Laboratory 7
week 7 (9 November – 14 November 2020)

TASKS:

A. Please continue to work on the assignment A3. The deadline of A3 is week 8 (16 - 20 November 2020).

B. After the seminar 7, Please start to work on the assignment A4. The deadline of the assignment A4 is week 9 (23-27 November 2020).

Assignment A4

You must extend the JAVA project from **A3**. Please implement the followings:

1. Heap Management

1.1. Add a new Type class, that we called RefType. A reference type is a recursive type such that, in our ToyLanguage we can allow the following types:

Ref(int) x1;
Ref bool x2;
Ref (Ref (int)) x3;
Ref Ref Ref string x4;
Ref Ref Ref Ref bool x5;

Therefore you may want to implement as follows:

```
class RefType implements Type{
        Type inner;
   RefType(Type inner) {this.inner=inner;}
   Type getInner() {return inner;}
   boolean equals(Object another){
                    if (another instanceof  RefType)
                            return inner.equals(another.getInner());
                    else
                            return false;
                }
    String toString() { return "Ref(" +inner.toString()+")";}
    Value defaultValue() { return new RefValue(0,inner);}
}
```

1.2. Add a new Value class, that we called RefValue. A reference value consists of a heap address (that is an int) and the Type of the location having that address. For example:

- a value of type Ref int can be new RefValue(1,new IntType()) where the heap address is 1 and the location has the type IntType
- a value of type Ref Ref int can be new RefValue(10,new RefType(new IntType())) where the heap address is 10 and the location has the type RefType having its inner field of type IntType

Therefore you may want to implement as follows:

```
class RefValue implements Value{
   int address;
```

```
  Type locationType;
.....
  int getAddr() {return address;}
  Type getType() { return new RefType(locationType);}
}
```

1.3. Implement Heap table to manage the heap memory. In order to implement the heap you need to add one more data structure named **Heap** into the program state. Heap is a dictionary of mappings (address, content) where the address is an integer (the index of a location in the heap) while the content is a Value. Heap must manage the unicity of the addresses, therefore it must have a field that denotes the new free location. The addresses start from 1. The address 0 is considered an invalid address (namely null). You have to implement the Heap in the same manner as ExeStack, SymTable, FileTable and Out namely an interface plus a class that implements that interface (PrgState class will contain a field of type heap interface). Heap must be defined as as HashMap and must be integrated in all the previous functionalities (ex. Printing the prgstate in a log file).

1.4. Heap Allocation: Define and integrate the following statement which allocates heap memory:

new(var_name, expression)

- it is a statement which takes a variable name and an expression
- check whether var_name is a variable in SymTable and its type is a RefType. If not, the execution is stopped with an appropriate error message.
- Evaluate the expression to a value and then compare the type of the value to the locationType from the value associated to var_name in SymTable. If the types are not equal, the execution is stopped with an appropriate error message.
- Create a new entry in the Heap table such that a new key (new free address) is generated and it is associated to the result of the expression evaluation
- in SymTable update the RefValue associated to the var_name such that the new RefValue has the same locationType and the address is equal to the new key generated in the Heap at the previous step

Example: Ref int v;new(v,20);Ref Ref int a; new(a,v);print(v);print(a)
At the end of execution: Heap={1->20, 2->(1,int)}, SymTable={v->(1,int), a->(2,Ref int)} and Out={(1,int),(2,Ref int)}

1.5. Heap Reading: Define and integrate the following expression

rH(expression)

- the expression must be evaluated to a RefValue. If not, the execution is stopped with an appropriate error message.
- Take the address component of the RefValue computed before and use it to access Heap table and return the value associated to that address. If the address is not a key in Heap table, the execution is stopped with an appropriate error message.
- In order to implement the evaluation of the new expression, you have to change the signature of the eval method of the expressions classes as follows

Value eval(MyIDictionary<String,Value> tbl, MyIHeap<Integer,Value> hp)

Example: Ref int v;new(v,20);Ref Ref int a; new(a,v);print(rH(v));print(rH(rH(a))+5)
At the end of execution: Heap={1->20, 2->(1,int)}, SymTable={v->(1,int), a->(2,Ref int)} and Out={20, 25}

1.6.Heap Writing: Define and integrate the following statement

wH(Var_Name, expression)

- it is a statement which takes a variable and an expression, the variable contains the heap

address, the expression represents the new value that is going to be stored into the heap

- first we check if var_name is a variable defined in SymTable, if its type is a Ref type and if the address from the RefValue associated in SymTable is a key in Heap. If not, the execution is stopped with an appropriate error message.
- Second the expression is evaluated and the result must have its type equal to the locationType of the var_name type. If not, the execution is stopped with an appropriate message.
- Third we access the Heap using the address from var_name and that Heap entry is updated to the result of the expression evaluation.

Example: Ref int v;new(v,20);print(rH(v)); wH(v,30);print(rH(v)+5);
At the end of execution: Heap={1->30}, SymTable={v->(1,int)} and Out={20, 35}

2. Garbage Collector: In general, the garbage collector removes those addresses which are not refferred from the SymTable and from other Heap table entries. Here, we provide an unsafe implementation of the garbage collector which eliminates all the Heap entries whose keys (addresses) do not occur in the address fields of the RefValues from the SymTable.

```
Map<Integer,Value> unsafeGarbageCollector(List<Integer> symTableAddr, Map<Integer,Value>
heap){
return heap.entrySet().stream()
        .filter(e->symTableAddr.contains(e.getKey())
        .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue));}

List<Integer> getAddrFromSymTable(Collection<Value> symTableValues){
   return symTableValues.stream()
                .filter(v-> v instanceof RefValue)
                .map(v-> {RefValue v1 = (RefValue)v; return v1.getAddr();})
                .collect(Collectors.toList());
}
```

The garbage collector has to be called in the method allStep() of the Controller class, as follows:

```
 void allStep() throws MyException{
          PrgState prg = repo.getCrtPrg();
           repo. logPrgStateExec();
          while (!prg.getStk().isEmpty()){
               oneStep(prg);
                repo. logPrgStateExec();
               prg.getHeap().setContent(unsafeGarbageCollector(
                          getAddrFromSymTable(prg.getSymTable().getContent().values()),
                           prg.getHeap().getContent()));
               repo. logPrgStateExec();
           }}
```

Please integrate the above code in your project. If you use it on the following example, the execution will throw an error when the last rH expression is executed:
Example: Ref int v;new(v,20);Ref Ref int a; new(a,v); new(v,30);print(rH(rH(a)))

Therefore, please complete the above implementation of the garbage collector such that it takes into account the possible references from the Heap cells.

3. While Statement: Define and integrate the following While statement:

while (expression) statement

The statement evaluation rule is as follows:

Stack1={while (exp1) Stmt1 | Stmt2|...}
SymTable1
Out1
HeapTable1
FileTable1
==>

If exp1 is evaluated to BoolValue then
If exp1 is evaluated to False then Stack2={Stmt2|...}
Else Stack2={Stmt1 | while (exp1) Stmt1 | Stmt2|...}
Else
throws new MyException("condition exp is not a boolean")
SymTable2=SymTable1
Out2=Out1
HeapTable2=HeapTable1
FileTable2=FileTable1

Example: int v; v=4; (while (v>0) print(v);v=v-1);print(v)