



Assignment 3

Assignment 3

Due date

- 11.59 PM EST on October 29th
- Submit your code as per the provided instructions.

Updates

Github link

<https://classroom.github.com/a/kn2pIRpQ>

Assignment Goal

Apply the design principles you have learned so far to develop software for the given problem.

Team Work

- You need to work alone on this assignment.


Programming Language

You are required to use Java.

Compilation Method

- You are required to use ANT to compile the code.

Policy on sharing of code

-  EVERY line of code that you submit in this assignment should be written by you. Do NOT show your code to any other group. Our code-comparison software can very easily detect similarities.
- Post to piazza if you have any questions about the requirements. Do NOT post your code asking for



Project Description

Load Balancer

We have services that we want to host on the cloud. To guarantee high availability, multiple instances of the services are hosted on multiple machines in the cluster and a load balancer is placed to route incoming requests to any one of the instances.

The number of machines constituting the cluster is not fixed. New machines can be added to the cluster or some existing machines can be removed from the cluster.

A user can request for a service by providing the name of the service (ex: binghamton.csx42.grades). This is fed to the load balancer that returns the following.

- URL for the service.
- Hostname of the machine on which an instance of the service is running.

The goal of the assignment is to design an observer of the cluster and guarantee that a consistent mapping of service URLs to host names is maintained.

Machine

A machine has a hostname and a set of hosted services keyed by their respective names. A machine can be considered to be an instance of the below class.

```
public class Machine {
    private String hostname;
    // Service name to hosted services.
    private Map<String, Service> hostedServices;

    // Rest of the code.
}
```

Service

Each service has a name and a URL associated to it.

A service can be considered to be an instance of the below class.

```
public class Service {
    // Service URL.
    private String URL;
    // Service name.
    private String name;

    // Rest of the code.
}
```

The following operations can be performed with regard to services.

- **SERVICE_OP__ADD_SERVICE** <service name> <URL> <host₁, host₂, ... host_r>
 - Add a service with the given name to a machine in the cluster with the given hostname.
 - If a machine does not exist for any of the given hostnames, then do not add the service and write a message stating the same to the Results class and continue.
 - If a service with the given name already exists on the cluster, then write a message stating the same the Results class.
 - On successful processing, write the message **Service Added** to the Results class.
- **SERVICE_OP__REMOVE_SERVICE** <service name>





- On successful processing, write the message **Service Removed** to the Results class.
- **SERVICE_OP__ADD_INSTANCE** <service name> <host name>
 - Add an instance of the service on the machine with the given hostname.
 - If an instance of the service already exists on the machine with the given hostname, then (1) do not add the instance to the machine and (2) write a message stating the same to the Results class and continue.
 - If the service was not previously added to the cluster using SERVICE_OP__ADD_SERVICE operation, then (1) do not add the instance to the machine and (2) write a message stating the same to the Results class and continue.
 - On successful processing, write the message **Instance Added** to the Results class.
- **SERVICE_OP__REMOVE_INSTANCE** <service name> <host name>
 - Remove the instance of the service on the machine with the given hostname.
 - If no instance of service is present on the machine with the given hostname, then write a message stating the same to the Results class and continue.
 - After removing the instance, if no more instances of the service exist anywhere in cluster, mark the service as *inactive* but **do not remove it**. *Note that a service with ≥ 1 instances is considered as an active service.*
 - On successful processing, write the message **Instance Removed** to the Results class.

Cluster

You can assume that an instance of the class shown below represents the cluster.

```
public class Cluster {
    // Hostnames to corresponding machine instances.
    private Map<String, Machine> machines;

    // Rest of the code.
}
```

Note: There should be only one cluster.

Note: The operations pertaining to services are performed on the cluster.

The following operations can also be performed on the cluster.

- **CLUSTER_OP__SCALE_UP** <hostname of new machine>
 - Add new machine with the given hostname to the cluster.
 - If a machine with the given hostname already exists, then write a message stating the same to the Results class.
 - On successful processing, write the message **Cluster Scaled Up** to the Results class.
- **CLUSTER_OP__SCALE_DOWN** <hostname of machine to remove>
 - Remove machine with the given hostname from the cluster.
 - If no machine with the given hostname exists, then write a message stating the same to the Results class and continue.
 - On successful processing, write the message **Cluster Scaled Down** to the Results class.

