Task 2 Writeup

I. Code Explanation

TypedValue:
(1) Implementing equals function. As the argument “o” passed to this function is an “Object” type which is vague, so a judgement of “o”’s instance is implemented first and return false if it not a “TypedValue”. Then the condition of whether “o” to be a “Hole” is considered, compare the type only if “o” is a “Hole” and compare both the type and value if “o” is a “TypedValue”.

Template:
(1) Implementing constructor. The code uses “Vector” to initialise the “elements” member.
(2) Implementing “boolean matches(aTuple t)” function. It tests the lengths of two instance first to improve efficiency, as if the lengths are not the same, then they will not match. Followed by traversing the template and compare each “TypedValue” in the two tuples, return false if any of them returns false from function “equals()”. Return true if reaches the end of the function.

TupleSpace:
Using Collections.synchronizedList(new ArrayList<aTuple>()) to initialise “theSpace”, which is a data structure designed for concurrency, also refers to “SynchronizedRandomAccessList”. One of the reason to choose this data structure is to make it concurrency compatible since in some functions the original collection needs to be modified thus synchronisation is required. The other reason is for the random access due to the non-deterministic feature of Linda, this will be explained in following explanation.
(1) Implementing function “out(aTuple v)”. Simply call “add()” to add the tuple to the tuple space, with “theSpace” synchronised.
(2) Implementing private function “aTuple fetchTuple(aTemplete t, boolean delete)”. This additional function covers the fundamental requirement for both “in()” and “rd()”. In this function, “theSpace” is always synchronised. Initially, create an “ArrayList” as an index pool which holds all the unchosen indexes. Then pick a random index from the index pool and remove it. Access “theSpace” with the index and test if template “t” matches it, this insure the tuple to be chosen is non-deterministic if multiple tuples matches. At last, if “delete” is true, copy the matched tuple and return, delete it from “theSpace”. If “delete” is false, simply return the matched tuple.
(3) Implementing function “in()”. Keep calling fetchTuple with “delete” true until a valid tuple is returned.
(4) Implementing function “out()”. Keep calling fetchTuple with “delete” false until a valid tuple is returned.

II. Test Explanation

Using ant for sources control. Instructions are in Appendix A.

A. Single Thread Test.
   1. Basic Functionality Test
      (1) Put two tuples each holds a unique TypedValue into the TupleSpace from a single thread, and fetch one of them to see if the correct one is fetched.
      (2) Do (1) 10 times.
   2. Non-determinism Test
      (1) Fill the TupleSpace with 100 tuples each holds an integer TypedValue and 4 tuples each holds a string TypedValue. Try fetch an integer TypedValue, will get a random one from the 100 integer tuples.
      (2) Calculate the probabilities of the four tuples hold string values. Get around 25% for each.
   3. Null Test
      (1) Call out() with a null tuple.
      (2) Call in() with a null template which can match the null tuple.
   4. Test with JPF

B. Multi-Thread Test
   1. Basic Functionality Test
      (1) That is the test provided by the assignment.
   2. Non-determinism Test
      (1) Test with two threads try to get the same tuple instance.
   3. Test with JPF

Appendix A

Compile & Run & Test Instructions

Compile by run:

```
$ ant
```

or

```
$ ant compile
```

Run by run:

```
$ ./runTask2
```

or

```
$ ant run
```

Compile and run by run:

```
$ ant go
```

Test with jpf by run:

```
$ ant jpf
```

If Apache ant is not installed, please find source code in ./src/ and compiled classes in ./build/classes