STEREOTYPE ANNOTATIONS

1. **@Component** - Generic annotation applicable for any class. A general-purpose stereotype annotation used to indicate that a class is a Spring-managed component.
    a. Base for all Spring Stereotype Annotations.
    b. Specializations of **@Component**:
        i. **@Service** - Indicates that an annotated class has business logic. Used to indicate that a class is a service component.
        ii. **@Controller** - Indicates that an annotated class is a "Controller" (e.g., a web controller). Used to define controllers in your web applications and REST API. Used to indicate that a class is a Spring MVC controller.
        iii. **@Repository** - Indicates that an annotated class is used to retrieve and/or manipulate data in a database. Used to indicate that a class is a repository component.
2. Use the most specific annotation possible.
3. By using a specific annotation, you are giving more information to the framework about your intentions.
4. In the Java Spring Framework, stereotype annotations are used to define and configure the roles of specific classes in a Spring application context.
5. In Spring, a stereotype is a way of adding metadata to a class to indicate its purpose or role in the application. Stereotype annotations are used to apply these metadata tags to classes, which can then be used by the Spring container to configure and manage those classes.
6. By using these stereotype annotations, developers can easily configure and manage their Spring application components, and the Spring container can automatically detect and manage those components based on their defined roles.

Early we have did one exercise. We can enhance the exercise using stereotype annotations.

DataService.java

```
package com.naveen.learnspringframework.dependencyInjection.ExerciseDemo;

public interface DataService {

    int[] retrieveData();

}
```

MongoDbDataService.java

```java
package com.naveen.learnspringframework.dependencyInjection.ExerciseDemo;

import org.springframework.stereotype.Repository;

//@Component Instead of writing @component annotation, if we are using database
or data access in a class, then we can use @Repository annotation.
@Repository
public class MongoDbDataService implements DataService {

    @Override
    public int[] retrieveData() {
        // TODO Auto-generated method stub
        return new int[] {50, 60, 70, 80};
    }

}
```

MySQLDataService.java

```java
package com.naveen.learnspringframework.dependencyInjection.ExerciseDemo;

import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Repository;

//@Component
@Repository
@Primary
public class MySQLDataService implements DataService {

    @Override
    public int[] retrieveData() {
        // TODO Auto-generated method stub
        return new int[] {10, 20, 30, 40};
    }

}
```

BusinessCalculationService.java

```java
package com.naveen.learnspringframework.dependencyInjection.ExerciseDemo;

import java.util.Arrays;

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

//@Component - Instead of writing component annotation, if we are using
business logic in a class, then we can use @Service annotation.
```

```java
@Service
public class BusinessCalculationService {

    DataService dataService;

    @Autowired
    public BusinessCalculationService(DataService dataService) {
        super();
        this.dataService = dataService;
    }


    public int findMax() {
        return Arrays.stream(dataService.retrieveData()).max().orElse(0);
    }
}
```

DependencyInjectionApp.java

```java
package com.naveen.learnspringframework.dependencyInjection.ExerciseDemo;

import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class DependencyInjectionApp {

    public static void main(String[] args) {

        try(var context =
                new AnnotationConfigApplicationContext
                    (DependencyInjectionApp.class)){
            System.out.println(context.getBean
                    (BusinessCalculationService.class).findMax());

        }
    }
}
```

OUTPUT:

```
11:06:03.128 [main] DEBUG org.springframework.beans.factory.support.DefaultI
11:06:03.133 [main] DEBUG org.springframework.beans.factory.support.DefaultI
11:06:03.147 [main] DEBUG org.springframework.beans.factory.support.DefaultI
11:06:03.151 [main] DEBUG org.springframework.beans.factory.support.DefaultI
11:06:03.151 [main] DEBUG org.springframework.beans.factory.support.DefaultI
40
11:06:03.200 [main] DEBUG org.springframework.context.annotation.AnnotationC
```