SPRING BOOT FUNDAMENTALS

DAY 1

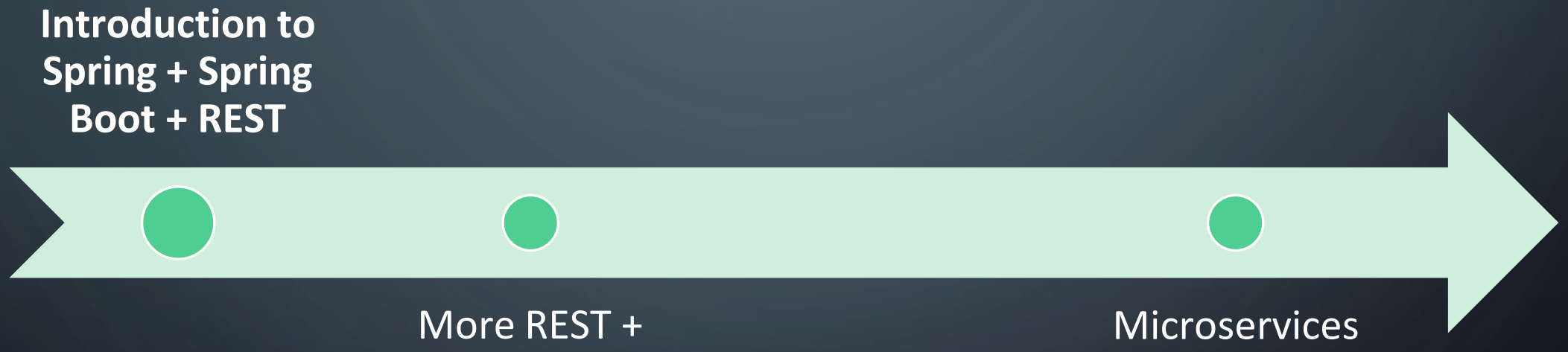
SPRING BOOT FUNDAMENTALS

DAY 1

GOAL OF SPRING DAYS OF THE COURSE

- Solid basic understanding of Spring and Spring Boot
- Know different projects and be able to create implementations with them
- Be able to independently modify an existing Spring application

COURSE OVERVIEW



GOAL OF TODAY, AFTER THIS SESSION YOU…

- Know the difference between Spring and Spring Boot
- Can explain what a Spring bean is
- Can explain what the application context is
- Can explain what the Spring container is
- Can explain what dependency injection and inversion of control is
- Can instantiate a Spring Boot application

TODAY'S SCHEDULE

- Introduction to Spring
- Spring framework
- Spring Bean
- Application context
- @Component
- Dependencies and dependency injection
- Spring boot overview
- Maven and Spring
- POM and adding Spring projects

WHAT CAN YOU DO WITH SPRING?

Creating scalable modular enterprise applications

- Spring handles the infrastructure, developer can focus on specific requirements for the application
- Examples of Spring modules: Web, Data, Security and many more: https://spring.io/projects

- Examples of Spring usages:
  - MVC web application
  - Applications with many different databases
  - Big data analysis
  - REST APIs

