Spring Framework - DI

Piya Lumyong

What's Spring

- Goals
  - make Enterprise Java easier to use
  - promote good programming practice
  - enabling a POJO-based programming model that is applicable in a wide range of environments
- Some said Spring is just a “glue” for connecting all state of the art technologies together (a.k.a Integration Framework) via it’s Application Context.
- Heart and Soul of Spring is Dependency Injection and Aspect Oriented Programming.

What is Spring Framework today?

- an open source application framework
- a lightweight solution for enterprise applications
- non-invasive (POJO based)
- is modular
- extendible for other frameworks
- de facto standard of Java Enterprise Application

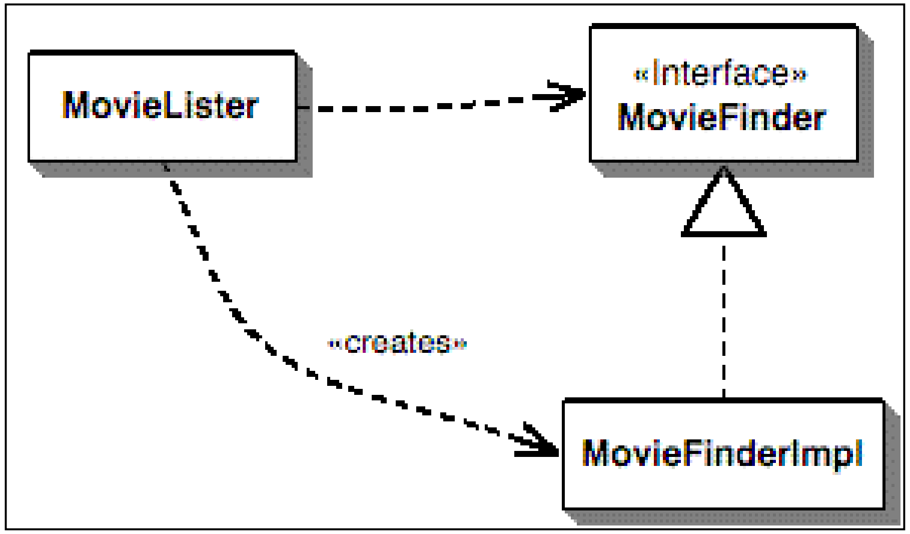
Dependency Injection/Inversion of Control

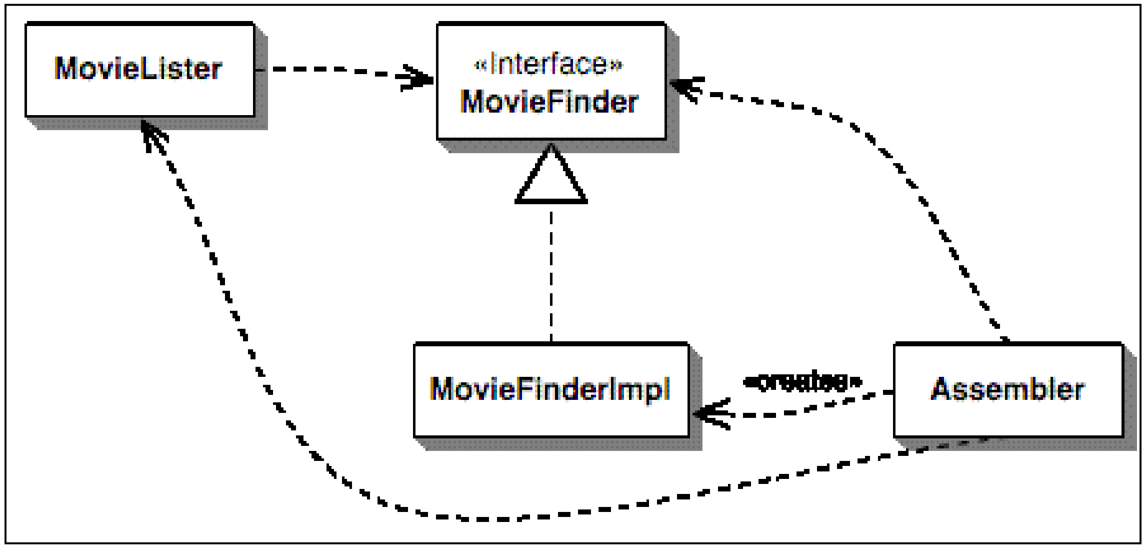
Inversion of Control in Action
ALRIGHT BUDDY, EJB NO MORE. NOW *I* DRIVE!

Dependency Injection in Action
SO HOW DO YOU LIKE IT: SETTER, CONSTRUCTOR, OR INTERFACE?
THOUGHT LEADER
ORM
POJO

Dependency Injection/Inversion of Control

- Naive
- DI





Hollywood Principle: "Don't call me, I'll call you."

