1. QUESTION

[RESPECTABLE]

Proof of Concept to demonstrate an application designed using Dependency Injection design pattern. Expected the architecture and an implementation approach with a sample project

Prerequisites:

- DILauncher.java, Employee.java, SkillSets.java

Source Code GIT Repository:

Web Browser Link:

https://github.com/Ajay-Kumar-Aspiring-Minds-Round-2/DependencyInjectionApp

GIT Clone Link:

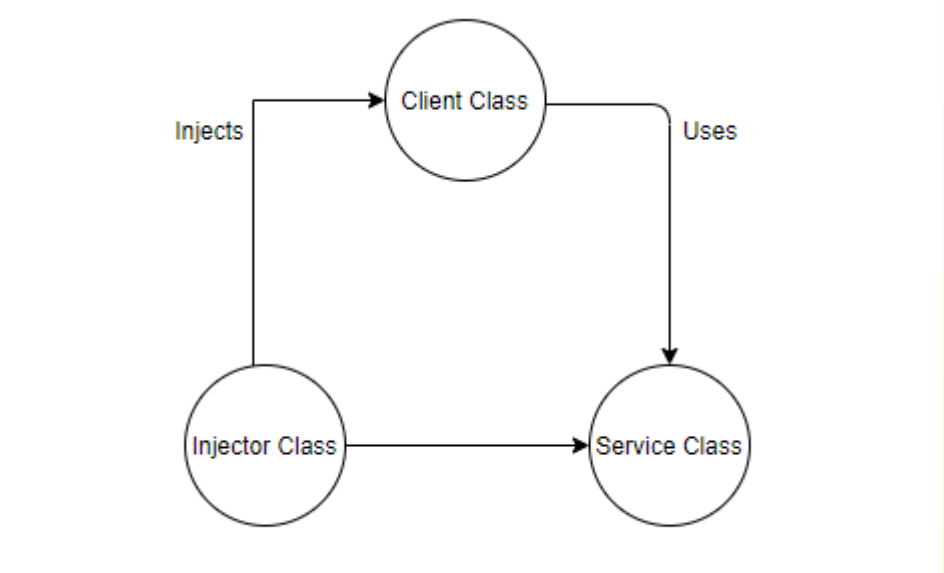
https://github.com/Ajay-Kumar-Aspiring-Minds-Round-2/DependencyInjectionApp.git

Technology Stack:

What is Dependency Injection?

- Dependency Injection is the ability of an object to supply dependencies of another object.
- Dependency in programming is an approach where a class uses specific functionalities of another class. So, for example, If you consider two classes A and B, and say that class A uses functionalities of class B, then its implied that class A has a dependency of class B. Now, if you are coding in Java then you must know that, you have to create an instance of class B before the objects are being used by class A.
- In the below diagram we can see that the injector class creates an object of the service class, and injects that object to a client object. In this way, the DI pattern separates the responsibility of creating an object of the service class out of the client class.

Below diagram shows the relationship between the classes (DI Components):



DI Components:

- **Client Class:** This is the dependent class and is dependent on the Service class.
- **Service Class:** This class provides a service to the client class.
- **Injector Class:** This class is responsible for injecting the service class object into the client class

Types of Dependency Injection:

There are mainly three types of Dependency Injection:

- **Constructor Injection:** In this type of injection, the injector supplies dependency through the client class constructor.
- **Setter Injection / Property Injection:** In this type of injection, the injector method injects the dependency to the setter method exposed by the client.

- **Interface Injection:** In this type of injection, the client class implements an interface which declares the methods to supply the dependency and the injector uses this interface to supply the dependency to the client class.

Benefits of using Dependency Injection:

- Helps in Unit testing.
- Boiler plate code is reduced, as initializing of dependencies is done by the injector component.
- Extending the application becomes easier.
- Helps to enable loose coupling, which is important in application programming.

Dependency Injection using Spring Core Container:

.

