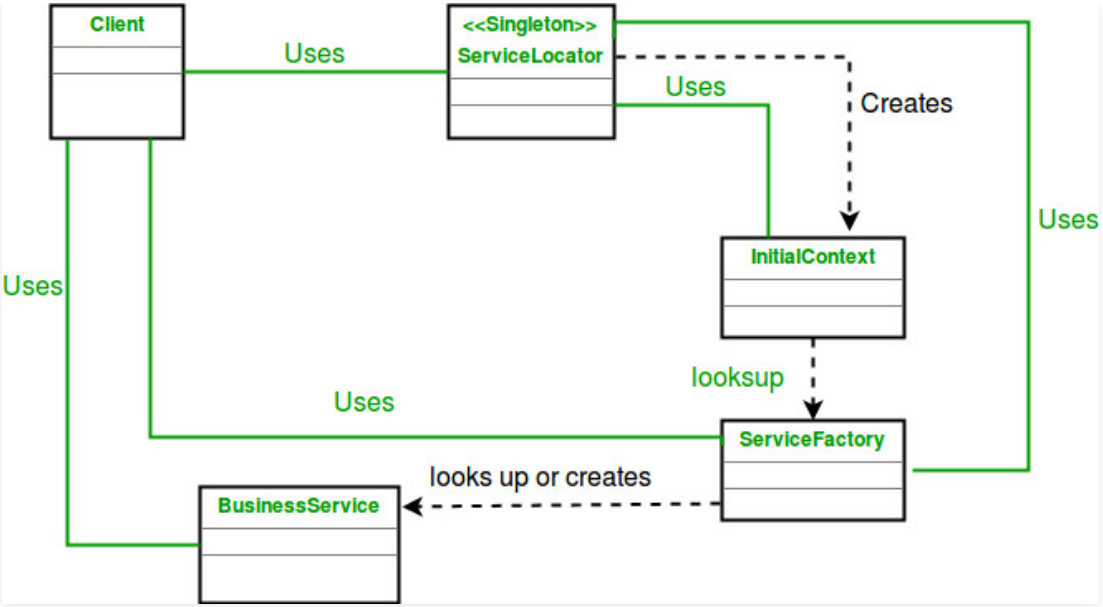
Service Locator Pattern

Difficulty Level : Medium • Last Updated : 06 Mar, 2018

The service locator pattern is a design pattern used in software development to encapsulate the processes involved in obtaining a service with a strong abstraction layer. This pattern uses a central registry known as the "service locator" which on request returns the information necessary to perform a certain task.
The ServiceLocator is responsible for returning instances of services when they are requested for by the service consumers or the service clients.

