Exercise 4

1. Examine the Scribble protocol in Fig. **??** (left) and the Java program given in Fig. **??** (right). Give the projection for the Voter role. Is the Java program correct w.r.t. the protocol? If not, correct and explain the errors. (Read the Instruction section to learn how to set up your Java environment for this exercise)

```
module exercise.voting;
type <java> "str" from "java" as string;
type <java> "int" from "java" as int;

global protocol EVoting(role Voter as V,
role Server as S){
Authenticate(name:string) from V to S;

choice at S {
  Ok(token:string) from S to V;
  choice at V {
    Yes(string) from V to S;
    // returns the number of yes and no voters
    Result(int, int) from S to V;
  } or {
    No(reason:string) from V to S;
  }
} or {
  Reject(reason:string) from S to V;
}
}
```

```
void  vote(SessionEndpoint s1, String name) {
  VCases cases = s1.receive(S, Authenticate, name)
                   .branch(S);
  EVoting_V_4 s3 = null;

  switch(cases.op){
    case Ok: Buf<String> token = new Buf<>();
        s3 = cases.receive(Ok, token)
                  .send(S, Maybe, token);
        break;

    case Reject:Buf<String> reason = new Buf<>();
        s3 = cases.receive(S, Reject,reason)
                  .send(S, Yes, name);
        break;
      }
    Buf<String> results = new Buf<>();
    s3.receive(S, Result, results);
}
```

Figure 1: Voting Service in Scribble (left) and Java (right)

2. Implement a Java program for the Server role from the Scribble protocol in Fig. **??**. (Read the Instruction section to learn how to set up your Java environment for this exercise)

```
global protocol Calculator(role Server as S, role Client as C) {
rec Loop {
  choice at C {
    sum(int, int) from C to S;
    result(int) from S to C;
    continue Loop;
  } or {
    multiply(int, int) from C to S;
    result(int) from S to C;
    continue Loop;
  } or {
    quit() from C to S;
    terminate() from S to C;
  }
}}
```

Figure 2: A Calculator protocol in Scribble

3. Fig. **??** shows an implementation of a math server and its type in Sepi.

   (a) Explain what the server is doing.

   (b) Write a client that requests a sum of two elements from the server.

   (c) Write a client that requests multiplication of two elements (hint: use delegation)

   Use the Sepi Online Interpreter to implement and run your program: http://gloss.di.fc.ul.pt/tryit/SePi

```
type Calc = &{ sum: ?integer. ? integer. ! integer. Calc,
               multiplication: ?(integer, integer).! integer. Calc,
               quit: end }

def mathServer(x: Calc) = {
  case x of
  sum →
    x ? a. x ? b. x !(a+b).
    mathServer!(x) // recursion
  multiplication →
    x ?(a, b).x !(a*b).
    mathServer !(x)
  quit →
    printStringLn !"the client is done"
}
```

Figure 3: MathServer in SEPI

Instructions on how to configure Session Java

1. Clone the following github repo:
   git clone https://github.com/rumineykova/scribble-java.git
   cd scribble-java git checkout session-exercise

2. Open Eclipse

3. Go to File → Import → Maven → Existing Maven Projects. Then navigate to the scribble-java folder you just cloned. Several projects are imported. Right click on the project named *parent* (or *scribble-java*).

4. From the right click menu choose Run As/ Maven Install. Pop up will appear asking to install a amven plugin. Click Finish.

5. Go to the scribble-core. Right click on the scribble/core/src/scrib folder. Select Build Path/Use as Source Folder

6. Make sure Project/Build Automatically is selected from the top level menu in Eclipse

For Exercise 1: Open scribble-core/src/test/scrib/exercise/voting/Server.java
The file does not compile. Fix the errors following the Eclipse suggestions.
For Exercise 2: Open scribble-core/src/test/scrib/exercise/calculator/Server.java
Finish the implementation of the main function following the protocol.