DILauncher.java

```java
package com.aspiringminds.event;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

import com.aspiringminds.event.di.Employee;

@SpringBootApplication
public class DILauncher {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(DILauncher.class, args);
        Employee emp = context.getBean(Employee.class);
        emp.display();
    }
}
```

- getBean(Employee.class) method is used in injecting/creating the object for Employee class in the application.
- Spring core container uses the bean factory to create new objects. New objects are generally created as Singletons if not specified differently.
- The above code snippet, tells the compiler to return an object of the Employee class.
- As we need an object of the Employee class, we need to mention the **@Component annotation**, in the Employee class.

Dependency Injection Using Autowired Annotation:

DILauncher.java | Employee.java

```java
@Component
public class Employee {

    private int empId;
    private String empName;
    private String deptName;
    @Autowired
    private SkillSets skillSets;

    public int getEmpId() {..
    public void setEmpId(int empId) {..
    public String getEmpName() {..
    public void setEmpName(String empName) {..
    public String getDeptName() {..
    public void setDeptName(String deptName) {..
    public SkillSets getSkillSets() {..
    public void setSkillSets(SkillSets skillSets) {..

    public void display() {
        System.out.println("Object Returned Successfully");
        skillSets.skillSetsDisplay();
    }
}
```

SkillSets.java

```java
@Component
public class SkillSets {

    private int skillSetId;
    private String skillSetName;

    public int getSkillSetId() {
        return skillSetId;
    }
    public void setSkillSetId(int skillSetId) {
        this.skillSetId = skillSetId;
    }
    public String getSkillSetName() {
        return skillSetName;
    }
    public void setSkillSetName(String skillSetName) {
        this.skillSetName = skillSetName;
    }

    public void skillSetsDisplay() {
        System.out.println("THIS IS SKILL SETS DISPLAY METHOD");
    }
}
```

Console

```
DILauncher [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (22-Jan-2021, 1:07:00 am)
2021-01-22 01:07:06.961  INFO 13988 --- [          main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [cla
2021-01-22 01:07:07.003  INFO 13988 --- [          main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler
2021-01-22 01:07:07.070  INFO 13988 --- [          main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2021-01-22 01:07:07.139  INFO 13988 --- [          main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2021-01-22 01:07:07.141  INFO 13988 --- [          main] com.aspiringminds.event.DILauncher       : Started DILauncher in 5.066 seconds (JVM runnin
Object Returned Successfully
THIS IS SKILL SETS DISPLAY METHOD
```

- Create two classes Employee and SkillSets.
- SkillSets class has a skillSetsDisplay() method which prints some text.
- Now we need to call the skillSetsDisplay() method in the Employee class, so we have to create an object of SkillSets class.
- To use the skillSetsDisplay() method, you have to call skillSets.skillSetsDisplay(); under the display method of the Employee class. Also, to make sure that the skillSets object is instantiated mention **@Component annotation** is the SkillSets class.

- **@Autowired annotation** need to be added for skillSets object reference in Employee class to be able to recognize SkillSets class and create an instance and access its methods.

