Assignment 2

Team

James Juan Whei Tan - 40161156

Vithushon Maheswaran - 27052715

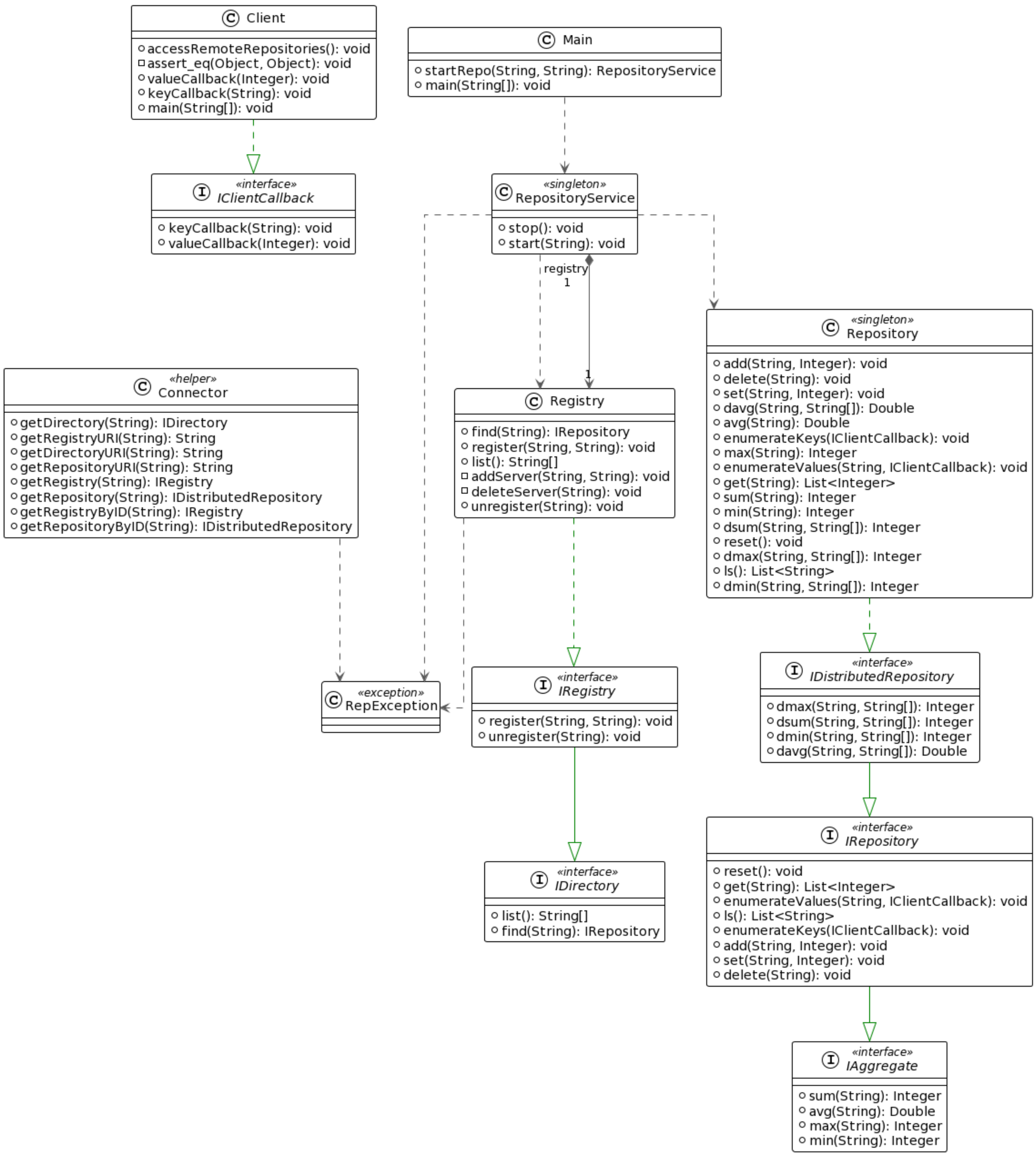
VCS

Development of the project was carried out in a Github repository.

