

Laboratory 8

(week 21-27 November 2017)

NOTE: Starting from week 10 (4-8 December 2017) all groups will have the same topics for seminars and laboratories. The week will start on Monday and end on Friday. Only the groups from Tuesday and Wednesday will have the Seminar 9 and Laboratory 9 due to the National Day Holiday.

TASKS:

- A. Please submit the Lab-Assignment 4 to your laboratory professor.**
- B. Please continue to work on the Lab-Assignment 5 . The deadline of Lab-Assignment 5 is week 10 (4 - 8 December 2017).**
- C. Please start to work on the Lab-Assignment 6.**
The deadline of Lab-Assignment 6 is week 11 (11- 15 December 2017).

Lab-Assignment 6

You must extend the JAVA project from Lab-Assignment 5.

Concurrent ToyLanguage: In order to support concurrent programming in our ToyLanguage you must do the following modifications in your current project from Lab-Assignment 5:

A. Repository Interface

1. **In the Repository there is a List<PrgState>.** Each PrgState corresponds to a thread. Initially you must introduce only one program (namely a PrgState) and the execution of that program will generate multiple PrgStates as you can see below.
NOTE: You are not allowed to introduce more than one program, only the main program is introduced. The other programs are generated by the fork statements!!!
2. You must add **one more method to the Repository interface:**
List<PrgState> getPrgList() that returns the list of the program states.
3. You must add **one more method to the Repository interface :**
void setPrgList(List<PrgState>) that replaces the existing list of program from the repository with one given as parameter in this method.
4. The method **getCrtPrg** must be removed since we are not longer using it.
5. You must change the existing method **void logPrgStateExec()** into **void logPrgStateExec(PrgState)** such that you are able to save the content of the given input PrgState into a text file.

B. PrgState Class

6. You must add **one more method to the class PrgState: Boolean isNotCompleted()** that returns true when the exeStack is not empty and false otherwise.
7. You must **move the method PrgState oneStep(PrgState) from the Controller into PrgState class.** The current version of method oneStep from Controller class looks like:

```
PrgState oneStep(PrgState state){
    MyIStack<IStmt> stk=state.getStk();
    if(stk.isEmpty()) throws MyStmtExecException;
    IStmt crtStmt = stk.pop();
    return crtStmt.execute(state);
}
```

The new version of oneStep method from PrgState class is the following:

```
PrgState oneStep(){
```

```

        if(exeStack.isEmpty()) throws MyStmtExecException;
        IStmt crtStmt = exeStack.pop();
        return crtStmt.execute(this);
    }

```

Note that the new version of oneStep has no argument since the argument of the old version is the receiver of the new version.

8. In the PrgState class **add one more field called id of type int**. Please modify all print, toString, toStr, toString and save (logPrgStateExec) into a text file methods such that the id of the program state to be printed first. In the concurrent settings we must know which program state is printed/saved on the screen/file.

C. IStmt interface and new forkStmt class (Creation of a new thread using the fork statement)

9. You must define a **new class forkStmt** that implements IStmt interface in order to define and integrate the following fork statement:

fork(Stmt)

It may be combined with any other statements (e.g. using either compound statement, or if statement, or loop statement or another fork statement, etc).

10. In the **class forkStmt the method execute** must implement the following rule:

ExeStack1={fork(Stmt1) | Stmt2|Stmt3|....}

SymTable1,

Heap1,

FileTable1,

Out1,

id1

==>

ExeStack2={Stmt2 | Stmt3|.....}

SymTable2=SymTable1

Heap2 = Heap1

FileTable2=FileTable1

Out2 = Out1

id2=id1

and a new PrgState is created with the following data structures:

ExeStack3={Stmt1}

SymTable3=clone(SymTable1)

Heap3=Heap1,

FileTable3=FileTable1

Out3=Out1

id3= id1*10 in order to be unique

The new PrgState is returned by the execute method. As you can see above, when a fork statement is on top of the ExeStack a new PrgState (thread) is generated having as ExeStack the argument of the fork, as SymTable a clone of the parent PrgState (parent thread) SymTable, as Heap a reference to the parent PrgState (parent thread) Heap, as FileTable a reference tot the parent PrgState (parent thread) FileTable and as Out a reference to the parent PrgState (parent thread) Out. **Please note that Heap, FileTable and Out are shared by all PrgStates. The SymTable of the new thread is a clone (or a new deep copy) and is not shared with the parent thread.**

NOTE: Please ensure (and correct if necessary) that the methods execute of all the previous statement classes return null. Only the method execute of the class forkStmt returns a non-null value, namely the new created PrgState.