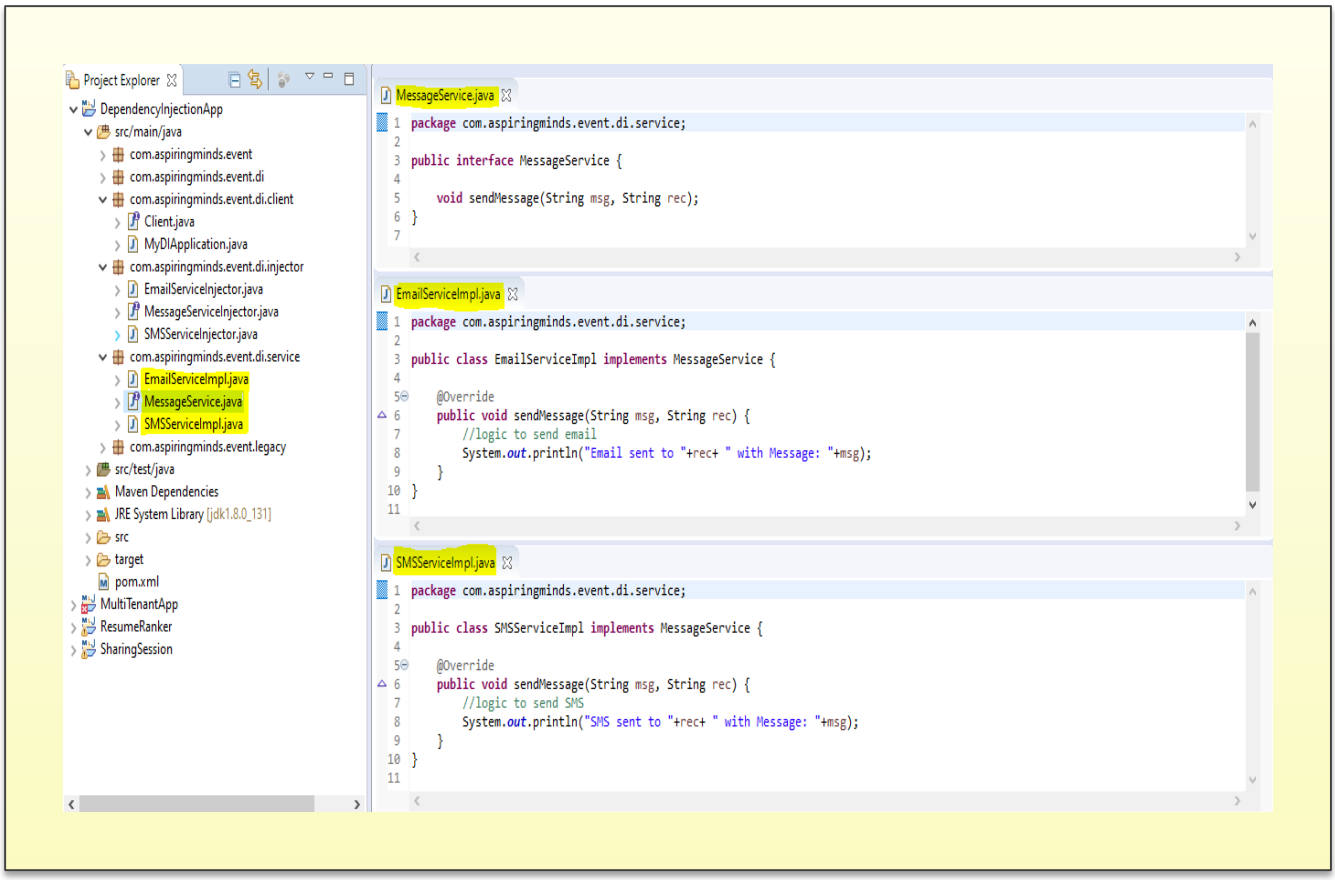
Java Dependency Injection:

Dependency Injection in java requires at least the following:

1. **Service components** should be designed with base class or interface. It's better to prefer interfaces or abstract classes that would define contract for the services.
2. **Client classes** should be written in terms of service interface.
3. **Injector classes** that will initialize the services and then the client classes.

