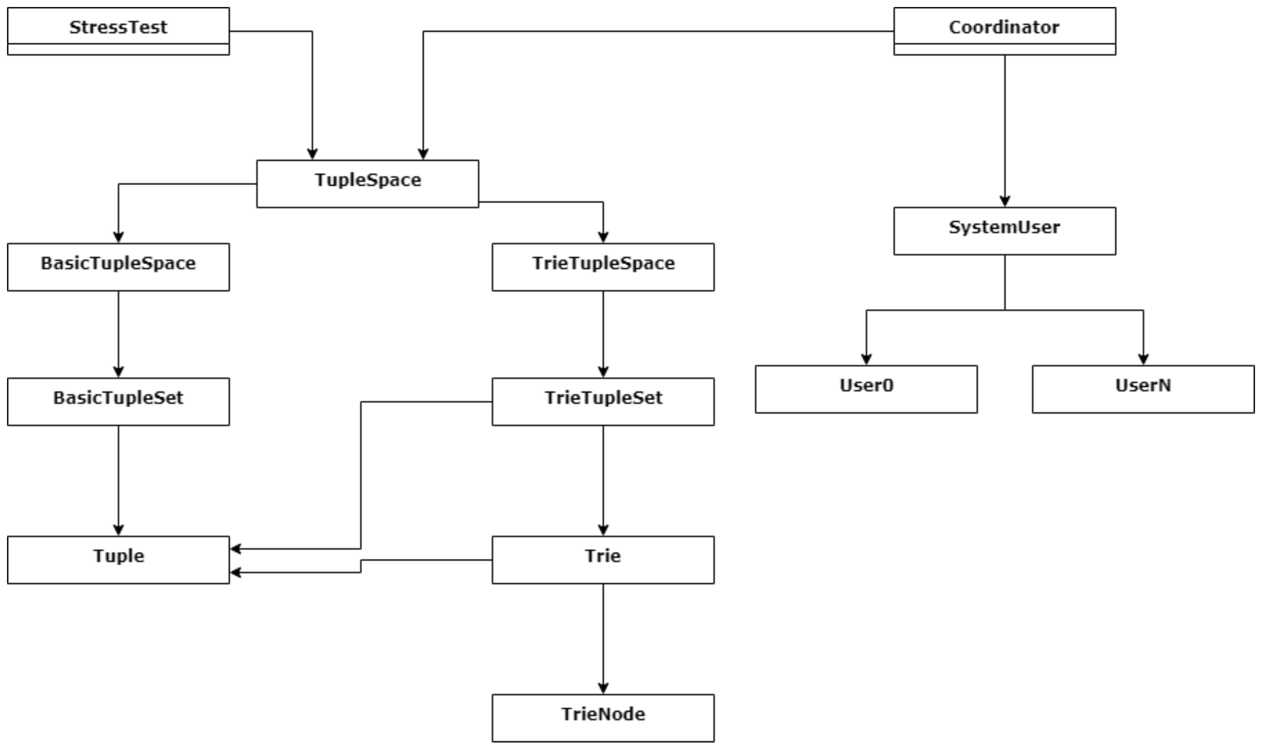
CHAT ROOM DESIGN DIAGRAM



StressTest

Stress Test class used to benchmark the performance difference between Basic Tuple Space and Trie Tuple Space. TEST_COUNT defines how many random tuples will be generated, and TUPLE_SPACE_SIZE defines the max size of any random tuple. Randomized tuples are generated with random objects, including randomized integers, strings, characters, and floats.

*The test will first add all randomized tuples to both tuple spaces, then remove all of the same tuples, printing the working-time to the console. Tests have proven that Trie Tuple Space far exceeds Basic Tuple Space in performance. Trie Tuple Space time complexity for all operations = O(M) where M is the size of the tuple. Basic Tuple Space time complexity for all operations = O(N * M) where N is the number of tuples in the TupleSpace already and M is the size of the tuple.*

TupleSpace (interface)

TupleSpace interface that creates a contract with all implementations of tuplespace to assure the following methods are used:

add(Tuple tuple)

remove(Object...obj)

read(Object...obj)

BasicTupleSpace

Basic tuple space implementation. Uses an array list to store all tupleSets. Initializes a tupleSet of size 1 through N, where N is passed in the BTS constructor. The following methods are implemented:

add(Tuple tuple) - adds a tuple to the tuple space by calling tupleSet add().

remove(Object...obj) - removes a tuple with matching pattern by calling tupleSet readOrRemove() with remove flag on.

read(Object...obj) - returns a tuple with matching pattern by calling tupleSet readOrRemove() with remove flag off.

