Spring Basics

Bernardo Cuteri

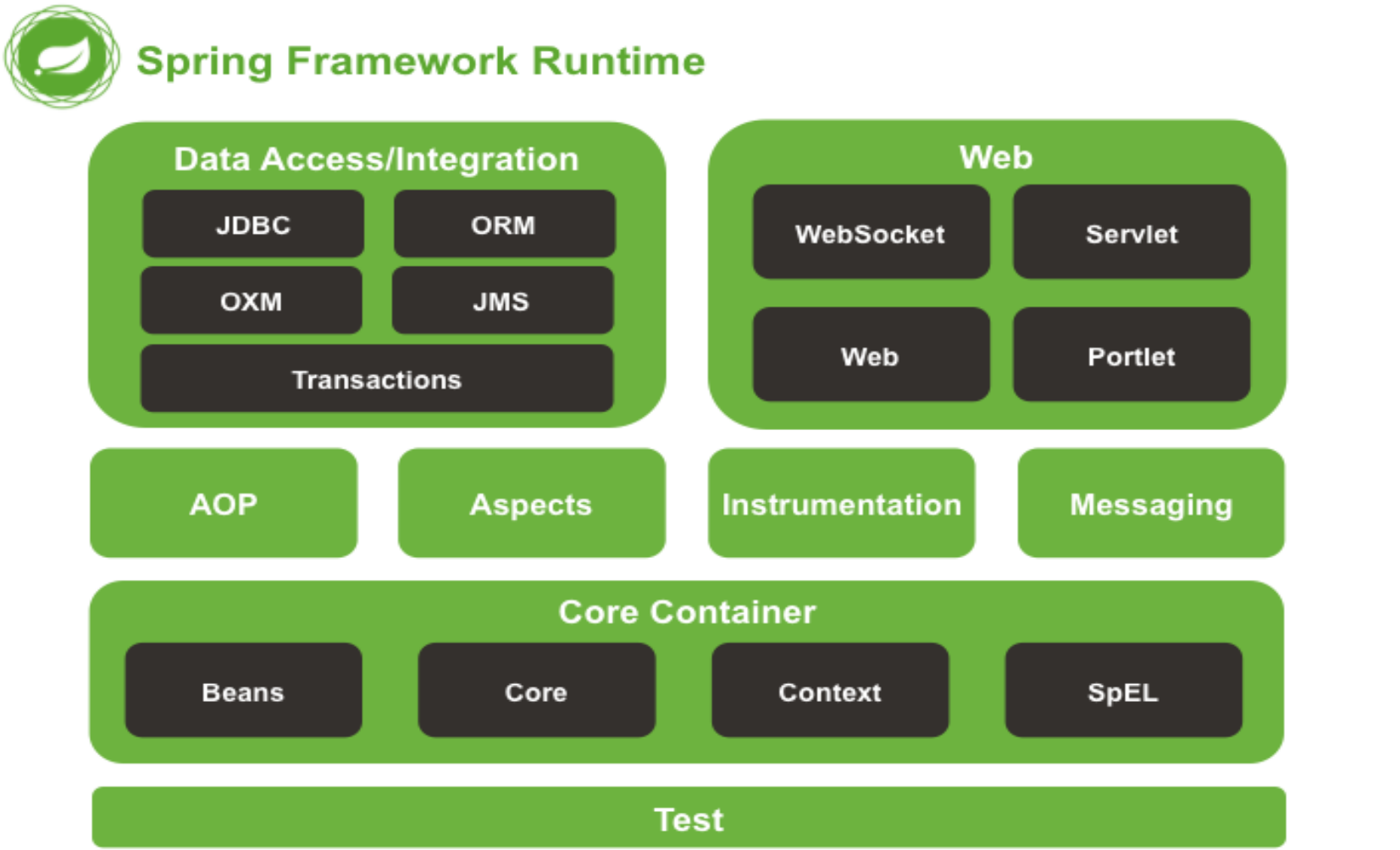
Spring

- Open source framework
- Created to address the complexity of enterprise application development
- Any Java application can benefit from Spring in terms of simplicity and testability

A spring project for bootstrapping spring applications:

- the user select the core modules needed
- Spring boot creates a project compliant with user's specifications