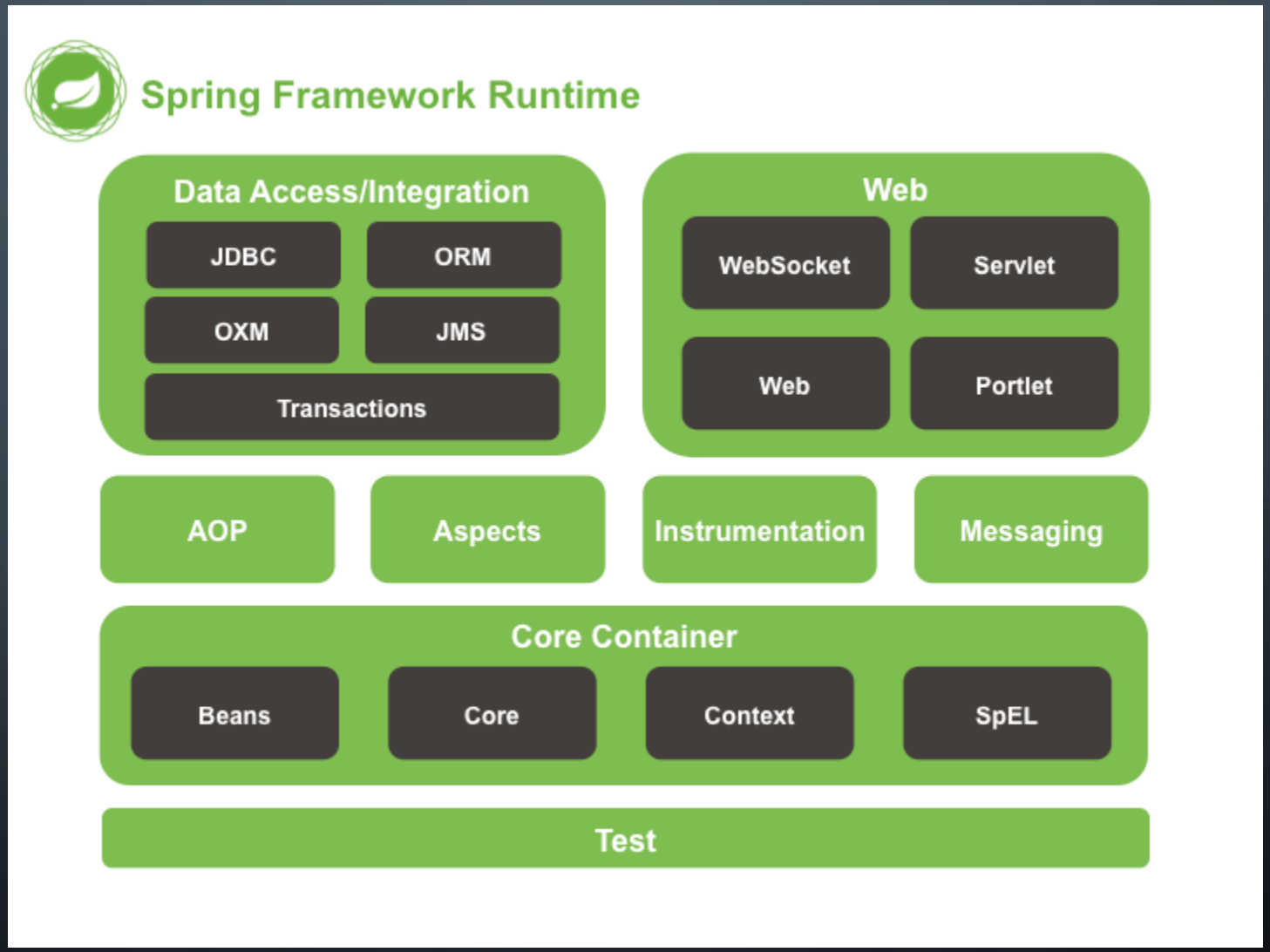
SPRING OR SPRING BOOT?

There is a difference!

- Spring is a framework for developing Java applications (just like Java EE). The Spring framework has different modules (also called ‘projects’) for very common functionalities of enterprise environments.

- Spring Boot configures all the modules/projects of Spring automatically in the best practice most common way.
- All you need to do is select the projects you need and press a button. The project and the configurations are made for you.

OVERVIEW OF THE MODULES IN THE SPRING FRAMEWORK



HOW DID WE DO THESE THINGS BEFORE WE HAD FRAMEWORKS LIKE SPRING?

We had to work… Much harder

- Manual work that has been automated includes:

- Database connections
- Database requests
- Endpoint handling
- Multi-threading / Auto-closeable
- Factory patterns
- Instantiations of entities
- And many more!

HOW DOES SPRING DO ITS MAGIC?

- Adds enterprise services to POJOs
- Dependency injection and inversion of control (coming soon)
- Spring container does most of the magic

EXERCISE

- Take 10 minutes to come up with a two sentence explanation of inversion of control and dependency injection.

INVERSION OF CONTROL AND DEPENDENCY INJECTION

Inversion of Control

Inversion of control: code written by developer doesn't call libraries and dependencies, but the framework calls custom code written by developers

Dependency injection: one way of applying inversion of control. External code injects the dependencies to the class.

Old days:

The classes (and manager classes for each class) take care of instantiation and accessibility and connections etc their selves by calling on libraries.

Now:

The framework is in control and does all these things!

```
public class ExampleController {
    private ExampleService exampleService;

    ExampleController(){
        this.exampleService = new ExampleService();
    }
}
```

```
@Controller
public class ExampleController {
    private ExampleService exampleService;

    @Autowired
    ExampleController(ExampleService exampleService){
        this.exampleService = exampleService;
    }
}
```

HOW DOES SPRING RELATE TO JAVA EE?