BasicTupleSet

Basic tuple set implementation. Creates an array list to store each tuple in. The following primary functions are called in the class:

add(Tuple tuple) - adds a tuple to the tuples array list as long as there is not already a matching tuple in the list.

readOrRemove(boolean remove, Object...obj) - searches array list for a matching pattern (when compared to the passed object array). If a match isn't found, the method returns null. If a match is found, the found tuple is returned. If the remove flag is set to true, that same tuple will also be removed from the tuples array list.

TrieTupleSpace

Tuple space implementation that uses a trie to store all of its tuples. Uses an array list to store all tupleSets. Initializes a tupleSet of size 1 through N, where N is passed in the TTS constructor. The following methods are implemented:

add(Tuple tuple) - adds a tuple to the tuple space by calling tupleSet add().

remove(Object...obj) - removes a tuple with matching pattern by calling tupleSet remove().

read(Object...obj) - returns a tuple with matching pattern by calling tupleSet read().

TrieTupleSet

The main tuple space is partitioned into tuple sets. A trie data structure is initialized for each tuple set. Each tuple set simply calls the add, remove, and read methods in the trie.

Trie

This class defines a Trie (retrieval tree) structure. The trie consists of trie nodes (defined by a hashmap with key = tuple object and value = hashmap of children). The trie data structure has the follow methods:

add(Tuple tuple) - adds each element of the passed tuple to the trie.

remove(Object[]) - removes pattern from trie if found, then returns the pattern as a Tuple object (or null if no matching pattern found).

read(Object[]) - looks for pattern passed through Object array, if found returns the pattern as a Tuple object. (or null if no matching pattern is found).

TrieNode

Each trie node has a hashmap that holds all of their child nodes, a boolean isTuple field that is marked true if the node is the last node in an added tuple, and an Object that defines the value of a certain Tuple element. Some important public methods include:

addChild(TrieNode child) - which adds a new node to this node's set of children.

removeChild(TrieNode child) - which removes a node from this node's set of children.

getSetOfKeys() - returns all keys of children as a set.

Tuple

Tuple objects hold a list of Objects. These can range from Strings, to ints, to booleans, to newly defined types. A tuple's size is fixed, may contain duplicates, but is unchangeable once created. Three important public methods are used in this class:

getSize() - returns k where the tuple in question is a k-tuple.

get(int index) - returns the Object in the array list, at the passed index.

convertToObjArr() - returns an Object[] of the array list tuple.

Coordinator

Coordinator for chat room program. Creates a tuple space and systemUser (which creates users on command). The program starts by giving the systemUser a turn. While it's the systemUser's turn, commands can be entered in the command line as follows:

"/create"	*- Create new user (and log them in).*
"/logon"	*- Log user onto the chat room.*
"/pass"	*- Pass control to first user (or return control to user that called "/system").*
"/kill"	*- Kill chat room program.*

When /pass is called, the program will then pass control to the first active user, which can execute the following commands:

"/printactive"	*- Print all active users to console.*
"/printall"	*- Print all users to console.*
"/logoff"	*- Log off this user.*
"/send"	*- Send message to all users.*
"/pass"	*- Pass control to next user.*
"/system"	*- Pass control to System.*

A user may printactive, printall, or call system as many times as they like during their turn. Once they send a message, log off, or pass, their turn is over and will pass control to the next user (or back to themselves if there is only one active user). When the user's turn is ended, the chat room's message log will be printed to the console. This cycle continues until systemUser "/kill" command is executed. If all users become inactive at any point, systemUser will re-gain control.

SystemUser

System user is created only one time. System user gets the first turn, and a turn each time a user types "/system". System user also gets a turn if all users become inactive. System User has the following public methods:

createNewUser(String userName) - creates a new user by initializing a new User object, then creates a user tuple and adds it to the tuple space in the following format:

[(User)UserReference, (String)UserName, (int)UserID, (boolean)ActiveStatus]

logOnUser(String userName) - changes a user's active status to true.

User0 through UserN (where there are N+1 users)

Users are initialized with a username, user ID, and passed a reference of the tuple space. Users can call the methods:

logoff() - sets their Tuple container's active status to false.

printActiveUsers() - prints all users with active status to console.

printAllUsers() - prints all users to console.

sendMessage() - creates a tuple (and adds it to the tuple space) for a message in the following format:

[(String)UserName, (String)TimeStamp, (String)Message, (int)MessageID]

printMessageLog() - prints the most recent 10 messages sent by all users in chronological order.