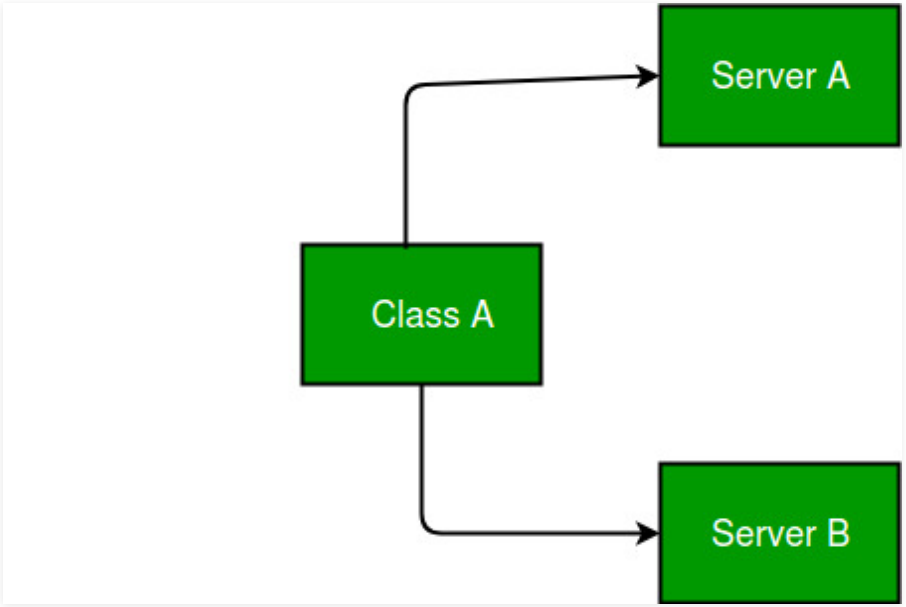
UML Diagram Service Locator Pattern



Design components

- **Service Locator :** The Service Locator abstracts the API lookup services, vendor dependencies, lookup complexities, and business object creation, and provides a simple interface to clients. This reduces the client's complexity. In addition, the same client or other clients can reuse the Service Locator.
- **InitialContext :** The InitialContext object is the start point in the lookup and creation process. Service providers provide the context object, which varies depending on the type of business object provided by the Service Locator's lookup and creation service.
- **ServiceFactory :** The ServiceFactory object represents an object that provides life cycle management for the BusinessService objects. The ServiceFactory object for enterprise beans is an EJBHome object.
- **BusinessService :** The BusinessService is a role that is fulfilled by the service the client is seeking to access. The BusinessService object is created or looked up or removed by the ServiceFactory. The BusinessService object in the context of an EJB application is an enterprise bean.

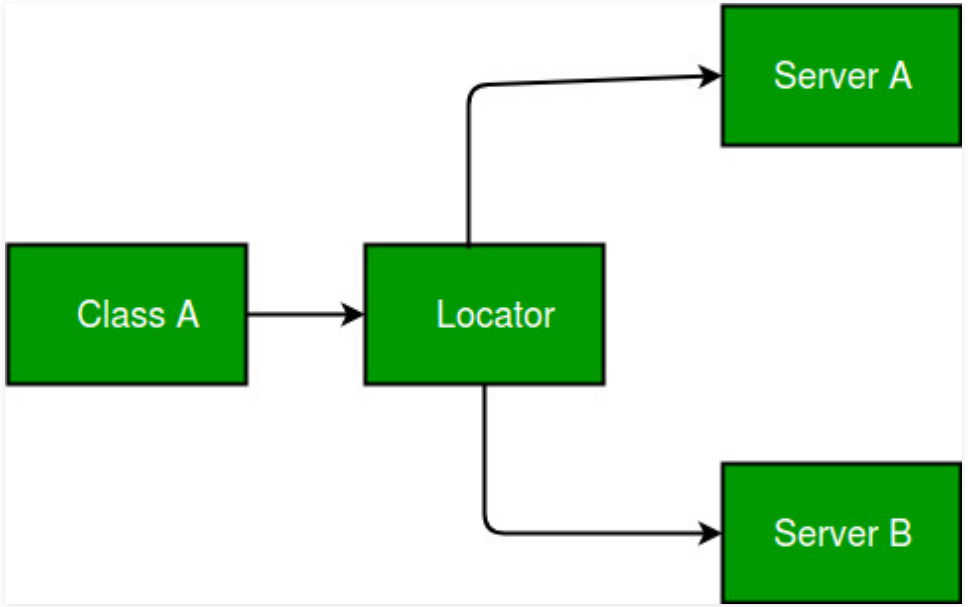
Suppose classes with dependencies on services whose concrete types are specified at compile time.



In the above diagram, ClassA has compile time dependencies on ServiceA and ServiceB.But this situation has drawbacks.

- If we want to replace or update the dependencies we must change the classes source code and recompile the solution.
- The concrete implementation of the dependencies must be available at compile time.

By using the Service Locator pattern :



In simple words, Service Locator pattern does not describe how to instantiate the services. It describes a way to register services and locate them.