- Simplifies enterprise-level project creation (bootstrapping)
- Generates a configured Maven or Gradle project from user specifications
- Integrates additional dependencies depending on app configuration
- Select proper dependencies versions, makes sure they work together
- Supports multiple development profiles
- Multiple supported interface types:
  - Spring CLI: command line interface
  - Spring Initializr: official web initializer `https://start.spring.io/`
  - IDE spring boot interfaces


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

- The Spring Framework is made up of several modules
- Do I need to use all of the modules? No, it is possible to use a fragment of Spring Framework
- Spring offers integration points with several other frameworks and libraries

Important Features

- Is Lightweight, in terms of both size (few MB) and overhead (negligible)
- Brings the principle of Inversion of Control (IoC): custom-written portions of a computer program receive the flow of control from a generic framework (as opposed to traditional programming with reusable libraries)
- Supports of Dependency Injection (DI), objects are passively given their dependencies instead of creating or looking for dependent objects for themselves
- Is a Container, in the sense that it contains and manages the lifecycle and configuration of application objects
- Is a Framework, since it allows to configure and compose complex applications from simpler components

Dependency Injection (1)

- All real-world applications are made up of two or more classes that collaborate with each other
- In general, each object is responsible for obtaining its own references to the objects it collaborates with
- Using Dependency Injection objects are given their dependencies at creation time by some external entity that coordinates each object in the system
- Objects are not responsible for finding or creating the other objects that they need
- Instead, they are given references to the objects that they collaborate with by the container
- The act of creating these associations between application objects is referred to as wiring

- Automatic beans wiring in Spring is obtained by using the @Autowired annotation
- Wiring resolution depends on types and names

Spring provides a set of extendable stereotypes for identifying certain special beans (at class-level) of the application. Such beans are typically referred to as components

- @**Component**: identifies a component that is a candidate for auto-detection
- @**Service**: Specialization of @*Component*, denotes classes intended to act as bussiness service facades.
- @**Repository**: Specialization of @*Component*, denotes classes that handle data access and manipulation (e.g. DAO classes)
- @**Controller**: Specialization of @*Component*, denotes classes that implement the controller layer of the app (e.g. web controllers)

Stereotype annotated classes are:

- subject to auto-detection
- collected in the application context (spring container) and can be wired into other classes
- singleton by default (i.e. only one instance will be constructed in the application context, so all classes that wire it will use the same instance).

@Bean annotation

@Bean annotation:

- used to specify beans that are collected in the application context (and can be subject to autowiring)
- method-level annotation
- provides more fine-grained control over @Component, but is less declarative
- necessary for beans integrated from external libraries

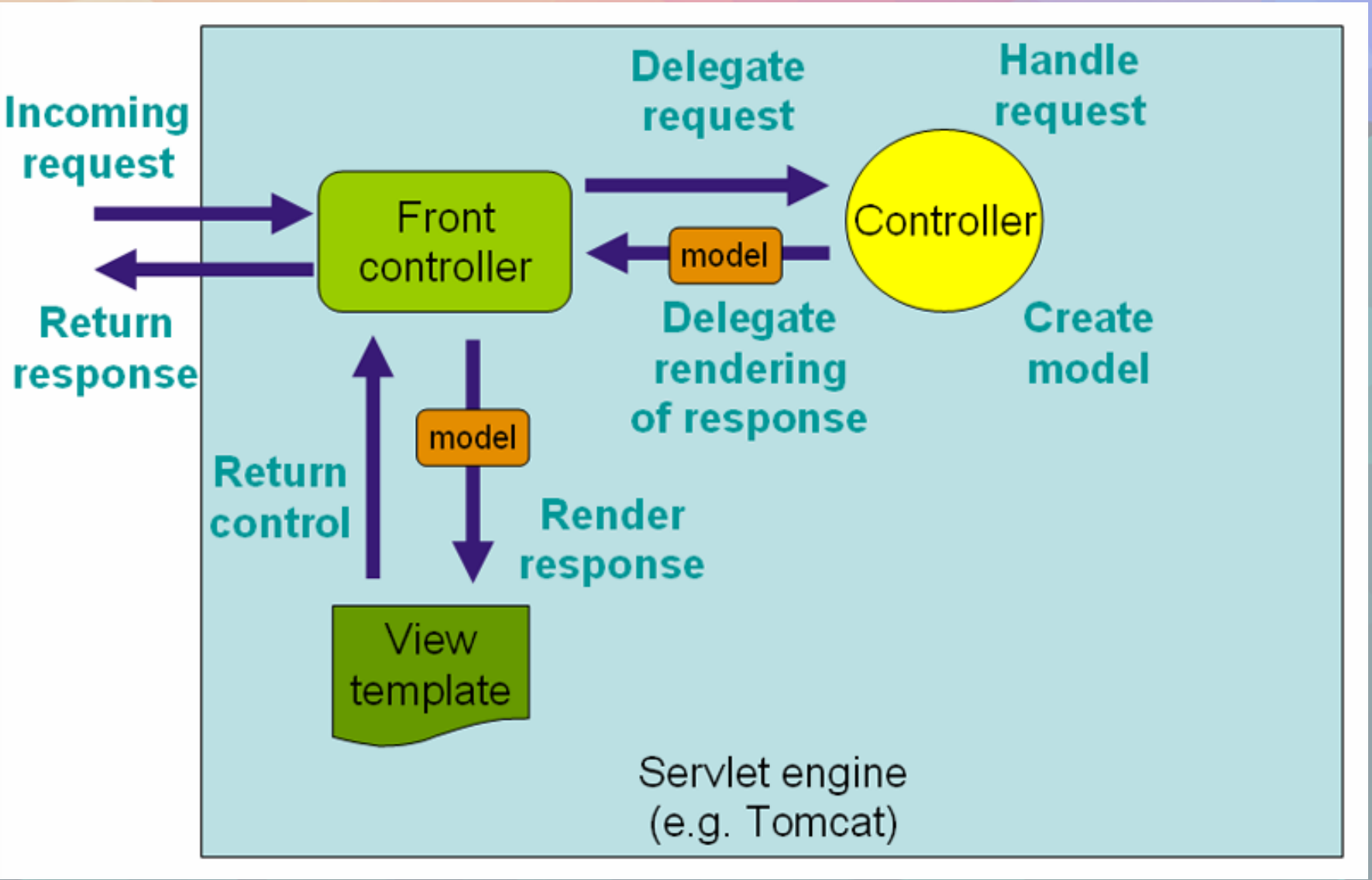
Application Context

- An ApplicationContext is responsible
  - to load bean definitions
  - to wire beans together
  - to dispense beans upon request
  - . . . and much more
- Many implementations of ApplicationContext exists (xml-based, annotation-based, ...)

What's Spring MVC

- Software component in the Spring framework
- MVC framework for web applications
- Lightweight and opensource

THE MVC DESIGN PATTERN

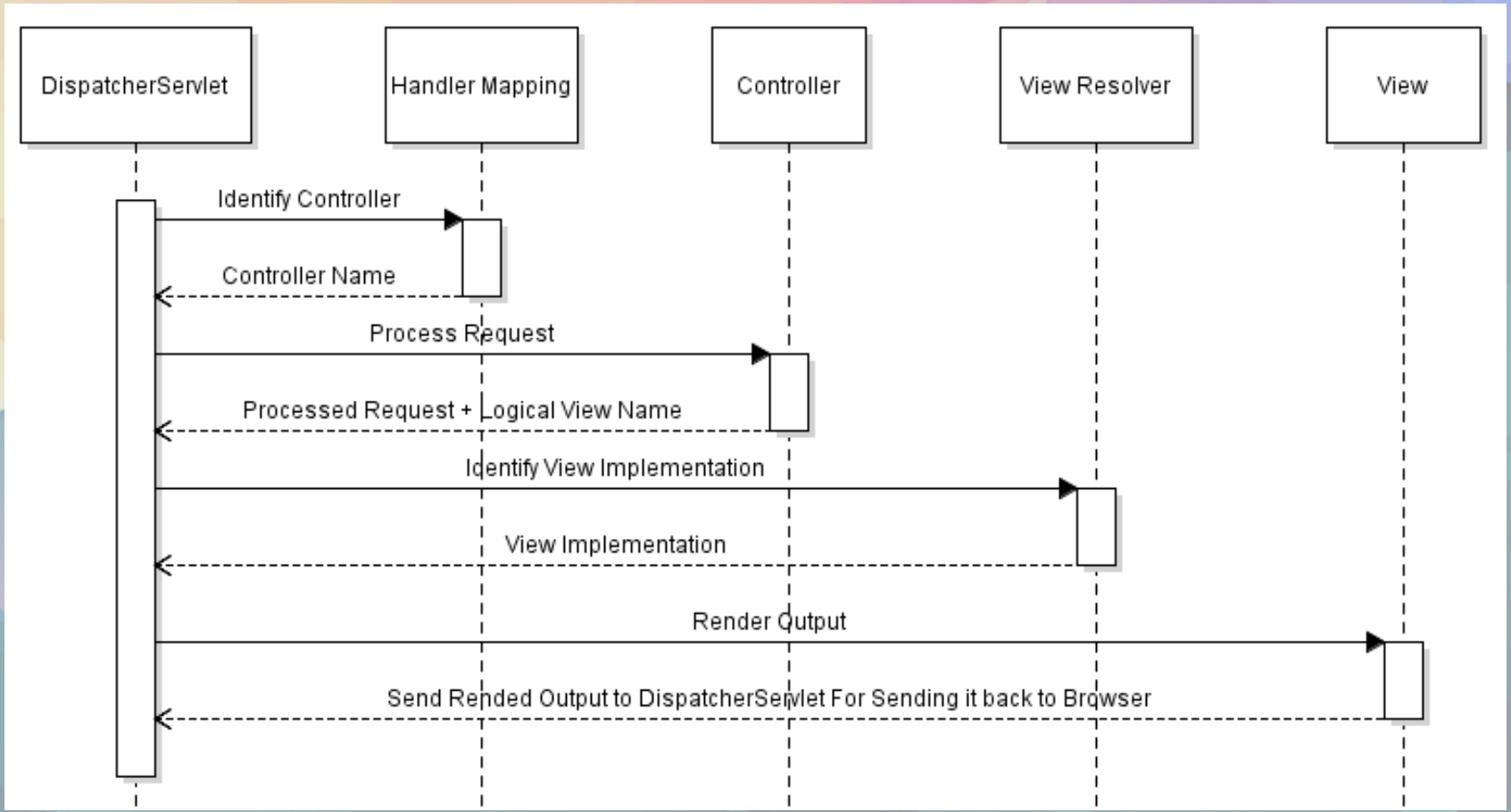


JAVA MVC WITHOUT SPRING

- model: Java POJO
- views: JSP pages
- controllers: java servlets

- model: Java Map
- views: pluggable view technologies
- controllers: controller classes

REQUEST HANDLING



REQUEST HANDLING

What requests are handled by what controller is specified by @RequestMapping annotate methods (or annotated by other verb-specific mappings like @GetMapping, @PostMapping, etc.)
@RequestMapping annotated methods:

- are customizable to accept parameters (@Parameter), body content (@RequestBody), a Model object, Session/Request objects and others
- return the view responsible to render the result (or the view name that will be resolved)..
- unless they are annotated also with @RequestBody to denote that their return an object that is intended to be returned as the message body

THE DISPATCHER SERVLET

- Handles all incoming requests and routes them to Spring controllers
- Uses customizable logic to determine which controllers should handle which requests

THE MODEL

- Used to pass objects from the controller tier up into the view
- It is just a java.util.Map
- Attributes in the Model will be passed as request attributes and available in the application PageContext
- In Spring we can use a simple Map or Spring specific classes: ModelMap or Model

An example of controller class

Code on eclipse IDE

Web services

Web services are a mean to implement web APIs

- provide a service that is accessible through the HTTP protocol
- web service oriented architecture typically have a good separation of concern
- stateless web services scale well: knowing which specific instance is serving the request is not important then more instances can be spawned at any time without affecting sessions
- sometimes are improperly called REST services or RESTful services, but they should adhere to the REST architectural principle to be called so, which is often not the case

Web services in spring are developed by annotating classes with the @RestController stereotype

- the same mechanism of @Controller applies, but there is no view nor view resolution
- @RequestBody is assumed by default on @RequestMapping annotated methods

An example of REST controller class

Code on eclipse IDE

Consuming web services with Spring

Spring offers facilities for consuming web/REST services

- implemented by the RestTemplate class