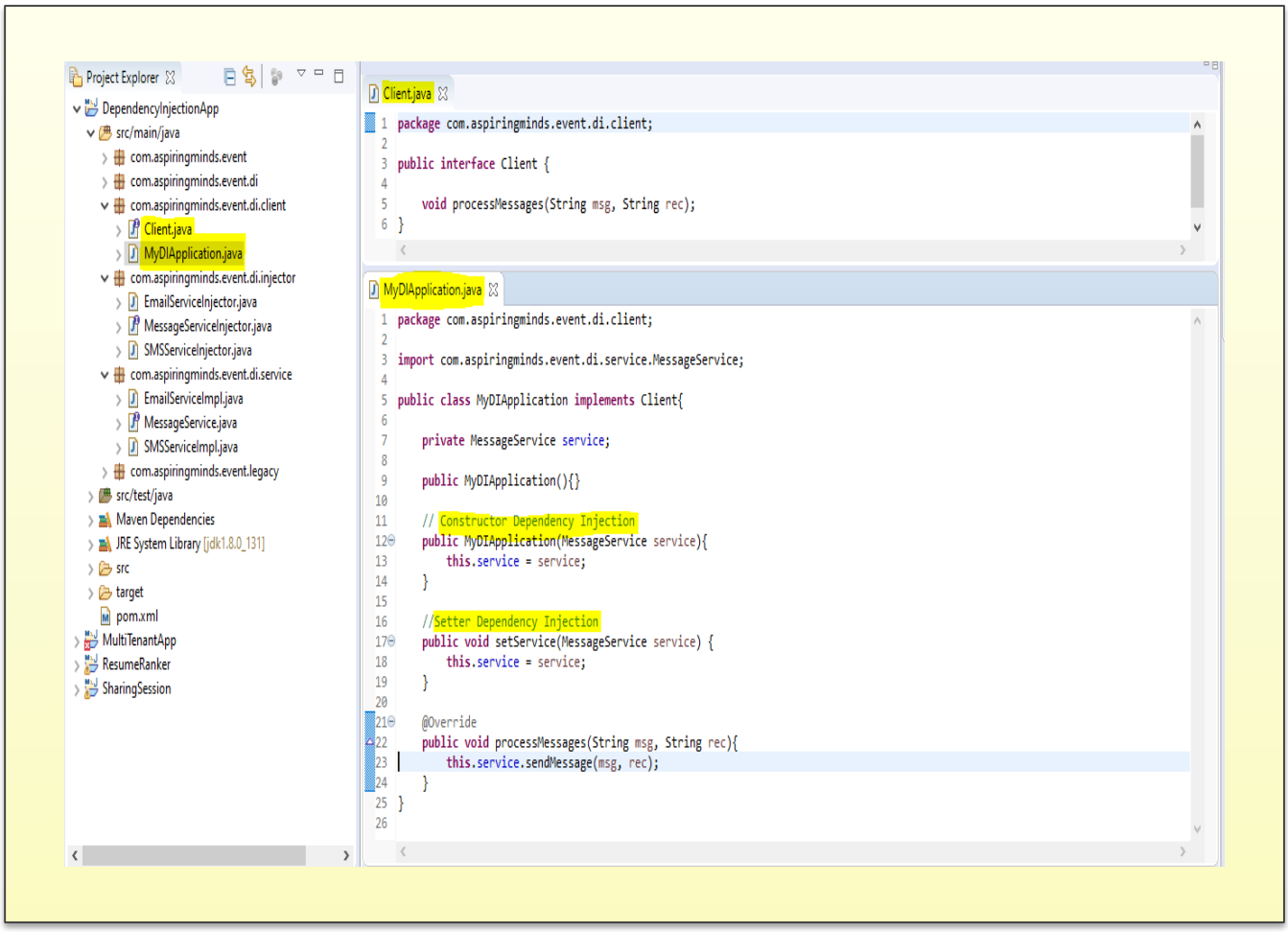
Java Dependency Injection – Service Components:

- We have the **MessageService** interface that will declare the contract for service implementations.
- We have **Email** and **SMS** service impl classes that implement the **MessageService** interface.



