1.8 Homework

1. ***Make a list of all the annotations you have learned so far.***

1. ***@Component*** – Marks a class as a **generic Spring-managed bean**.
2. ***@Service*** – Marks a class as a **service layer component** (business logic).
3. ***@Repository*** – Marks a class as a **DAO** (data access object), **adds persistence exception handling**.
4. ***@Controller*** – Marks a class as a **web controller** in **Spring MVC**.
5. ***@Bean*** – **Declares** a **bean** manually inside a **@Configuration** class.
6. ***@Configuration*** – Marks a **class** that **contains bean definitions**.
7. ***@Autowired*** – **Injects dependencies** automatically.
8. ***@PostConstruct*** – Runs a method **after** the **bean** is **initialized**.
9. ***@PreDestroy*** – Runs a method **before** the **bean** is **destroyed**.
10. ***@Scope("prototype")*** – Specifies **bean scope** as **prototype** (*new instance* per *request*).
11. ***@Scope("singleton")*** – Specifies **bean scope** as **singleton** (*one instance* per *container*).
12. ***@ConditionalOnProperty*** – **Loads** a **bean** only if a **specific property** is **set**.
13. ***@ConditionalOnBean(DataSource.class)*** – **Loads** a **bean** if a **specific bean exists**.
14. ***@ConditionalOnClass(DataSource.class)*** – **Loads** a **bean** if a **specific class** is on the **classpath**.
15. ***@EnableAutoConfiguration*** – **Enables Spring Boot's auto-configuration**.
16. ***@SpringBootApplication*** – Combines *@Configuration*, *@EnableAutoConfiguration*, and *@ComponentScan*.
17. ***@ComponentScan*** – Enables **component scanning** for *@Component*, *@Service*, etc.
18. ***@SpringBootConfiguration*** – **Marks** the **main class** as **Spring Boot's configuration class**.

2. ***Spring Boot vs NodeJS.***

Aspect	**Spring Framework (Java)**	**Node.js (JavaScript)**
Language	Java (**statically** typed)	JavaScript (**dynamically** typed)
Project Type	Best for **large-scale enterprise applications**	Suitable for **lightweight**, fast, **I/O-bound apps**
Dependency Injection (IoC)	Built-in and powerful **via annotations** (@Autowired, @Component, etc.)	Available via **external libraries** like InversifyJS
Configuration	Declarative **via annotations** and **application.properties**	**Manual** or with **configuration libraries**
Database Integration	Rich integration with **JPA**, **Hibernate**, **Spring Data**	Requires **separate ORMs** (Sequelize, TypeORM, etc.)
Security	Comprehensive **via Spring Security**	**Basic auth** via **middleware** (e.g., Passport.js)
Multi-threading & Concurrency	Excellent support via **Java's threading model**	**Single-threaded**, event-driven
Microservices Support	First-class with **Spring Boot + Spring Cloud**	Requires additional tools like **Express**, **PM2**, and **Kubernetes**
Build Tooling	**Maven** or **Gradle**	**npm** or **yarn**
Community & Ecosystem (Enterprise)	Very strong in **large-scale**, **banking**, **telecom**, and **government** projects	Strong in **startups**, **real-time apps**, **web APIs**
Error Handling	**Compile-time safety** due to static typing	**More runtime errors** due to dynamic typing
Auto Configuration	Available via *@SpringBootApplication* and *@EnableAutoConfiguration*	Needs more **manual setup**
Learning Curve	**Steeper** due to **Java** and Spring's **vast ecosystem**	**Easier** for **beginners** familiar with **JavaScript**

