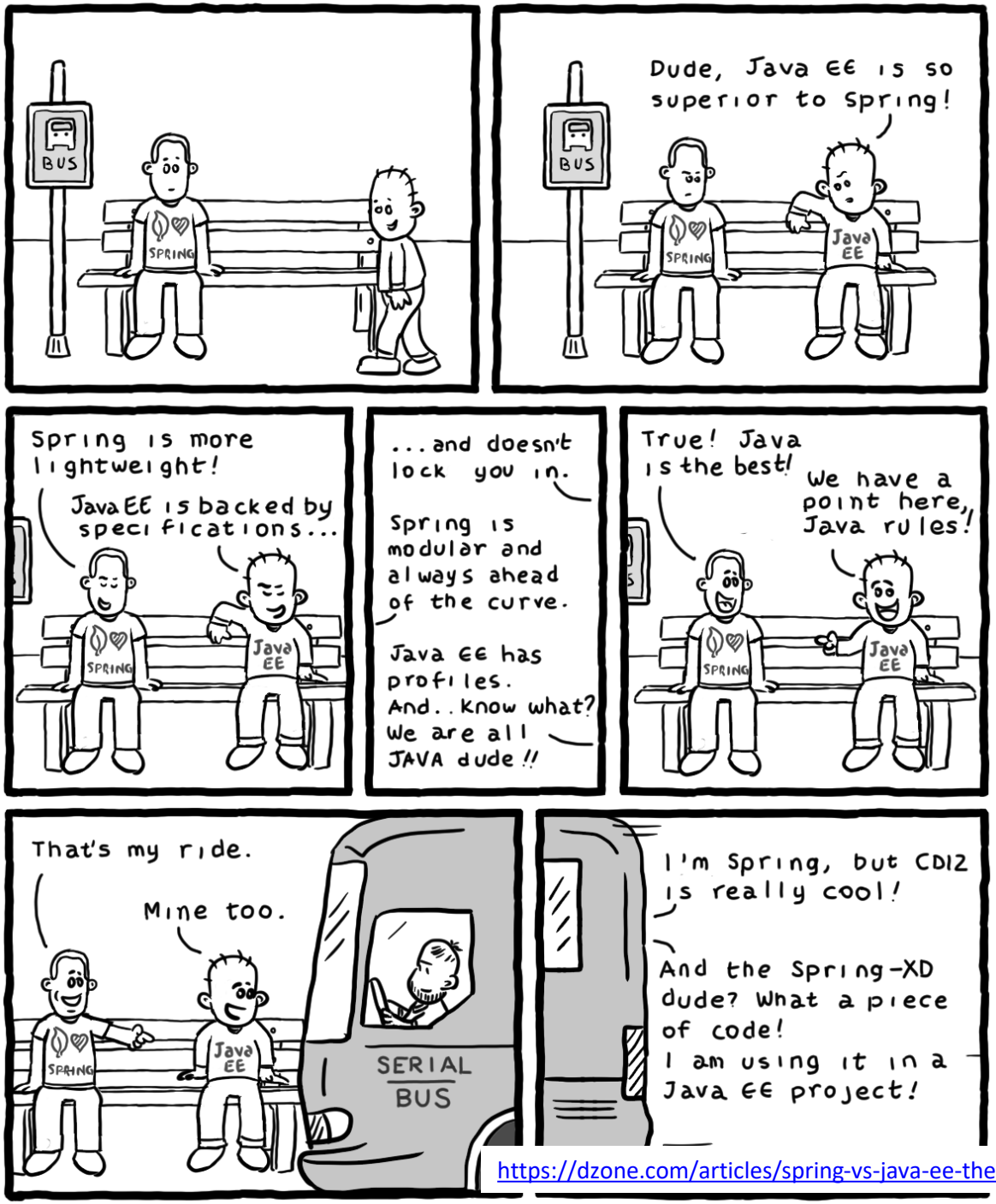
Spring Boot, Java EE and components



https://dzone.com/articles/spring-vs-java-ee-the-real-story-comic

Java EE & Spring boot

- Both have similar solution
  - Code, annotations & configuration files
  - Relies also on JNDI / Java EE runtime
  - Spring on explicit configurations
    - Spring boot is a big change – code
- Java EE has Context & Dependency injection (CDI)
  - Part of the Java EE (since version 6)
  - Solutions provide lookup without containers
    - Weld an implementation supported in Java SE.
    - http://weld.cdi-spec.org/
  - Xml
- Spring boot
  - Depends on Transparent for the developer (once ok)
  - External: java properties and yaml files

https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-configuration-classes.html
https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html

Spring boot

Spring

- spring framework
- IoC Container
  - Core for IoC / DI
  - Lifecycle interceptor….

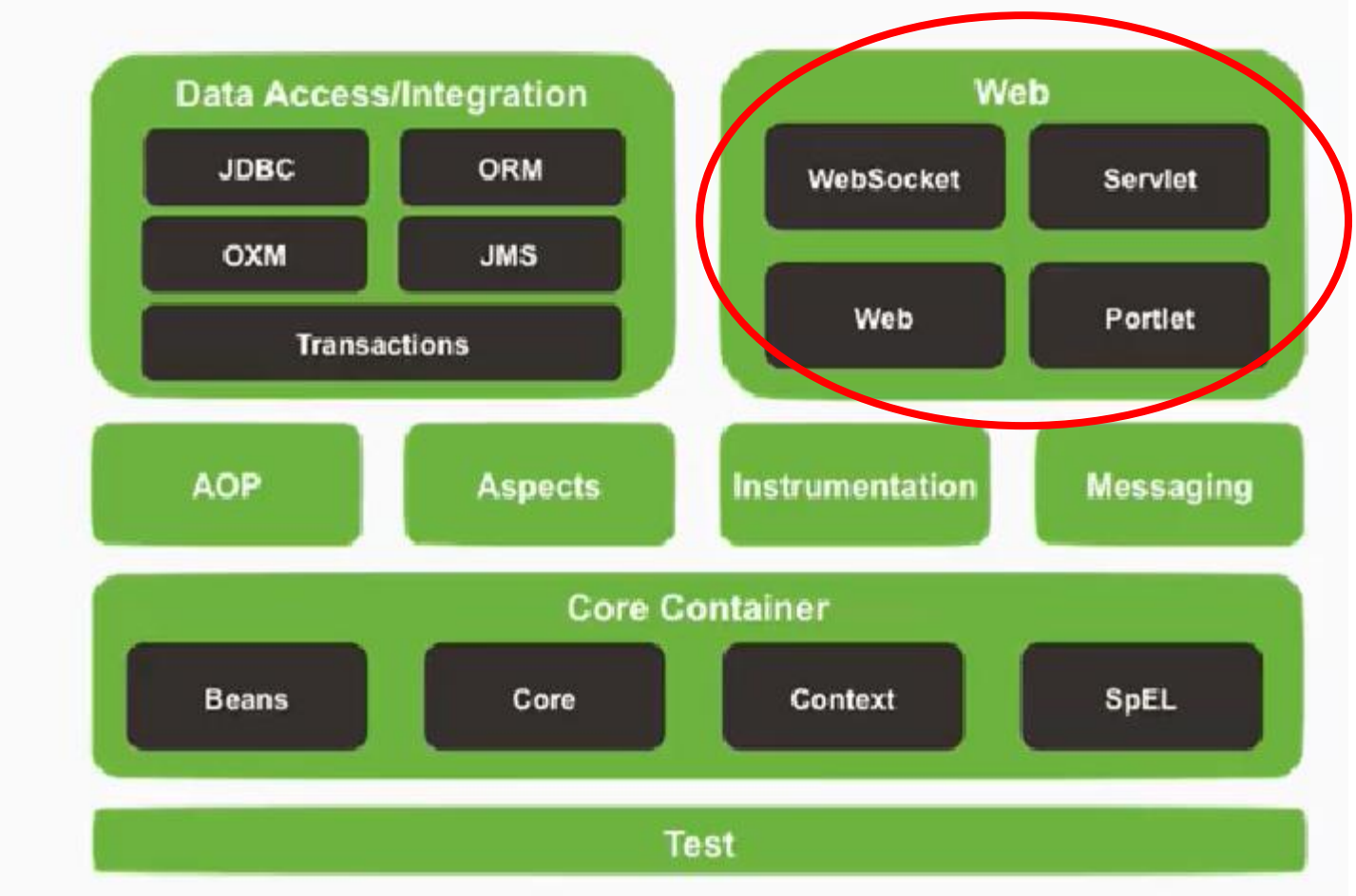
bootstrap

- Embedded server
- **Relies on servlet containers for web**

Name	Servlet Version
Tomcat 8.5	3.1
Jetty 9.4	3.1
Undertow 1.4	3.1

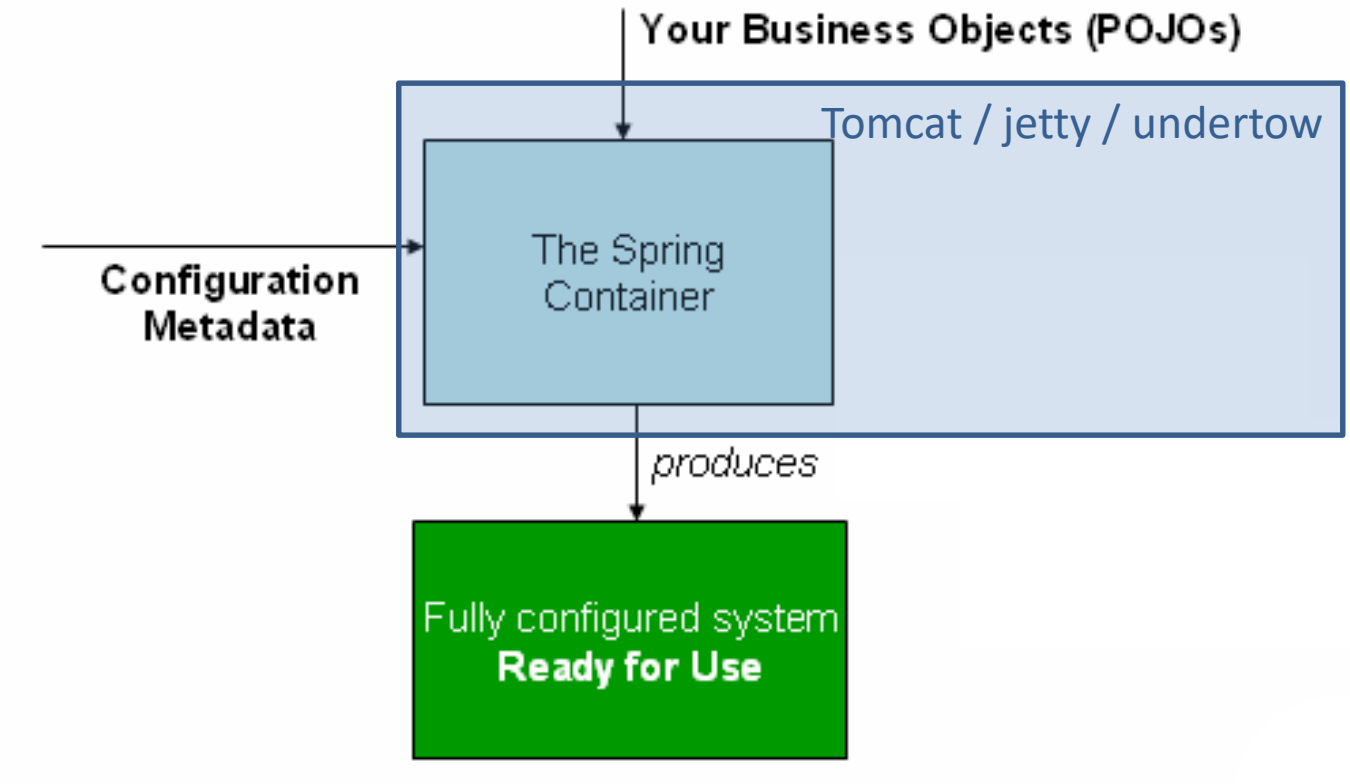
Java EE 8 Microservices : Introduction to Spring and Spring Boot
https://www.youtube.com/watch?v=5ALiSgHxHIE

Spring boot



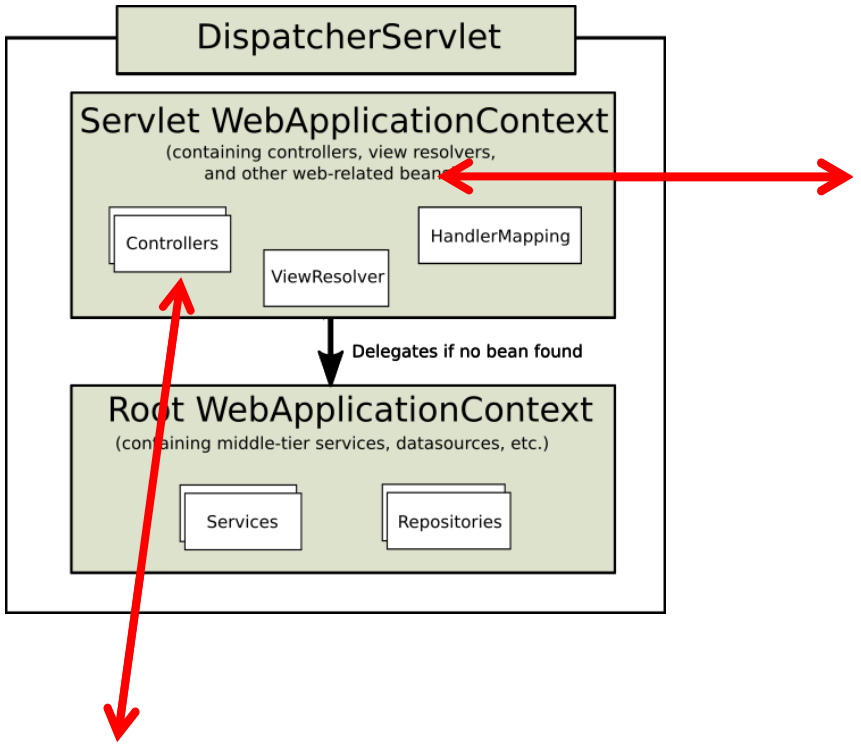
For specifics refer to official documentation
https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/

Spring boot



For specifics refer to official documentation
https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/

Spring MVC: similar to Java EE




```
<web-app>

    <listener>
        <listener-class>org.springframework.web.context.ContextL
    </listener>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/root-context.xml</param-value>
    </context-param>

    <servlet>
        <servlet-name>app1</servlet-name>
        <servlet-class>org.springframework.web.servlet.Dispatche
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/app1-context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>app1</servlet-name>
        <url-pattern>/app1/*</url-pattern>
    </servlet-mapping>

</web-app>
```

servlets" a.k.a. as controllers

https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html#m

A first look at Spring Boot, is it time to leave XML based configuration behind?

15 APRIL 2014 // MAGNUS LARSSON

Spring Boot makes it very easy to create Spring based applications. Spring Boot takes an opinionated view of the Spring platform and third-party libraries, allowing us to get started with a minimal configuration. For example we can develop a Jave EE based web application without any configuration files. Not even a `web.xml` file is required!

When required, however, we can take control over parts of the configuration and override the conventions that Spring Boot puts in play. We can also, if we really must, use traditional XML configuration files for some parts of the configuration. Please read the excellent documentation for extensive information.

In this blog we will develop a plain REST service using the Spring Web MVC framework (shortened to *Spring MVC* in this blog) and package it in a web application, ready to be deployed in any Servlet 3.0 compliant web server. We will also, of course, do it a test run in the embedded Servlet container that Spring Boot provides. We are going to use Gradle as our build system.

GET THE SOURCE CODE

If you want to check out the source code and test it on your own you need to have Java SE 7 and Git installed. Then perform:

```
$ git clone git@github.com:callistaenterprise/blog-a-first-look-at-spring-boot.git
$ cd blog-a-first-look-at-spring-boot/spring-boot-one
```

A first look at Spring Boot, is it time to leave XML based configuration behind?
http://callistaenterprise.se/blogg/teknik/2014/04/15/a-first-look-at-spring-boot/

A stop on annotations

Annotations i.e. @

- form of metadata that can be added to classes, fields, and methods.
- start with the “@” sign

```
@Deprecated
public class
MyClass { ... }
```

- **Built in**
  - ***@Deprecated***
  - ***@Override***
  - ***@FunctionalInterface***
  - ***…***
- **Meta-Annotations**
  - ***@Target***
  - ***@Inherited***
  - ***@Documented***

The @

- Annotations serve multiple purposes like:
  - Compiler Information: they can be read by it and used for additional processing
  - Runtime Information: annotations can be read during runtime (for example, this approach is used heavily in the Spring ecosystem)
- ~ Pre “compiler”
  - Similar to define / typedef in C/C++
- Semantics defined by use
  - Framework support

Annotations

definition

```
@Documented
@Target(ElementType.METHOD)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotation{
    int studentAge() default 18;
    String studentName();
    String stuAddress();
    String stuStream() default "CSE";
}
```

use

```
@MyCustomAnnotation(
    studentName="Chaitanya",
    stuAddress="Agra, India"
)
public class MyClass {
...
}
```

Out of scope addressing definition of annotations

https://beginnersbook.com/2014/09/java-annotations/
https://docs.oracle.com/javase/tutorial/java/annotations/index.html

ORACLE Java Documentation

Hide TOC

The Java™ Tutorials

Annotations

- Annotations Basics
- Declaring an Annotation Type
- Predefined Annotation Types
- Type Annotations and Pluggable Type Systems
- Repeating Annotations
- Questions and Exercises

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases.

Lesson: Annotations

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Annotations have a number of uses, among them:

- **Information for the compiler** — Annotations can be used by the compiler to detect errors or suppress warnings.
- **Compile-time and deployment-time processing** — Software tools can process annotation information to generate code, XML files, and so forth.
- **Runtime processing** — Some ann… …ilable to be examined at runtime.

This lesson explains where annotations ca… …otation types are available in the Java Platform, Standard Edition (Java… …n with pluggable type systems to write code with stronger type checking, and how to imp…

Out of scope addressing definition of annotations

Lesson: Annotations
https://docs.oracle.com/javase/tutorial/java/annotations/

NetBeans | NetBeans IDE | NetBeans Platform | Enterprise | Plugins | Docs & Support | Community

HOME / Docs & Support

Working with Injection and Qualifiers in CDI

PRINTABLE VERSION

Contributed by Andy Gibson

Contexts and Dependency Injection

1. *Getting Started with CDI and JSF 2.0*
2. ***Working with Injection and Qualifiers in CDI***
   - *Injection: the 'I' in CDI*
   - *Working with Qualifiers*
   - *Alternative Injection Methods*
   - *See Also*
3. *Applying @Alternative Beans and Lifecycle Annotations*
4. *Working with Events in CDI*

creating commonly used CDI artifacts.

REQUIRES NetBeans 8.0/7.4/7.3

Contexts and Dependency Injection (CDI), specified by JSR-299, is an integral part of Java EE 6 and provides an architecture that allows Java EE components such as servlets, enterprise beans, and JavaBeans to exist within the lifecycle of an application with well-defined scopes. In addition, CDI services allow Java EE components such as EJB session beans and JavaServer Faces (JSF) managed beans to be injected and to interact in a loosely coupled way by firing and observing events.

This tutorial is based on the blog post by Andy Gibson, entitled Getting Started with CDI part 2 – Injection. It demonstrates how you can use CDI injection to *inject* classes or interfaces into other classes. It also shows how to apply CDI *qualifiers* to your code, so that you can specify which class type should be injected at a given injection point.

NetBeans IDE provides built-in support for Contexts and Dependency Injection, including the option of generating the `beans.xml` CDI configuration file upon project [illegible] navigation support for annotations, as well as various wizards for

Out of scope addressing definition of annotations

To complete this tutorial, you need the following software and resources:

Software or Resource	Version Required
NetBeans IDE	7.2, 7.3, 7.4, 8.0, Java EE version

https://netbeans.org/kb/docs/javaee/cdi-inject.html

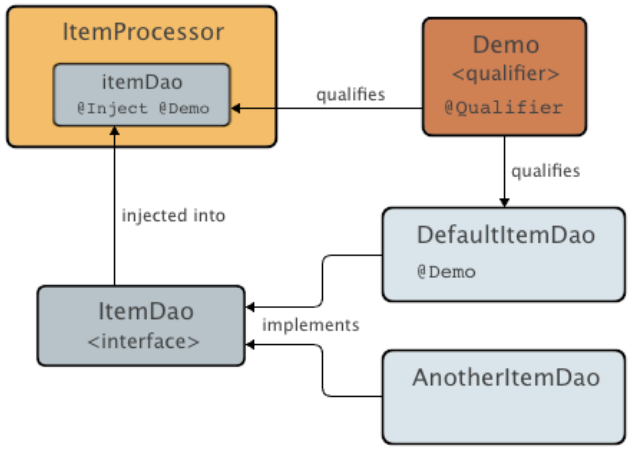
NetBeans | NetBeans IDE | NetBeans Platform | Enterprise | Plugins | Docs & Support | Community

HOME / Docs & Support

PRINTABLE VERSION

Working with Injection and Qualifiers in CDI

Contributed by Andy Gibson



Dependency Injection with qualifier

```
@Named
@RequestScoped
public class ItemProcessor {

@Inject @Demo
private ItemDao itemDao;

public void execute() {
    List<Item> items = itemDao.fetchItems();
    for (Item item : items) {
        System.out.println("Found item " + item);
    }
}
}
```

Out of scope addressing specifics on scope resolution

https://netbeans.org/kb/docs/javaee/cdi-inject.html

An example: lombok

- “java library that automatically plugs into your editor and build tools, spicing up your java.
  Never write another getter or equals method again. Early access to future java features such as val, and much more.”

```
<dependencies>
    ...
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.8</version>
        <scope>provided</scope>
    </dependency>
    ...
</dependencies>
```

Lombok features

The Lombok javadoc is available, but we advise these pages.

`val`
Finally! Hassle-free final local variables.

`@NonNull`
or: How I learned to stop worrying and love the NullPointerException.

`@Cleanup`
Automatic resource management: Call your `close()` methods safely with no hassle.

`@Getter/@Setter`
Never write `public int getFoo() {return foo;}` again.

`@ToString`
No need to start a debugger to see your fields: Just let lombok generate a `toString` for you!

`@EqualsAndHashCode`
Equality made easy: Generates `hashCode` and `equals` implementations from the fields of your object..

`@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor`
Constructors made to order: Generates constructors that take no arguments, one argument per final / non-nullfiel or one argument for every field.

https://projectlombok.org/

“explode” code

written

```
@Entity
@Getter @Setter @NoArgsConstructor // <--- THIS is it
public class User implements Serializable {

    private @Id Long id; // will be set when persisting

    private String firstName;
    private String lastName;
    private int age;

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }
}
```

What it is really

```
@Entity
public class User implements Serializable {

    private @Id Long id; // will be set when persisting

    private String firstName;
    private String lastName;
    private int age;

    public User() {
    }

    public User(String firstName, String lastName, int age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    // getters and setters: ~30 extra lines of code
}
```

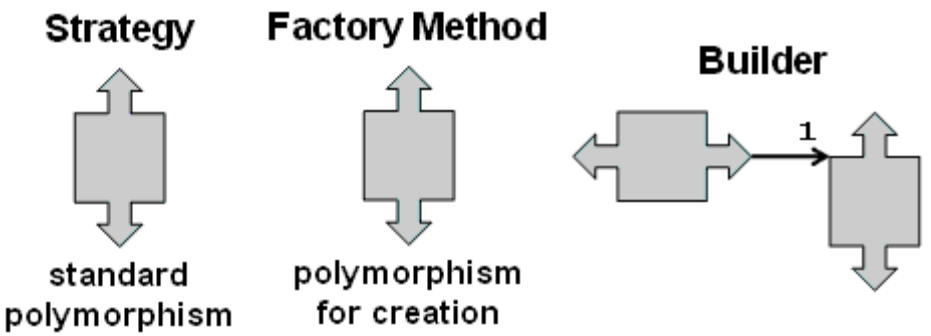
http://www.baeldung.com/intro-to-project-lombok

Leverage patterns

```java
@Builder
public class ApiClientConfiguration {

    // ... everything else remains the same

}
```

```java
public class ApiClientConfiguration {

    private String host;
    private int port;
    private boolean useHttps;

    private long connectTimeout;
    private long readTimeout;

    private String username;
    private String password;

    // Whatever other options you may thing.

    // Empty constructor? All combinations?

    // getters... and setters?
}
```

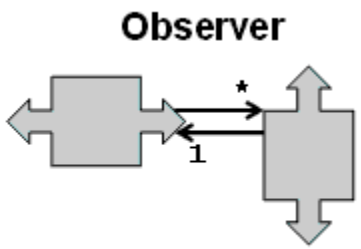
```java
ApiClientConfiguration config =
    new ApiClientConfigurationBuilder()
        .host("api.server.com")
        .port(443)
        .useHttps(true)
        .connectTimeout(15_000L)
        .readTimeout(5_000L)
        .username("myusername")
        .password("secret")
    .build();
```



```java
public class ApiClientConfiguration {

    private static Logger LOG = LoggerFactory.getLogger(ApiClientConfiguration.class);

    // LOG.debug(), LOG.info(), ...

}
```

```java
@Slf4j // or: @Log @CommonsLog @Log4j @Log4j2 @XSlf4j
public class ApiClientConfiguration {

    // log.debug(), log.info(), ...

}
```

```java
@Data
@ToString(includeFieldNames = true, of = {"firstName", "secondName"} )
public class PersonDTO {

  @NonNull private String firstName;
  @NonNull private String secondName;
  @NonNull private Date dateOfBirth;
  final private final String profession;
  private BigDecimal salary;

}
```



Inversion of Control (IoC)

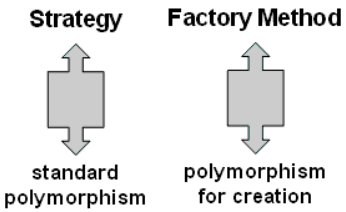
- Is a design principle
  - also known as the Hollywood Principle - "Don't call us, we'll call you").
- Control of the object is inverted.
  - It is not the programmer, but someone else who controls the object.
- “IoC is when you have someone else create objects for you”.
  - So instead of writing “***new MyObject”*** on your code, the object is created by someone else.
  - This *‘someone else’* is normally referred to as IoC Container.

InversionOfControl
https://martinfowler.com/bliki/InversionOfControl.html
Inversion of Control vs Dependency Injection
http://blog.bytecode.tech/inversion-of-control-vs-dependency-injection/

(IoC) Containers once more…

- The programmer must identify the classes whose instances are to be managed by the Container.
  - There are several ways to do this: **with annotations, by extending some specific classes, using external configuration**.

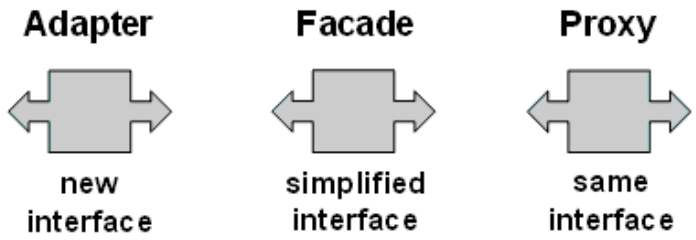
```
@WebServlet("/Simple")
public class Simple extends HttpServlet {
   private static final long serialVersionUID = 1L;
   @GET
   protected void doGet(HttpServletRequest request, HttpServletRe
sponse response)
                  throws ServletException, IOException {

(…)
}
// AWT or Swing context
public class MouseEventDemo ... implements MouseListener {
(…)
   public void mousePressed(MouseEvent e) {
     saySomething("Mouse pressed”);
   }
(…)
}
```



(IoC) Containers

- The programmer can influence, to some extent, the way the objects are managed by the Container.
  - **Normally, this is achieved by overriding the default behaviour of the object callbacks.**

```
@WebServlet("/Simple")
public class Simple extends HttpServlet {
   private static final long serialVersionUID = 1L;
  @GET
   protected void doGet(HttpServletRequest request, HttpServletR
esponse response)
                  throws ServletException, IOException {

(…)
}
// AWT or Swing context
public class MouseEventDemo ... implements MouseListener {
(...)
   public void mousePressed(MouseEvent e) {
     saySomething("Mouse pressed”);
   }
(...)
}
```



(dis)advantage?

Traditional code

- our custom code makes calls to a library,
- **We control the flow**

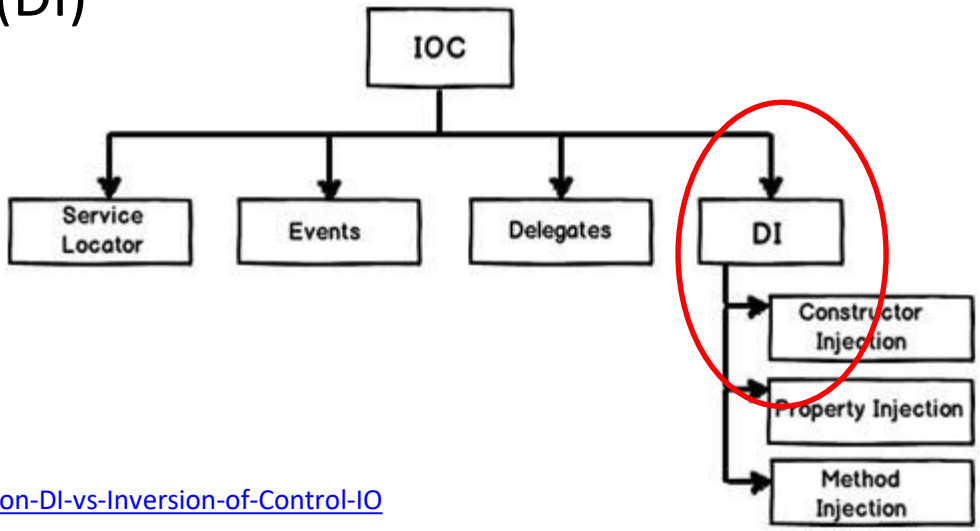
IoC

- **framework takes control of the flow** of a program and make calls to our custom code.
- In order to make this possible, frameworks are defined through abstractions with additional behavior built in. **If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.**

Intro to Inversion Control and Dependency Injection with Spring
http://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

How implement IoC?

- Inversion of Control can be achieved through various mechanisms such as:
  - Strategy design pattern,
  - Service Locator pattern,
  - Factory pattern
  - Dependency Injection (DI)



Dependency Injection (DI) vs. Inversion of Control (IOC)
https://www.codeproject.com/Articles/592372/Dependency-Injection-DI-vs-Inversion-of-Control-IO

DI: I need, I ask…

- Don’t ask for implementation
- Ask for “something” that fulfils my need
  - my focus is on interface
  - my focus is not on specific resources
- Ask the expert / context (GRASP?)
  - Assume context will be reasonable
  - Assume context semantics are clear

> “Always expect to be called. If you ever need to call, never call direct. Always ask the framework. The framework knows.”

Why dependency injection?

- Loose coupling
  - Law of Demeter
- Programming to an interface
  - Interface segregation
- Inversion of Control (IoC)

- Helps managing complexity, change ...

https://blogs.endjin.com/2014/04/understanding-dependency-injection/

Configurations vs annotations

- Precedence may depend
  - On specification
  - on framework
  - …
- Read documentation
- **Not only for servlets**

One example from JAX-RS

```
@ApplicationPath("api")
public class MyRestApp extends Application {
}
```

@ApplicationPath for our example resource, http://example.com/**api/**orders

```
@Path("/orders")
public class Order {

    @GET
    public String getOrders() {
        return "returning all orders";
    }
}
```

The method getOrders (we can give any name to this method) will get called when an HTTP GET request is made for http://example.com/api**/orders.**

JAX-RS - Creating Resources using @ApplicationPath, @Path, @PathParam
https://www.logicbig.com/tutorials/java-ee-tutorial/jax-rs/path-annotion-resource-mapping.html

One example from JAX-RS

```
@Path("/orders")
public class Order {


    @GET
    public String getOrders() {
        return "returning all orders";
    }


    @GET
    @Path("{orderId}")
    public String getOrderById(
            @PathParam("orderId") String orderId) {
        return "return order with id " + orderId;
    }
}
```

http://example.com/api/orders/345 can be mapped to http://example.com/api/orders**/{orderId}**

JAX-RS - Creating Resources using @ApplicationPath, @Path, @PathParam
https://www.logicbig.com/tutorials/java-ee-tutorial/jax-rs/path-annotion-resource-mapping.html

Returning to spring boot

Spring annotations…

```java
@Configuartion
public class DataConfig {
    @Bean
    public DataSource source() {
        DataSource source = new OracleDataSource();
        source.setURL();
        source.setUser();
        return source;
    }
    @Bean
    public PlatformTransactionManager manager() {
        PlatformTransactionManager manager = new BasicDataSourceTransactionManager();
        manager.setDataSource(source());
        return manager;
    }
}
```

```java
@Controller
@RequestMapping("/welcome")
public class WelcomeController {
    @RequestMapping(method = RequestMethod.GET)
    public String welcomeAll() {
        return "welcome all";
    }
}
```

A Guide to Spring Framework Annotations
https://dzone.com/articles/a-guide-to-spring-framework-annotations

Spring autowired

- Using annotations like *@Autowired*, the Container is asked to inject a dependency where it is needed, and the programmers do not need to create/manage those dependencies by themselves.

```
public class MyClass1 {

    @Autowired
    private MyClass2 myClass2;

    public void doSomething(){
        myClass2.doSomething();
    }
}
```

```
public class MyClass2 {

    @Autowired
    private MyClass3 myClass3;
    @Autowired
    private MyClass4 myClass4;

    public void doSomething(){
        myClass3.doSomething();
        myClass4.doSomething();
    }
}
```

@SpringBootApplication

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

- Adds following annotations:
  - @Configuration –
    - Marks class as a source of bean definitions
  - @EnableAutoConfiguration
    - add beans based on the dependencies on the classpath automatically
  - @ComponentScan –
    - scans for other configurations and beans in the same package as the Application class or below

Migrating from Spring to Spring Boot
http://www.baeldung.com/spring-boot-migration

@SpringBootApplication

- We use this annotation to mark the main class of a Spring Boot application:

```
@SpringBootApplication
class VehicleFactoryApplication {

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

- **@SpringBootApplication encapsulates @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.**

Spring beans

- objects managed by the Spring IoC container
- created container
  - With configuration metadata
  - Provided by developer

```java
package com.example.service;

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

@Service
public class DatabaseAccountService implements AccountService {

        private final RiskAssessor riskAssessor;

        @Autowired
        public DatabaseAccountService(RiskAssessor riskAssessor)
                this.riskAssessor = riskAssessor;
        }

        // ...

}
```

configurations

Deploying Spring Apps to Tomcat (Without web.xml)

For those out there who haven't made the jump to Servlet's annotations, a simple sample Spring app to Tomcat without a web.xml file.

by Mohamed Sanaulla MVB · Nov. 21, 17 · Java Zone

Like (7) Comment (2) Save Tweet

Just released, a free O'Reilly book on Reactive Microsystems: The Evolution of Microservices a in partnership with Lightbend.

Since the Servlet 3 specification, `web.xml` is no longer needed for configuring your has been replaced by annotations. In this article, we will look at how to deploy a s application without `web.xml` to Tomcat 8.5.x.

Creating an Empty Application

Use the following command to create an empty web application using the Maven

```
mvn archetype:generate -DgroupId=info.sanaulla -DartifactId=spring-tomcat-sample
   -Dversion=1.0 -DarchetypeArtifactId=maven-archetype-webapp
```

Delete the `web.xml` created in `src\main\webapp\WEB-INF`. Then, we need to update the `m` fail if `web.xml` is missing. This can be done by updating the plugin information in t shown below:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>3.1.0</version>
    <executions>
        <execution>
            <id>default-war</id>
            <phase>prepare-package</phase>
            <configuration>
                <failOnMissingWebXml>false</failOnMissingWebXml>
            </configuration>
        </execution>
```

Deploying Spring Apps to Tomcat (Without web.xml)
https://dzone.com/articles/deploying-spring-apps-to-t

```
@Configuration
public class ViewConfiguration {

    @Bean
    public ClassLoaderTemplateResolver templateResolver() {
        ClassLoaderTemplateResolver templateResolver =
            new ClassLoaderTemplateResolver();
        templateResolver.setPrefix("templates/");
        templateResolver.setSuffix(".html");
        templateResolver.setTemplateMode(TemplateMode.HTML);
        templateResolver.setCacheable(false);
        return templateResolver;
    }

    @Bean
    public SpringTemplateEngine templateEngine() {
        SpringTemplateEngine templateEngine =
            new SpringTemplateEngine();
        templateEngine.setTemplateResolver(templateResolver());
        templateEngine.addDialect(new LayoutDialect());
        return templateEngine;
    }

    @Bean
    public ViewResolver viewResolver() {
        ThymeleafViewResolver viewResolver =
            new ThymeleafViewResolver();
        viewResolver.setTemplateEngine(templateEngine());
        viewResolver.setCharacterEncoding("UTF-8");
        return viewResolver;
    }

}
```

@Configuration & @ComponentScan

- @Configuration
  - used by the Spring IoC container
  - source of bean definitions
- @ComponentScan
  - search packages for Components.

```java
@Configuration
public class SpringConfigJavaCode {

  @Bean
  public App app() {
    return new App(logger());
  }

  /* @Bean
  public App app() {
    App app = new App();

    Service[] services = {database(), logger(), mail()};
    app.setServices(services);
    app.setMainService(services[1]);
    app.setId(1234);
    return app;
  } */

  @Bean
  public Database database() {
   return new Database();
  }
```

Without IoC / DI

https://www.tutorialspoint.com/spring/spring_java_based_configuration.htm

@Configuration

- Configuration classes can contain bean definition methods annotated with @Bean:

```
@Configuration
class VehicleFactoryConfig {

    @Bean
    Engine engine() {
        return new Engine();
    }
}
```

@Bean

- Spring @Bean annotation tells that a method produces a bean to be managed by the Spring container.
- It is a method-level annotation.
  - During Java configuration (@Configuration),
  - the method is executed and its
  - return value is registered as a bean within a BeanFactory.

http://zetcode.com/articles/springbootbean/

Beans have scopes

singleton

This scopes the bean definition to a single instance per Spring IoC container (default).

prototype

This scopes a single bean definition to have any number of object instances.

request

This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

session

This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

global-session

This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

https://www.tutorialspoint.com/spring/spring_bean_scopes.htm

@Component & @Bean

- @Component
  - Most generic Spring annotation.
  - Specializations
    - @Service, @Repository, and @Controller
- @Bean
  - Object/method that should be registered as a bean

```
@Configuration
public class SpringConfigJavaCode {

  @Bean
  public App app() {
    return new App(logger());
  }

  /* @Bean
  public App app() {
    App app = new App();

    Service[] services = {database(), logger(), mail()};
    app.setServices(services);
    app.setMainService(services[1]);
    app.setId(1234);
    return app;
  } */

  @Bean
  public Database database() {
   return new Database();
  }
```

@Component & @Bean

- @Component
  - Spring scans all the classes
  - register it as a bean
  - inject the dependencies
  - @Autowired annotation.
- @Bean
  - similar to @Component.
  - not autodetected.
  - produce a bean to be managed by the Spring container during configuration stage.

```java
@Configuration
public class SpringConfigJavaCode {

  @Bean
  public App app() {
    return new App(logger());
  }

  /* @Bean
  public App app() {
    App app = new App();

    Service[] services = {database(), logger(), mail()};
    app.setServices(services);
    app.setMainService(services[1]);
    app.setId(1234);
    return app;
  } */

  @Bean
  public Database database() {
   return new Database();
  }
```

Without IoC / DI

Spring: A Head Start — Beans Configuration (Part 2)
https://medium.com/omarelgabrys-blog/spring-a-head-start-beans-configu

Injection & Annotations

- Not specific to Spring
- 3 options to inject
  - Constructor Injection
  - Setter Injection
  - Field Injection
- Out of scope comparison

```
@Autowired
public App(Service[] services){
  this.setServices(services);
}
```

```
@Autowired
@Qualifier("logger")
public void setMainService(Service mainService) {
  this.mainService = mainService;
}
```

```
@Autowired
@Qualifier("logger")
private Service mainService;
```

Spring: A Head Start 🔥 — Beans Configuration (Part 2)
https://medium.com/omarelgabrys-blog/spring-a-head-start-beans-configuration-part-2-4a8c239b070a

ConcretePage.com

API CONFERENCE April 11 – 13, 2018 The Conference for Web AP

HOME CORE JAVA JAVA EE FRAMEWORK UTILITY WEB SERVICES TOOLS WEB INTERVIEW QUIZ ANDROID FORUM

Spring Auto-Detection with @Component, @Service, @Repository and @Controller Stereotype Annotation Example using @ComponentScan and component-scan

By Arvind Rai, January 17, 2016

This page will walk through spring auto-detection with @Component, @Service, @Repository and @Controller stereotype annotation example using @ComponentScan and component-scan. The spring stereotype @Component is parent stereotype. The other stereotypes i.e @Service, @Repository and @Controller are the specialization of @Component annotation. For component auto-detection, we need to provide base package name. In java configuration @ComponentScan is used and in XML component-scan is used for auto component scanning. Here we will describe the usability of these annotations with complete example.

Contents

Spring Auto-Detection …
https://goo.gl/6D6dPv

@Component

- @Component is a class level annotation. During the component scan, Spring Framework automatically detects classes annotated with @Component.

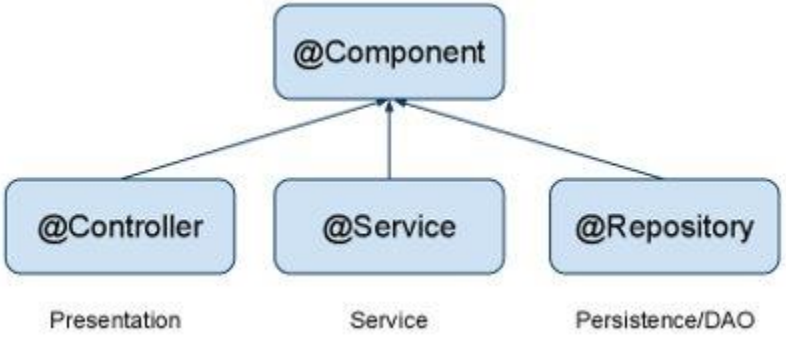
```
@Component
class CarUtility {
    // ...
}
```

- meta-annotations of @Component
  - @Repository, @Service, @Configuration, and @Controller
  - same bean naming behavior.
- Spring automatically picks them up during the component scanning process.

https://www.baeldung.com/spring-bean-annotations

@Component

```java
package com.concretepage.utility;
import org.springframework.stereotype.Component;
@Component
public class BookUtility {
        public int calculateArea(int length, int width) {
                return length * width;
        }
}
```



Spring Auto-Detection …
https://goo.gl/6D6dPv

@Service

- The business logic of an application usually resides within the service layer

@Service

"an operation offered as an interface that stands alone in the model, with no encapsulated state."

specialization of @Component

```java
@Service
public class BookService {
        @Autowired
        private BookDAO bookDAO;
        @Autowired
        private BookUtility utility;
        public String largestAreaBookName() {
                int lgarea = 0;
                String bookName = "";
                for (int i=0;i < 2;i++) {
                        Book book = bookDA
                        int area = utility
                        if (lgarea < area)
                                lgarea = a
                                bookName =
                        }

                }
                return bookName;
        }
}
```

```java
@Service
public class CountryService implements ICountryService {

    @Autowired
    private CountryRepository repository;

    @Override
    public List<Country> findAll() {

        List<Country> countries = (List<Country>) repository.findAll();
        return countries;
    }
}
```

Spring Auto-Detection ...
https://goo.gl/6D6dPv

Spring Boot @Repository
http://zetcode.com/springboot/repository/

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Service.html

@Repository

```
@Repository
public class BookDAO {
        List<Book> bookList = new ArrayList<>();
        {
                Book book1 = new Book(1, "Ramayana",5,4);
                Book book2 = new Book(2, "Mahabharat",6,3);
                bookList.add(book1);
                bookList.add(book2);
        }
        public Book getB
                //In rea
                return b
        }
}
```

"a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects"

specialization of [@Component](),

```
package com.zetcode.repository;

import com.zetcode.bean.Country;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CountryRepository extends CrudRepository<Countr

}
```

Spring Auto-Detection …
https://goo.gl/6D6dPv

Spring Boot @Repository
http://zetcode.com/springboot/repository/

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Repository.html

@Repository

```
public interface JpaRepository<T, ID extends Serializable> extends
  PagingAndSortingRepository<T, ID> {

    List<T> findAll();

    List<T> findAll(Sort sort);

    List<T> save(Iterable<? extends T> entities);

    void flush();

    T saveAndFlush(T entity);

    void deleteInBatch(Iterable<T> entities);
}
```

```
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

    <S extends T> S save(S entity);

    T findOne(ID primaryKey);

    Iterable<T> findAll();

    Long count();

    void delete(T entity);

    boolean exists(ID primaryKey);
}
```

Return to this later

CrudRepository, JpaRepository, and PagingAndSortingRepository in Spring Data
http://www.baeldung.com/spring-data-repositories

@Repository

- DAO or Repository classes usually represent the database access layer in an application, and should be annotated with @Repository:

```
@Repository
class VehicleRepository {
    // ...
}
```

- One advantage of using this annotation is that it has automatic persistence exception translation enabled. When using a persistence framework such as Hibernate, native exceptions thrown within classes annotated with @Repository will be automatically translated into subclasses of Spring's DataAccessExeption.

https://www.baeldung.com/spring-bean-annotations

@Controller

- @Controller is a class level annotation which tells the Spring Framework that this class serves as a controller in Spring MVC:

```
@Controller
public class VehicleController {
    // ...
}
```

@Controller

```java
@Controller
@RequestMapping(value = "/book")
public class BookController {
        @Autowired
        private BookService service;
        @RequestMapping(value = "/service")
        public ModelAndView bookName(ModelAndView  modelAndView) {
                modelAndView.addObjec
                modelAndView.setViewN
                return modelAndView;
        }
}
```

```java
@Controller
public class MyController {

    @Autowired
    ICountryService countryService;

    @RequestMapping("/showCountries")
    public ModelAndView findCities() {

        List<Country> countries = (List<Country>) countryService.findAll();

        Map<String, Object> params = new HashMap<>();
        params.put("countries", countries);

        return new ModelAndView("showCountries", params);
    }
}
```

Spring Auto-Detection …
https://goo.gl/6D6dPv

Spring Boot @Repository
http://zetcode.com/springboot/repository/

@RestController

- Combines @Controller and @ResponseBody.

- Therefore, the following declarations are equivalent:

```
@Controller
@ResponseBody
class VehicleRestController {
    // ...
}

@RestController
class VehicleRestController {
    // ...
}
```

@RestController

```java
@RestController
public class RegistrationController {

    @RequestMapping(method = RequestMethod.POST,
                    value = "/register",
                    produces = APPLICATION_JSON_VALUE)
    public UserData register(@RequestBody User user) {
        ...
        if(usernameAlreadyExists) {
            throw new IllegalArgumentException("error.username");
        }
        ...
        return new UserData(...);
    }

    @ExceptionHandler
    void handleIllegalArgumentException(
                    IllegalArgumentException e,
                    HttpServletResponse response) throws IOExcepti

        response.sendError(HttpStatus.BAD_REQUEST.value());

    }
}
```

```java
public class User {

    private String mail;
    private String passwo
    private String lastNa
    private String name;
    private String addres

    public Registration()

    //... getter and sett
}
```

Spring Boot tutorial: REST services and microservices
https://jaxenter.com/spring-boot-tutorial-rest-services-and-microservices-135148.html

Want more annotations?

- Spring Boot Annotations
  - https://www.baeldung.com/spring-boot-annotations
- Spring Web Annotations
  - https://www.baeldung.com/spring-mvc-annotations
- Spring Bean Annotations
  - https://www.baeldung.com/spring-bean-annotations
- Spring Core Annotations
  - https://www.baeldung.com/spring-core-annotations

@ and IoC/DI no only on Spring Boot

Android lifecyle

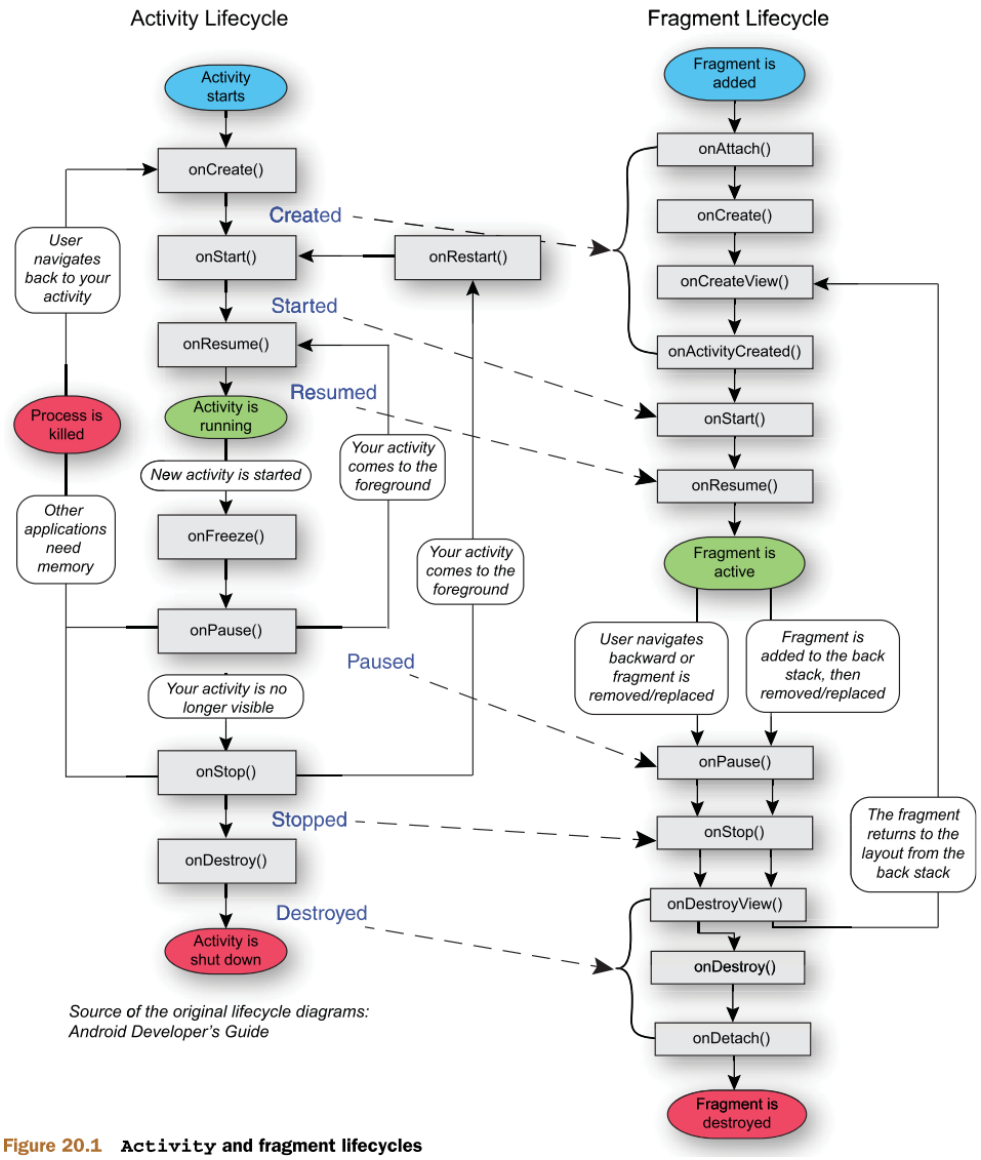
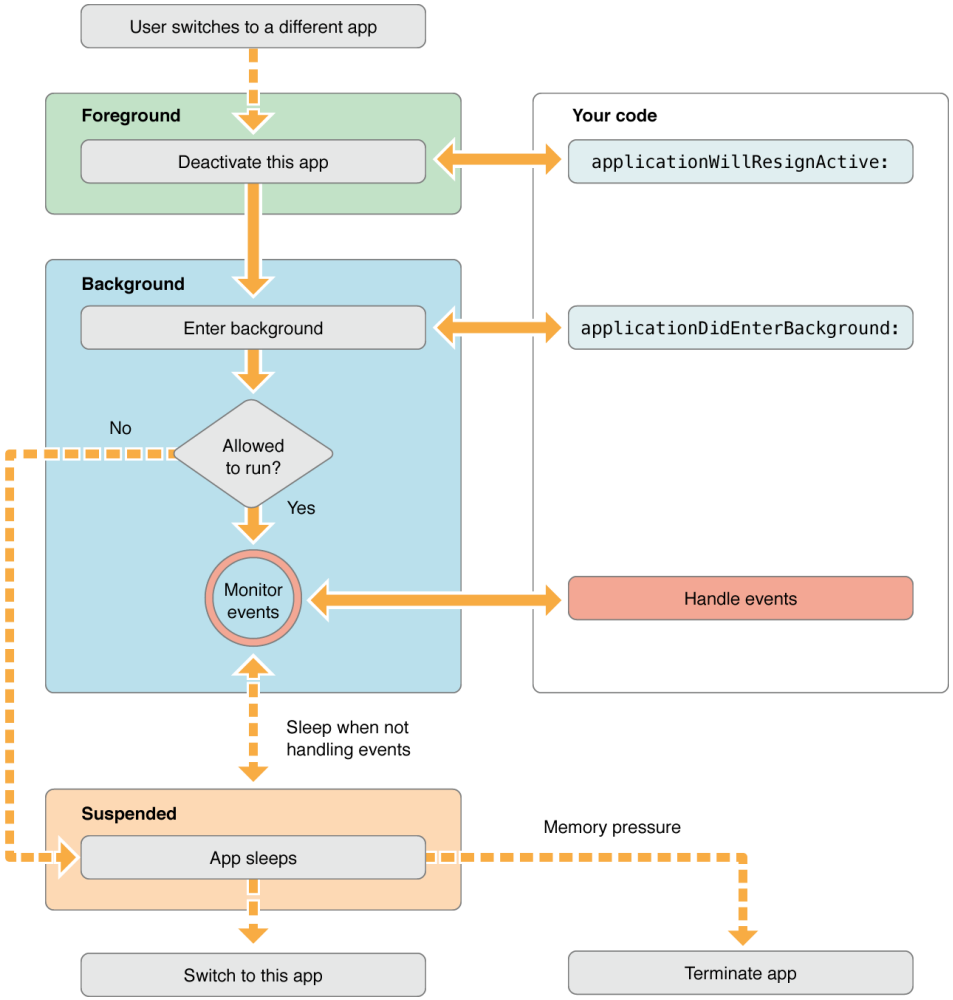
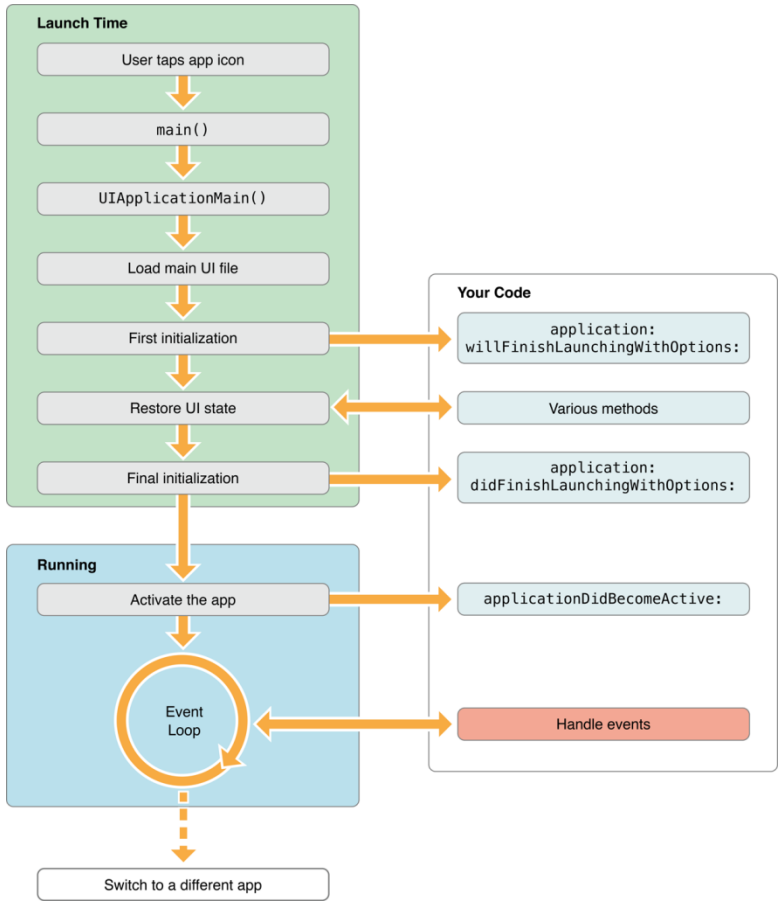


Figure 20.1 `Activity` and fragment lifecycles

https://github.com/xxv/android-lifecycle

iOS application lifecycle

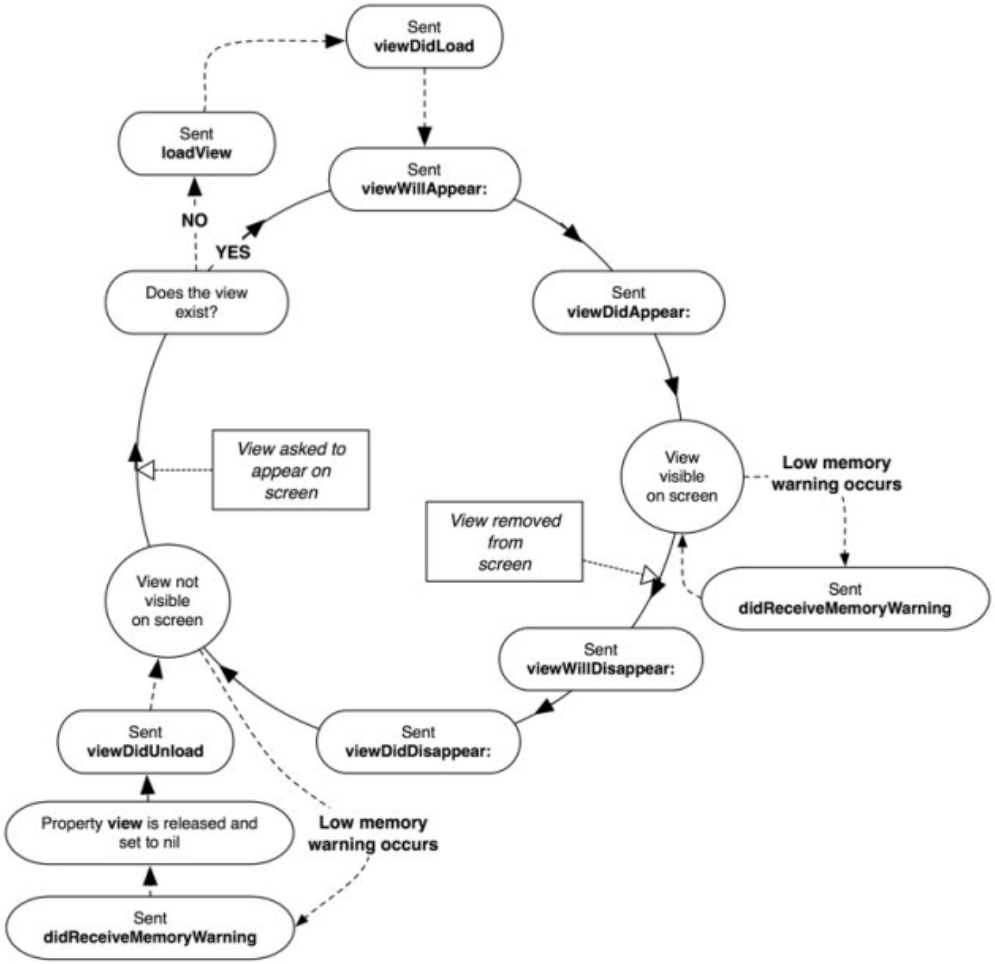




https://stackoverflow.com/questions/6519847/what-is-the-life-cycle-of-an-iphone-application

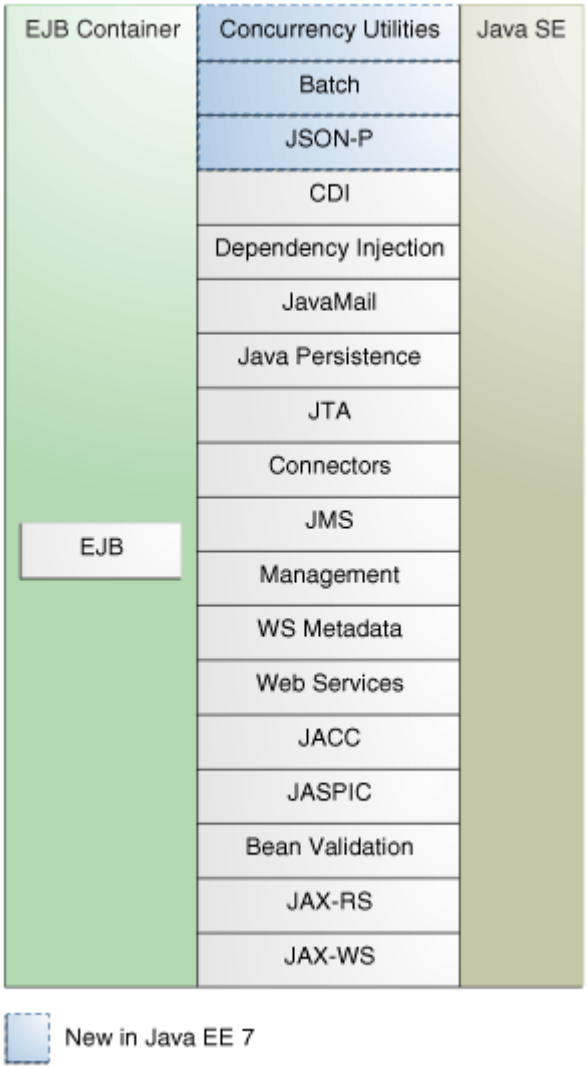
https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html

Ios viewcontroller lifecycle

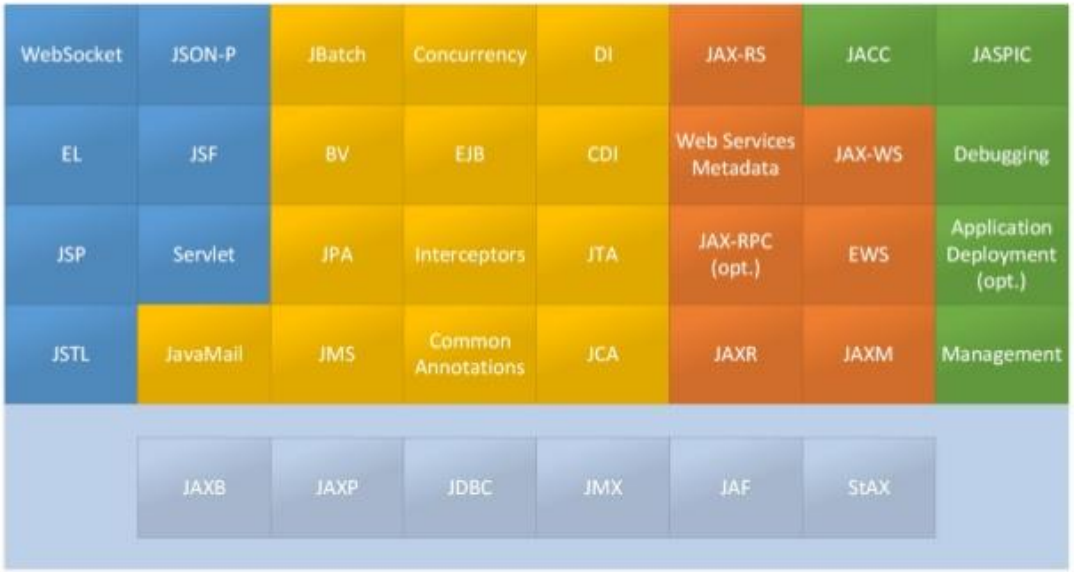


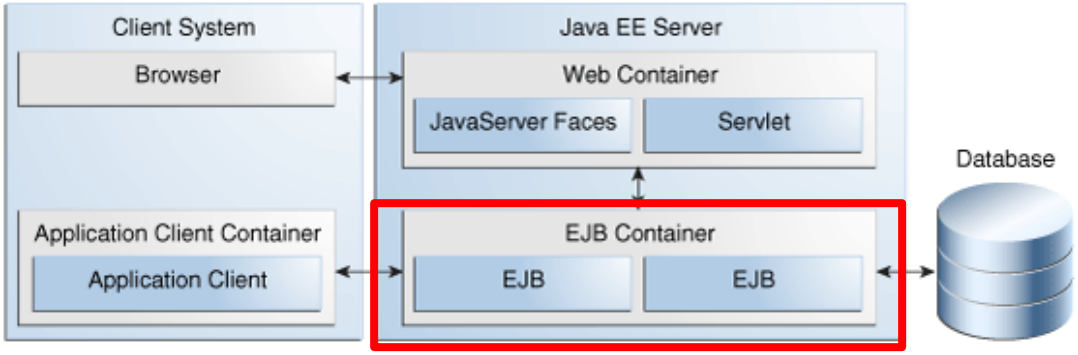
https://stackoverflow.com/questions/6519847/what-is-the-life-cycle-of-an-iphone-application

Now focus on Java EE

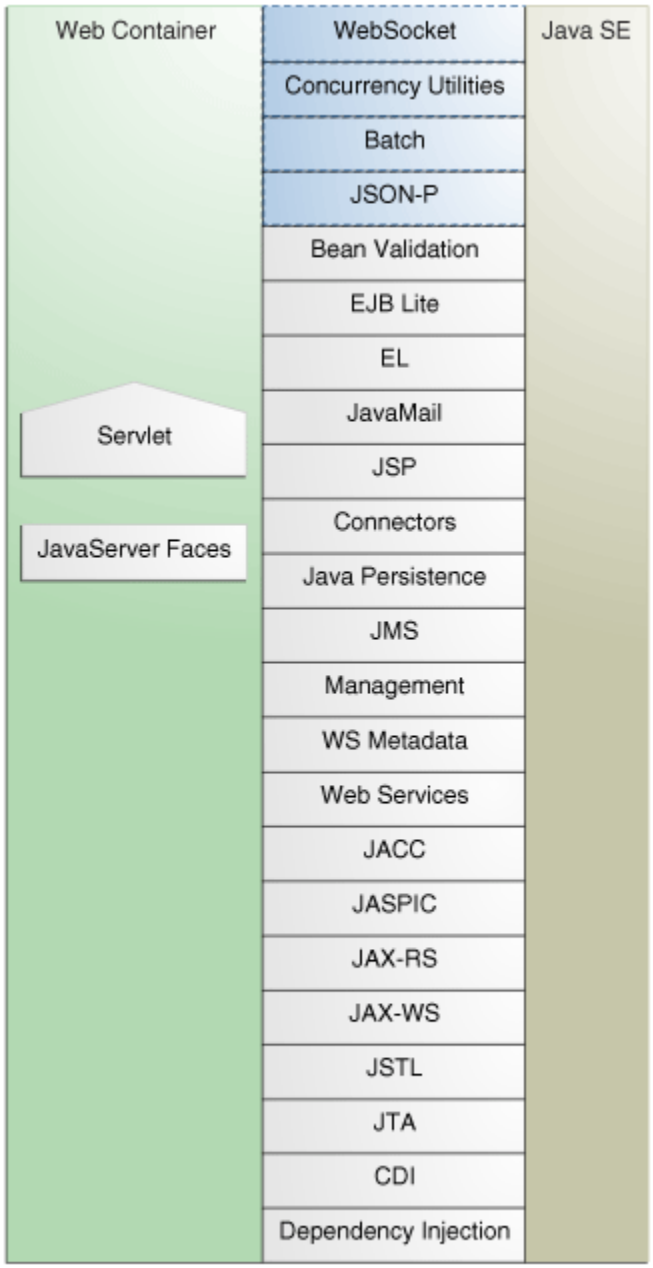


Java EE 7 Platform

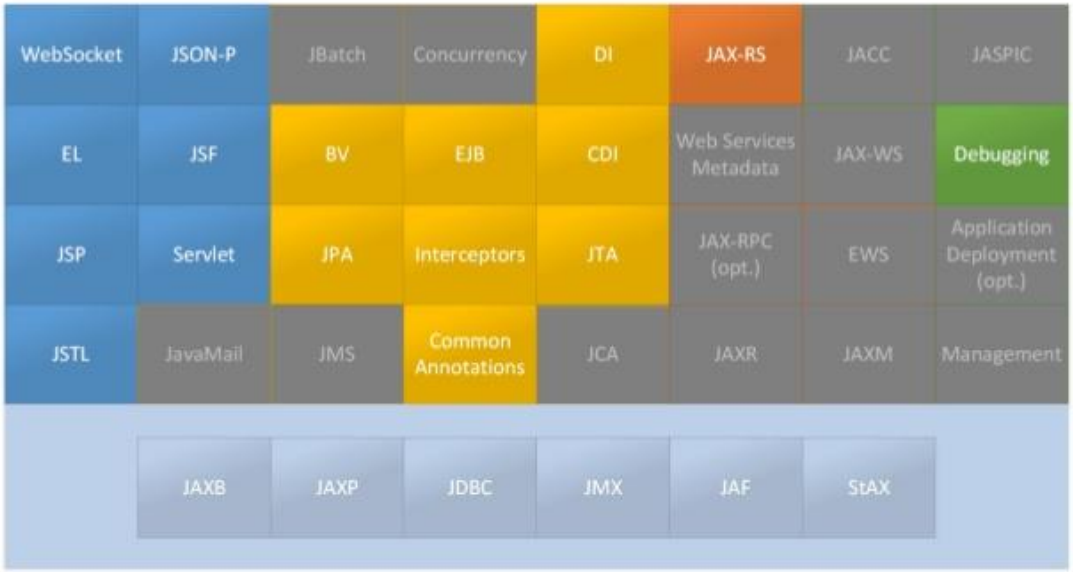


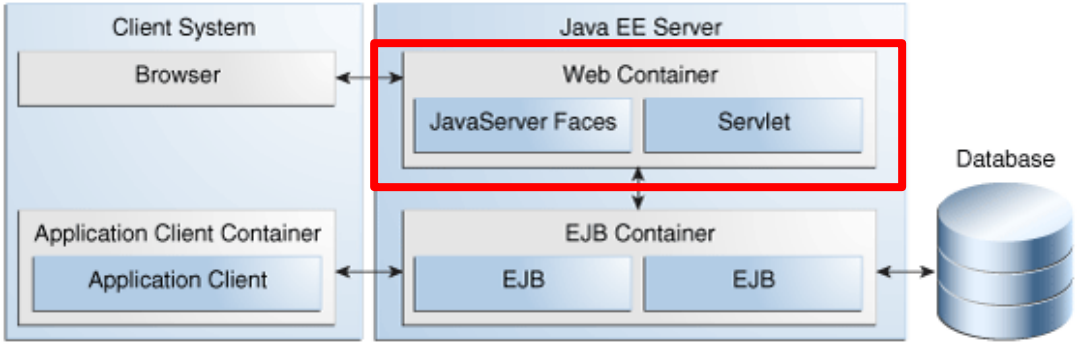


https://www.slideshare.net/khasunuma/java-eewebprofile-75404485



Java EE 7 Web Profile





ewebprofile-75404485

One example: Wilfdfly coverage

Getting Started with WildFly 10

WildFly 10 is the latest release in a series of JBoss open-source appl
specifications. The state-of-the-art architecture built on the Modular Service Container enables services on-demand when your application re
and the technologies available in WildFly 10 server configuration profiles.

Java EE 7 Platform Technology	Java EE 7 Full Profile	Java EE 7 Web Profile	WildFly 10 Full Profile	WildFly 10 Web Profile
JSR-356: Java API for Web Socket	X	X	X	X
JSR-353: Java API for JSON Processing	X	X	X	X
JSR-340: Java Servlet 3.1	X	X	X	X
JSR-344: JavaServer Faces 2.2	X	X	X	X
JSR-341: Expression Language 3.0	X	X	X	X
JSR-245: JavaServer Pages 2.3	X	X	X	X
JSR-52: Standard Tag Library for JavaServer Pages (JSTL) 1.2	X	X	X	X
JSR-352: Batch Applications for the Java Platform 1.0	X	--	X	--
JSR-236: Concurrency Utilities for Java EE 1.0	X	X	X	X
JSR-346: Contexts and Dependency Injection for Java 1.1	X	X	X	X
JSR-330: Dependency Injection for Java 1.0	X	X	X	X
JSR-349: Bean Validation 1.1	X	X	X	X
JSR-345: Enterprise JavaBeans 3.2	X CMP 2.0 Optional	X (Lite)	X CMP 2.0 Not Available	X (Lite)
JSR-318: Interceptors 1.2	X	X	X	X
JSR-322: Java EE Connector Architecture 1.7	X	--	X	X
JSR-338: Java Persistence 2.1	X	X	X	X
	X	X	X	X
JSR-343: Java Message Service API 2.0	X	--	X	--

https://docs.jboss.org/author/display/WFLY10/Getting+Started+Guide

@stateless, @stateful, @EJB


```java
@Stateless
public class StatelessEJB {

    public String name;

}
```

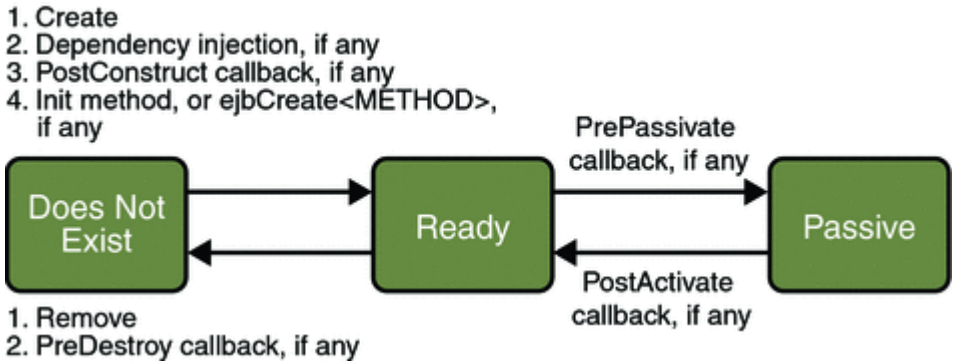
```java
@Stateful
public class StatefulEJB {

    public String name;

}
```

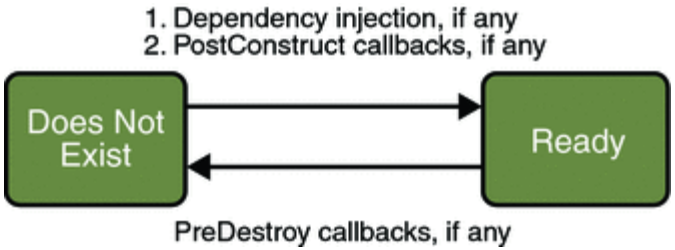
Concurrency and resource management
Transparent (Binary / text)

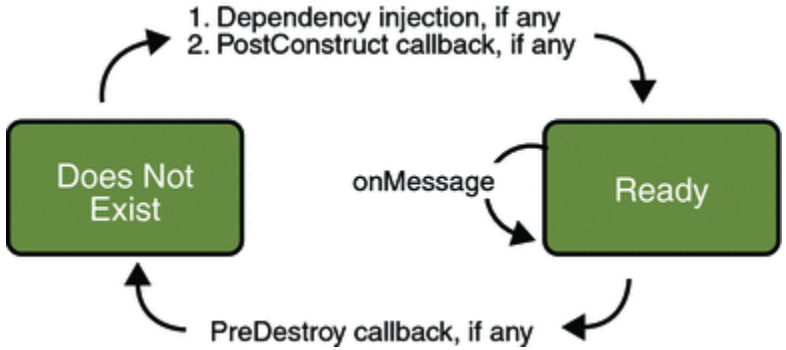
```java
public class EJBClient1 {

    @EJB
    public StatefulEJB statefulEJB;

}
```

http://www.baeldung.com/ejb-session-beans

Java EJB lifecycle





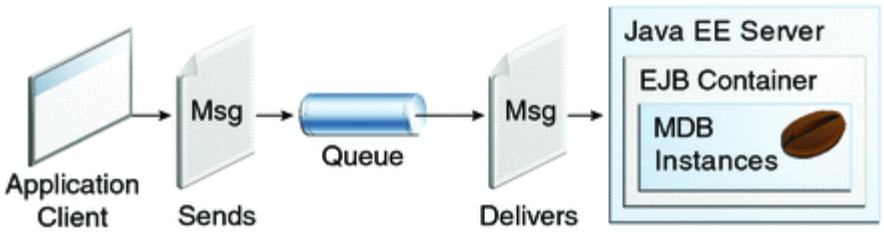


https://docs.oracle.com/javaee/5/tutorial/doc/bnbmt.html

MDB: Listeners on JavaEE

- A message driven bean (MDB) is a bean that contains business logic. But, it is invoked by passing the message. So, it is like JMS Receiver.
- MDB asynchronously receives the message and processes it.
- A message driven bean r… message from q… pic, so you must ha… wledge of JMS API.
- De… DI from application server