Implementation Platform

The implementation platform is Java

Class diagram


Client
accessRemoteRepositories(): void
assert_eq(Object, Object): void
valueCallback(Integer): void
keyCallback(String): void
main(String[]): void
Main
startRepo(String, String): RepositoryService
main(String[]): void
«interface»
IClientCallback
keyCallback(String): void
valueCallback(Integer): void
«singleton»
RepositoryService
stop(): void
start(String): void
registry
1
1
«singleton»
Repository
add(String, Integer): void
delete(String): void
set(String, Integer): void
davg(String, String[]): Double
avg(String): Double
enumerateKeys(IClientCallback): void
max(String): Integer
enumerateValues(String, IClientCallback): void
get(String): List<Integer>
sum(String): Integer
min(String): Integer
dsum(String, String[]): Integer
reset(): void
dmax(String, String[]): Integer
ls(): List<String>
dmin(String, String[]): Integer
«helper»
Connector
getDirectory(String): IDirectory
getRegistryURI(String): String
getDirectoryURI(String): String
getRepositoryURI(String): String
getRegistry(String): IRegistry
getRepository(String): IDistributedRepository
getRegistryByID(String): IRegistry
getRepositoryByID(String): IDistributedRepository
Registry
find(String): IRepository
register(String, String): void
list(): String[]
addServer(String, String): void
deleteServer(String): void
unregister(String): void
«exception»
RepException
«interface»
IRegistry
register(String, String): void
unregister(String): void
«interface»
IDistributedRepository
dmax(String, String[]): Integer
dsum(String, String[]): Integer
dmin(String, String[]): Integer
davg(String, String[]): Double
«interface»
IDirectory
list(): String[]
find(String): IRepository
«interface»
IRepository
reset(): void
get(String): List<Integer>
enumerateValues(String, IClientCallback): void
ls(): List<String>
enumerateKeys(IClientCallback): void
add(String, Integer): void
set(String, Integer): void
delete(String): void
«interface»
IAggregate
sum(String): Integer
avg(String): Double
max(String): Integer
min(String): Integer

Repository Core

A repository allows the storage of values (**integers only** for the sake of simplifying things) corresponding to keys (any string is a valid key as long as it does not contain a dot).

The Repository Core package defines the core interfaces that are implemented in the service implementation.

Repository Implementation (Basic Functionality)

Set

Make a particular key correspond to a list containing just the given value.

Add

Adds a value to the list of values that correspond to a particular key. If the key does not exist yet, then a new list will be created for it.

List

Lists all keys

Get

Returns a particular list of values corresponding to a particular key, If the key does not exist, returns error. (Returns an empty list if the key does not exist)

Sum

Returns the sum of values corresponding to a particular key. This returns 0 if the key does not exist.

Delete

Delete a particular key-value pair by specifying the key. This is a no-op if the key does not exist.

Min

Returns the minimum value corresponding to a particular key. This returns `null` if the key does not exist.

Max

Returns the maximum value corresponding to a particular key. This returns `null` if the key does not exist.

Avg

Returns the average value (in floating point) corresponding to a particular key. This returns 0.0 if the key does not exist.

Reset

Wipes out the entire repository

Enum Keys

By receiving a callback, enumerates all keys currently stored in the repository.

Enum Values

By receiving a key, and a callback, enumerates all values associated with the given key, currently stored in the repository.

Repository Implementation (Extended Functionality)

By calling the method `aggregate()`, the client informs the Distributed Repository that which repositories are to be used.

Distributed sum

`Integer dsum(String key, String[] repids);`

Returns the sum of values corresponding to a particular key. This returns 0 if the key does not exist.

This does not return an error if the key does not exist on any of the servers, if the key is absent on all the servers the sum would be 0.

e.g.

```
r1.dsum("a", new String[] { "R2", "R3" })
```

Distributed average

Suppose that the client is connected to `R1` and wants to find the average value of `a` across the repositories `R1`, `R2`, and `R3`. The below command should be used

`Double davg(String key, String[] repids);`

e.g.

```
r1.davg("a", new String[] { "R2", "R3" })
```

Distributed min/max

Suppose that the client is connected to `R1` and wants to find the min/max value of `a` across the repositories `R1`, `R2`, and `R3`. The below command should be used

`Integer dmin(String key, String[] repids);`

`Integer dmax(String key, String[] repids);`

e.g.

```
r1.dmin("a", new String[] { "R2", "R3" })
```

Service Deamon

The deamon registers the RMI repository. The service uses the distribution module, as specified in section 4.4 of the Assignment specification. Upon stopping the service, the service unregisters itself from the network of known repositories

Repository Distribution and Registry

By extending the `IDirectory` interface, the `IRegistry` interface provides a distributed registry mechanism for all repositories. Each repository registers itself through the `IRegistry.register()` method. Each repository is responsible to keep track of all other registries.

Peer Registration

Implemented using RMI, every repository has a registry that keeps track of all active peers. When a new node joins the network, it communicates its entry to a known peer. The said peer takes note of the addition of this new node and spreads this information to its other peers. This propagation of information only occurs when a new and unknown peer registers itself to a node, i.e. if a node encounters the registration of an already registered peer, the request is ignored. This allows us to achieve a full mesh and all repositories will have references to all the other repositories.

Client

The client may connect to any repository. Access to all repositories are available through the `IDistributedRepository` interface. This interface provides local access to the local repository, including the local aggregate method(s). Additionally, the aggregate interface enables the client to query aggregate methods over multiple repositories.

Error Handling

Errors are handled in the repository implementation. In case of errors, `RepException` is thrown, by which, the exception is eventually caught and processed by the client.