Very closely!

- **You can do very similar things with both**
- **Java EE is a collection of protocols**
- **Spring is a collection of modules/projects**

- Java EE is a very extensive guideline of best practices (protocols) to build and structure your application. Hibernate is an example of an implementation for a JPA (but it’s not exclusive to Java EE)

- Spring follows these guidelines of Java EE, but uses an implementation with modules to leave out parts you don’t need

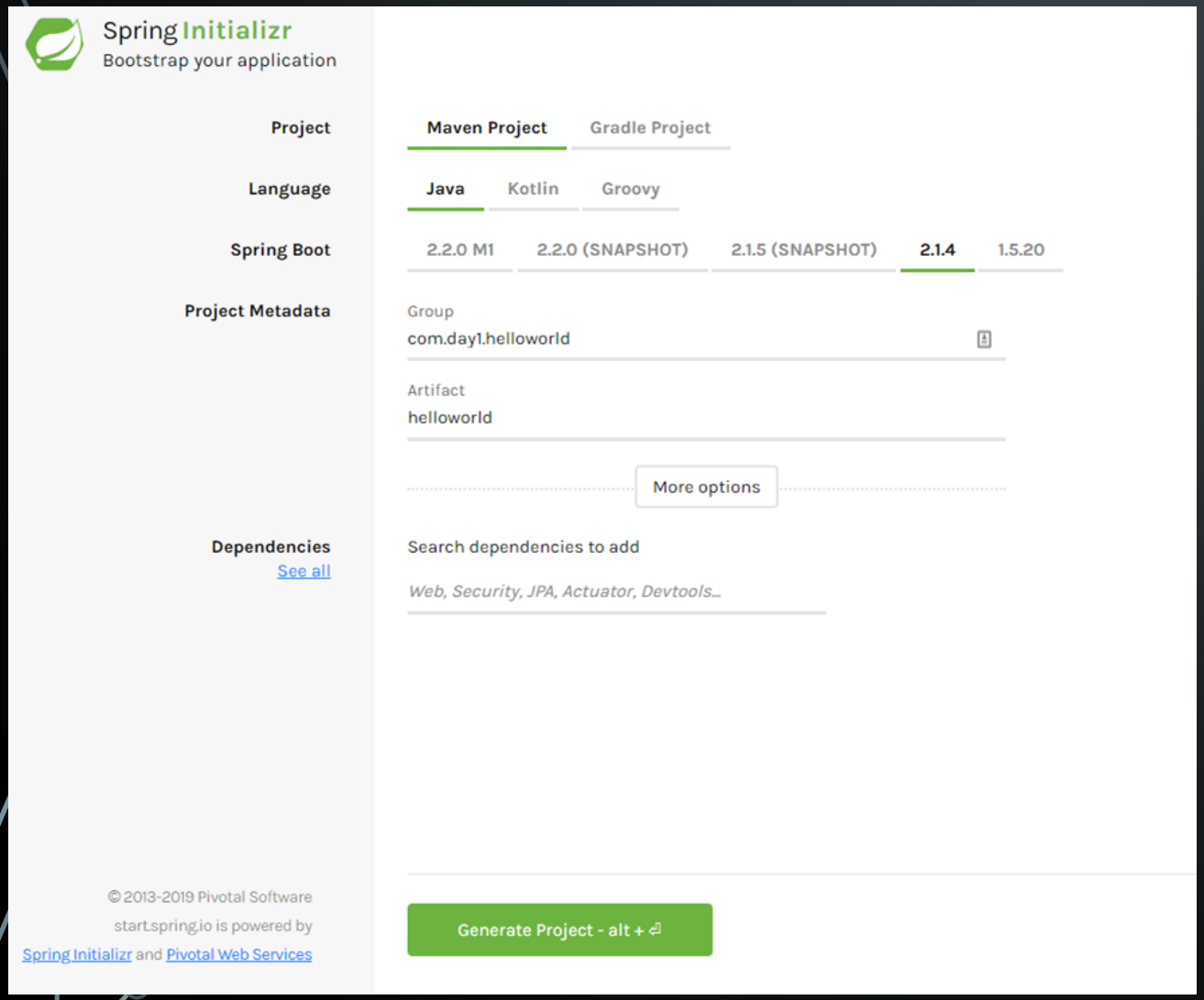
- The greatest difference is that Spring is an actual library while Java EE is an API that has to be implemented.

