as shown above.) Here are the expressions that you must deal with:

i. The `Literal`s corresponding to integers 3, 12 and -6.

ii. The `CompoundExpression`s corresponding to $1 - 2$ and $12/2$.

iii. The `CompoundExpressions` corresponding to $(1-2)\times 3$ and $-6\times(12/2)$.

[15 marks]

(e) Further extend the `main()` method of `CalculatorLauncher` to represent and evaluate the `CompoundExpressions` corresponding to $((1-2)\times 3)+(12/2)$. Make this method then print out the equation and the result it calculates using the relevant methods. Your program should provide output as shown below:

```
((1 - 2) * 3) + (12 / 2) = 3
```

[5 marks]

The files that you must submit for this question are the following:

(a) `Expression.java`

(b) `Division.java`

(c) `Multiplication.java`

(d) `CompoundExpression.java`

(e) `CalculatorLauncher.java`

Final Checklist

Here is a complete list of all the files required for this exam:

```
OneA.java
OneB.java
OneC.java
QuestionOneLauncher.java
Expression.java
Division.java
Multiplication.java
CompoundExpression.java
CalculatorLauncher.java
```