Exercise Basic DI

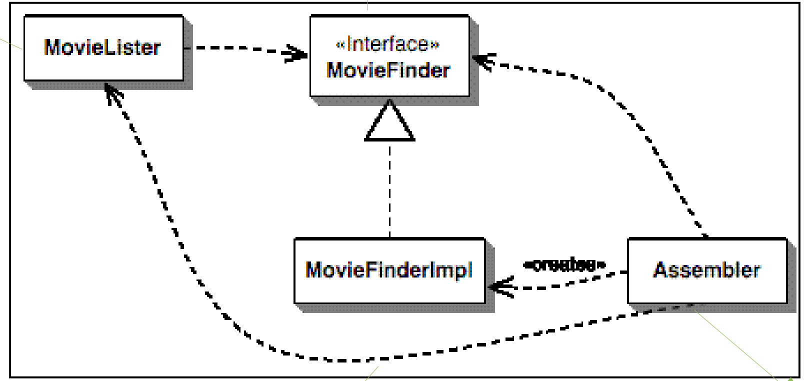
LAB1 Setup/Basic DI

```java
public class MovieLister {
  private MovieFinder finder;

  public MovieLister(MovieFinder finder) {
    this.finder = finder;
  }

  public void listMovie() {
    ...
  }
}
```

```java
public interface MovieFinder {
        public List<Movie> findMovie();
}
```



```java
public class MovieListerTest {
  @Test
  public void testListMovie() {
    //given
    MovieFinder finder = mock(MovieFinder.class);
    when(finder.findMovie()).thenReturn(new ArrayList<Movie>());
    MovieLister lister = new MovieLister(finder);

    //when
    lister.listMovie();

    //then
    verify(finder).findMovie();
  }
}
```

```java
public class Assembler {
  public static void main(String[] args) {
    InputStream is = Assembler.class.getClass()
           .getResourceAsStream("/movie.csv");
    MovieFinder finder = new MovieFinderImpl2(is);
    MovieLister lister = new MovieLister(finder);

    lister.listMovie();
  }
}
```

```java
public class MovieFinderImpl2 implements MovieFinder {
  private List<Movie> movies;

  public MovieFinderImpl2(InputStream is) {
    movies = new ArrayList<Movie>();

    try {
      InputStreamReader isr = new InputStreamReader(is);
      BufferedReader br = new BufferedReader(isr);
      String line;
      while ((line=br.readLine())!=null) {
        String[] csv = line.split("\\|\\|");
        movies.add(new Movie(csv[0], csv[1], Float.parseFloat(csv[2])));
      }
    } catch (IOException e) {}
  }

  @Override
  public List<Movie> findMovie() {
    return movies;
  }
}
```

```java
public class MovieFinderImpl2Test {
  @Test
  public void testFindMovie() {
    //given
    InputStream is = this.getClass().getResourceAsStream("/test-movie.csv");
    MovieFinder finder = new MovieFinderImpl2(is);

    //when
    List<Movie> results = finder.findMovie();

    //then
    assertEquals(6, results.size());
  }
}
```

What is bean?

- The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.
- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions which you have already seen in previous chapters.

Annotation

Java-based container configuration

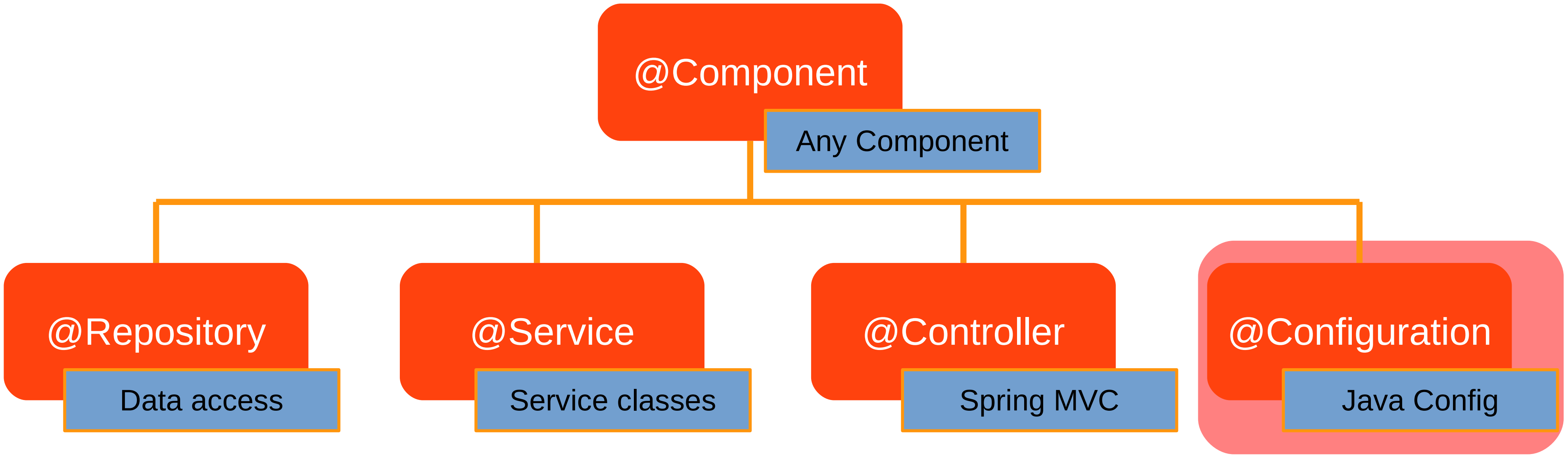
- Annotation-based configuration metadata

```
@Configuration
public class BankConfg {
  @Bean
  public TransferService createTransferService() {
    return new TransferServiceImpl();
  }

  @Bean(name = "exchangeService", initMethod = "init")
  public Exchange createExchangeService() {
    return new ExchangeServiceImpl();
  }
}
```

Annotation-based container configuration

- Stereotypical



Dependency injection

```
@Service
public class OrderBuilder {
  @Autowired
  private CurrencyService currencyService;

  public void action() {
    // do something
  }
}
```

Dependency injection

```
@Configuration
public class BankServiceConfig {
  @Autowired
  private CurrencyRepository currencyRepository;

  @Bean
  public CurrencyService currencyService() {
    return new CurrencyServiceImpl(currencyRepository);
  }

  @Bean(name = {"orderBuilder", "builder"})
  public OrderBuilder orderBuilder() {
    OrderBuilder builder = new OrderBuilder(currencyService());
    builder.initial();
    builder.setup();
    builder.something();
    ...
    Return builder;
  }
}
```

Additional

A Guide to Spring Framework Annotations

Spring Framework 4 Cheat Sheet by danielfc