# Scala-Interview-Test

**Author: Jian Yang**

**File organization**

The project is built by sbt in IntelliJ IDEA.

The source code files are under /src/main/scala/ and the test code file is in /src/test/scala/.
For convenience, all code files are duplicated in /source_code.

All responses to questions and explanation of solutions are in below and duplicated in Solution.pdf.

## Project Preference

I prefer the **giter8 templates integration** project. For one thing, web development is always my interest, and I have developed many web application, like news collections website and online stock trading system. Furthermore, I am familiar with the REST interface and I have used it several times. For another, I am an experienced git user and I know how git works, such as version control, branches, etc.

In a word, this project not only fits my interest well but also matches my skill sets, so I prefer this one.

By the way, I am good with other projects, for I did a research program with sbt and I understand how sbt works. I also have taken compiler courses, so I believe I am able to catch up the other projects quickly.

## Warmup

File: WarmUp.scala

Assumption: the parameter is an integer.

1. Time complexity: O(2^N).
   Space complexity: O(N) if we consider the function call stack size, otherwise O(1).
2. Yes, there are better solutions.
   First solution can be dynamic programming. We store the f(x) of every x from 0 to n, and thus we only need to calculate f(x) once for every x. Therefore, the time complexity will be O(N) while the space complexity is still O(N).
   In addition, in this specific problem, f(n) = 2^n, so we can use the power function directly, then the time complexity and space complexity both become O(lgN).

## Using APIs and testing: JSON serialization

### JSON serialization and deserialization

Files: BooleanJson.scala, BooleanExpression.scala

The Json4s (http://json4s.org/) library is used.

Json format:
And: {"jsonClass": "And", "e1": JObject, "e2": JObject}
Or: {"jsonClass": "Or", "e1": JObject, "e2", JObject}
Not: {"jsonClass": "Not", "e": JObject}
Variable: {"jsonClass": "Variable", "symbol", JString}
True: {"jsonClass": "True"}
False: {"jsonClass": "False"}

As shown above, a key named "jsonClass" is used to distinguish between different Boolean Expressions.
A custom serializer is used to process Boolean Expressions and JSON Expressions recursively.

Example:
And(Or(Variable(a),True),Not(Variable(b))) =>
{"jsonClass":"And","e1":{"jsonClass":"Or","e1":{"jsonClass":"Variable","symbol":"a"},"e2":
{"jsonClass":"True"}},"e2":{"jsonClass":"Not","e":{"jsonClass":"Variable","symbol":"b"}}}"

### Test

File: BooleanJsonTest.scala

The ScalaTest (http://www.scalatest.org/) library is used.

The Boolean Expressions and JSON Expressions are checked recursively. The checking is based on type, for example, "e1" should be a JObject in a JSON Expression while an extracted Boolean Expression should belong to (And, Or, Not, Variable, True, False).

**Error Handling**

In the serialization, a regular expression is used to check whether the symbol of Variable is legal. If there is an illegal symbol, an exception will be thrown. The information of exception will be printed on the screen so that the user can find out how to fix it.
Especially, in the second bonus assignment, the exception information is sent to the client side, but the connection would not lose.

## Bonus assignment

### Algebraic transformation

File: BooleanAlgebra.scala

The pattern match is used to convert a Boolean Expression into an Algebra Expression recursively.

Formats:
And: (e1)∧(e2)
Or: (e1)v(e2)
Not: ¬(e)
Variable: symbol
True: $T
False:$F

Example:
And(Or(Variable("a"), True), Not(Variable("b"))) =>
((a)v($T))∧(¬(b))

## Boolean algebra server

Files: BooleanServer.scala, BooleanClient.scala

The socket library is used.

The server is bound to a localhost port and accepting a connection. When a message is received, it will convert the message into an Algebra Expression and send it to the client. If an exception is thrown, it will send the error message to the client.

The client established a connection to the server. It will read input from the console, send the content to the server, and print the output.

## Bells and whistles

1. On the client side, the JSON expression is printed in pretty formats.
   Example:
   ```
   {
     "jsonClass" : "Not",
     "e" : {
       "jsonClass" : "Not",
       "e" : {
         "jsonClass" : "Variable",
         "symbol" : "a"
       }
     }
   }
   ```
2. On the server side, the logging function is added.
   Example:
   23:49:33.337 [main] INFO BooleanServer$ - Boolean ExpressionNot(Not(Variable(a)))