Work Time: 3 hours
Please implement in Java the following two problems.
If a problem implementation does not compile or does not run you will get 0 points for that problem (that means no default points)!!!
If for one problem you have only a text interface to display the program execution you are penalized with 1.25 points for that problem.

1. **(0.5p by default). Problem 1:Implement a CountDownLatch mechanism in ToyLanguage.**
   a. (0.5p). Inside PrgState, define a new global table (global means it is similar to Heap, FileTable and Out tables and it is shared among different threads), LatchTable that maps an integer to an integer. LatchTable must be supported by all of the previous statements. It must be implemented in the same manner as Heap, namely an interface and a class which implements the interface. **b. (0.75p).** Define a new statement newLatch(var,exp)
   which creates a new countdownlatch into the LatchTable. The statement execution rule is as follows:

   Stack1={newLatch(var, exp)| Stmt2|...}
   SymTable1
   Out1
   Heap1
   FileTable1
   LatchTable1
   ==>
   Stack2={Stmt2|...}
   Out2=Out1
   Heap2=Heap1
   FileTable2=FileTable1
   - evaluate the expression exp using SymTable1 and Heap1 and let be number the result of this evaluation
   LatchTable2 = LatchTable1 synchronizedUnion {newfreelocation ->number}
   if var exists in SymTable1 *then*
   SymTable2 = update(SymTable1,var, newfreelocation)
   else SymTable2 = add(SymTable1,var, newfreelocation) Note that you must use the lock mechanisms of the host language Java over the LatchTable in order to add a new latch to the table. **c. (0.75p).** Define the new statement

   await(var)

   where var represents a variable from SymTable which is mapped to a number into the LatchTable. Its execution on the ExeStack is the following:
   - pop the statement

- foundIndex=lookup(SymTable,var). If var is not in SymTable print an error message and terminate the execution.
- *if* foundIndex is not an index in the LatchTable *then* print an error message and terminate the execution *elseif* LatchTable[foundIndex]==0 *then* do nothing

  else push back the await statement(that means the current PrgState must wait for the countdownlatch to reach zero)

d. (0.75p) Define the new statement: countDown(var) where var represents a variable from SymTable which is mapped to an index into the LatchTable. Its execution on the ExeStack is the following:

- pop the statement
- foundIndex=lookup(SymTable,var). If var is not in SymTable print an error message and terminate the execution.
- *if* foundIndex is not an index in the LatchTable *then* do nothing *elseif* LatchTable[foundIndex] > 0 *then*

  LatchTable[foundIndex]=LatchTable[foundIndex]-1;

  write into Out table the current prgState id

  else do nothing

Note that the lookup and the update of the LatchTable must be an atomic operation, that means they cannot be interrupted by the execution of the other PrgStates. Therefore you must use the lock mechanisms of the host language Java over the LatchTable in order to read and write the values of the LatchTable entrances .

e. (1p) Extend your GUI to suport step-by-step execution of the new added features. To represent the LatchTable please use a TableView with two columns location and value.

f. (0.75p). Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a readable log text file.

The following program must be hard coded in your implementation.

new(v1,2);new(v2,3);new(v3,4);newLatch(cnt,rH(v2));
fork(wh(v1,rh(v1)*10));print(rh(v1));countDown(cnt);
fork(wh(v2,rh(v2)*10));print(rh(v2));countDown(cnt);
fork(wh(v3,rh(v3)*10));print(rh(v3));countDown(cnt))));
await(cnt); print(100); countDown(cnt); print(100)

The final Out should be {20,id-first-child,30,id-second-child,40, id-third-child, 100,100} where id-first-child, id-second-child and id-third-child are the unique identifiers of those three new threads created by fork.

2. (0.5p by default) Problem 2: Implement Repeat...Until statement in Toy Language.

a. **(2.75p).** Define the new statement: repeat stmt1 until exp2

The body stmt1 is executed as long as exp2 is not true.
Its execution on the ExeStack is the following:

- pop the statement
- create the following statement: stmt1;(while(!exp2) stmt1)
- push the new statement on the stack

b. **(1.75p).** Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a readable log text file.

The following program must be hard coded in your implementation:

```
v=0;
(repeat (fork(print(v);v=v-1);v=v+1) until v==3);
x=1;y=2;z=3;w=4;
print(v*10)
```

The final Out should be {0,1,2,30}