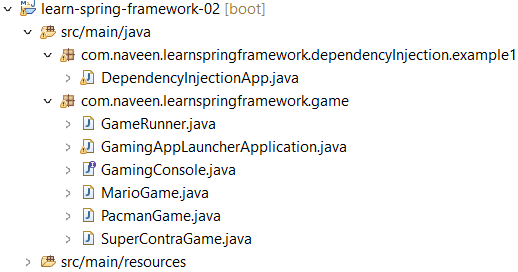
**DIFFERENT TYPES OF DEPENDENCY INJECTION**

**PROJECT STRUCTURE:**



**DependencyInjectionApp.java**

package com.naveen.learnspringframework.dependencyInjection.example1;

import java.util.Arrays;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import com.naveen.learnspringframework.game.GameRunner;

import com.naveen.learnspringframework.game.GamingConsole;

@Configuration

@ComponentScan // automatically scan on current package when we doesn't mention package.

public class DependencyInjectionApp {

    public static void main(String[] args) {

        try(var context =

                new AnnotationConfigApplicationContext

                    (DependencyInjectionApp.class)){

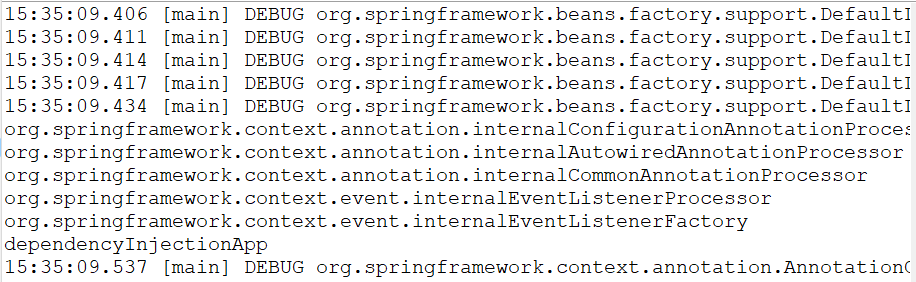
        Arrays.stream(context.getBeanDefinitionNames())

        .forEach(System.out::println);

        }

    }

}

OUTPUT:

In this case, **DependencyInjectionApp** is being treated as a bean by the Spring framework. When the **AnnotationConfigApplicationContext** is created with **DependencyInjectionApp.class** as a parameter, it registers the **DependencyInjectionApp** class as a bean in the Spring application context. The **context.getBeanDefinitionNames()** method call retrieves the names of all the beans in the context, which includes the name of the **DependencyInjectionApp** bean, and prints it to the console.

**FIELD INJECTION**

**Field injection** is one of the ways to perform dependency injection in the Spring framework. In field injection, the dependencies of a class are injected directly into its fields using Spring's **@Autowired** annotation.

**DependencyInjectionApp.java**

package com.naveen.learnspringframework.dependencyInjection.example1;

import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.stereotype.Component;

@Component

class YourBusiness{

    @Autowired

    Dependency1 dependency1;

    @Autowired

    Dependency2 dependency2;

    public String toString() {

        return "Using " + dependency1 + " and " + dependency2;

    }

}

@Component

class Dependency1{

}

@Component

class Dependency2{

}

@Configuration

@ComponentScan // automatically scan on current package

public class DependencyInjectionApp {

    public static void main(String[] args) {

        try(var context =

                new AnnotationConfigApplicationContext

                    (DependencyInjectionApp.class)){

        Arrays.stream(context.getBeanDefinitionNames())

        .forEach(System.out::println);

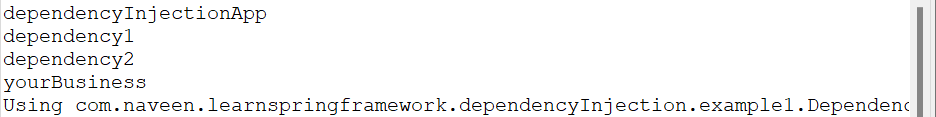
        System.out.println(context.getBean(YourBusiness.class));

        }

    }

}

OUTPUT:



**YourBusiness** class has two dependencies, **Dependency1** and **Dependency2**, which are marked with **@Autowired** annotation. This tells Spring that these dependencies should be injected automatically at runtime by Spring's dependency injection mechanism.

**SETTER INJECTION**

In setter injection, the dependencies of a class are injected through its setter methods using Spring's **@Autowired** annotation.

**DependencyInjectionApp.java**

@Component

class YourBusiness{

    Dependency1 dependency1;

    Dependency2 dependency2;

    @Autowired

    public void setDependency1(Dependency1 dependency1) {

        System.out.println("Setter Injection - setDependency1");

        this.dependency1 = dependency1;

    }

    @Autowired

    public void setDependency2(Dependency2 dependency2) {

        System.out.println("Setter Injection - setDependency2");

        this.dependency2 = dependency2;

    }

    public String toString() {

        return "Using " + dependency1 + " and " + dependency2;

    }

}

@Component

class Dependency1{

}

@Component

class Dependency2{

}

@Configuration

@ComponentScan // automatically scan on current package

public class DependencyInjectionApp {

    public static void main(String[] args) {

        try(var context =

                new AnnotationConfigApplicationContext

                    (DependencyInjectionApp.class)){

        Arrays.stream(context.getBeanDefinitionNames())

        .forEach(System.out::println);

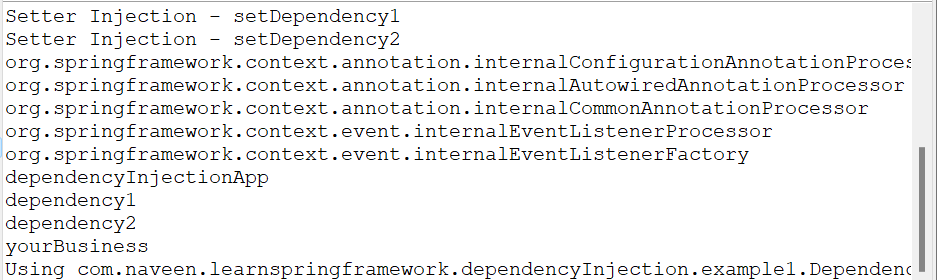
        System.out.println(context.getBean(YourBusiness.class));

        }

    }

}

**OUTPUT:**

****

**CONSTRUCTOR INJECTION**

In constructor injection, the dependencies of a class are injected through its constructor with or without using Spring's **@Autowired** annotation.

**DependencyInjectionApp.java**

package com.naveen.learnspringframework.dependencyInjection.example1;

@Component

class YourBusiness{

    Dependency1 dependency1;

    Dependency2 dependency2;

    @Autowired

    public YourBusiness(Dependency1 dependency1, Dependency2 dependency2) {

        super();

        System.out.println("Constructor Injection - YourBusiness");

        this.dependency1 = dependency1;

        this.dependency2 = dependency2;

    }

    public String toString() {

        return "Using " + dependency1 + " and " + dependency2;

    }

}

@Component

class Dependency1{

}

@Component

class Dependency2{

}

@Configuration

@ComponentScan // automatically scan on current package

public class DependencyInjectionApp {

    public static void main(String[] args) {

        try(var context =

                new AnnotationConfigApplicationContext

                    (DependencyInjectionApp.class)){

        Arrays.stream(context.getBeanDefinitionNames())

        .forEach(System.out::println);

        System.out.println(context.getBean(YourBusiness.class));

        }

    }

}

**OUTPUT:**

