**WHY DO WE HAVE A LOT OF DEPENDENCIES?**

1. In Game Application example, we have very few classes.
2. But Real – World applications are much more complex:
   1. Multiple Layers (Web, Business, Data etc)
   2. Each layer is dependent on the layer below it!
      1. **Example**: Business Layer class talks to a Data Layer class.
         1. Data Layer class is a dependency of Business Layer class.
      2. There are thousands of such dependencies in every application!
3. With Spring Framework:
   1. Instead of focusing on objects, their dependencies and wiring,
      1. You can focus on the business logic of your application!
   2. Spring Framework manages the lifecycle of objects:
      1. Mark components using annotations: @Component (and others...)
      2. Mark dependencies using @Autowired.
      3. Allow Spring Framework to do its magic!

**Chart

Description automatically generated with medium confidence**A picture containing graphical user interface

Description automatically generated

**EXERCISE:**

1. Create classes and interfaces as needed.
   1. Use constructor injection to inject dependencies.
   2. Make MongoDbDataService as primary.
   3. Create a Spring Context
      1. Prefer annotations.
      2. Retrieve BusinessCalculationService bean and run findMax method.

**SOLUTION**

1. **Create classes and interfaces as needed.**

**MongoDbDataService.java**

package com.naveen.learnspringframework.DependencyExercise;

public class MongoDbDataService implements DataService {

    @Override

    public int[] retrieveData() {

        return new int[] {50, 60, 70, 80};

    }

}

**MySQLDataService.java**

package com.naveen.learnspringframework.DependencyExercise;

public class MySQLDataService implements DataService {

    @Override

    public int[] retrieveData() {

        return new int[] {10, 20, 30, 40};

    }

}

**BusinessCalculationService.java**

package com.naveen.learnspringframework.DependencyExercise;

import java.util.Arrays;

public class BusinessCalculationService {

    private DataService dataService;

    public int findMax() {

        return Arrays.stream(dataService.retrieveData()).max().orElse(0);

    }

}

1. Use constructor injection to inject dependencies.

**BusinessCalculationService.java**

package com.naveen.learnspringframework.DependencyExercise;

import java.util.Arrays;

import org.springframework.stereotype.Component;

@Component

public class BusinessCalculationService {

    private DataService dataService;

    public BusinessCalculationService(DataService dataService) {

        super();

        this.dataService = dataService;

    }

    public int findMax() {

        return Arrays.stream(dataService.retrieveData()).max().orElse(0);

    }

}

1. Make MongoDbDataService as primary.

**MongoDbDataService.java**

package com.naveen.learnspringframework.DependencyExercise;

import org.springframework.context.annotation.Primary;

import org.springframework.stereotype.Component;

@Component

@Primary

public class MongoDbDataService implements DataService {

    @Override

    public int[] retrieveData() {

        return new int[] {50, 60, 70, 80};

    }

}

1. Create a Spring Context.
   1. Prefer annotations.
   2. Retrieve BusinessCalculationService bean and run findMax method.

**DependencyNeedExerciseApp.java**

package com.naveen.learnspringframework.DependencyExercise;

import java.util.Arrays;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan

public class DependencyNeedExerciseApp {

    public static void main(String[] args) {

        try (var context

                = new AnnotationConfigApplicationContext

                    (DependencyNeedExerciseApp.class)) {

            Arrays.stream(context.getBeanDefinitionNames())

                .forEach(System.out::println);

            System.out.println(

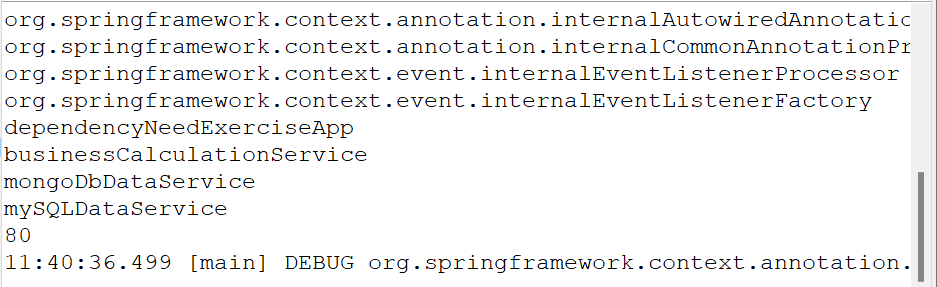
            context.getBean(BusinessCalculationService.class).findMax());

        }

    }

}

**Output:**

****