Java Dependency Injection – Client Classes:

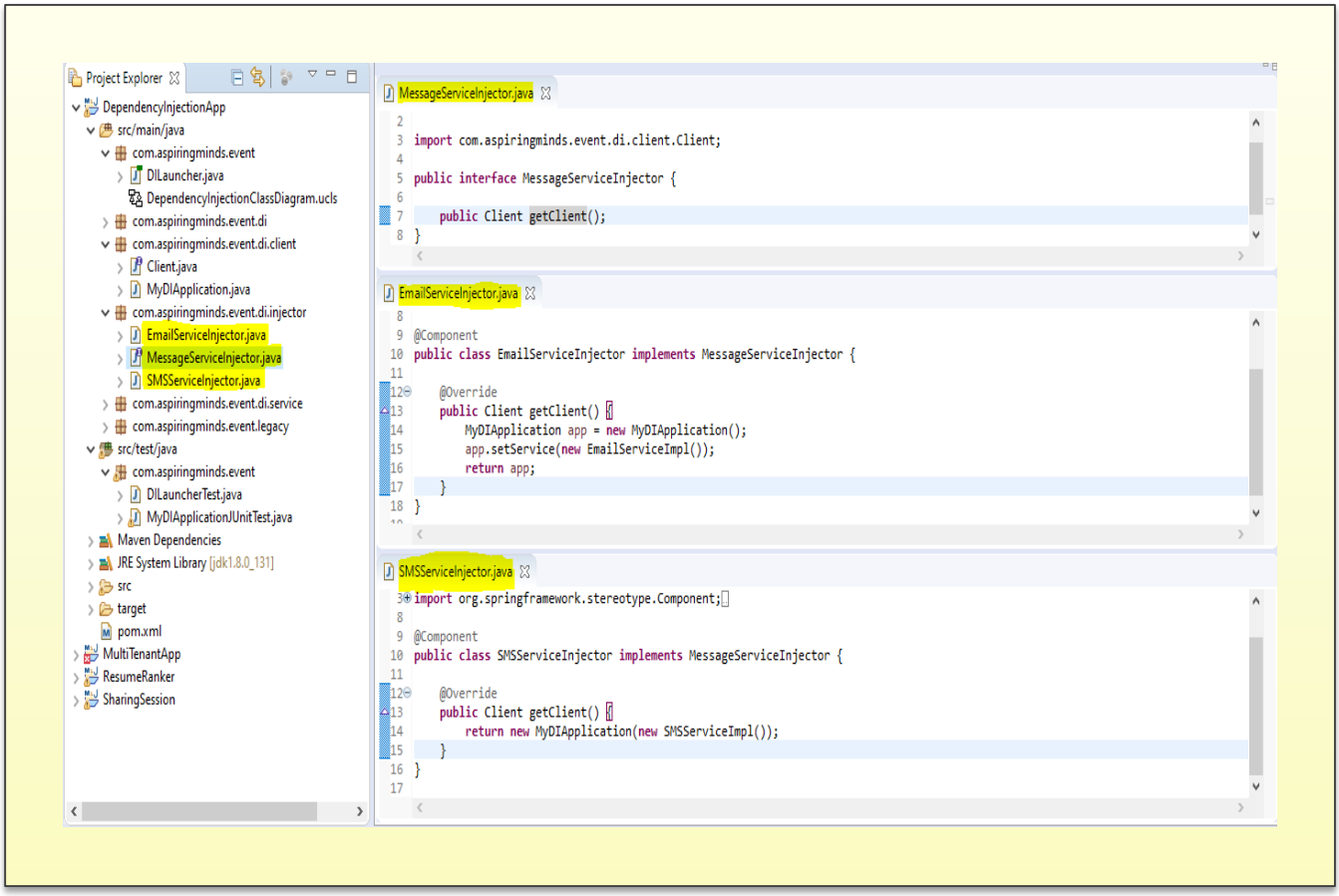
- We have a **Client** interface declaring contract for client classes.
- Here the application class is just using the service. It does not initialize the service that leads to better **"separation of concerns".** Also use of service interface allows us to easily test the application by mocking the **MessageService** and bind the services at runtime rather than compile time.



