—————————————————
Project Title: P2
Tingting Bao, w1270485
—————————————————

—————————————————
How to run
—————————————————
Before running program, please use rm –rf /tmp/tbao/ to remove documents first to insure document exists.
use javac to compile P2.java, and then use java P2 host_1 (host_1 is the hostName, which must be included, and must start with "host_") to run program.

———————————————
Input format
———————————————
java P2 host_1 //(host_1 is host name, must start with "host_", which must be unique)

add (host_1, 129.210.16.80, 9998)(…..)(….) //add command which is used to add hosts into serverList.
"add" must be in lowercase. All the parameters have to be wrapped inside a pair of parentheses. "add" can have one or more parameters. Host name must be unique, and duplicate host name would be denied. The ip address should be valid, which have four parts, and each part's range is from 0~255. port number must be larger than 1023.

out("abc", 3)   //out command: add tuple("abc", 3) into tuple space.
All the parameters must be wrapped inside a pair of parentheses. and "out" must be lowercase. and the data type of parameters can be string, integer, float. It can also have variables, which must match format like ?i:int, ?f:float or ?s:string. All of these should be in lowercase.

in("abc", 3) //in command: match and remove tuple in tuple space.
The parameters following after "in" must match requirement as above. And in must be in lowercase.

rd("abc", 3) // rd command: find tuple in tuple space.
The parameters following after "rd" must match requirement as above. And rd must be in lowercase

delete(host_1, host_2) // delete command: delete host from cluster and exit host. Will check host exists or not.

Kill host // you can input "ctr+c" to kill a host process. And you can restart this host in the same machine again. And this host will return the cluster automatically.

—————————————————
Project introduction:
—————————————————

This is a distributed Linda platform implemented in java. Linda provides a conceptually “global” tuple space (TS) which remote processes can access the matched tuples in TS by atomic operations(in, rd, out).

A tuple is an ordered list of values with types(integer, float, string). A match can be exact match (value only) or variable match (in the form of ?variable_name:type) for all operations.

The “out” simply put tuples in TS, The “in” will match and remove tuples from TS, but the “rd” is not destructive. If multiple tuples are available to an “in” or “rd” call, one is selected non-deterministically. If no tuple is available, the “in” or “rd” call blocks.

Every tuple will store two copies in the Tuplespace, which stores in backup host and original host. Every host will have the probability to crash at any time. When the origin host is crashed, you can read/delete tuples in its backup host. You can also add tuples to its backup host. When the crashed/killed host reboot, it will get update from its backup and original host. You can also delete host, the tuples in this deleted host need to be redistributed again.

——————————————————
Modular design
——————————————————
This project contains the following classes: P2, Client, Server, ErrorChecking ServerItem, ServerList, TupleSpace.

P2 class: It is the entry of the whole project. It first accept P2 command from terminal, and then start Server class and Client class. Use while loop to continue to accept request and pass request to client.

Client class: Accept add/out/in/rd requests from P2, check whether the request is valid or not, and hand request to particular function according to different requests.
Add request: add all the added hosts into this host’s serverList, and send request to all the connected hosts to update their serverList.
out request: using hashing function to get particular host’s ID, and send request to that host’s server to store tuple in its tuple space.
in request: If it is exact-match, using hashing function to get the host’s ID which stores this tuple, and send request to that host to match and remove that tuple. If it is variable-match, send request to all connected host to match and remove that tuple. If not found, it will block until one host find it.
rd request: If it is exact-match, using hashing function to get the host’s ID which stores this tuple, and send request to that host to find that tuple. If it is variable-match, send request to all connected host to find that tuple. If not found, it will block until one host find it.

Server class: Accept request from Client, and execute different commands like add/out/in/rd. It will implement commands like: add hosts into serverList, find and match tuples, and save tuples into disks.

ServerItem class: define and store server’s information: hostName, IP address, Port number

ConsistentHash class: create lookuptable and backUptable, and get ids of tuple.

Md5sum class: use this class to do hashing.

ServerList: deal with the functions related to serverList. like get server file's path, load serverList from disk, save serverList into disk, get all the contents of serverList files, map server's name to serverItem.

TupleSpace class: implement functions related to Tuple Space. Get tuple file's path, load tuple files from disk into memory, save tuples into disk, append a simple tuple into tuples file, create tuples from string and so on.

————————————————
file/directory organizing
————————————————
nets file: Store in path /tmp/<login>/linda/<name>/nets, which store all the information about Servers, including hostName, IP address, port number.
For example, the format is: host_1 172.20.216.90 53695
tuples file: Store in path /tmp/<login>/linda/<name>/tuples, which store all the tuples in that host.
For example ("abc", 3), the data it stores would be 2:si:("abc" 3)&hashVal&(backup OR origin)
—-> 2 means how many elements it has, si represent data types, and ("abc" 3) is the exact tuples.