Let's see an example of Service Locator Pattern.

```
// Java program to
// illustrate Service Design Service
// Locator Pattern
```

```
import java.util.ArrayList;
import java.util.List;

// Service interface
// for getting name and
// Executing it.

interface Service {
    public String getName();
    public void execute();
}

// Service one implementing Locator
class ServiceOne implements Service {
    public void execute()
    {
        System.out.println("Executing ServiceOne");
    }

    @Override
    public String getName()
    {
        return "ServiceOne";
    }
}

// Service two implementing Locator
class ServiceTwo implements Service {
    public void execute()
    {
        System.out.println("Executing ServiceTwo");
    }

    @Override
```

```
    public String getName()
    {
        return "ServiceTwo";
    }
}

// Checking the context
// for ServiceOne and ServiceTwo
class InitialContext {
    public Object lookup(String name)
    {
        if (name.equalsIgnoreCase("ServiceOne")) {
            System.out.println("Creating a new ServiceOne object");
            return new ServiceOne();
        }
        else if (name.equalsIgnoreCase("ServiceTwo")) {
            System.out.println("Creating a new ServiceTwo object");
            return new ServiceTwo();
        }
        return null;
    }
}

class Cache {
    private List<Service> services;

    public Cache()
    {
        services = new ArrayList<Service>();
    }

    public Service getService(String serviceName)
    {
        for (Service service : services) {
            if (service.getName().equalsIgnoreCase(serviceName)) {
```

```
                System.out.println("Returning cached "
                                   + serviceName + " object");
                return service;
            }
        }
        return null;
    }

    public void addService(Service newService)
    {
        boolean exists = false;
        for (Service service : services) {
            if (service.getName().equalsIgnoreCase(newService.getName())) {
                exists = true;
            }
        }
        if (!exists) {
            services.add(newService);
        }
    }
}

// Locator class
class ServiceLocator {
    private static Cache cache;

    static
    {
        cache = new Cache();
    }

    public static Service getService(String name)
    {
        Service service = cache.getService(name);
```

```
            if (service != null) {
                return service;
            }

            InitialContext context = new InitialContext();
            Service ServiceOne = (Service)context.lookup(name);
            cache.addService(ServiceOne);
            return ServiceOne;
        }
    }

    // Driver class
    class ServiceLocatorPatternDemo {
        public static void main(String[] args)
        {
            Service service = ServiceLocator.getService("ServiceOne");
            service.execute();

            service = ServiceLocator.getService("ServiceTwo");
            service.execute();

            service = ServiceLocator.getService("ServiceOne");
            service.execute();

            service = ServiceLocator.getService("ServiceTwo");
            service.execute();
        }
    }
```

Output:

```
Creating a new ServiceOne object
Executing ServiceOne
```

```
Creating a new ServiceTwo object
Executing ServiceTwo
Returning cached ServiceOne object
Executing ServiceOne
Returning cached ServiceTwo object
Executing ServiceTwo
```

Advantages :

- Applications can optimize themselves at run-time by selectively adding and removing items from the service locator.
- Large sections of a library or application can be completely separated. The only link between them becomes the registry.

Disadvantages :

- The registry makes the code more difficult to maintain (opposed to using Dependency injection), because it becomes unclear when you would be introducing a breaking change.
- The registry hides the class dependencies causing run-time errors instead of compile-time errors when dependencies are missing.

Strategies

The following strategies are used to implement service Locator Pattern :

- **EJB Service Locator Strategy :** This strategy uses EJBHome object for enterprise bean components and this EJBHome is cached in the ServiceLocator for future use when the client needs the home object again.
- **JMS Queue Service Locator Strategy :** This strategy is applicable to point to point messaging

requirements. The following the strategies under JMS Queue Service Locator Strategy.

- JMS Queue Service Locator Strategy
- JMS Topic Service Locator Strategy

- **Type Checked Service Locator Strategy :** This strategy has trade-offs. It reduces the flexibility of lookup, which is in the Services Property Locator strategy, but add the type checking of passing in a constant to the ServiceLocator.getHome() method.

Previous

Find closest number in array

Next

Puzzle | Haunted House Escape

RECOMMENDED ARTICLES

Page : 1 2 3

01 Design Video Sharing Service System like Youtube

05 Decorator Pattern | Set 1 (Background)
25, Apr 16

02, Feb 18

02 **Command Pattern**
06, May 16

03 **Observer Pattern | Set 1 (Introduction)**
04, Apr 16

04 **Observer Pattern | Set 2 (Implementation)**
05, Apr 16

06 **The Decorator Pattern | Set 2 (Introduction and Design)**
25, Apr 16

07 **Decorator Pattern | Set 3 (Coding the Design)**
26, Apr 16

08 **Strategy Pattern | Set 1 (Introduction)**
29, Apr 16

Article Contributed By :

saketkumr
@saketkumr

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Article Tags : Design Pattern

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Contact Us
Copyright Policy

CS Subjects
Video Tutorials

Practice

Courses
Company-wise
Topic-wise
How to begin?

Contribute

Write an Article
Write Interview Experience
Internships
Videos