D. Controller class

11. You must **add one more method**

List<PrgState> removeCompletedPrg(List<PrgState> inPrgList)

which takes a list of PrgState as input , removes all PrgState for which isNotCompleted returns false and then returns as result a list where all PrgState are not completed. You must implement it in functional manner, as follows:

```
return inPrgList.stream()
    .filter(p -> p.isNotCompleted())
    .collect(Collectors.toList())
```

12. As you have seen above in the section of PrgState, **you must move the method PrgState oneStep(PrgState)** from the Controller into PrgState class.
13. You must **add a new field named "executor"** of type **ExecutorService** in Controller class.
14. You **must replace the method allStep**. The current version of the method allStep looks like:

```
void allStep(){
    PrgState prg = repo.getCrtPrg();
    try{
        while(true){
            oneStep(prg);
            prg.getHeap().setContent(conservativeGarbageCollector(
                prg.getSymTable().getContent().values(),
                prg.getHeap().getContent()));
            repo.logPrgStateExec();
        }
    }catch(MyStmtExecException e) {}
    catch(...) ...
    finally { // the code to close all files from FileTable }
}
```

The new version of the method allStep is described in the next steps:

15. You **must define the method void oneStepForAllPrg(List<PrgState>)** which executes one step for each existing PrgState (namely each thread), as follows:

```
void oneStepForAllPrg(List<PrgState> prgList) {
    //before the execution, print the PrgState List into the log file
    prgList.forEach(prg ->repo.logPrgStateExec(prg));

    //RUN concurrently one step for each of the existing PrgStates
    //-----
    //prepare the list of callables
    List<Callable<PrgState>> callList = prgList.stream()
        .map((PrgState p) -> (Callable<PrgState>)(() -> {return p.oneStep();}))
        .collect(Collectors.toList())

    //start the execution of the callables
    //it returns the list of new created PrgStates (namely threads)
    List<PrgState> newPrgList = executor.invokeAll(callList). stream()
        . map(future -> { try { return future.get();}
                        catch(Exception e) {
                            //here you can treat the possible
                            // exceptions thrown by statements
                            // execution, namely the green part
                            // from previous allStep method}
                        })
        .filter(p -> p!=null)
        .collect(Collectors.toList())
}
```

```

//add the new created threads to the list of existing threads
prgList.addAll(newPrgList);
//-----

//after the execution, print the PrgState List into the log file
prgList.forEach(prg ->repo.logPrgStateExec(prg));

//Save the current programs in the repository
repo.setPrgList(prgList);
}

```

16. You must define the new version of the method `void allStep(void)`, as follows:

```

void allStep() {
    executor = Executors.newFixedThreadPool(2);
    //remove the completed programs
    List<PrgState> prgList=removeCompletedPrg(repo.getPrgList());
    while(prgList.size() > 0){
        oneStepForAllPrg(prgList);
        //remove the completed programs
        List<PrgState> prgList=removeCompletedPrg(repo.getPrgList())
    }
    executor.shutdownNow();
    //HERE the repository still contains at least one Completed Prg
    // and its List<PrgState> is not empty. Note that oneStepForAllPrg calls the method
    //setPrgList of repository in order to change the repository

    // update the repository state
    repo.setPrgList(prgList);
}

```

17. **OPTIONAL:** Garbage collector. The method `conservativeGarbageCollector` can be still used, as follows:

```

void allStep() {
    executor = Executors.newFixedThreadPool(2);
    //remove the completed programs
    List<PrgState> prgList=removeCompletedPrg(repo.getPrgList());
    while(prgList.size() > 0){
        //HERE you can call conservativeGarbageCollector
        oneStepForAllPrg(prgList);
        //remove the completed programs
        List<PrgState> prgList=removeCompletedPrg(repo.getPrgList())
    }
    executor.shutdownNow();
    //HERE the repository still contains at least one Completed Prg
    // and its List<PrgState> is not empty. Note that oneStepForAllPrg calls the method
    //setPrgList of repository in order to change the repository

    // update the repository state
    repo.setPrgList(prgList);
}

```

When you prepare the arguments of the `conservativeGarbageCollector` call you must take

into account the fact that now there is one HEAP shared by multiple PrgStates and multiple SymbolTables(one for each PrgState).

18. **OPTIONAL: Closing the remaining files from FileTable.** You can reuse the code from Lab-Assignment 5 and you have to take into account the fact that FileTable is shared by all PrgStates. The code can be placed as follows:

```
void allStep() {
    executor = Executors.newFixedThreadPool(2);
    //remove the completed programs
    List<PrgState> prgList=removeCompletedPrg(repo.getPrgList());
    while(prgList.size() > 0){
        //HERE you can call conservativeGarbageCollector
        oneStepForAllPrg(prgList);
        //remove the completed programs
        List<PrgState> prgList=removeCompletedPrg(repo.getPrgList())
    }
    executor.shutdownNow();
    //HERE the repository still contains at least one Completed Prg
    // and its List<PrgState> is not empty. Note that oneStepForAllPrg calls the method
    //setPrgList of repository in order to change the repository

    //HERE you can put the code for closing the files from FileTable
    List<PrgState> tmpList= repo.getPrgList();
    //get the acces to FileTable and call the code from Lab-Assignment 5

    // update the repository state
    repo.setPrgList(prgList);
}
```

Example:

```
v=10;new(a,22);
fork(wH(a,30);v=32;print(v);print(rH(a)));
print(v);print(rH(a))
```

At the end:

Id=1

SymTable_1={v->10,a->1}

Id=10

SymTable_10={v->32,a->1}

Heap={1->30}

Out={10,30,32,30}