Requests and Load Balancing

Requests to a service specify the name of the service.

Requests are submitted using the syntax **REQUEST** <Service name>.

A load balancer maintains an internal data structure (**please see the requirements for the data structure to be used in Guidelines and Tips section.**) to store the mapping of Service names to service managers.

Services and Service Managers share a 1:1 relationship with each other.

Each service manager stores the URL and the hostnames of machines on which an instance of the



```

        // Index to find the URL and hostname for a given service name.
        //
        // Trie is optional for under-graduate students.
        // Graduate students have to use a Trie datastructure.
        private Trie ServiceURLAndHostnameFetcher;

        // Cluster on which the services are hosted.
private Cluster cluster;

        // Rest of the code.
    }

    public class Trie {
        // Code for Trie data structure.
        // Used as a key value store.
        // Key is the service name. Value is an instance of ServiceManager.

        // Rest of the code.
    }

    public class ServiceManager {
        private String key;

        // Information pertaining to the service.
        private String URL;
        private List<String> hostnames;

        // Rest of the code.
    }

```

Requests are submitted to the load balancer searches the internal data structure to fetch a hostname of the machine on which an instance of the service with the given name is running on.

If the service has been marked inactive (no instances left on the cluster), then print a message stating

Service Inactive - Service::<service name>

If the service name is invalid, then write a message stating **Invalid Service** to the Results class and continue.

The service manager selects the hostname, from the available list of hosts for a service, in a round-robin fashion. The selected hostname and URL are returned back to the load balancer.

The hostname and URL returned from the service manager are returned by the load balancer as the response to the request.

On successful processing of each request, the URL and the hostname should be written to the Results class with the format **Processed Request - Service_URL::<URL> Host::<hostname>**

Implement the observer pattern to capture this design and make sure that the load balancer always provides the URL and hostname of a machine on which an instance of the service is running.

Guidelines and Tips

1. The cluster is the one being observed so it is Subject.
2. Service managers maintain service information and an up to date list of available hosts for that service. A service manager is therefore an Observer of the cluster.
3. The load balancer maintains an index of all the available service names for which requests can be made. The load balancer is therefore also an Observer of the cluster.
4. **Trie data structure is optional for under-graduate students. Instead, they can use a Map.**
5. Load balancer should be notified when the following operations are performed on the cluster.
 - SERVICE_OP_ADD_SERVICE
 - SERVICE_OP_REMOVE_SERVICE
6. Service managers registered as observers should be notified when the following operations are performed on the cluster.
 - CLUSTER_OP_SCALE_DOWN



8. Other than CLUSTER_OP__SCALE_UP operation, when performing any of the operations the registered observers need to be notified of the same. When notifying the observers, necessary information will need to be passed.

Control Flow for Reference

- CLUSTER_OP__SCALE_UP
 1. Call specific method on cluster.
 2. Cluster adds a new machine to the available machines.
- SERVICE_OP__ADD_SERVICE
 1. Call specific method on cluster.
 2. Cluster adds an instance of the service to all the machines with the given hostnames.
 3. Notify registered observers by passing the name of the operation, service name, service URL, and the hostnames of the machines which each host an instance of the service.
 4. Observers update their state.
- SERVICE_OP__REMOVE_SERVICE
 1. Call specific method on cluster.
 2. Cluster removes all instances of the service.
 3. Notify registered observers by passing the name of the operation and the name of the service that was removed.
 4. Observers update their state.
- CLUSTER_OP__SCALE_DOWN
 1. Call specific method on cluster.
 2. Cluster removes machine from the available machines.
 3. Notify registered observers by passing the name of the operation and the hostname of the machine removed from the cluster.
 4. Observers update their state.
- SERVICE_OP__ADD_INSTANCE
 1. Call specific method on cluster.
 2. Cluster adds instance of service to the machine with the given hostname
 3. Notify registered observers by passing the name of the operation and the hostname of the machine on which an instance of the service was added.
 4. Observers update their state.