HELLO WORLD!

- Baby steps
- Create a Spring Boot application that reads console input and prints “Hello world + <console input>!” to the console.

- Giant steps: Assignment ++: Write a Spring Boot REST service application that returns “Hello World” from any endpoint.

BOOTSTRAPPING



MANUAL MAGIC – IMPORT INTO INTELLIJ

- Easy to start using your new project:
- Download and extract the zip file to your training dedicated projects folder (custom made ;-)).
- Try to run the Application.

spring-training [D:\IdeaProjects\spring-training] - ...\src\main\java\nl\brightboost\springtraining\SpringTrainingApplication.java [spring-training] - IntelliJ IDEA

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

spring-training > src > main > java > nl > brightboost > springtraining > SpringTrainingApplication

Add Configuration...

Project

- spring-training D:\IdeaProjects\spring-training
  - .idea
  - .mvn
  - src
    - main
      - java
        - nl.brightboost.springtraining
          - SpringTrainingApplication
      - resources
    - test
  - .gitignore
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml
- External Libraries
- Scratches and Consoles

SpringTrainingApplication.java

```
1  package nl.brightboost.springtraining;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class SpringTrainingApplication {
8
9      public static void main(String[] args) { SpringApplication.run(SpringTrainingApplication.class, args); }
12
13 }
14
```

SpringTrainingApplication

Terminal 6: TODO Event Log

7:14 LF UTF-8

HELLO WORLD! – EXPLAINED

- What just happened?
- You generated a Spring Boot application.
- Did some manual magic...
- Got yourself a working application by running the main method.

WHY SPRING? WHY SPRING BOOT?

Spring!? (of course also for Java EE)

- Splitting up your application into layers making the interchangeability easier (frontend, backend, controllers, data access objects, facades containing business logic, repositories handling the database interaction etc...).
- But also removing all the hardcoded connections to the ‘outside world’ -> Annotations! So you can focus on what you want, writing the handling logic.

Spring Boot!?

- Remove the work to get all correct packages, you can start developing the business logic fast. Default configuration is provided

EXERCISE

- Take 15 minutes to come up with a two sentence explanation of Spring beans.

SPRING BEANS

- So what are beans?

Bean is an object instance, that is instantiated by the IoC container.

The framework is in control of which beans are instantiated and ended.

In code you can recognize beans like this: @..... (and then some Bean annotation like @Bean, @Component, @Controller, @Repository etc.)

Not all classes are beans, only the ones that need to be managed by the container. Examples of beans in everyday enterprise applications are REST controllers, models/entities and repositories.

The container manages the lifecycle of the beans, the transactions, security and other system-level services.

EXERCISE

- Take 5 minutes to come up with a two sentence explanation of Spring container.

SPRING CONTAINER

- Core of the Spring framework
- Manages difficult tasks that developers had to do by hand in the olden days, such as managing beans and their life cycle, managing threads, etc

APPLICATION CONTEXT

- Spring’s container (together with BeanFactory (later))
- Loads and holds and wires bean definitions
- Add enterprise-specific functionality
- Different implementations available

COMPONENT SCANNING IN SPRING

- Spring automatically scans for beans
- @ComponentScan scans the package of the class that annotions is on and all the subpackages
- If main method class is also the configuration class (so no custom @Configuration) the scanning will start from there
- If Spring needs to inject something that is not a bean, you'll get an exception

DEMO – EXCEPTIONAL DEMO

LET'S DO THE COMPONENT SCANNING WRONG AND SEE WHAT HAPPENS.

ASSIGNMENT 1 - BEANSOUP

- Imagine... That we like beansoup, and the more different beans we use the better!

- Write a Spring Boot console application that has different types of ‘Bean’ Classes (at least 2), and let it print out your favourite type of bean in either the console or as a response on a controller endpoint call.

- Assignment ++: Can you explain what happens with the different Beans you use in your application when you run it?

- Assignment theory investigation: So in regards to lifecycle and what the container uses as default settings/configuration, can you make a distinction between certain types of Beans?

XML

Olden days..