@sibasunderj8

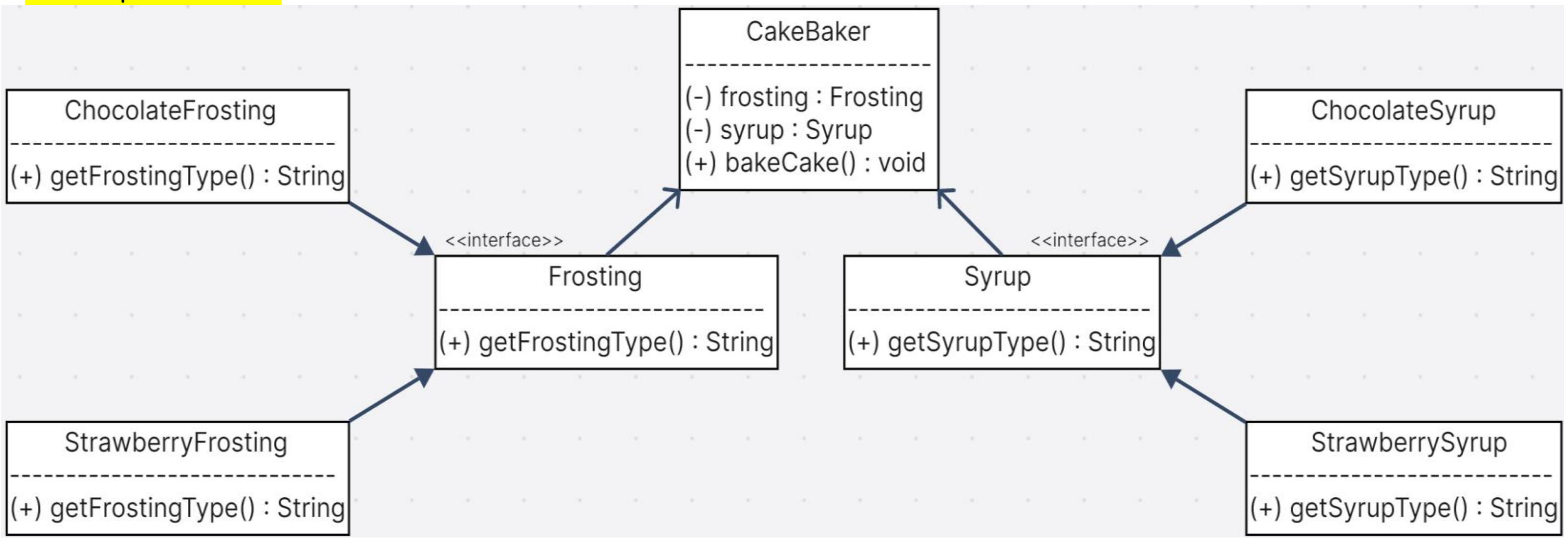
3. Spring Framework vs Spring Boot

Feature	Spring Framework	Spring Boot
Setup & Configuration	Requires extensive **XML** or **Java-based configuration**	**Zero** or **minimal configuration** using ***auto-configuration***
Starter Template	Not provided; you **build dependencies manually**	Provides ***starter dependencies*** (e.g., **spring-boot-starter-web**)
Dependency Management	**Manual** (risk of version mismatch)	**Built-in** dependency management with *compatible versions*
Application Server	Needs to be **deployed manually** on **external server** (e.g., *Tomcat*)	Comes with **embedded servers** (*Tomcat*, *Jetty*)
Entry Point	**No default entry point**, *requires* **boilerplate**	Has a default **main() method** using ***@SpringBootApplication***
Auto-Configuration	**Not available**, everything **configured manually**	Smart **auto-configuration** using ***@EnableAutoConfiguration***
Development Speed	**Slower** due to **verbose setup**	**Faster** with **default settings** and sensible **configuration**
Production Readiness	**Not built-in**	Includes **production features**: ***metrics***, ***health checks***, ***externalized config***
Microservices Support	Requires **manual integration**	First-class support for **microservices** with **Spring Cloud**
Command-Line Interface (CLI)	**Not available**	Spring Boot includes a **CLI** for running **Groovy/Java apps** quickly
Use Case	Good for **fine-grained**, **fully controlled setups**	Best for **rapid development** and **microservices architecture**
Learning Curve	**Steeper** due to **manual setup**	**Easier** for **beginners** due to **defaults** and **embedded components**

4. ***Alice and her Bakery***

- Create a class called *CakeBaker*, that is dependent on two other classes called *Frosting* and *Syrup*. This class has a function called *bakeCake()*.
- Create two **interfaces** of type *Frosting* and *Syrup* with a **function** called *getFrostingType()* and *getSyrupType()* respectively.
- Create two implementations of these two interfaces (so total **4** classes) for **Chocolate** and **Strawberry flavours**.
- Use ***Dependency injection*** to inject the **Frosting** and **Syrup dependencies** into *CakeBaker* and also to call the *bakeCake()* of the *CakeBaker* class.

UML representation:



@sibasunderj8

Code:

Frosting

```java
@Component
public interface Frosting {
   String getFrostingType();
}
@Component
@ConditionalOnProperty(name = "frosting.env", havingValue = "chocolate")
public class FrostingChocolate implements Frosting {
   @Override
   public String getFrostingType() {
      return "Chocolate Frosting";
   }
}
```

```java
@Component
@ConditionalOnProperty(name = "frosting.env", havingValue = "strawberry")
public class FrostingStrawberry implements Frosting {
   @Override
   public String getFrostingType() {
      return "Strawberry Frosting";
   }
}
```

CakeBaker

```java
@Component
public class CakeBaker {
   @Autowired
   Frosting frosting;

   @Autowired
   Syrup syrup;

   public void bakeCake() {
      System.out.println(
            "Preparing Cake using " +
              frosting.getFrostingType() +
              " and " +
              syrup.getSyrupType()
      );
   }
}
```

Syrup

```java
@Component
public interface Syrup {
   String getSyrupType();
}
```

```java
@Component
@ConditionalOnProperty(name = "syrup.env", havingValue = "chocolate")
public class SyrupChocolate implements Syrup {
   @Override
   public String getSyrupType() {
      return "Chocolate Syrup";
   }
}
```

```java
@Component
@ConditionalOnProperty(name = "syrup.env", havingValue = "strawberry")
public class SyrupStrawberry implements Syrup {
   @Override
   public String getSyrupType() {
      return "Strawberry Syrup";
   }
}
```

MyApp

```java
@SpringBootApplication
public class MyApp implements CommandLineRunner {

   @Autowired
   CakeBaker cakeBaker;

   public static void main(String[] args) {
     SpringApplication.run(MyApp.class, args);
   }

   @Override
   public void run(String[] args) throws Exception {
     cakeBaker.bakeCake();
   }
}
```