Web Frameworks
Spring

UA.DETI.IES

Recap

1. Software development process
   - Different models (sequential, incremental, evolutionary, …)
2. Agile development methods
   - Agile principles and project management
3. DevOps Technical benefits
   - Continuous integration and software delivery

Team manager

Product owner

Architect

DevOps master

Recap

1. Software development process
   - Different models (sequential, incremental, evolutionary, …)
2. Agile development methods
   - Agile principles and project management
3. DevOps Technical benefits
   - Continuous integration and software delivery

Team manager

Product owner

Developer

Architect

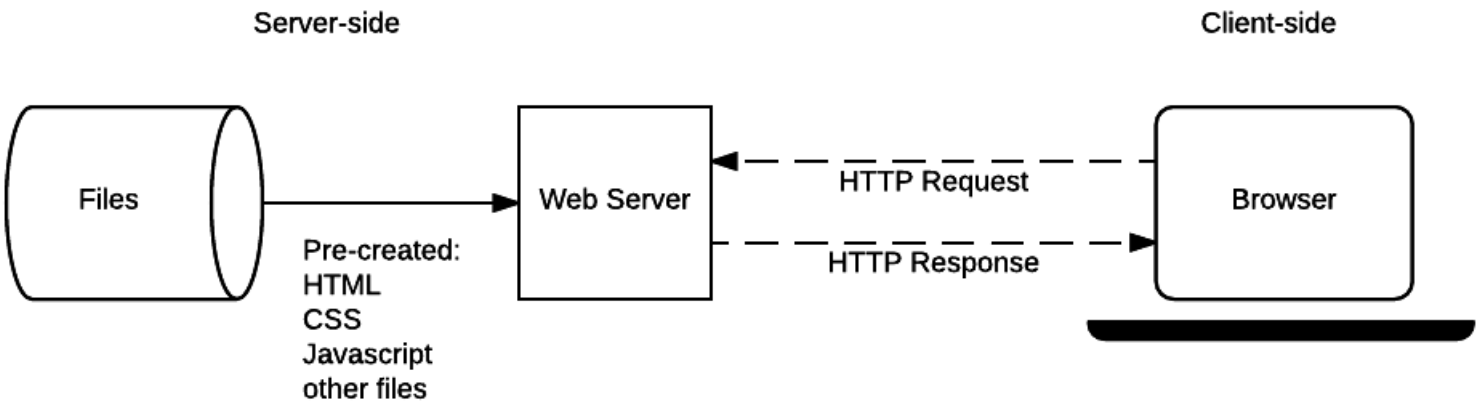
DevOps master

Main topics

- Web application
  - Static and dynamic
- Frontend development
  - HTML, CSS, JavaScript, JavaScript libraries
  - Frameworks
- Backend development
  - Frameworks
  - N-Tier
  - MVC
- Spring framework

Static sites

- Returns the same hard-coded content from the server whenever a particular resource is requested.

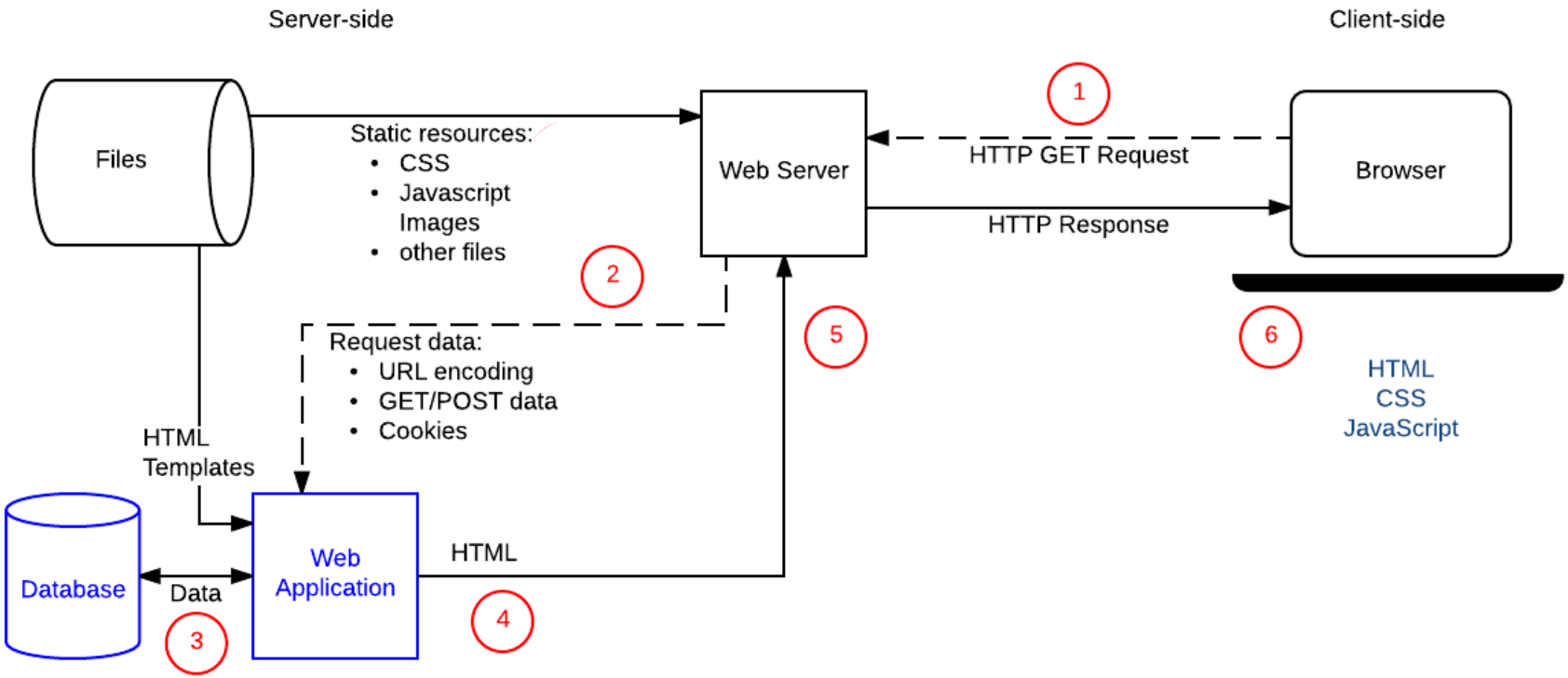


https://developer.mozilla.org/en-US/docs/Learn/Server-side

Dynamic sites

- The response content can be **generated dynamically**, only when needed
  - HTML pages are normally created by inserting data from a database into **placeholders in HTML templates**
- A dynamic site can return different data for a URL
  - based on information provided by the user or stored preferences and can perform other operations as part of returning a response (e.g., sending notifications).
- Most of the code to support a dynamic website runs on a web server
- Creating this code is known as **server-side** programming or **backend** programming

Dynamic sites



https://developer.mozilla.org/en-US/docs/Learn/Server-side

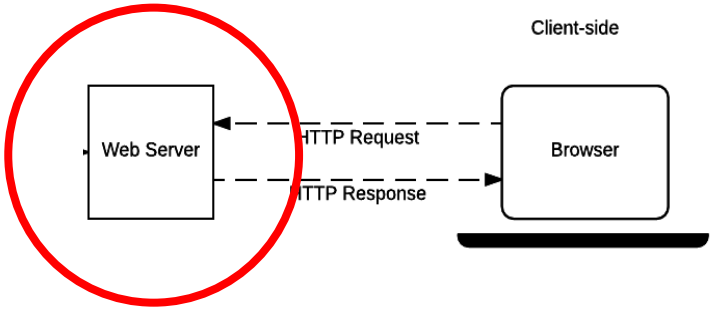
Why did we need Backend?

- But programs execute binary code
  - Well, also text (interpreted languages)

```
Hello world!
```

- A browser understands HTML, not binary code
  - Well, some other scripts (e.g., JavaScript)

```
<!DOCTYPE html>
<html><head>
<title>Page Title</title>
</head><body>
<p>Hello world!</p>
</body>
</html>
```



Common Tasks in Web Development

- Manage interface components
- Access control management
- Session and authentication management
- Handle access requests
- Validating inputs
- Error handling
- Access and manipulate data
- …

Web Development Frameworks

https://www.scnsoft.com/blog/web-application-framework

Frameworks - definitions

Definition 1 (structure)

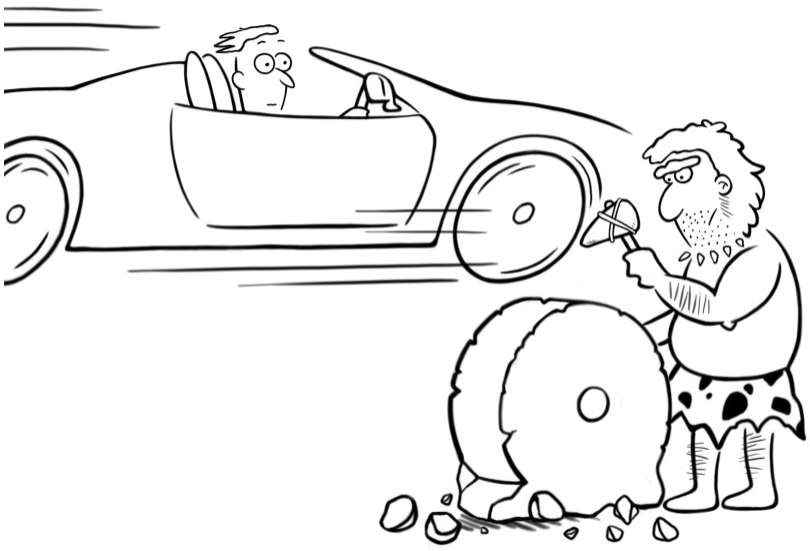
- A framework is a **reusable design** of all or part of a system that is represented by a set of abstract classes and the way their instances interact.

Definition 2 (purpose)

- A framework is the **skeleton** of an application that can be **customised** by an application developer.

Frameworks

- Don't reinvent
- And don't run from it



https://boarsheadeastcheap.com/2018/04/04/not-reinventing-the-wheel-after-all/

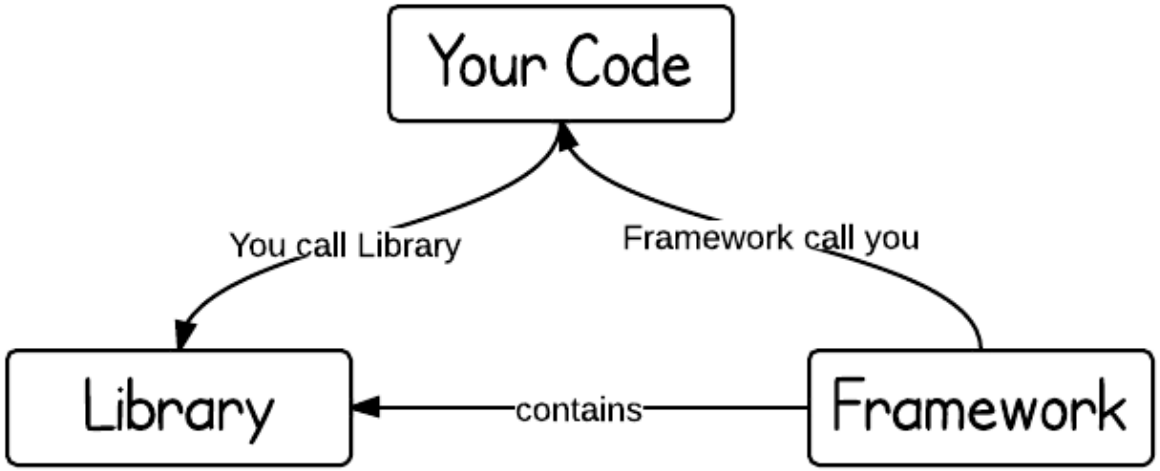
Software Frameworks

- A framework provide a generic software foundation over which **custom application-specific code** can be written
  - They often include multiple libraries, in addition to tools and rules on how to structure and use these components.

- Frameworks differ from other class libraries by reusing high-level design
  - **Libraries are used by the application-specific code** to support specific features.
  - **Frameworks control the application flow** and call application-specific code.

Frameworks and Libraries

- Libraries
  - *call the functions in the API to do things for you*
- Frameworks
  - *don't call us, we'll call you*
  - Inversion of Control containers



https://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/

Software Frameworks

Advantages

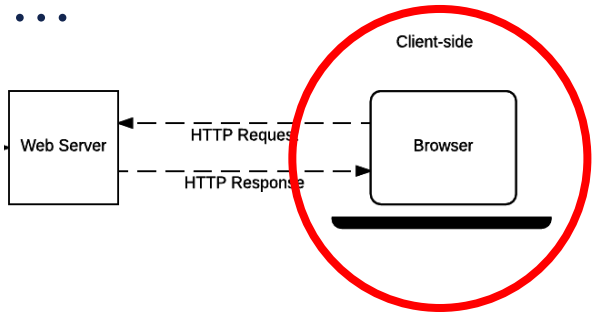
- Powerful and complex
- Implementation speed
- Tested, proven solutions
- Access to expertise and off-the-shelf solutions
- Maintenance (updates, ..)

Disadvantages

- Difficult to learn
- Lower performance
- Reduced independence
- Dependence on external entities
- Technological lock-in

Client-side frameworks (Frontend)

- React
- jQuery
- Express
- Vue
- Angular
- Node.js
- Ember.js
- …



Client-side APIs

- Manipulate **documents** loaded into the browser
  - DOM (Document Object Model) allows the manipulation of a document together with HTML and CSS.
- **Fetch data** to update small sections of a webpage
  - e.g., AJAX had a huge impact on the performance and behaviour of sites
- **Drawing** and manipulating graphics
  - e.g., Canvas and WebGL
- Client-side **storage**
  - e.g., IndexedDB API allows saving an app state between page loads, or when the device is offline.

Server-side frameworks

- Python
  - Django
- JavaScript
  - node.js
- Java
  - Spring
- Ruby
  - Ruby on Rails
- PHP
  - Symfony
- C#
  - ASP.NET

Server-side/Web Frameworks – Java

- Spring
- JSF (Java Server Faces)
- GWT (Google Web Toolkit)
- Play!
- Struts
- Vaadin
- Grails

https://www.dreamsoft4u.com/top-10-java-frameworks-you-should-know

Framework Components

- Core components
  - **Request Routing** — Match incoming HTTP requests to code
  - **Data Access** — Uniform data access, mapping and configuration
  - **Template Engine** — Structure and separate presentation from logic
- Common components
  - **Security** — Protection against common web security attacks
  - **Sessions** — Session management and configuration
  - **Error Handling** — Capture and manage application-level errors
  - **Scaffolding** — Quickly generate CRUD interfaces based on data model

Request Routing – SpringBoot example

```
@RestController
@SpringBootApplication
public class BookApplication {

  @RequestMapping(value = "/available")
  // ou  @GetMapping("/available")
  public String available() {
    return "Spring in Action";
  }

  @RequestMapping(value = "/checked-out")
  public String checkedOut() {
    return "Spring Boot in Action";
  }

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

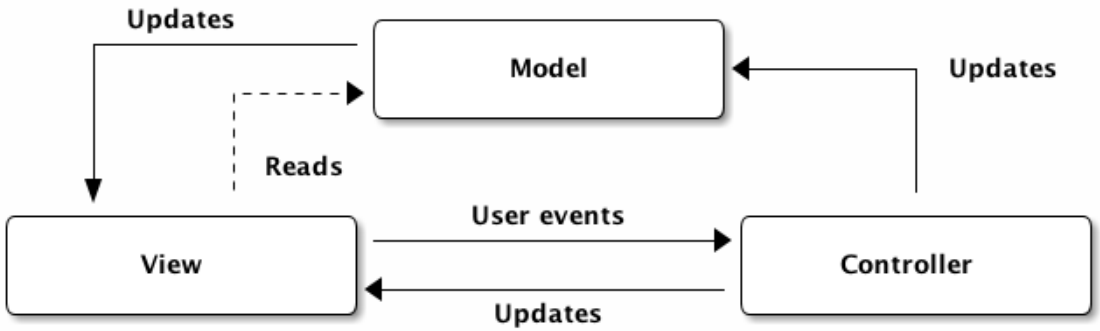
Data Access

- **Logical Independence**: The ability to change the logical schema without changing the external schema or application programs
  - Can add new fields, new tables without changing views
  - Can change structure of tables without changing view
- **Physical Independence**: The ability to change the physical schema without changing the logical schema
  - Storage space can change
  - Type of some data can change for reasons of optimization

- Solution: Keep the VIEW (what the user sees) independent of the MODEL (domain knowledge)

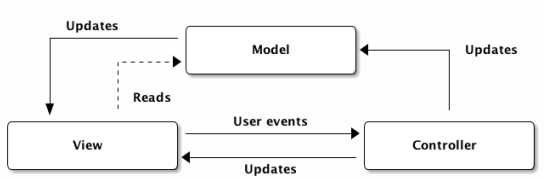
Model-View-Controller (MVC)

- Many Backend Frameworks follow, or are an evolution of, a design pattern called Model-View-Controller, or MVC.
  - Separates core functionality from the presentation and control logic that uses this functionality
  - Allow multiple views to share the same data model
  - Make supporting multiple clients easier to implement, test, and maintain



Model-View-Controller (MVC)

- **Model**: The program representation of the object/database
  - Users, Products, Cars, …



- **View**: The visual representation of data. This is what the web app user sees
  - HTML, CSS, and JavaScript
- **Controller**: The go-between Model and View
  - One way: takes input from the View and gives it to the Model to act upon.
  - Other way: Gathers data from Model(s) and gives it to the View to incorporate in the visual representation.
  - Examples: New User signup, listing all products that are less than some dollar amount, redirecting a user.

Template Engine

- Template engines take in tokenized strings and produce rendered strings with values in place of the tokens as output.
  - Templates are typically used as an intermediate format written by developers to programmatically produce one or more desired output formats, commonly HTML, XML or PDF.
- Examples
  - Java Server Pages
  - Thymeleaf
  - FreeMarker
  - Groovy
  - Jade4j
  - JMustache
  - Pebble
  - ...

JSF Templates

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<html>
    <head>
        <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1">
        <title>User Registration</title>
    </head>
    <body>
        <form:form method="POST" modelAttribute="user">
            <form:label path="email">Email: </form:label>
            <form:input path="email" type="text"/>
            <form:label path="password">Password: </form:label>
            <form:input path="password" type="password" />
            <input type="submit" value="Submit" />
        </form:form>
    </body>
</html>
```

Thymeleaf Templates

```
<html>
    <head>
        <meta charset="ISO-8859-1" />
        <title>User Registration</title>
    </head>
    <body>
        <form action="#" th:action="@{/register}"
          th:object="${user}" method="post">
            Email:<input type="text" th:field="*{email}" />
            Password:<input type="password" th:field="*{password}" />
            <input type="submit" value="Submit" />
        </form>
    </body>
</html>
```

The Spring Framework

- Architecture
- Annotations
- Inversion of Control (IoC)
- Dependency Injection (DI)
- Beans
- Aspect-Oriented Programming (AOP)

Java frameworks

- Spring was not the first framework trying to solve the problems around building enterprise Java applications

- **J2EE or Java Enterprise Edition** which preceded Spring tried to do the same thing
  - On paper, J2EE was meant to become the standard way of building Java enterprise applications.
  - The major guiding principle behind J2EE was to provide a clean separation between various parts of an application.

- **Spring Framework** is a Java application framework that started in the early 2000s.
  - Since then, it has grown from an (already sophisticated) dependency injection container into an **ecosystem of frameworks that cover a broad set of use cases** in application development

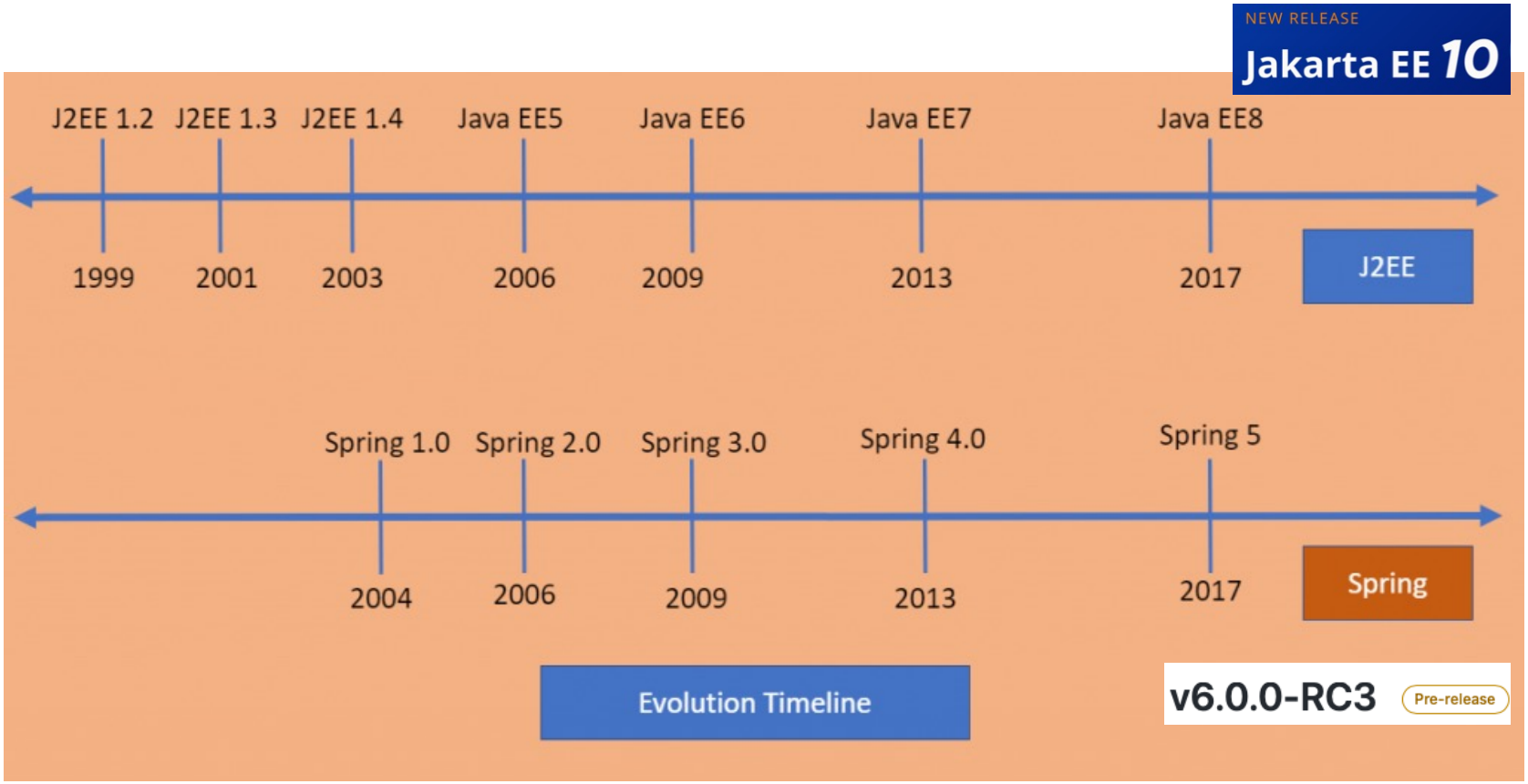
J2EE, Java EE, Jacarta EE – history…

- In the first version of Java, Java enterprise extensions were simply a part of the **core JDK**.
- 1999 – as part of Java 2, these extensions were broken out of the standard binaries
  - **J2EE**, or Java 2 Platform Enterprise Edition, was born. It would keep that name until 2006.
- 2006 – with Java 5, it was renamed to **Java EE** or Java Platform Enterprise Edition.
- 2017 – Oracle give away the rights to the Eclipse Foundation
  - But.. the language was/is still owned by Oracle.
  - Eclipse Foundation legally had to choose a new name: **Jakarta EE** (2018).

https://www.baeldung.com/java-enterprise-evolution

Jakarta EE and Spring

- **Jakarta Enterprise Edition (JEE)** – since 2018
  - formerly **Java Enterprise Edition (JEE)**



Jakarta EE and Spring

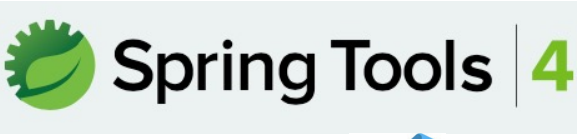
- The Spring programming model does not embrace the Java EE platform specification
- But, it integrates selected specifications from EE:
  - Servlet API (JSR 340)
  - WebSocket API (JSR 356)
  - Concurrency Utilities (JSR 236)
  - JSON Binding API (JSR 367)
  - Bean Validation (JSR 303)
  - JPA (JSR 338)
  - JMS (JSR 914)
  - Spring also supports the Dependency Injection (JSR 330) and Common Annotations (JSR 250) specifications
    - But it provides specific mechanisms for this.

JSR: Java Specification Request.

The JSR was a bit like the *interface* for an EE feature.

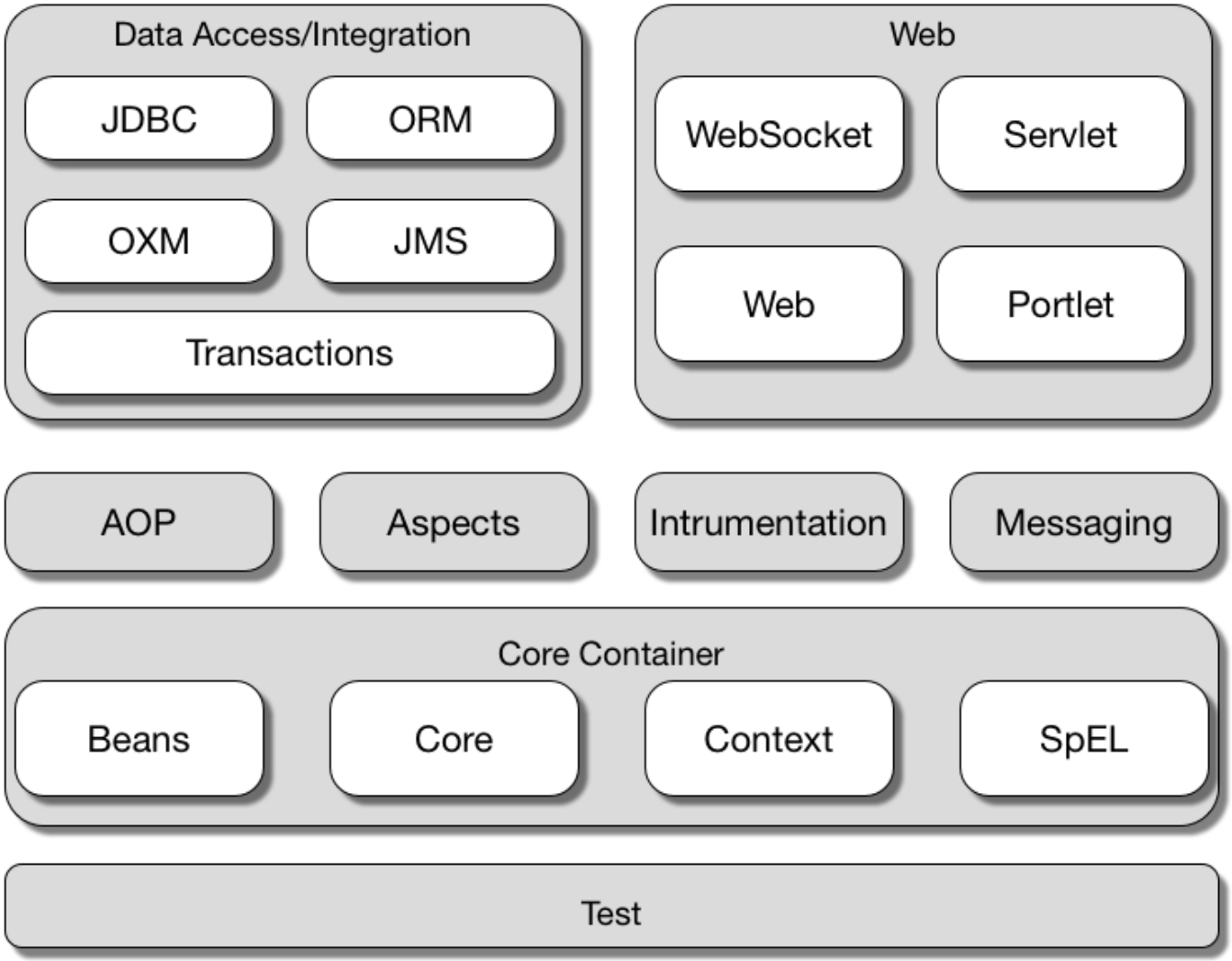
Spring Framework

- Development tools

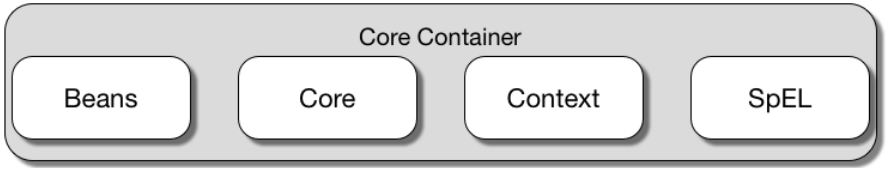


- Beyond the Spring Framework
  - there are other projects such as Spring Boot, Spring Security, Spring Data, Spring Cloud, Spring Batch, among others.
    - *See spring.io/projects for the complete list of Spring projects.*
  - Each project has its own source code repository, issue tracker, and release cadence

Spring Framework Architecture



Core Container modules



- **Core (spring-core)**
  - The core of the framework provides Inversion of Control and dependency injection with singleton design pattern.
- **Beans (spring-beans)**
  - This module provides implementations for the factory design pattern through *BeanFactory* (and others).
- **Context (spring-context)**
  - Builds on Core and Beans and provides a medium to access defined objects and provides support for third-party interactions such as caching, mailing, and template engines.
  - The *ApplicationContext* interface is the core part of the Context module.
- **Spring Expression Languages – SpEL (spring-expression)**
  - Enables users to use the Spring Expression Language to query and manipulate the object graph at runtime.

Recap... Java Annotations

- Annotations provide metadata about a program or about its components.

  `@Override, @SuppressWarning, @Deprecated, ..`

- They can be applied on declarations of:
  - classes, fields, methods, and other program elements.
- Common usage:
  - Information for the compiler - to detect errors or suppress warnings.
  - Compile-time and deployment-time processing - to generate code, javadoc, XML files, configs.
  - Runtime processing – some annotations are available to be examined at runtime
- **Annotations are largely used in Spring Framework**

Inversion of Control (IoC)

- IoC is a process in which, we defined a class
  - e.g., class Person

```
public class Person {
   private String id;
   private String name;
   // ...
}
```

- But … **we do not create the objects!**
  - We just define how they should be created by the IoC container.

Inversion of Control (IoC) – example

Old way

Person.java

```
public class Person {
   private String id;
   private String name;
   // ...
}
```

beans.xml

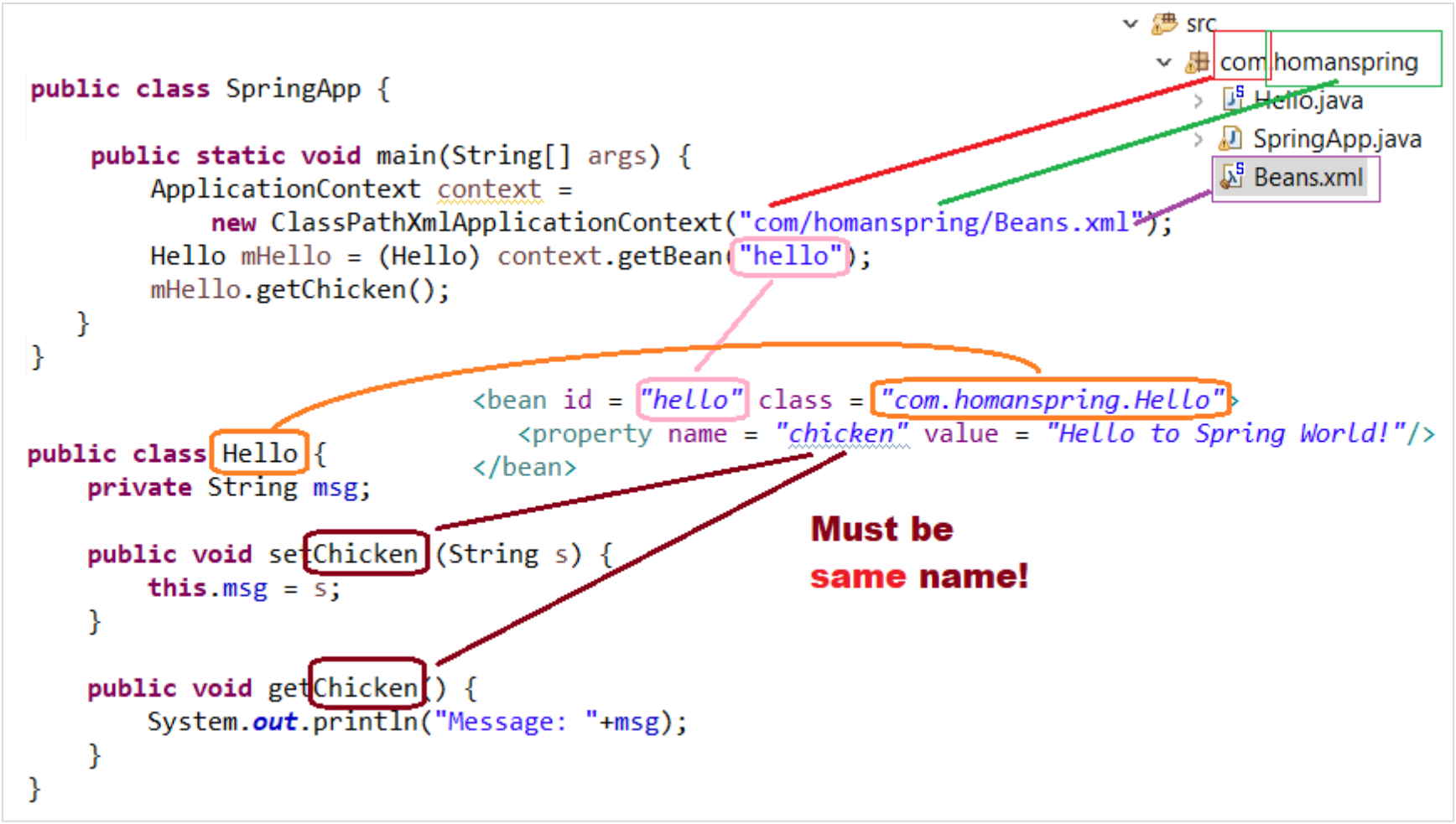
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
<!--  other stuffs ... -->
<beans ...>
   <bean id="person1" class="ua.ies.Person">
     <property name="id" value="12345"/>
     <property name="name" value="Leonor"/>
   </bean>
</beans>
```

Main.java

```
public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
    Person p1 = context.getBean("person1", Person.class);
    System.out.println(p1.getId() + ", " + p1.getName()));
}
```

IoC through XML definition is tricky!

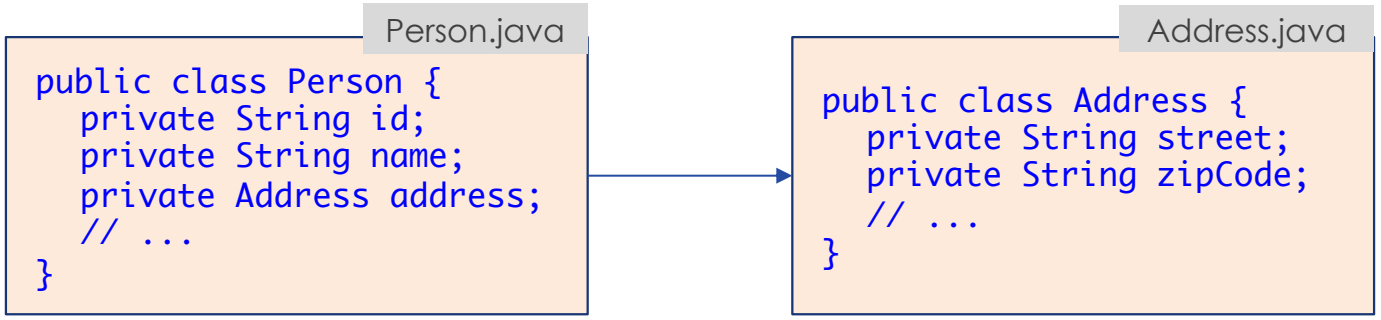
Old way



https://homanhuang.medium.com/java-spring-on-windows-10-part-1-spring-framework-b017ee39aed5

Inversion of Control (IoC) – example

- How, if we have **dependencies?**



- ... the IoC container needs to create address before person

Dependency Injection

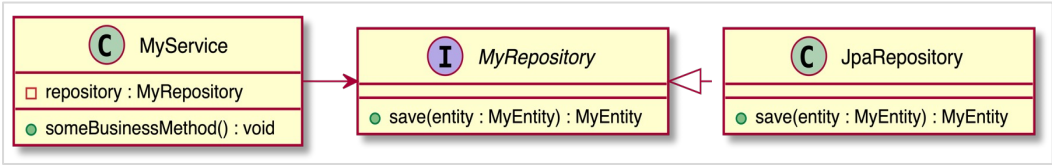
- A design pattern that **removes the dependency from the code**
  - We may specify classes' dependencies through **annotations** or from external source, such as **XML** file

```
<constructor-arg value="101" type="int"></constructor-arg>
<property name="id" value="101"></property>
```

- Dependencies can be injected in two ways
  - By **constructor**
  - By **setter** method

Example (POJO)

(Plain Old Java Object)



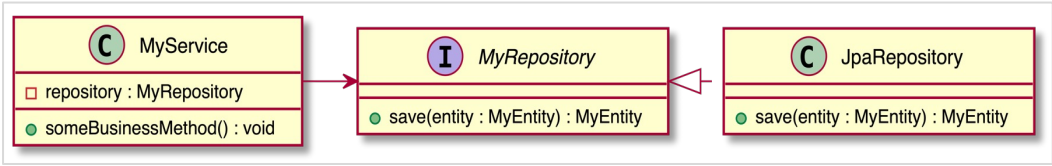

```
interface MyRepository {
    MyEntity save(MyEntity entity);
}

class JpaRepository implements MyRepository {
   MyEntity save(MyEntity entity) { ... }
}

class MyService {
    private final MyRepository repository;
    MyService(MyRepository repository){
      this.repository = repository;
    }
    void setRespository(MyRepository newRepository) {
      this.repository = newRepository;
   }
}

class Main(String ...) {
   MyRepository repository = new JpaRepository(...);
   MyService service = new MyService(repository);
}
```

Example (POJO)




```
interface MyRepository {
    MyEntity save(MyEntity entity);
}

class JpaRepository implements MyRepository {
    MyEntity save(MyEntity entity) { ... }
}

class MyService {
    private final MyRepository repository;
    MyService(MyRepository repository){
      this.repository = repository;
    }
    void setRespository(MyRepository newRepository) {
      this.repository = newRepository;
    }
}
// Manual dependency injection
class Main(String ...) {
    MyRepository repository = new JpaRepository(...);
    MyService service = new MyService(repository);
}
```

MyService expresses a dependency to MyRepository instead of creating it itself:

- by constructor

- by setter method

The manual setup code creates as instance of MyRepository to the service instance to be created.

Example (with Spring DI)

```
@Component
class MyService { ... }

@Component
class JpaRepository { ... }

...
// Spring-driven dependency injection
    ApplicationContext context =
        new AnnotationConfigApplicationContext(Config.class);
    MyService service = context.getBean(MyService.class);
...
```

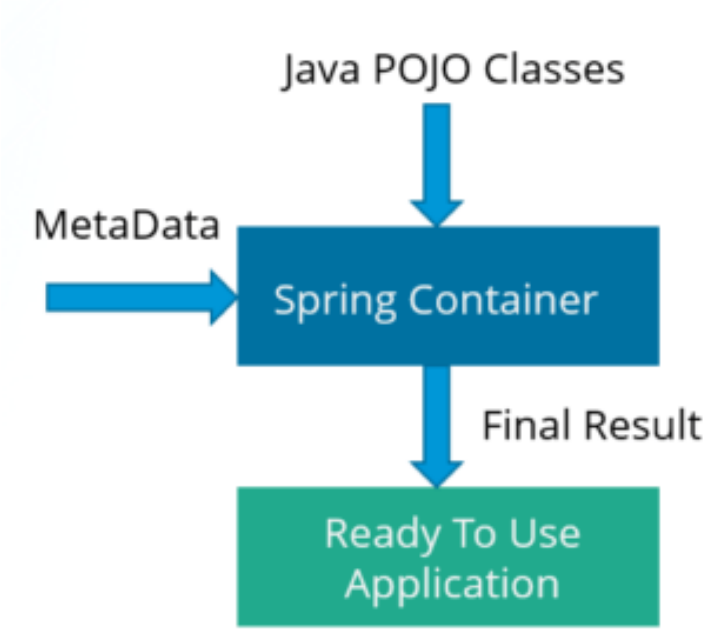
Classes are annotated with framework specific annotations so that it can discover the components of an application that it's supposed to handle.

The detection is triggered by the framework, pointing it to the code written by the user.

The framework API is then used to access the components.

Spring IoC Container

- Spring IoC is the heart of the Spring Framework
- Each object delegates the **job of constructing** to an IoC container.
- The IoC container receives metadata from either
  - an XML file,
  - Java annotations, or
  - Java code
- and produces a fully configured and executable system or application



Spring IoC Container

- The **main tasks** performed by the IoC container are:
  - Instantiating the bean
  - Wiring the beans together
  - Configuring the beans
  - Managing the bean's entire life-cycle

- Two types of Spring IoC containers (interfaces):
  - the **BeanFactory** is the simplest container
    - providing the configuration framework and basic functionality
  - the **ApplicationContext** built on top of the BeanFactory interface
    - It adds more enterprise-specific functionality

ApplicationContext

- org.springframework.context.**ApplicationContext**.
  - It follows eager-initialization technique
    - instance of beans are created with the ApplicationContext.
  - Spring framework provides several implementations

All Known Implementing Classes:

AbstractApplicationContext, AbstractRefreshableApplicationContext, AbstractRefreshableConfigApplicationContext, AbstractRefreshableWebApplicationContext, AbstractXmlApplicationContext, AnnotationConfigApplicationContext, AnnotationConfigWebApplicationContext, ClassPathXmlApplicationContext, FileSystemXmlApplicationContext, GenericApplicationContext, GenericGroovyApplicationContext, GenericWebApplicationContext, GenericXmlApplicationContext, GroovyWebApplicationContext, ResourceAdapterApplicationContext, StaticApplicationContext, StaticWebApplicationContext, XmlWebApplicationContext

```
ApplicationContext context =
   new ClassPathXmlApplicationContext("applicationContext.xml");

ApplicationContext context =
   new AnnotationConfigApplicationContext(Config.class);
```

What is a bean?

- Spring documentation definition
  - "The objects that form the backbone of an application and that are managed by the Spring IoC container are called beans. A bean is instantiated, assembled, and managed by the IoC container."

- Some key characteristics:
  - All fields should be **private**
  - all fields must have **setters** and **getters**
  - there should be a **no-arg constructor**
  - fields are accessed exclusively by the **constructor** or the **getter/setter methods**

https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-introduction

What is a bean?

- Spring is responsible for creating bean objects. But first, we need to tell the framework which objects it should create
- **Bean definitions** tell Spring which classes the framework should use as beans
- Bean definitions are like recipes
  - They also describe the properties of a bean

Defining a Spring bean

- There are **three different ways** to define a Spring bean:
  - declaring a bean definition in an **XML configuration** file
  - annotating a class with the ***@Component*** annotation (or its derivatives, *@Service, @Repository, @Controller*)
  - writing a bean factory method annotated with the ***@Bean*** annotation in a custom Java configuration class
- In modern projects, we use only the last two, component annotations and bean factory methods
  - Spring still allows the XML configuration for the backward compatibility
- The **choice between bean definition methods** is mainly dictated by access to the source code of a class that we want to use as a Spring bean

Spring bean as @Component

- If **we own the source code**, we'll usually use the @Component annotation directly on a class.

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

- At runtime, Spring finds all classes annotated with @Component or its derivatives and uses them as bean definitions

- The process of finding annotated classes is called **component scanning**

Using @Bean for factory methods

- For **classes we don't own**, we must create factory methods with the @Bean annotation in a bean configuration class
  - If we don't want to make a class dependent on Spring, we can also use this option for classes we own

```
@Configuration
class MyConfigurationClass {

   @Bean
   public static NotMyClass notMyClass() {
       return new NotMyClass();
   }
}
```

- The @Configuration annotation also comes from Spring.
  - The annotation marks the class as a container of @Bean definitions.
  - We may write multiple factory methods inside a single configuration class.

Component scanning – Example

```
@Configuration
@ComponentScan
public class SpringComponentScanApp {
    private static ApplicationContext applicationContext;

    @Bean
    public ExampleBean exampleBean() {  return new ExampleBean();    }

    public static void main(String[] args) {
        context =
            new AnnotationConfigApplicationContext(SpringComponentScanApp.class);

        for (String beanName : context.getBeanDefinitionNames()) {
            System.out.println(beanName);
        }
    }
}
```

```
springComponentScanApp
cat
dog
exampleBean
```

```
@Component
public class Cat {}
```

```
@Component
public class Dog {}
```

Spring Expression Language (SpEL)

- SpEL can be used for manipulating and querying an object graph at runtime.
  - SpEL is available via XML or annotations and is evaluated during the bean creation time.
- There are several operators available in the language:
  - Arithmetic +, -, *, /, %, ^, div, mod
  - Relational <, >, ==, !=, <=, >=, lt, gt, eq, ne, le, ge
  - Logical and, or, not, &&, | |, !
  - Conditional ?:
  - Regex matches

SpEL examples

- 1: Create *employee.properties* file in the resources' directory.

```
employee.names=Petey Cruiser,Anna Sthesia,Paul Molive,Buck Kinnear
employee.type=contract,fulltime,external
employee.age={one:'26', two : '34', three : '32', four: '25'}
```

- 2: Create a class EmployeeConfig as follows:

```
@Configuration
@PropertySource (name = "employeeProperties",
                 value = "employee.properties")
@ConfigurationProperties
public class EmployeeConfig {
}
```

- SpEL: The @Value annotation can be used for injecting values into fields in Spring-managed beans,
  - it can be applied at the field, constructor, or method parameter level.

```
@Value ("#{'${employee.names}'.split(',')}")
private List<String> employeeNames;
```

```
[Petey Cruiser, Anna Sthesia, Paul Molive, Buck Kinnear]
```

SpEL examples

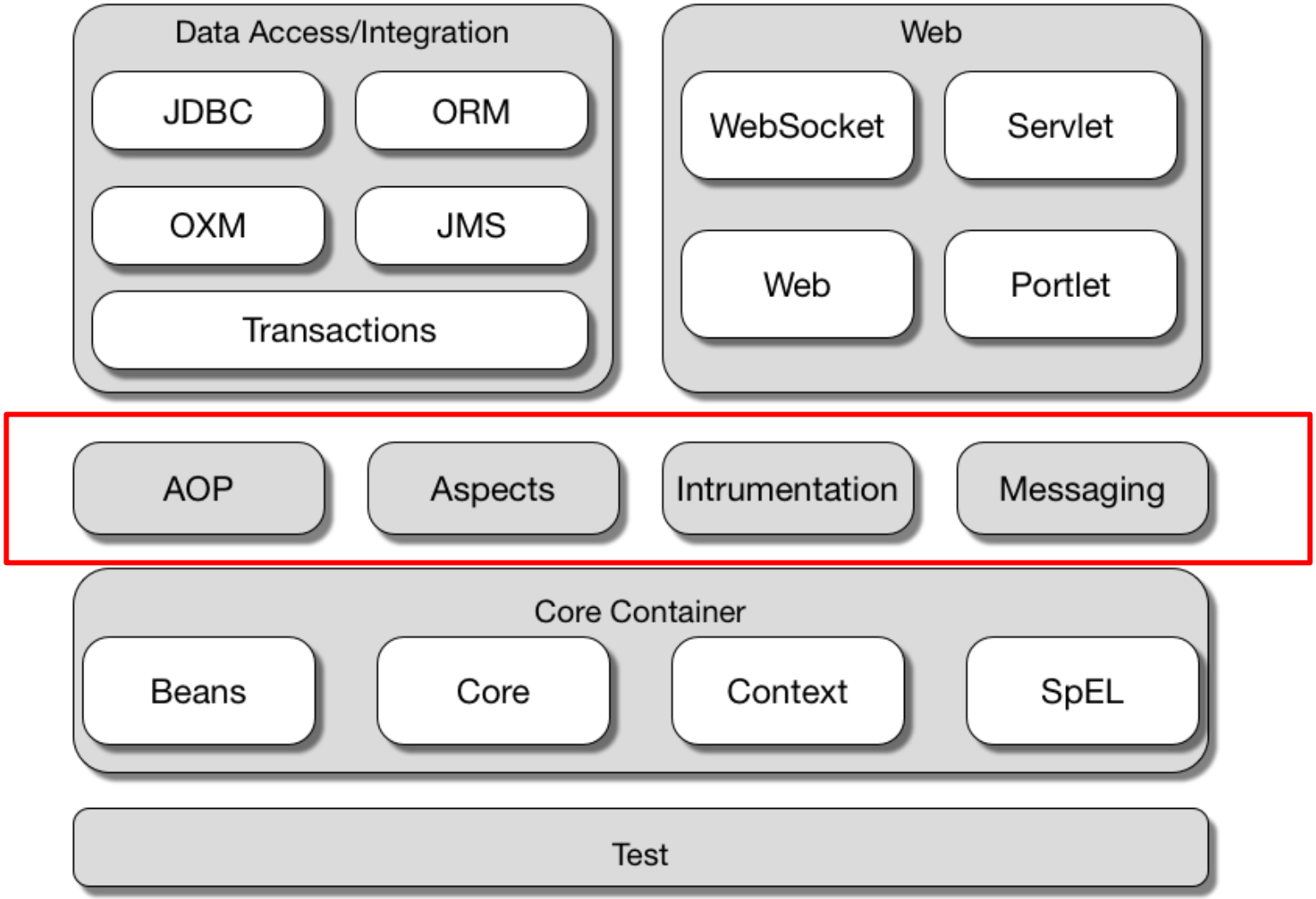
```
@Value ("#{'${employee.names}'.split(',')[0]}")
private String firstEmployeeName;

@Value ("#{systemProperties['java.home']}")
private String javaHome;

@Value ("#{systemProperties['user.dir']}")
private String userDir;

@Value("#{someBean.someProperty != null ?
someBean.someProperty : 'default'}")
private String ternary;
```

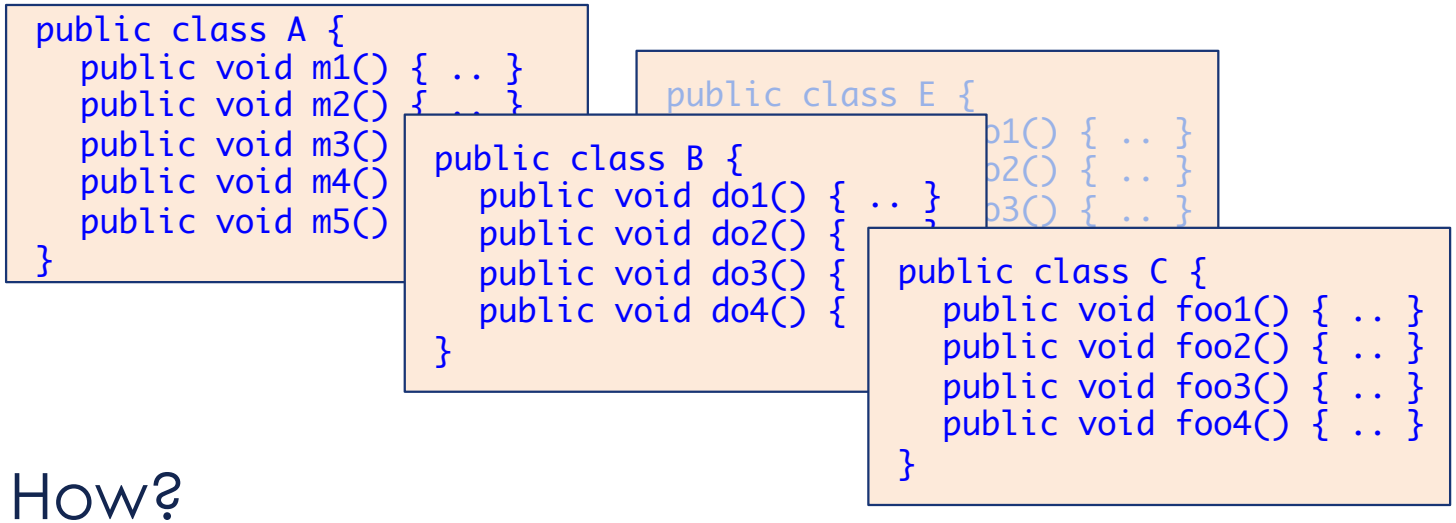
Spring Framework Architecture



Aspect Oriented Programming (AOP)

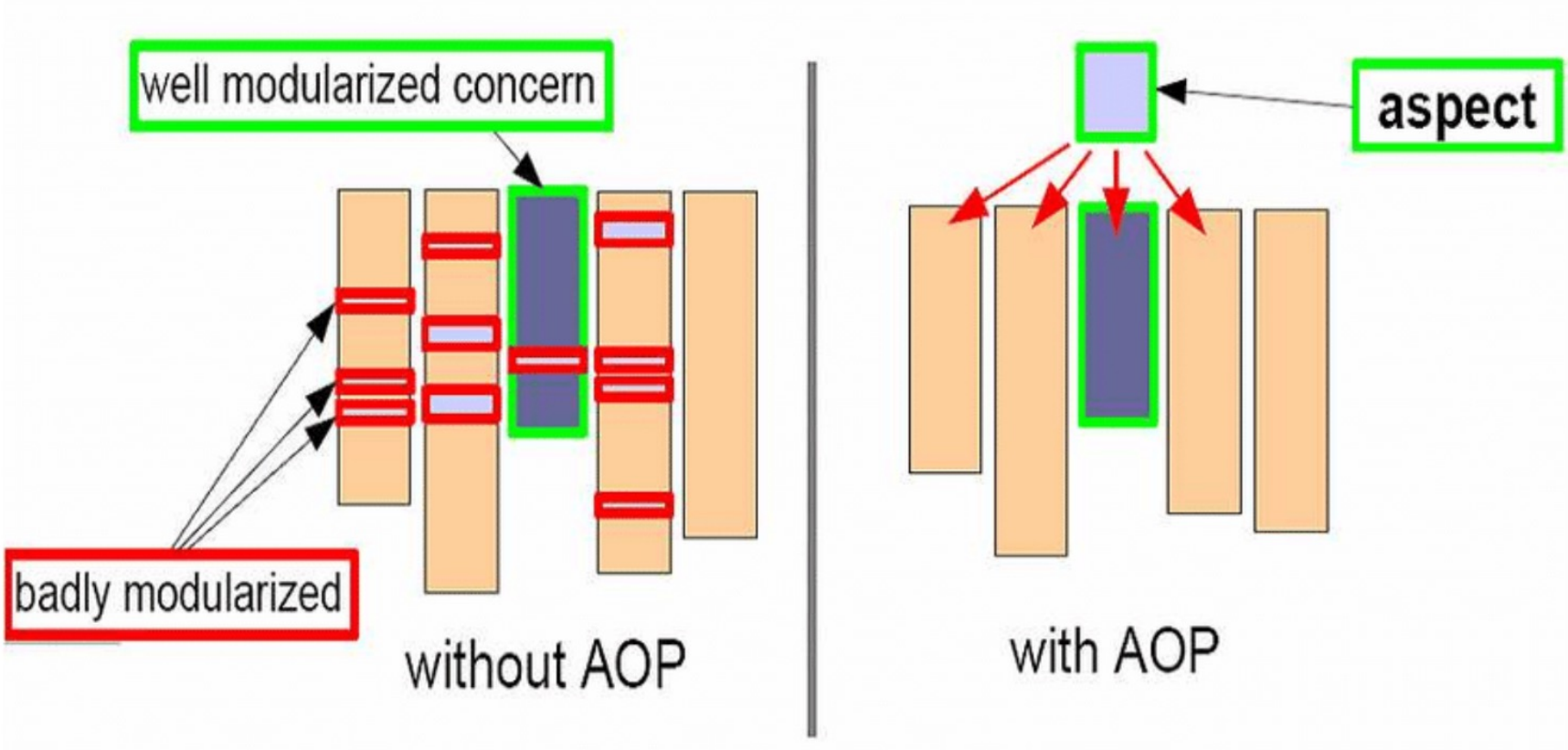
- Why AOP? An example:

- We want to keep a log, or send a notification, after(or before) calling a class method (or several classes methods)



- How?

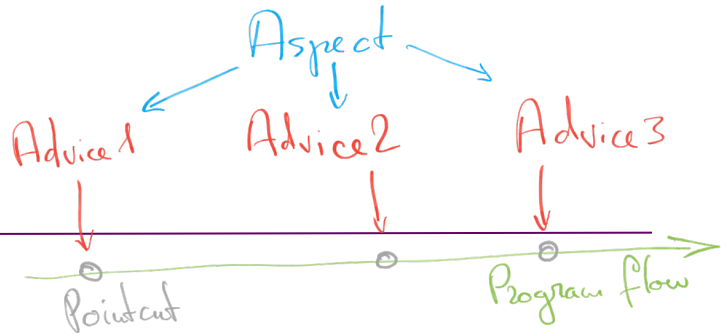
Aspect Oriented Programming (AOP)



Spring AOP

- AOP is a programming approach that allows global properties of a program to **determine how it is compiled** into an executable program
  - AOP compliments OOPs in the sense that it also provides modularity. But here, the key unit of modularity is an aspect rather than a class.
- AOP **breaks down the logic of program** into distinct parts called *concerns*. This increases modularity by **cross-cutting concerns**.
  - A **cross-cutting concern** is a concern that affects the whole application and is centralized in one location in code like transaction management, authentication, logging, security etc.
- AOP can be considered as a **dynamic decorator** design pattern.
  - The decorator pattern allows additional behavior to be added to an existing class by wrapping the original class and duplicating its interface and then delegating to the original.

Core AOP Concepts

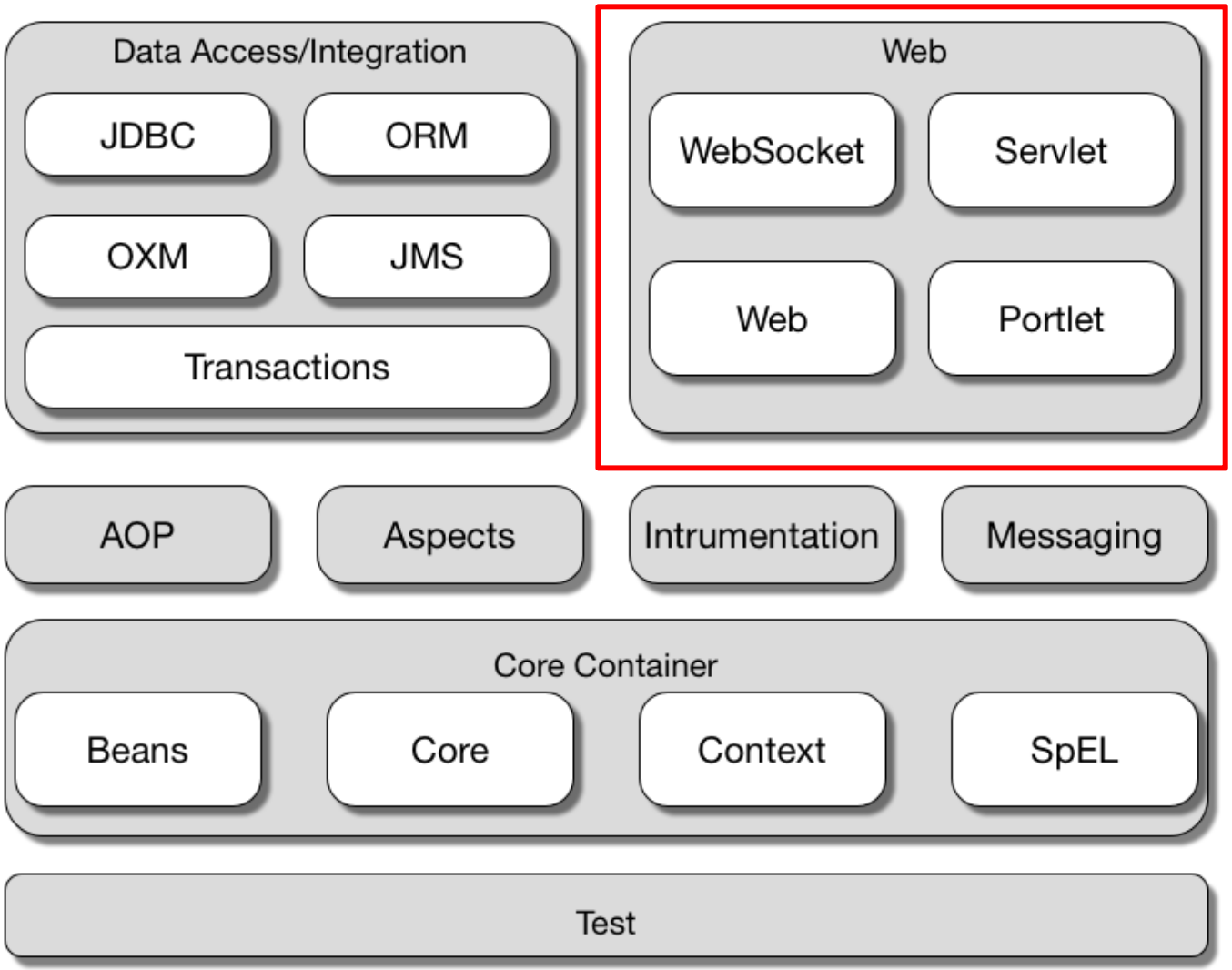


- **Aspect:** a modularization of a concern that cuts across multiple classes (e.g., transaction, logger).
  - It can be a normal **class**, configured through XML configuration or through the annotation **@Aspect**.
- **JoinPoint**: is a point during the execution of a program, where flow of execution got changed like exception catching or calling other method.
- **Pointcut:** are basically Joinpoints with an advice (or call aspect)
  - expressions that are matched at join points to determine whether advice needs to be executed or not.
- **Advice:** action taken by an aspect at a particular join point.
  - @Before, @After, @Around, @AfterReturning, @AfterThrowing.

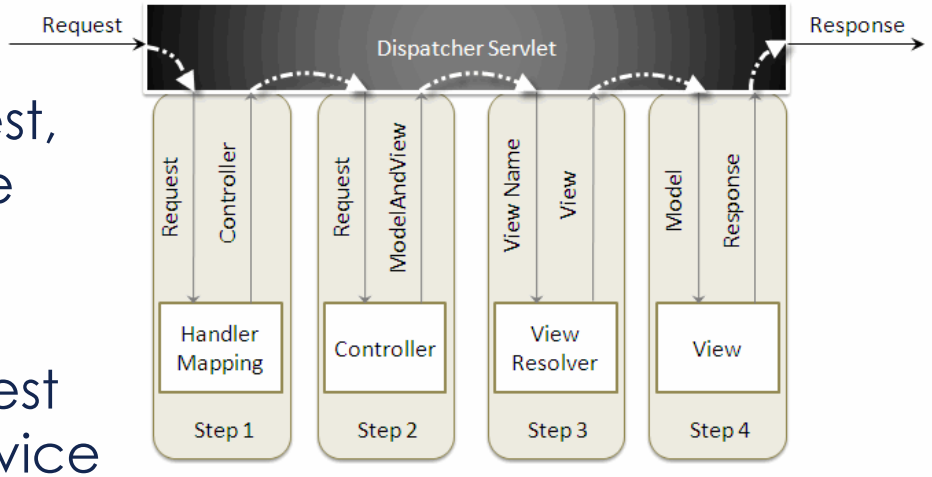
AOP Examples

```
package foo
import org.aspectj.lang.*;
import org.springframework.util.StopWatch;
import org.springframework.core.annotation.Order;
@Aspect
public class ProfilingAspect {
    @Around("methodsToBeProfiled()")   // action taken at the join point.
    public Object profile(ProceedingJoinPoint pjp) throws Throwable {
        StopWatch sw = new StopWatch(getClass().getSimpleName());
         try {
             sw.start(pjp.getSignature().getName());
             return pjp.proceed();
         } finally {
             sw.stop();
             System.out.println(sw.prettyPrint());
         }
     }
     @Pointcut("execution(public * foo..*.*(..))")
     public void methodsToBeProfiled(){}
}
```

Spring Web


Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Intrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Spring MVC (Model-View-Controller)



- After receiving an HTTP request, DispatcherServlet consults the **HandlerMapping** to call the appropriate Controller.
- The **Controller** takes the request and calls the appropriate service methods based on used GET or POST method.
  - The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
- The DispatcherServlet will take help from **ViewResolver** to pickup the defined view for the request.
- Once view is finalized, The DispatcherServlet passes the **model data to the view** which is finally rendered on the browser.

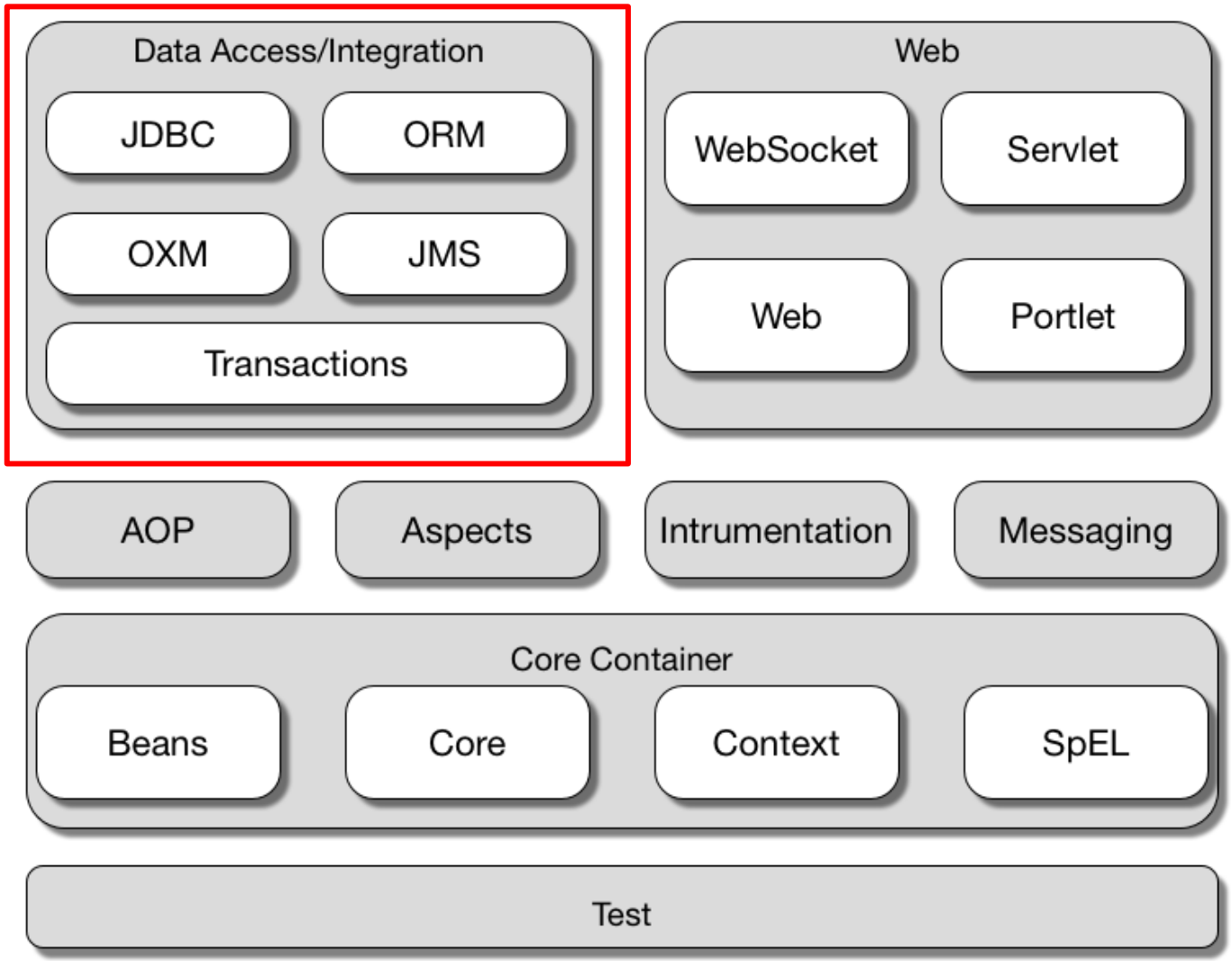
Controller example

```
@Controller
@RequestMapping("/funcionarios")  // a kind of path..
public class FuncionariosController {
    @Autowired
    private FuncionarioService funcionarioService;
    @Autowired
    private Funcionarios funcionarios;

    @ResponseBody
    @GetMapping("/todos")  // curl -i http://localhost:8080/iesapp/funcionarios/todos
    public List<Funcionario> todos() {
         return funcionarios.findAll();
    }
    @GetMapping("/lista")
    public ModelAndView listar() {
        ModelAndView modelAndView = new ModelAndView("funcionario-lista.jsp");
        modelAndView.addObject("funcionarios", funcionarios.findAll());
        return modelAndView;
    }

// ...
}
```

Spring Framework Architecture



Spring Data Access modules

- **JDBC:** This module provides JDBC abstraction layer which eliminates the need of repetitive and unnecessary exception handling overhead.
- **ORM:** ORM stands for **O**bject **R**elational **M**apping. This module provides consistency/ portability to our code regardless of data access technologies based on object oriented mapping concept.
- **OXM:** OXM stands for **O**bject **X**ML **M**appers. It is used to convert the objects into XML format and vice versa. The Spring OXM provides an uniform API to access any of these OXM frameworks.
- **JMS:** JMS stands for **J**ava **M**essaging **S**ervice. This module contains features for producing and consuming messages among various clients.
- **Transaction:** This module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs. All the enterprise level transaction implementation concepts can be implemented in Spring by using this module.

References

- Introduction to web frameworks
  - https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction
  - https://www.pixelcrayons.com/blog/web/best-web-development-frameworks-comparison/
  - https://en.wikipedia.org/wiki/Comparison_of_web_frameworks
  - https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks
- Spring framework
  - https://docs.spring.io/spring/docs/current/spring-framework-reference/
  - https://www.baeldung.com/spring-tutorial
  - https://www.edureka.co/blog/spring-tutorial