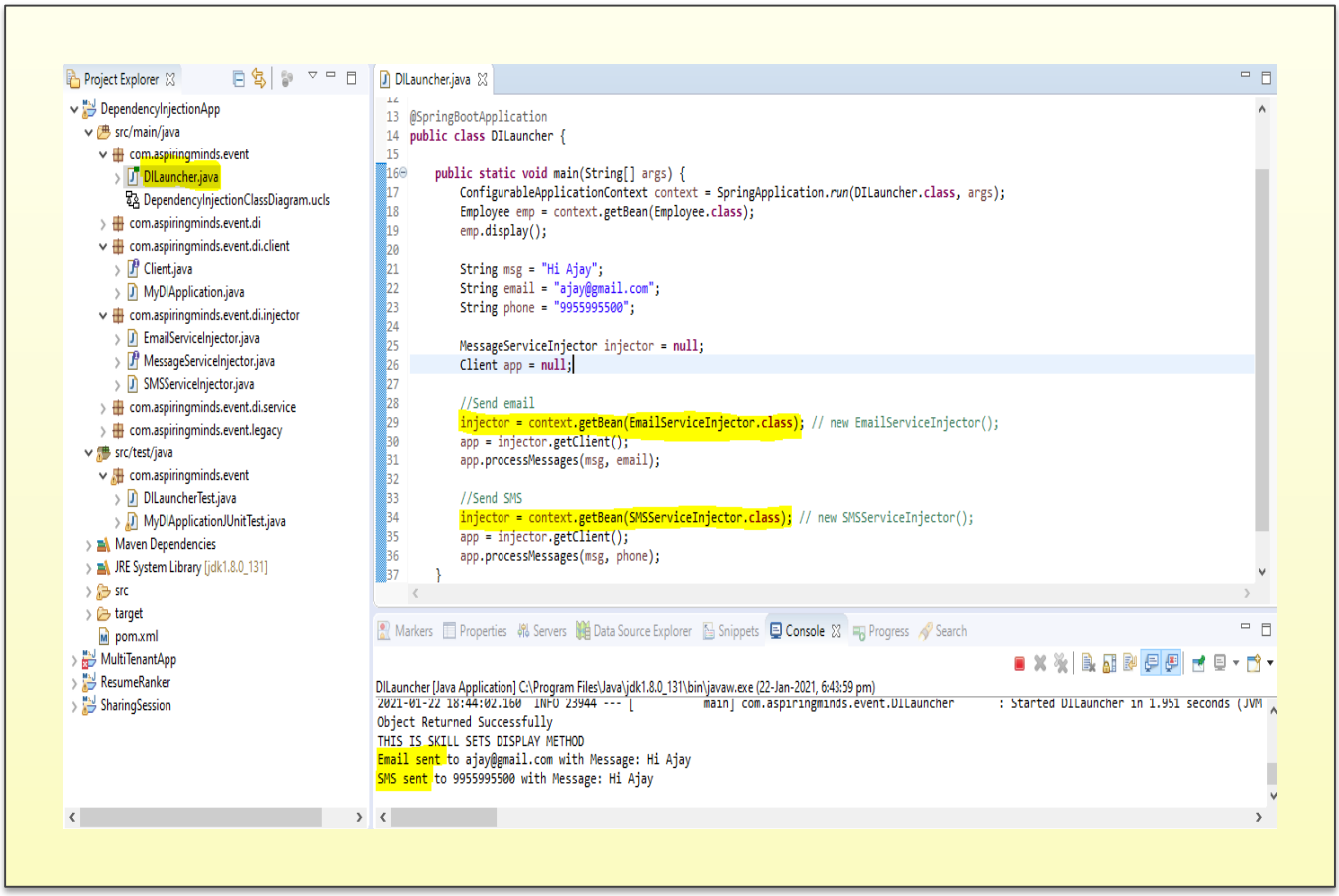
Java Dependency Injection – Injectors Classes:

- We have an interface **MessageServiceInjector** with method declaration that returns the **Client** class.
- For every service classes, we have to create injector classes.
- In Injector classes, we have initialized the **Service** and also the **Client** classes.

- We have used **setter** method to inject the EmailService.
- We have used **constructor** to inject the SMSService.



Application Launcher Class - DILauncher.java



Class Diagram: