

Spring Dependency Injection & Java 5

Alef Arendsen

- Alef Arendsen
 - Spring committer since 2003
 - Now Principal at SpringSource (NL)



- A big pile of car parts
 - Workers running around uncontrollably
 - The parts don't connect at all
 - Creating cars was all done by hand
-
- That's what car manufacturing was like before Ford introduced the assembly line!

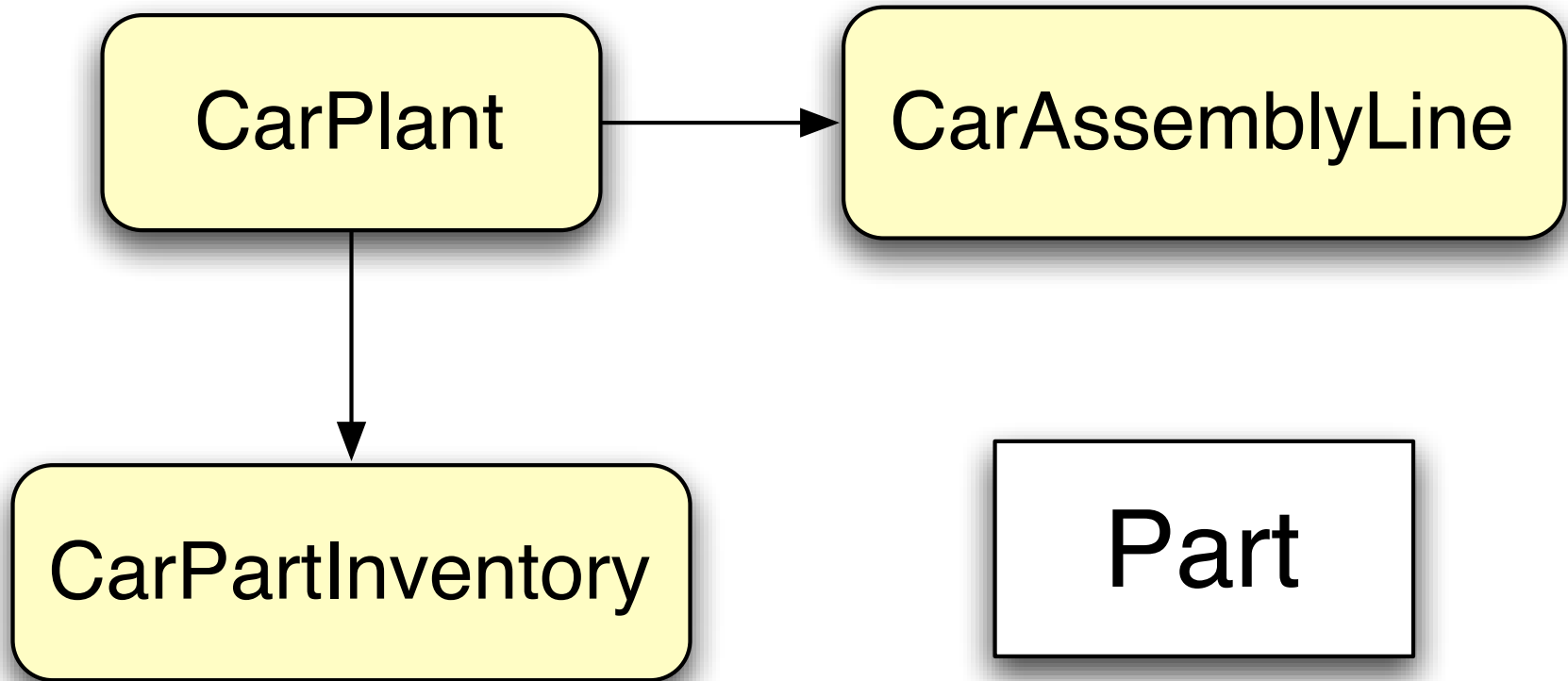
The Model T was the first automobile mass produced on assembly lines with completely interchangeable parts...

By 1914, the assembly process for the Model T had been so streamlined it took only 93 minutes to assemble a car.

Source: Wikipedia

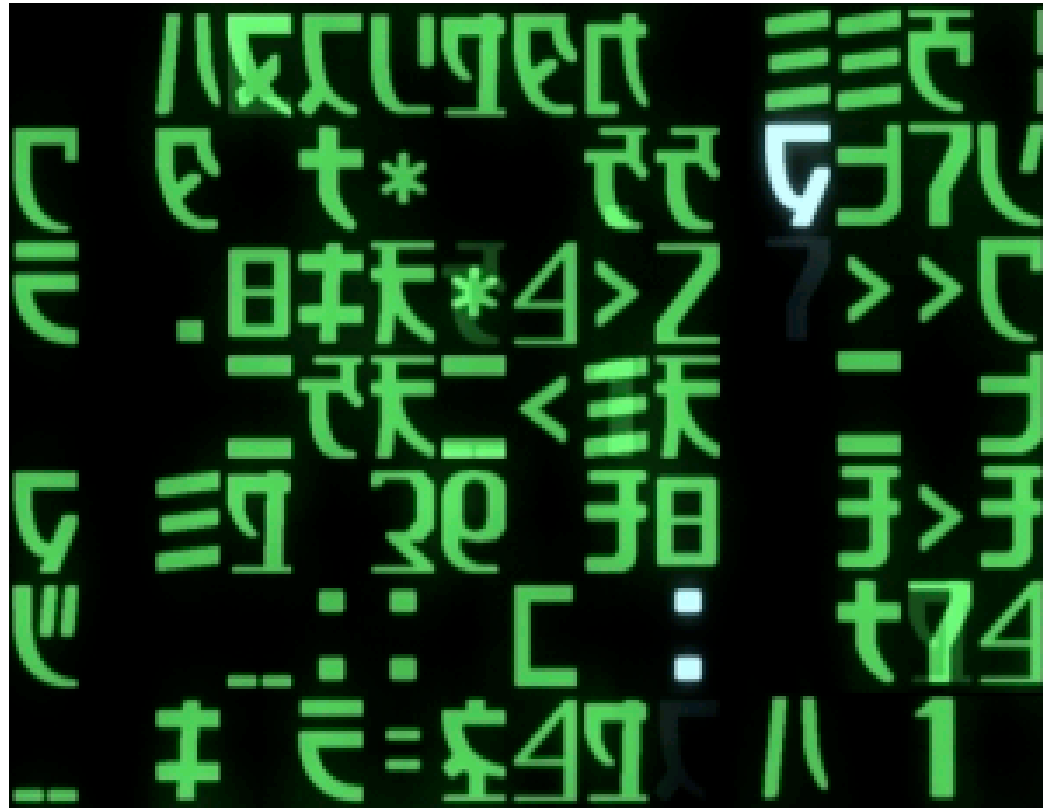
Part I

<bean>



- Spring == XML
- XML == Evil
- Evil == Sucks

- Therefore, Spring == Sucks





- Easy for tooling to generate graphs
- Central location for all config data
- Configuration separate from Java code only option for code you don't control
- Easy solution for ambiguity

```
<bean class="org.apache.commons.dbcp.BasicDataSource">  
  <property name="username" value="sa"/>  
  <property name="password" value="manager"/>  
</bean>
```

- Perceived XML hell (partially true)
- Lack of type safety (at compile time)
 - Tooling helps us a bit here
- Less refactoring friendly
- Names needed to solve ambiguity

Part II

@Autowired and <bean>

- Candidate for auto-detection

```
@Component
public class HibernateCarPartsInventory
    implements CarPartsInventory {

    private SessionFactory sessionFactory;
    ...
}

<context:component-scan base-package="com.carplant"/>
```

- Constructor
- Field
- Property

```
@Autowired
public HibernateCarPartsInventory(
    SessionFactory factory) {
    this.sessionFactory = factory;
}
```

- 'Config' code in the Java code
- More type safe experience
- Elegant annotation-based solution for solving ambiguity (requires XML)

```
<bean class="com.carplant.inventory.HibernateCarPartsInventory">  
  <qualifier type="com.carplant.util.Offline"/>  
</bean>
```

```
<bean class="com.carplant.inventory.HibernateCarPartsInventory">  
  <qualifier type="com.carplant.util.Online"/>  
</bean>
```

```
@Autowired @Offline CarPartsInventory offlineInventory;  
@Autowired @Online CarPartsInventory onlineInventory;
```

- 'Config' code in the Java code
- Extra (sometimes complex) measures needed for solving ambiguity

```
<bean class="com.carplant.inventory.HibernateCarPartsInventory">  
  <qualifier type="com.carplant.util.Offline"/>  
</bean>
```

```
<bean class="com.carplant.inventory.HibernateCarPartsInventory">  
  <qualifier type="com.carplant.util.Online"/>  
</bean>
```

```
@Autowired @Offline CarPartsInventory offlineInventory;  
@Autowired @Online CarPartsInventory onlineInventory;
```

Part III

@Bean and <bean>

- On type-level
- Identifies a class as a configuration class
- @Bean methods represent beans

```
@Configuration
public class MyConfig {

    public @Bean Service service() {
        return new Service();
    }
}
```

- Method-level
- Identifies a method returning an external bean

```
public abstract @ExternalBean DataSource dataSource();
```

- 'Config' code completely separate from Java code
- Entirely type safe approach
- Easy solution for ambiguity problem
- Allows for context inheritance
- Allows for 100% of all Java constructs

```
public CarPartsInventory offlineInventory() {  
    // configure and return offline inventory  
    return new HibernateCarPartsInventory(null,null);  
}
```

```
public CarPartsInventory onlineInventory() {  
    // configure and return online inventory  
    return new HibernateCarPartsInventory(null,null);  
}
```

- Harder to make it work in tooling
- It's 'configuration with a twist'
- Requires a little bit more code

Conclusion

- There's something for everyone in Spring
- Type safe and separate configuration
 - JavaConfig (@Bean)
- Type safe and config in Java code
 - @Autowired / @Component
- For external code and XML fans
 - <bean/>
- For specification-minded people
 - EJB 3

- All three approaches build on Spring's proven and solid foundation
 - Just mix and match all approaches
 - A moving model, not a fixed static snapshot of the current state of DI
- Plus all the other benefits
 - Easy JMX exporting
 - Ease AOP configuration