Past Year PE1 Question: Expression

Adapted from PE1 of 20/21 Semester 2

Instructions to Past-Year PE1 Question:

1. Accept the repo on GitHub Classroom here
2. Log into the PE nodes and run `~cs2030s/get py1` to get the skeleton for all available past year PE1 questions.
3. The skeleton for this question can be found under `2021-s2-q1`. You should see the following files:
   - The files `Test1.java`, `Test2.java`, and `CS2030STest.java` for testing your solution.
   - The skeleton files for this question: `Operand.java` `InvalidOperandException.java` `Operation.java`

Background

An expression is an entity that can be evaluated into a value.

We consider two types of expression in this question:

- An operand, which itself is a value.
- A binary operation, which is a mathematical function that takes in two expressions and produces an output value.

For instance,

- 3 is an expression that evaluates to 3.
- 3 + 2 is an expression that evaluates to 5
- (3 + 2) + 3 is also an expression that evaluates to 8

An operand is not necessarily an integer. It can be of any type. An expression can be evaluated to any type.

Three skeleton files are provided for you: `Operand.java`, `Operation.java`, and

`InvalidOperandException.java`. If you need extra classes or interfaces, create the necessary additional Java files yourself.

Operand

Create a class called `Operand` that encapsulates the operands of an operation. The `Operand` class can contain references to a value of any reference type.

You may create additional parent classes or interfaces if you think it is appropriate.

The Operand has an `eval` method that returns its value.

```
jshell> new Operand(5).eval()
$.. ==> 5
jshell> new Operand("string").eval()
$.. ==> "string"
jshell> new Operand(true).eval()
$.. ==> true
```

Expression.java

```
abstract class Expression {
    abstract Object eval();
}
```

Operand.java

```
class Operand extends Expression {
    private Object o;

    public Operand(Object o) {
        this.o = o;
    }

    public Object eval() {
        return this.o;
    }
}
```

InvalidOperandException

Create an unchecked exception named `InvalidOperandException` that behaves as follows:

```
jshell> InvalidOperandException e = new InvalidOperandException('!')
jshell> e.getMessage();
$.. ==> "ERROR: Invalid operand for operator !"
```

The constructor for `InvalidOperandException` takes in a `char` which is the corresponding symbol for the operation that is invalid.

Recall that all unchecked exceptions are a subclass of `java.lang.RuntimeException`. The class `RuntimeException` has the following constructor:

```
RuntimeException(String message)
```

that constructs a new runtime exception with the specified detail message `message`. The message can be retrieved by the `getMessage()` method.

You can test your code by running the `Test1.java` provided. Make sure your code follows the CS2030S Java style.

```
$ javac Test1.java
$ java Test1
$ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
*.java
```

InvalidOperandException.java

```
class InvalidOperandException extends RuntimeException {
    InvalidOperandException(char operator) {
        super("ERROR: Invalid operand for operator " + operator);
    }
}
```

Operation

Create an abstract class called `Operation` with the following fields and methods:

- two private fields that correspond to two expressions (an expression is as defined at the beginning of this question).
- a class factory method `of`, which returns the appropriate subclass that implements a specific operation. The first parameter of the `of` methods is a `char` to indicate the

operation to be performed. You need to support three operations:

- if the first parameter is `*`, return an operation that performs multiplication on integers
- if the first parameter is `+`, return an operation that performs concatenation on strings
- if the first parameter is `^`, return an operation that performs XOR on booleans
- if the first parameter is none of the above, return `null`

Note that the operator to perform XOR on two boolean variables is `^`.

For instance,

```
jshell> Operation o = Operation.of('*', new Operand(2), new Operand(3));
jshell> o.eval()
$.. ==> 6

jshell> Operation o = Operation.of('+', new Operand("hello"), new
Operand("world"));
jshell> o.eval()
$.. ==> "helloworld"

jshell> Operation o = Operation.of('^', new Operand(true), new
Operand(false));
jshell> o.eval()
$.. ==> true

jshell> Operation.of('!', new Operand(2), new Operand(3));
$.. ==> null

jshell> Operation o1 = Operation.of('*', new Operand(2), new Operand(3));
jshell> Operation o = Operation.of('*', o1, new Operand(4));
jshell> o.eval()
$.. ==> 24

jshell> Operation o2 = Operation.of('*', new Operand(2), new Operand(4));
jshell> Operation o = Operation.of('*', o1, o2);
jshell> o.eval()
$.. ==> 48
```

If the operands are not of the correct type, `eval` must throw an unchecked `InvalidOperandException` exception.