We will see in future classes




```
@MessageDriven(mappedName="myTopic")
public class MyListener implements
MessageListener{
    @Override
    public void onMessage(Message msg)
{
        TextMessage m=(TextMessage)msg;
        try{
        System.out.println("message
received: "+m.getText());
        }catch(Exception
e){System.out.println(e);}
    }
}
```

@Singleton

```
@Singleton
public class DatabaseBean {
        @PostConstruct
        public void initialize() {
                // Create database tables.
        }
}

@Singleton
@DependsOn({"DatabaseBean"})
public class ConfigurationBean implements Configuration {
        @PostConstruct
        public void initialize() {
                // Initialize settings from a database table.
        }

        // ...
}
```

@Singleton

```
@Singleton
public class DatabaseBean {
        @PostConstruct
        public void initialize() {
                // Create database tables.
        }
}
```

```
@Singleton
@DependsOn({
public class
        @Pos
        publ

        }

        // .
}
```

```
@Singleton
@Lock(READ)
public class ConfigurationBean implements Configuration {

        public Object businessMethod(long value) {
                // ...
        }

        public Object businessMethod(long value, int i, Object value) {
                // ...
        }
```

Java EE Interceptors

```
@Stateless
@Interceptors({ //global to EJB class
    PreNormizedInterceptor.class,
    ContactsNormalizerInterceptor.class,
    PostNormizedInterceptor.class,
})
public class ContactsEJB implements ContactsRemote {
```

```
public class LifecycleInterceptor {
    @AroundConstruct
    public void ctor(InvocationContext ctx) {
       ...
    }

    @PostConstruct
    public void init(InvocationContext ctx) {
       ...
    }

    @PreDestroy
    public void destory(InvocationContext ctx) {
    ...
    }
```

https://examples.javacodegeeks.com/enterprise-java/ejb3/ejb-interceptors-example/

JavaEE interceptors

```
public class LoggingIntercept {
   ...
   @PostConstruct
   private Object doLog(InvocationContext ic) {... }
}
```

```
@Transactional
@Interceptors(LoggingIntercept.class)
public class BookLibraryService {
   @Inject
   private EntityManager entityManager;
   @Inject
   private Logger logger;

   public void createPatron(Patron patron) {
      entityManager.persist(patron);
   }

   @ExcludeClassInterceptors
   public Patron searchPatron(Long patronId) {
      return entityManager.find(Patron.class, patronId);
   }
}
```

http://webdev.jhuep.com/~jcs/ejava-javaee/coursedocs/content/html/ejb-interceptor.html
https://www.developer.com/java/understanding-interceptors-for-java-ee.html

JPA lifecycle

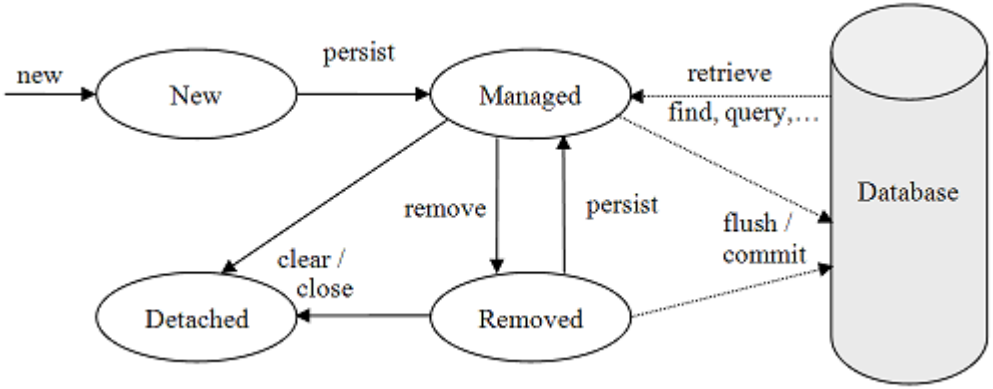


Table 6.1. Callbacks

Type	Description
@PrePersist	Executed before the entity manager persist operation is actually executed or cascaded. This call is synchronous with the persist operation.
@PreRemove	Executed before the entity manager remove operation is actually executed or cascaded. This call is synchronous with the remove operation.
@PostPersist	Executed after the entity manager persist operation is actually executed or cascaded. This call is invoked after the database INSERT is executed.
@PostRemove	Executed after the entity manager remove operation is actually executed or cascaded. This call is synchronous with the remove operation.
@PreUpdate	Executed before the database UPDATE operation.
@PostUpdate	Executed after the database UPDATE operation.
@PostLoad	Executed after an entity has been loaded into the current persistence context or an entity has been refreshed.

https://docs.jboss.org/hibernate/orm/4.0/hem/en-US/html/listeners.html
http://www.objectdb.com/java/jpa/persistence/managed

Hands on examples

Schedule tasks


```
import javax.ejb.Schedule;
import javax.ejb.Singleton;

@Singleton
public class DeclarativeScheduler {

    @Schedule(second = "*/5", minute = "*", hour = "*", persistent = false)
    public void atSchedule() throws InterruptedException {
        System.out.println("DeclarativeScheduler:: In atSchedule()");
    }
}
```

https://examples.javacodegeeks.com/enterprise-java/ejb3/ejb-schedule-example/
https://www.baeldung.com/scheduling-in-java-enterprise-edition

Schedule tasks


```
import javax.ejb.Schedule;
import javax.ejb.Singleton;

@Singleton
public
    @S
    pu

    }
}
```

```
@Startup
@Singleton
public class ProgrammaticScheduler {

    @Resource
    TimerService timerService;

    @PostConstruct
    public void initialize() {
        timerService.createTimer(0, 4000, "Every four second timer with no de
    }

    @Timeout
    public void programmaticTimeout(Timer timer) {
        System.out.println("ProgrammaticScheduler:: in programmaticTimeout");
    }
}
```

https://exam
https://www.

Schedule tasks

```java
@SpringBootApplication
@EnableScheduling
public class Application {

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class);
    }
}
```

```java
@Component
public class ScheduledTasks {

    private static final Logger log = LoggerFactory.getLogger(ScheduledTasks.class);

    private static final SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss

    @Scheduled(fixedRate = 5000)
    public void reportCurrentTime() {
        log.info("The time is now {}", dateFormat.format(new Date()));
    }
}
```

https://spring.io/guides/gs/scheduling-tasks/

Thinking in Java EE (at least trying to!)

Write Once, Read Forever

Java EE™

Me Java EE 7 Books, articles etc.. Java Magazine Hub

← New release: Java WebSocket API handbook

Quick tip: managing Stateful EJBs in WebSocket endpoints →

WebSocket endpoint as Singleton EJB

Posted on February 14, 2017

By default ...

... a WebSocket implementation creates a **new** (server) endpoint instance per client. In case you need a **single** instance, you can implement this using a custom `ServerEndpointConfig.Configurator` (by overriding the `getEndpointInstance` method)

The catch: you might have to sacrifice some of the (Java EE) platform related services like dependency injection and interceptors

Java EE eBooks on Leanpub

JAX-RS, EJB & WebSocket bundle

DZone Java Caching Refcard

WebSocket endpoint as Singleton EJB
https://abhirockzz.wordpress.com/2017/02/14/websocket-endpoint-as-singleton-ejb/

Thinking in Java EE (at least trying to!)

Write Once, Read Forever

```
@Singleton
@ServerEndpoint("/singleton/")
public class SingletonEndpoint {
    @OnOpen
    public void onopen(Session s) throws IOException {
        s.getBasicRemote().sendText(String.valueOf(hashCode()));
    }
    @PreDestroy
    public void onDestroy() {
        System.out.println("Singleton bean " + hashCode() + " will be destroyed");
    }
    @OnClose
    public void onClose(Session session, CloseReason closeReason) {
        System.out.println("Closed " + session.getId() + " due to " + closeReason.getCloseCode());
    }
}
```

The catch: you might have to sacrifice some of the (Java EE) platform related services like dependency injection and interceptors

WebSocket endpoint as Singleton EJB
https://abhirockzz.wordpress.com/2017/02/14/websocket-endpoint-as-singleton-ejb/

Websocket in JavaEE: singleton

```
@Singleton
@ServerEndpoint("/singleton/")
public class SingletonlEndpoint {

    @OnOpen
    public void onOpen(Session s) throws IOException {
        s.getBasicRemote().sendText(String.valueOf(hashCode()));
    }

    @PreDestroy
    public void onDestroy() {
        System.out.println("Singleton bean " + hashCode() + " will be destroyed");
    }

    @OnClose
    public void onClose(Session session, CloseReason closeReason) {
        System.out.println("Closed " + session.getId() + " due to " + closeReason.getCloseCode());
    }
}
```

java websocket api handbook
https://abhirockzz.gitbooks.io/java-websocket-api-handbook/content/

Websocket in JavaEE: stateful


```
@Singleton
@ServerEndpoint("/singleto
public class SingletonlEnd

    @OnOpen
    public void onOpen(Ses
        s.getBasicRemote()
    }

    @PreDestroy
    public void onDestroy(
        System.out.println
    }

    @OnClose
    public void onClose(Se
        System.out.println
    }
}
```

```
@Stateful
@ServerEndpoint("/chat/{user}")
public class StatefulChat {
    private transient Session s;
    private String userID;
    private List<History> history;

    @OnOpen
    public void onOpen(@PathParam("user") String user, Session s) throws IOException {
        this.userID= user;
        this.s = s;
    ....
    }

    @OnMessage
    public void chat(String msg) {
        history.add(msg);
       //route message to intended recipient(s)
    }
```

java websocket api handbook
https://abhirockzz.gitbooks.io/java-websocket-api-handbook/content/

Websocket in JavaEE: stateful


```
@Singleton
@ServerEndpoint("/singleto
public class SingletonlEnd

    @OnOpen
    public void onOpen(Ses
        s.getBasicRemote()
    }

    @PreDestroy
    public void onDestroy
        System.out.pr
    }

    @OnClose
    public void onClose(Se
        System.out.println
    }
}
```

```
@Stateful
@ServerEndpoint("/chat/{user}")
public class StatefulChat {
    private transient Session s;
    private String userID;
    private List<History> history

    @OnOpen
                    @PathParam("user") String user, Session s) throws IOException {
            userID= user;
            this.s = s;
    ....
    }

    @OnMessage
    public void chat(String msg) {
        history.add(msg);
       //route message to intended recipient(s)
    }
```

We will address websockets in more detail

java websocket api handbook
https://abhirockzz.gitbooks.io/java-websocket-api-handbook/content/

Intro to Inversion Control and Dependency Injection with Spring

Last modified: December 28, 2016

by Loredana Crusoveanu

For those curious

Spring

I just announced the new *Spring 5* modules in REST With Spring:

>> CHECK OUT THE COURSE

Intro to Inversion Control and Dependency Injection with Spring
http://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

Take home

- The @
- Some useful “tools” – lombok
- Regardless of technology / framework
  - Same principles
  - Same patterns
  - Similar “implementations”
- But not always the same way
  - Configuration / annotation
- Typically critical for developer
  - Either Spring boot or JavaEE
  - Creator of headaches and sleepless nights ;)

Some examples

spring by Pivotal

GETTING STARTED

Consuming a RESTful Web Service

This guide walks you through the process of creating an application that consumes a RESTful web service.

What you'll build

You'll build an application that uses Spring's `RestTemplate` to retrieve a random Spring Boot quotation at http://gturnquist-quoters.cfapps.io/api/random.

