/ JAVA DB SCHOOL
Spring Boot

Spring Boot

- Spring Boot is an open source Java-based framework used to create a microservices
- It is developed by Pivotal Team and is used to build standalone and production ready Spring applications
- Spring Boot provides a good platform for Java developers to develop a standalone and production-grade Spring application that you can just run
- You can get started with minimum configurations without the need for an entire Spring configuration setup

Advantages

- Easy to understand and develop Spring applications
- Increases productivity
- Reduces the development time

Spring Boot starters

- Handling dependency management is a difficult task for big projects
- Spring Boot resolves this problem by providing a set of dependencies for developers convenience

- https://start.spring.io/

First Spring Boot project

- https://start.spring.io/

- Dependencies:
  - Spring Data JDBC
  - MySQL Driver
  - Spring Web
  - Thymeleaf

/ Project review

Project review

- Project structure
- Main class
- @SpringBootApplication – convenience annotation
  - @Configuration - Tags the class as a source of bean definitions for the application context
  - @EnableAutoConfiguration - Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.
  - @ComponentScan - Tells Spring to look for other components, configurations, and services in the main package, letting it find the controllers

Application properties

- application.properties
- Possible configuration:
  - spring.application.name
  - server.port
  - server.address
  - spring.datasource.url
  - spring.datasource.username
  - spring.datasource.password

/ Web application

Web application

- Web application = an application accessible from the web
- Components
  - Backend – servlets, JSP, filters etc.
  - Frontend – HTML, CSS, JavaScript

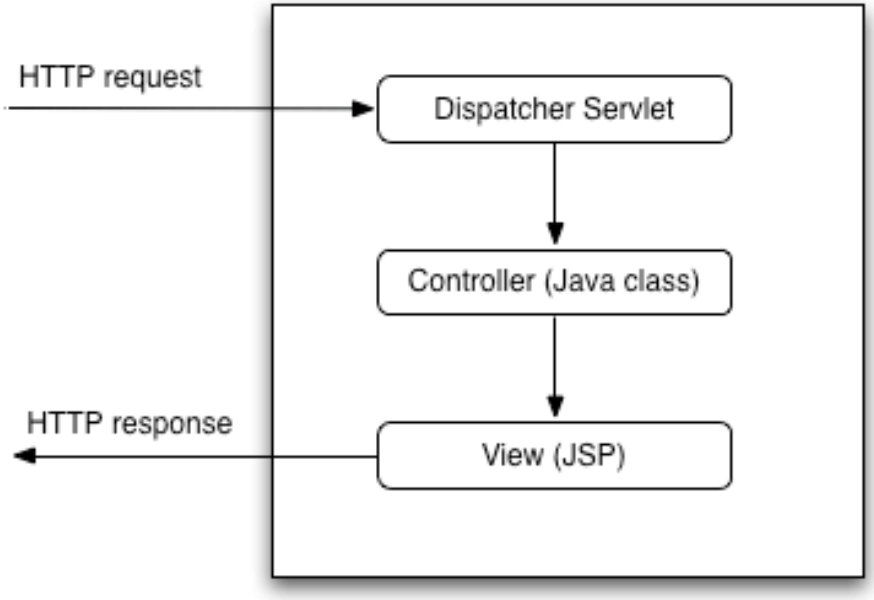
Servlets

- Depending on the context, servlets can be defined as:
  - Technology to create a web application
  - API that provides many interfaces and classes including documentation
  - A class that extends the capabilities of the servers and responds to the incoming requests
  - A web component that is deployed on the server to create a dynamic web page

Spring Dispatcher Servlet

- Front Controller design pattern – a single controller is responsible for directing incoming HttpRequests to an application's controllers and handlers

- Coordinates the HttpRequests to the corresponding handlers (methods)



Handler annotations (1)

- @Controller – marks a class as a request handler
- For web services:
  - @GetMapping
  - @PostMapping
  - @PutMapping
  - @DeleteMapping
- Common annotation parameters:
  - path – the mapped URL
  - consumes – narrows the media types the handler can process
  - produces - narrows the media types the handler can return

Handler annotations (2)

- For interacting with the request/response parameters:
  - @RequestBody – for extracting the parameters from the request body
  - @PathParam – for extracting the parameters from the URL
  - @RequestParam – for extracting query parameters from the URL
  - @ResponseBody – the method returns an object that should be serialized as JSON

Example

```
@GetMapping("/employees")
public List<Employee> getAllEmployees() {
    return employeeRepository.findAll();
}

@GetMapping("/employees/{id}")
@ResponseBody
public Employee getEmployeeById(@PathVariable(value = "id") Long employeeId) {
    return employeeRepository.findById(employeeId);
}

@PostMapping("/employees")
public Employee createEmployee(@RequestBody Employee employee) {
 return employeeRepository.save(employee);
}
```

IOC – Inversion of Control

- Inversion of Control – principle in software engineering which transfers the control of objects or portions of a program to a container or framework
- IoC enables a framework to take control of the flow of a program and make calls to custom code
- If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes

IOC advantages

- Decoupling the execution of a task from its implementation
- Making it easier to switch between different implementations
- Greater modularity of a program
- Greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

Spring IOC

- The interface ApplicationContext represents the IoC container
- The Spring container is responsible for instantiating, configuring and assembling objects known as beans, as well as managing their life cycles
- Bean = reusable software component

Dependency Injection (1)

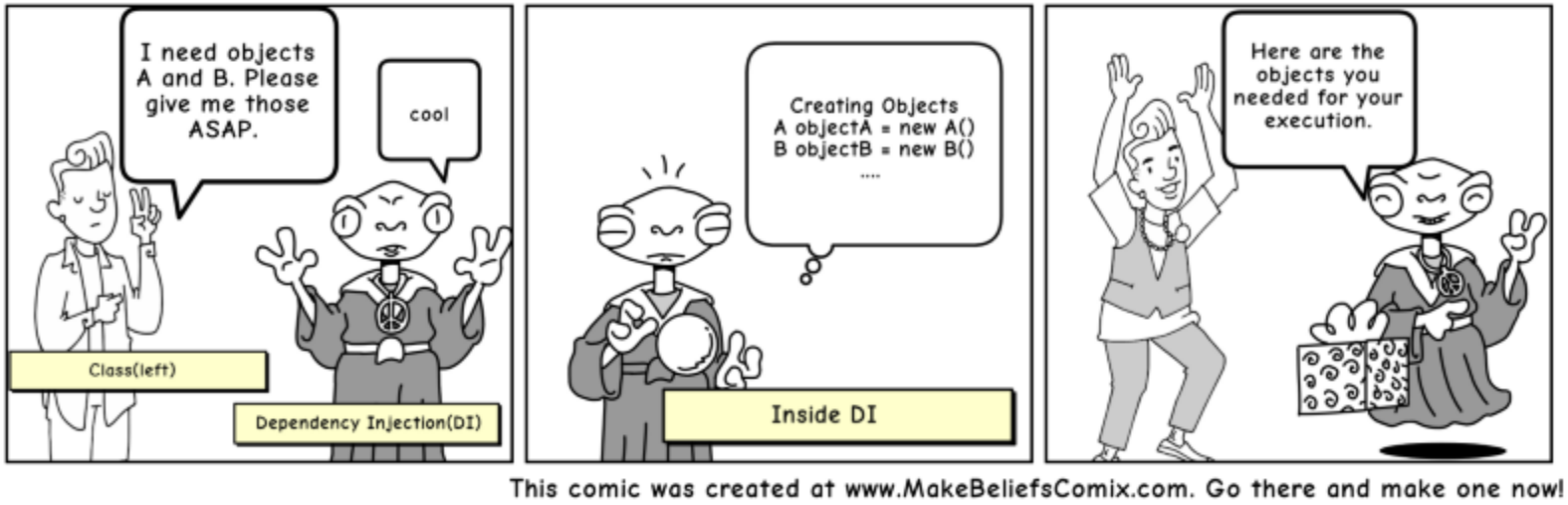
- Pattern used to implement IOC
- Transferring the task of creating the object to someone else and directly using the dependency

```
public class Store {
    private Item item;

    public Store() {
        item = new ItemImpl1();
    }
}
```

```
public class Store {
    private Item item;
    public Store(Item item) {
        this.item = item;
    }
}
```

Dependency Injection (2)



Reference: https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/

Dependency Injection (3)

- @Autowired – annotation used for injecting / requesting a bean

- @Autowired MyClass myObject;

  vs.

- MyClass myObject = new MyClass();

/ JDBCTemplate

JDBCTemplate (1)

- From Spring Data JDBC
- Used for interacting with the database
- Common methods
  - update – for insert / update / delete

```
jdbcTemplate.update( "INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)",
id, "Bill", "Gates", "USA");
```

  - queryForObject – returns a single element

```
jdbcTemplate.queryForObject( "SELECT COUNT(*) FROM EMPLOYEE",
Integer.class);
```

  - query – returns a list of elements, uses a RowMapper

JDBCTemplate (2)

- RowMapper – interface that maps a row from a table to a Java object

```
public class EmployeeRowMapper implements RowMapper<Employee> {
   @Override public Employee mapRow(ResultSet rs, int rowNum)
                              throws SQLException {
      Employee employee = new Employee();
      employee.setFirstName(rs.getString("FIRST_NAME"));
      employee.setLastName(rs.getString("LAST_NAME"));
      return employee;
} }
String query = "SELECT * FROM EMPLOYEE";
List<Employee> employees = jdbcTemplate.query(query, new EmployeeRowMapper());
```

/ Semantic annotations

Semantic annotations

- Semantic annotations
  - @Controller
  - @Service – annotates classes at the service (business logic) layer
  - @Repository – annotates classes at the persistence layer
  - @Component – generic annotation for any Spring-managed component

/ Practice, practice, practice

Practice

- Create controller classes for customers, products and orders
- Implement methods to get all values, get by id, insert, update and delete

/ Q&A


MOBILE/ACADEMY