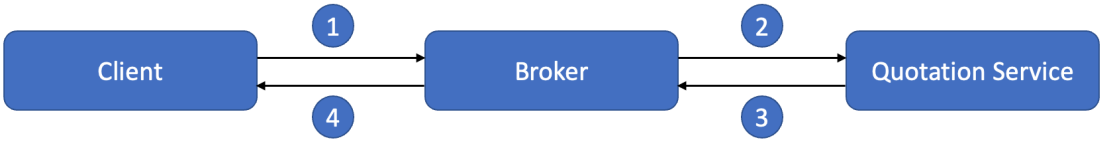
Outline of LifeCo Insurance System Scenario

Introduction

You have recently started working for an insurance broker and have joined the team responsible for creating a web-based life insurance portal.

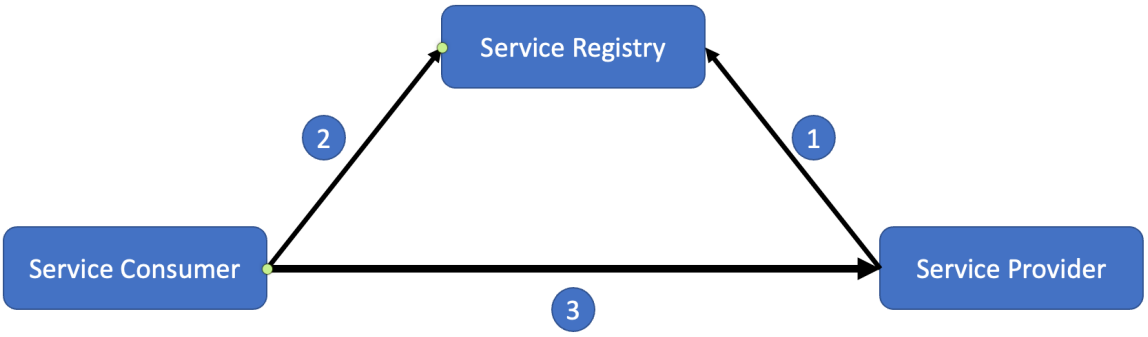


The idea is to create a distributed system that implements a broker service. As is illustrated in the figure above, the purpose of this service is to (1) receive customer information from a client and then (2) forward that information on to insurance quotation services that will be hosted by the various insurance companies that have partnered with your company. These services will receive the customer information and use it to generate a quote. The individual quotes will then (3) be returned to the broker which will (4) return a collated set of quotations to the client for presentation to the customer.

One of the complications is that there will be more than one quotation service. Currently, the insurance company has 3 insurance partners: **Girls Allowed Incorporated**, **Dodgy Geezers Corporation**, and **Auld Fellas Limited**. Your boss requires that the system attempt to get a quotation from each of these services before they are returned to the client.

Architectural Decision: Service-Oriented Architecture

The companies software architect has decided that a service-oriented approach should be adopted. Unfortunately, the architect doesn't really know what this means except that the resulting code should be as decoupled as possible. They also understand that the system should be broken into 3 types of component: service consumers (code that uses a service), service providers (code that implements a service), and a service registry (a place where service providers advertise the existence of the services they provide and service consumers discover the services they want to use).



As is indicated in the figure above, there are 3 main steps involved in service-oriented architecture: (1) the service provider registers a service with the service registry, (2) the service consumer explores the service registry looking for a service that it needs to use, and

(3) one a relevant service is located, the service consumer directly interacts with the service provider to access the service.

To support this service, the architect develops a simple service-oriented architecture that can be used by the team. This architecture consists of two classes that are organised into the `service.registry` package:

- `ServiceRegistry.java`: This class implements the service registry. The architect has decided that the registry will map service names (strings) to services and all information about the service will be encoded in the service name. To deliver this, the service will provide 4 basic functions:
  - `bind(<service-name>, <service>)`: registers the service under the given name.
  - `unbind(<service-name>)`: removes the service with the given name from the registry.
  - `lookup(<service-name>)`: locate the service with the given name (if it exists).
  - `list()`: return a list of the service names for all the currently registered services.
- `Service.java`: An empty java interface that must be implemented by all classes that implement a service that is to be registered with the service registry.

A prototype application was built using this architecture, but unfortunately, the architect quit their job before the distribution could be added or even designed.

The Problem

You have been given the enjoyable task of understanding the code that was written, and finding the best way to distribute it. Your boss has given you a list of several technologies they would like you to consider using. They have asked you to develop solutions using each one so that the team can compare them and decide which is the best.

As a starting point, you have been told to go an read the code to understand what the current prototype does…