- All configuration happened via XML
- Application context stored in applicationContext.xml
- You won’t find this file in new projects

- Developers didn’t like all this XML config and therefore they’ve made Java configuration possible. More on this in session two ☺

SPRING ANNOTATIONS

- You have all used them
- Before the beans and annotations would be configured in xml, more and more frameworks are moving to annotation based configurations.
- Luckily we now have annotations.

```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"
   xsi:schemaLocation = "http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <!-- A simple bean definition -->
   <bean id = "..." class = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with lazy init set on -->
   <bean id = "..." class = "..." lazy-init = "true">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with initialization method -->
   <bean id = "..." class = "..." init-method = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- A bean definition with destruction method -->
   <bean id = "..." class = "..." destroy-method = "...">
      <!-- collaborators and configuration for this bean go here -->
   </bean>

   <!-- more bean definitions go here -->

</beans>
```

```java
@Configuration
public abstract class VisibilityConfiguration {

  @Bean
  public Bean publicBean() {
     Bean bean = new Bean();
     bean.setDependency(hiddenBean());
     return bean;
  }

  @Bean
  protected HiddenBean hiddenBean() {
     return new Bean("protected bean");
  }

  @Bean
  HiddenBean secretBean() {
     Bean bean = new Bean("package-private bean");
     // hidden beans can access beans defined in the 'owning' context
     bean.setDependency(outsideBean());
  }

  @ExternalBean
  public abstract Bean outsideBean()
}
```

SPRING ANNOTATIONS

- Way to easily configure behavior of a piece of code

Several kinds of annotations:

- Core spring framework annotations
- Stereotype annotations
- Spring Boot annotations
- REST and MVC annotations
- Mapping annotations
- Spring Cloud annotations
- And more…

- You can even make custom annotations!

HELLO WORLD REVISITED

- What annotations do you recognize?

- The startpoint of your application
- (convention is <applicationName>Application.java)

- Your automatically generated test has some annotations (we'll zoom into this further along the training)

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class HelloworldApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```

```
package com.dayl.helloworld.helloworld;

import ...

@SpringBootApplication
public class HelloworldApplication {

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

}
```

ANNOTATIONS

- If you have done the Assignment ++
- You might have split of your logic into a @Service

```java
package com.helloworld.helloworld.service;

import ...

@Service
public class HelloworldService {

    @Autowired
    HelloworldRepository<Helloworld> helloworldRepository;

    public List<Helloworld> helloworld() { return helloworldRepository.findAll(); }
}
```

```java
package com.helloworld.helloworld.controller;

import ...

@RestController
public class HelloworldController {

    @Autowired
    HelloworldService helloworldService;

    @RequestMapping(value = "/helloworld", method = RequestMethod.GET)
    public String helloworld() {
        //return helloworldService.helloworld();
        return "helloworld";
    }
}
```

ANNOTATIONS

- And even more…
- The @(Rest)Controller configures the application endpoints that we could call

```java
package com.helloworld.helloworld.service;

import ...

@Service
public class HelloworldService {

    @Autowired
    HelloworldRepository<Helloworld> helloworldRepository;

    public List<Helloworld> helloworld() { return helloworldRepository.findAll(); }
}
```

```java
package com.helloworld.helloworld.controller;

import ...

@RestController
public class HelloworldController {

    @Autowired
    HelloworldService helloworldService;

    @RequestMapping(value = "/helloworld", method = RequestMethod.GET)
    public String helloworld() {
        //return helloworldService.helloworld();
        return "helloworld";
    }
}
```

```java
@Entity
@Table(name="helloworld")
public class Helloworld {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name="helloworld")
    private String helloworld;

    public Helloworld() {}
```

KITCHEN CHAOS! (ASSIGNMENT 2)

Context:

- Our kitchen is one big chaos. The previous chef had a paper based system that we used to have. You can imagine sometimes the orders would simply get lost! Obviously this is no longer sufficient and we need some terminal application that will allow us to place an order (consisting of multiple items), the kitchen personnel to view what orders are placed and the servicing staff to see what orders can be served.

- Assignment ++: Can you explain what happens with the different Beans you use in your application when you run it?

- Assignment theory investigation: So in regards to lifecycle and what the container uses as default settings/configuration, can you make a distinction between certain types of Beans?

ASSIGNMENT 2

- Kitchen chaos!