For instance,

```
jshell> Operation o = Operation.of('*', new Operand("1"), new
Operand(3));
jshell> try {
   ...>   o.eval();
   ...> } catch (InvalidOperandException e) {
   ...>   System.out.println(e.getMessage());
   ...> }
```

```
ERROR: Invalid operand for operator *

jshell> Operation o = Operation.of('+', new Operand(1), new Operand(4));
jshell> try {
   ...>   o.eval();
   ...> } catch (InvalidOperandException e) {
   ...>   System.out.println(e.getMessage());
   ...> }
ERROR: Invalid operand for operator +

jshell> Operation o = Operation.of('^', new Operand(false), new
Operand(3));
jshell> try {
   ...>   o.eval();
   ...> } catch (InvalidOperandException e) {
   ...>   System.out.println(e.getMessage());
   ...> }
ERROR: Invalid operand for operator ^

jshell> Operation o1 = Operation.of('*', new Operand(1), new Operand(3));
jshell> Operation o2 = Operation.of('^', new Operand(false), new
Operand(false));
jshell> Operation o = Operation.of('+', o1, o2);
jshell> try {
   ...>   o.eval();
   ...> } catch (InvalidOperandException e) {
   ...>   System.out.println(e.getMessage());
   ...> }
ERROR: Invalid operand for operator +

jshell> Operation o1 = Operation.of('*', new Operand(1), new
Operand("3"));
jshell> Operation o2 = Operation.of('^', new Operand(false), new
Operand(false));
jshell> Operation o = Operation.of('+', o1, o2);
jshell> try {
   ...>   o.eval();
   ...> } catch (InvalidOperandException e) {
   ...>   System.out.println(e.getMessage());
   ...> }
ERROR: Invalid operand for operator *
```

You can test your code by running the `Test2.java` provided. Make sure your code follows the CS2030S Java style.

```
$ javac Test2.java
$ java Test2
$ java -jar ~cs2030s/bin/checkstyle.jar -c ~cs2030s/bin/cs2030_checks.xml
*.java
```

Operation.java

```java
abstract class Operation extends Expression {
    private Expression x;
    private Expression y;

    public Expression getX() {
        return this.x;
    }

    public Expression getY() {
        return this.y;
    }

    public Operation(Expression x, Expression y) {
        this.x = x;
        this.y = y;
    }

    public static Operation of(char c, Expression x, Expression y) {
        if (c == '*') {
            return new TimesOperation(x, y);
        } else if (c == '+') {
            return new ConcatOperation(x, y);
        } else if (c == '^') {
            return new XorOperation(x, y);
        }
        return null;
    }
}
```

TimesOperation.java

```java
class TimesOperation extends Operation {
    public TimesOperation(Expression x, Expression y) {
        super(x, y);
    }

    @Override
    public Object eval() {
        Object objX = this.getX().eval();
        Object objY = this.getY().eval();
        if (objX instanceof Integer && objY instanceof Integer) {
            int x = (Integer) objX;
            int y = (Integer) objY;
            return x * y;
        } else {
            throw new InvalidOperandException('*');
        }
    }
}
```

ConcatOperation.java

```java
class ConcatOperation extends Operation {
    public ConcatOperation(Expression x, Expression y) {
        super(x, y);
    }

    @Override
    public Object eval() {
        Object objX = this.getX().eval();
        Object objY = this.getY().eval();
        if (objX instanceof String && objY instanceof String) {
            String x = (String) objX;
            String y = (String) objY;
            return x + y;
        } else {
            throw new InvalidOperandException('+');
        }
    }
}
```

XorOperation.java

```java
class XorOperation extends Operation {
    public XorOperation(Expression x, Expression y) {
        super(x, y);
    }

    @Override
    public Object eval() {
        Object objX = this.getX().eval();
        Object objY = this.getY().eval();
        if (objX instanceof Boolean && objY instanceof Boolean) {
            boolean x = (Boolean) objX;
            boolean y = (Boolean) objY;
            return x ^ y;
        } else {
            throw new InvalidOperandException('^');
        }
    }
}
```