Spring

Overview

- Framework that makes it easy to create enterprise applications.
- Built for Java but supports Groovy and Kotlin as alternative languages on the JVM.
- Came out in 2003 as a response to the complexity of the early J2EE specification.
  - Complements Java EE, does not compete with it.
- **Configuration model** and **dependency injection (DI)** in its core.
- Divided into modules, most of them optional (developer can choose which ones to use).
  - E.g. persistence (RDBMS and/or NoSQL), web, messaging, security, etc.

The Inversion of Control (IoC) container

- Objects only **declare** their dependencies, that is, the other objects they work with (a.k.a. collaborators). The (Spring) container then **injects** these dependencies when creating the object.
- The object does not control the instantiation of its dependencies by using their class. It is the container that is in control (the inverse) and creates them, hence the name **inversion of control**.
- Dependencies are typically declared as **abstractions** i.e. **interfaces** with the container providing the concrete implementation. Class dependencies can be declared as well.
- Objects created by and managed by the Spring IoC container are called **beans**.
- Configuration models:
  - **XML-based** configuration
  - **Annotation-based** configuration
  - **Java-based** configuration

Annotation-based configuration

Annotations are placed on relevant classes, methods or fields.

Injecting beans

- Constructor-based injection

  Example:

```
public class BookService {
  private BookRepository bookRepository;

  @Autowired
  public BookService(BookRepository bookRepository) {
    this.bookRepository = bookRepository;
  }
}
```

Injecting beans

- Setter-based injection

  Example:

```
public class BookService {
  private BookRepository bookRepository;

  @Autowired
  public void setBookRepository(BookRepository bookRepository) {
    this.bookRepository = bookRepository;
  }
}
```

Declaring beans

Stereotype annotations:

- **@Component**: Generic component
- **@Service**: Application/business service
- **@Repository**: Repository/DAO component
- **@Controller**: Web controller component
- **@RestController**: REST web controller component

Example:

```
@Service
public class BookService {
}
```

Bean scopes

Bean scopes:

- **Singleton**: Single instance per container
- **Prototype**: Once instance per dependency declaration
- **Request**: Scopes a bean to a single HTTP request (web)
- **Session**: Scopes a bean to a single HTTP session (web)

Example:

```
@Component
@Scope("prototype")
public class BookScanner {
}
```

Automatically detecting (scanning) beans

Use component scan inside a configuration class.

Example:

```
@Configuration
@ComponentScan(basePackages = "org.example")
public class ApplicationConfig {
}
```

Java-based configuration

Used when:

- The bean is to be created from a third-party class and can’t be annotated with a stereotype annotation
- Instantiating the bean involves more than invoking the class constructor.

Example:

```
@Configuration
@ComponentScan(basePackages = "org.example")
public class ApplicationConfig {

  @Bean
  public void BookService bookService() {
    return BookServiceFactory.getInstance();
  }
}
```

Spring profiles

Mechanism for controlling which implementation of a dependency (abstraction) is to be instantiated and injected by the Spring container.

```
@Repository
@Profile("jpa")
public class JpaBookRepository
     implements BookRepository {
}
```

```
@Repository
@Profile("collections")
public class MapBookRepository
     implements BookRepository {
}
```

```
@Service
public class BookService {
    private BookRepository bookRepository;

    @Autowired
    public BookService(
            BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
}
```

“src/main/resources/application.properties”

```
spring.profiles.active = jpa
```