- **What are the constraints:**
- We have 3 types of output shown (all in 1 console is just fine)

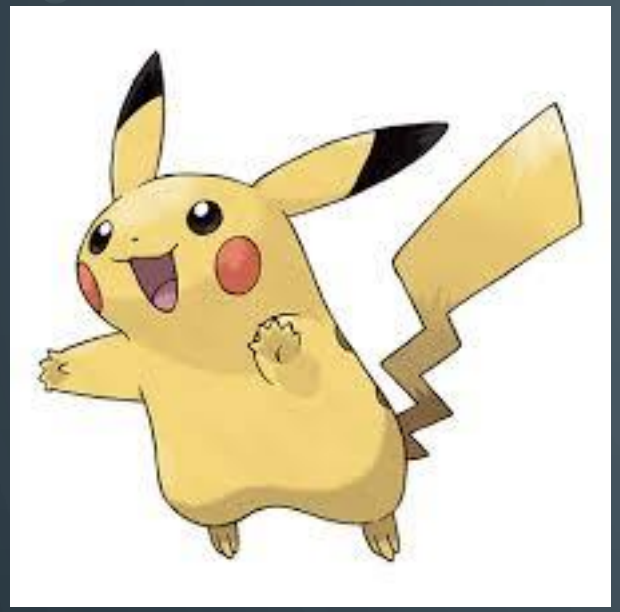
1. Order input + confirmation for the guests
2. Order overview for the kitchen + input to confirm a finished order
3. Order overview for the staff to see where to deliver the order

- Use Spring, SpringBoot to get your application up and running.
- Make sensible choices in designing your application (packages etc).
- Have at least 1 Controller and Service doing the work for you.

- **Tips**
- Split up the application into logical layers?
- Have you split the concerns into different classes?

ASSIGNMENT 2

- Present the awesomeness!
- You have all put in hard work and some awesome effort; who's up for the challenge to show us what you made!?



- 4 Don’t belong and one these 4 is a Pokemon, do you know which?

- **Annotations**
  - @Repository
  - @Singleton
  - @Magneton
  - @Multiton
  - @Controller

- **Annotations**
  - @PathVariable
  - @ModelAttribute
  - @Component
  - @Service
  - @RestController
  - @SecureController

- **Annotations**
  - @Autowired
  - @RequiredArgsConstructor
  - @Auto
  - @SpringBootApplication
  - @EnableAutoConfiguration
  - @ComponentScan
  - @Configuration
  - @EnableTransactionManagement
  - @EnableJpaRepositories
  - @PropertySource
  - @Entity

ADDING PROJECTS TO SPRING

- Add the correct dependency to your POM.xml
- And you're good to go!

- Using project? Usually annotations, sometimes a bit more work

JAVA CONFIGURATION - @CONFIGURATION

- Adding a Java class for config (often called AppConfig.java)
- This class has the annotation @Configuration
- Replacement for the old XML config files

QUESTION: IS THE CONFIGURATION CLASS A BEAN?

QUESTION: IS THE CONFIGURATION CLASS A BEAN?

- Yes!

@BEAN

- Add @Bean to a method in the configuration class
- This enables injection: setter, constructor, field
- Default all @Bean are singleton and only created the first time the method gets called

CONSTRUCTOR INJECTION

- @Autowired on top of constructor
- Dependencies are provided to the bean through the constructor

SETTER INJECTION

- Injection through the setter, autowired is on top of a setter
- Useful if you don’t have a constructor
- Allows reconfiguration later, new bean can be set as a dependency
- Third party code only supports setter injection

FIELD INJECTION

- Autowired on top of the field
- Recommended to not use this too much, because it’s heavy on the performance since it works with reflection

EXERCISE

- Can you come up with use cases for field injection?
- Show a code example

CAN YOU COME UP WITH USE CASES FOR FIELD INJECTION?

- When you have a @Configuration class with a certain bean, that depends on a bean of another @Configuration class
- Inject beans of the Spring container infrastructure that you'd need to modify (which is terrible, because you tie your code to the framework, but still)
- Tested bean in test classes, keeps it readable and performance less relevant

EXERCISE

- Some extra topics:
  - @Qualifier
  - @Value
  - Required property on @Autowired
  - @AliasFor
  - @Lazy
- Tell me in groups of 2/3:
  - What is it?
  - Demonstrate with a little piece of code
  - What are the pros/cons for using it?

HOW DOES SPRING DECIDE WHAT TO INJECT?

- First on type: this works if there's only one bean of that type
- If there is more than one, it is looking for the bean with @Qualifier annotation
- If nothing is found, it tries to autowire by name. Default name is classname starting with lowercase letter

WHAT EXCEPTION DO YOU GET IF SPRING CANNOT FIND A BEAN TO INJECT?