What you'll need

- About 15 minutes
- A favorite text editor or IDE
- JDK 1.8 or later
- Gradle 4+ or Maven 3.2+
- You can also import the code straight into your IDE:
  - Spring Tool Suite (STS)
  - IntelliJ IDEA

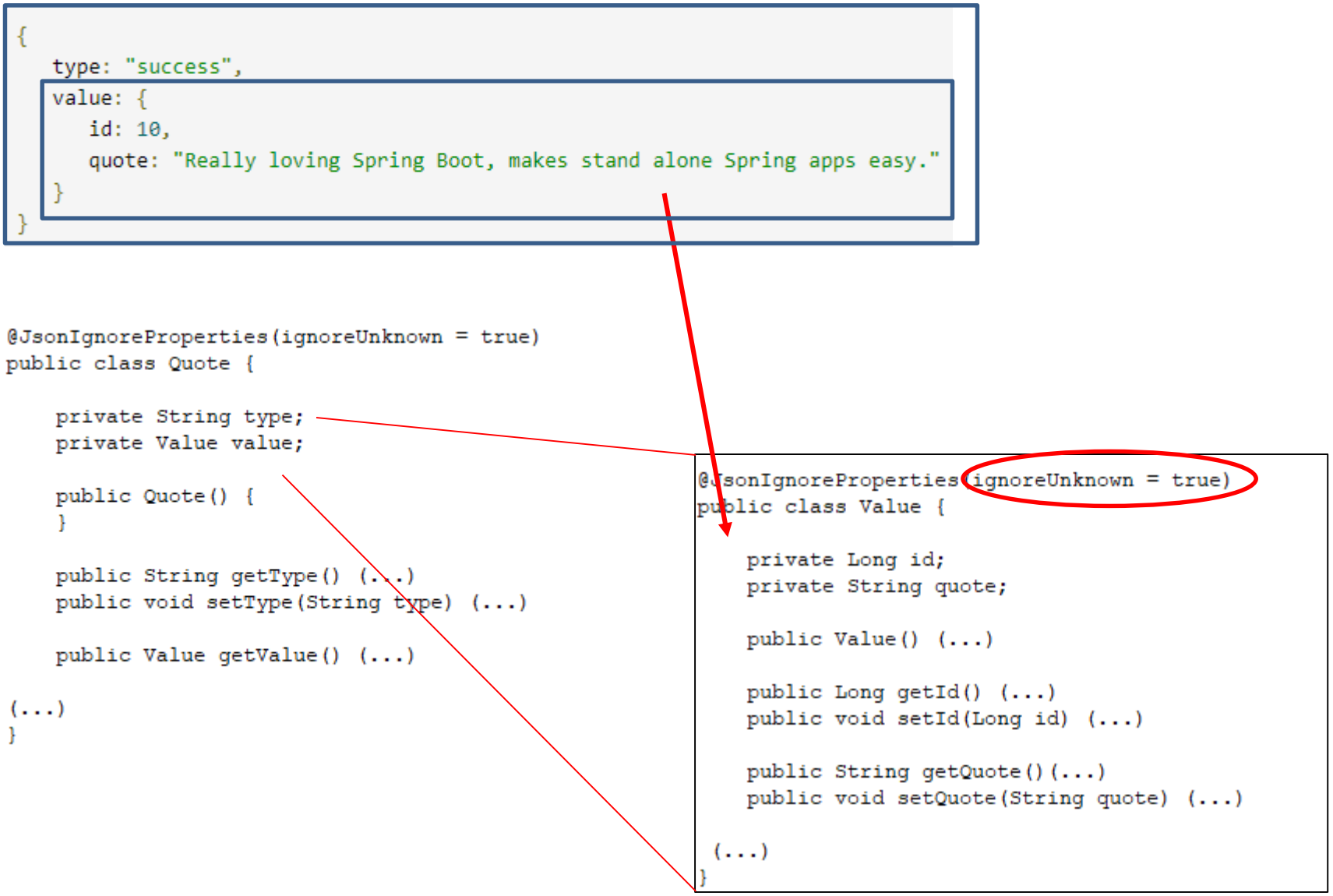
How

https://spring.io/guides/gs/consuming-rest/

Like most Spring Getting Started guides, you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with

```
{
   type: "success",
   value: {
      id: 10,
      quote: "Really loving Spring Boot, makes stand alone Spring apps easy."
   }
}
```

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class Quote {

    private String type;
    private Value value;

    public Quote() {
    }

    public String getType() (...)
    public void setType(String type) (...)

    public Value getValue() (...)

(...)
}
```


{
type: "success",
value: {
id: 10,
quote: "Really loving Spring Boot, makes stand alone Spring apps easy."
}
}
@JsonIgnoreProperties(ignoreUnknown = true)
public class Quote {
private String type;
private Value value;
public Quote() {
}
public String getType() (...)
public void setType(String type) (...)
public Value getValue() (...)
(...)
}
@JsonIgnoreProperties(ignoreUnknown = true)
public class Value {
private Long id;
private String quote;
public Value() (...)
public Long getId() (...)
public void setId(Long id) (...)
public String getQuote()(...)
public void setQuote(String quote) (...)
(...)
}

Spring: rest and schedule

Spring: rest and schedule

```
@SpringBootApplication
@EnableScheduling
public class Application {

    private static final Logger log = LoggerFactory.getLogger(Application.class);

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

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }

    @Bean
    public CommandLineRunner run(RestTemplate restTemplate) throws Exception {
        return args -> {
            Quote quote = restTemplate.getForObject(
                    "http://gturnquist-quoters.cfapps.io/api/random", Quote.class);
            log.info(quote.toString());
        };
    }
}
```

Spring: rest and schedule

```
@SpringBootApplication
@EnableScheduling
public class Application {

    private static final Logger log = LoggerFactory.getLogger(Application.class);

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

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }

    @Bean
    public CommandLineRunner
        return args -> {
            Quote quote =
                    "http:
            log.info(quote
        };
    }
}
```

```
@Component
public class ScheduledTasks {

    @Autowired
    private RestTemplate restTemplate;

    private static final Logger log = LoggerFactory.getLogger(ScheduledTasks.class);

    private static final SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss")

    @Scheduled(fixedRate = 5000)
    public void reportCurrentTime() {

        Quote quote = restTemplate.getForObject(
                    "http://gturnquist-quoters.cfapps.io/api/random", Quote.class);

        log.info(quote.toString());
    }
}
```

Spring: rest and schedule

```
@SpringBootApplication
@EnableScheduling
public class Application {

    private static final Logger log = LoggerFactory.getLogger(Application.class);

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

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }

    @Bean
    public CommandLineRunn
        return args -> {
            Quote quote =
                    "http:
            log.info(quote
        };
    }
}
```

```
@Component
public class ScheduledTasks {

    @Autowired
    private RestTemplate restTemplate;

    private static final Logger log = LoggerFactory.getLogger(ScheduledTasks.class);

    private static final SimpleDateFormat dateFormat = new SimpleDateFormat("HH:mm:ss")

    @Scheduled(fixedRate = 5000)
    public void reportCurrentTime() {

        Quote quote = restTemplate.getForObject(
                    "http://gturnquist-quoters.cfapps.io/api/random", Quote.class);

        log.info(quote.toString());
    }
}
```

Spring @Component, @Repository and @Service Annotations

By Lokesh Gupta | Filed Under: Spring Core

In spring autowiring, `@Autowired` annotation handles only wiring part. We still have to define the beans so the container is aware of them and can inject them for you.

With `@Component`, `@Repository`, `@Service` and `@Controller` annotations in place and **automatic component scanning** enabled, Spring will automatically import the beans into the container and inject to dependencies. These annotations are called **Stereotype annotations** as well.

https://howtodoinjava.com/spring-core/how-to-use-spring-component-repository-service-and-controller-annotations/

facts about these annotations which will help you in making a better

REST with Java (JAX-RS) using Jersey - Tutorial

Lars Vogel, (c) 2009, 2017 vogella GmbH – Version 2.7, 27.11.2017

Table of Contents

RESTful web services with Java (Jersey / JAX-RS). This tutorial explains how to develop RESTful web servi http://www.vogella.com/tutorials/REST/article.html *s tutorial Eclipse 4.7 (Oxygen), Java 1.8, Tomcat 6.0 and JAX-RS 2.0 (with Jersey 2.11) is*

REST v

Lars Vogel,

Table of

1. REST - Re
2. Installati
3. Web con
4. Requirec
5. Prerequi
6. Create y
7. Create a
8. RESTful v
9. CRUD RE
10. About t
11. Rest Re

Appendix A

```
public class Test {

    public static void main(String[] args) {
        ClientConfig config = new ClientConfig();

        Client client = ClientBuilder.newClient(config);

        WebTarget target = client.target(getBaseURI());

        String response = target.path("rest").
                          path("hello").
                          request().
                          accept(MediaType.TEXT_PLAIN).
                          get(Response.class)
                          .toString();


        String plainAnswer =

target.path("rest").path("hello").request().accept(MediaType.TEXT_PLAIN).get(String.class);
        String xmlAnswer =

target.path("rest").path("hello").request().accept(MediaType.TEXT_XML).get(String.class);
        String htmlAnswer=

target.path("rest").path("hello").request().accept(MediaType.TEXT_HTML).get(String.class);

        System.out.println(response);
        System.out.println(plainAnswer);
        System.out.println(xmlAnswer);
        System.out.println(htmlAnswer);
    }

    private static URI getBaseURI() {
        return UriBuilder.fromUri("http://localhost:8080/com.vogella.jersey.first").build();
    }
}
```

RESTful w *velop*

RESTful w *s*

tutorial Eclipse 4.7 (Oxygen), Java 1.8, Tomcat 6.0 and JAX-RS 2.0 (with Jersey 2.11) is

vogella

REST with Java (JAX-RS) using Jersey - Tutorial

Lars Vogel, (c) 2009, 2

Table of Content

1. REST - Representat
2. Installation of Jerse
3. Web container
4. Required setup for
5. Prerequisites
6. Create your first RE
7. Create a REST clien
8. RESTful web servic
9. CRUD RESTful web
10. About this websit
11. Rest Resources

Appendix A: Copyrigh

```
import java.net.URI;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;
import com.fasterxml.jackson.jaxrs.annotation.JacksonFeatures;

public class TodoTest {
    public static void main(String[] args) {
        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);

        WebTarget target = client.target(getBaseURI());

        // Get JSON for application
        String jsonResponse = target.request()
                .accept(MediaType.APPLICATION_JSON).get(String.class);

        System.out.println(jsonResponse);
    }

    private static URI getBaseURI() {

        String Url= "http://gturnquist-quoters.cfapps.io/api/random";

        return UriBuilder.fromUri( Url).build();
    }
```

RESTful web services with Java (Jersey / JAX-RS). This tutorial explains how to develop RESTful web servic http://www.vogella.com/tutorials/REST/article.html *s tutorial Eclipse 4.7 (Oxygen), Java 1.8, Tomcat 6.0 and JAX-RS 2.0 (with Jersey 2.11) is*

Baeldung Start Here Courses ▾ Guides ▾ About ▾

JAX-RS Client with Jersey

Last modified: August 10, 2018

by baeldung

Java EE REST Jersey

I just announced the new *Spring Boot 2* material, coming in REST With Spring:

>> CHECK OUT THE COURSE

1. Overview

Jersey is an open source framework for developing RESTful Web Services. It also has great inbuilt client capabilities.

In this quick tutorial, we will explore the creation of JAX-RS client using Jersey 2.

For a discussion on the creation of RESTful Web Services using Jersey, please refer to this article.

Further reading:

REST API with Jersey and Spring

Building Restful Web Services using Jersey 2 and Spring.

Read more →

CORS in JAX-RS

Learn how to implement Cross-Origin Resource Sharing (CORS) mechanism in JAX-RS based applications.

Read more →

Jersey Filters and Interceptors

Take a look at how filters and interceptors work in the Jersey framework.

Read more →

2. Maven Dependencies

Let's begin by adding the required dependencies (for Jersey JAX-RS client) in the *pom.xml*:

```
<dependency>
    <groupId>org.glassfish.jersey.core</groupId>
    <artifactId>jersey-client</artifactId>
    <version>2.25.1</version>
</dependency>
```

To use Jackson 2.x as JSON provider:

```
    <version>2.25.1</version>
</dependency>
```

https://www.baeldung.com/jersey-jax-rs-client

Useful details

How to Change the Default Port in Spring Boot

Last modified: May 1, 2018

by Loredana Crusoveanu

Spring

Spring Boot

I just announced the new *Spring Boot 2* material, coming in REST With Spring:

>> CHECK OUT THE COURSE

1. Introduction

Spring Boot provides sensible defaults for many configuration properties. Still, we sometimes need to customize these with our case-specific values.

A common use case is **changing the default port for the embedded server**.

In this quick tutorial, we'll cover several ways to achieve this.

2. Using Property Files

The fastest and easiest way to customize Spring Boot is by overriding the values of the default properties.

For the server port, the property we want to change is *server.port*.

By default, the embedded server start on port 8080. **Let's see how we can provide a different value in an *application.properties* file**:

```
server.port=8081
```

Now the server will

Similarly, we can do

```
server:
  port : 8081
```

How to Change the Default Port in Spring Boot
https://www.baeldung.com/spring-boot-change-port

Both files are loaded automatically by Spring Boot if placed in the *src/main/resources* directory of a Maven application.

2.1. Environment-Specific Ports

The END