- And what if it’s ambigue?

WHAT EXCEPTION DO YOU GET IF SPRING CANNOT FIND A BEAN TO INJECT?

- NoSuchBeanException

- UnsatisfiedDependencyException - NoUniqueBeanDefinitionException

BEAN SCOPES @SCOPE

- Singleton (default scope, valid in all configs) – one instance per Spring container
- Prototype (valid in all configs) – one instance per request
- Request (valid in web-aware Spring projects) – one instance per lifecycle
- Session (valid in web-aware Spring projects) – one instance per user session
- Global (valid in web-aware Spring projects) – one instance per global session (life of application on server, so new one after reboot)

- So e.g. @Scope(value="session") or @Scope(value=BeanDefinition.SCOPE_SESSION)

EXERCISE

- Ok sure, sounds fine… Right?
- But what happens if we have a Singleton bean that depends on a (for example) prototype bean?

EXERCISE

- Create a class Liar
- Liar depends on a bean Salary
- Salary is of type prototype
- Everytime liar’s getSalary gets called, a new value for salary is given

EXERCISE

- Is it better to create a bean using @Bean or @Component (or one of the stereotype annotations)?

- When should you be using each?

LIFECYCLE OF A BEAN

- Instantiation
- Populate properties
- BeanNameAware
- BeanFactoryAware
- BeanPostProcessors (pre initialize)
- InitializeBean
- Init
- BeanPostProcessors (post initialize) (@PostConstruct)

QUESTION

- Take 10 minutes to come up with a simple explanation for Bean Factory

BEAN FACTORY

Pattern to encapsulate object construction logic in a class.

It is a done in a way that the construction can be reused.

- Expands initMethod concept
- Factory method pattern
- Deal with legacy code
- Contract with constructor
- Static methods

https://www.youtube.com/watch?v=xlWwMSu5I70

MULTIPLE CONFIGURATION CLASSES

- You can import another configuration class to a configuration class with the @Import annotation on top of the config class

- @Import(MyOtherConfig.class)

- Exercise: Demonstrate this, by using a bean from another config file that gets imported by @Import

PROPERTIES

- @Value can be used to set the default using a properties file
- @Value(${prop.name})
- On top of class @PropertySource("…path to props file")
- Use prefix classpath for your path if it's in the resources

EXERCISE

- Create a configuration for a datasource
- Give it a driverclassname, url, username and password property
- Make sure this class gets it values for a property file called test-db.properties

PROFILES

- But what if we want to have different configs for our test and production environment?
- We can use profiles!
- On top of the configuration class we write @Profile(“test”)
- And on top of our @ContextConfiguration(classes = {ExampleConfigOne.class, ExampleConfigTwo.class})
- @ActiveProfile(“exampleone”) to activate the right config class
- Exercise: create a test with a production datasource and test datasource bean, give both a profile and use a simple runner to test the difference between the two active profiles

APPLICATION.PROPERTIES OR APPLICATION.YAML

- Properties for configuration
- Many different properties can be specified in the application.properties file
- https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

LOGGING IN SPRING BOOT

- Log4j
- Logback (default)

USING LOG4J

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

EXERCISE

- Get some REST
- Take 5 minutes to come up with a short explanation of what REST is.

WHAT IS A REST SERVICE?

Representational State Transfer

Follows 4 principles:

- **1. Resource identification through URI**
- e.g. <host>:<port>/application/persons)

- **2. Uniform interface**
- Manipulation of resources through CRUD operations.

- **3. Self-descriptive messages**
- Resources are decoupled from their representations so the content can be accessed in more than one format (plain text, HTML, XML, JSON etc etc). Metadata in the message is used to do authentication, detect errors, caching, access control etc etc…

- **4. Stateful interactions through hyperlinks**
- Every interaction with a resource is stateless. Stateful interactions can be done through URI manipulation, cookies or

QUIZZES

- What stuck?

1. What does @Autowired do?

1. Name 2 advantages of using Spring over ‘regular’ Java SE for your development.

1. What is the difference between a @Bean and a @Component?

1. What new thing did you learn today?

NEXT UP

- More REST
- Spring Data
- JPA + Hibernate

FOR NOW… GET SOME REST

QUESTIONS?

- Maaike.vanputten@brightboost.nl
- Or Whatsapp: +31683982426
- Don’t hesitate to contact me, I love to help!