Swedbank

# Spring Framework 5

2021.02.15

Go to www.menti.com and use the code 42 46 40 2

Mentimeter

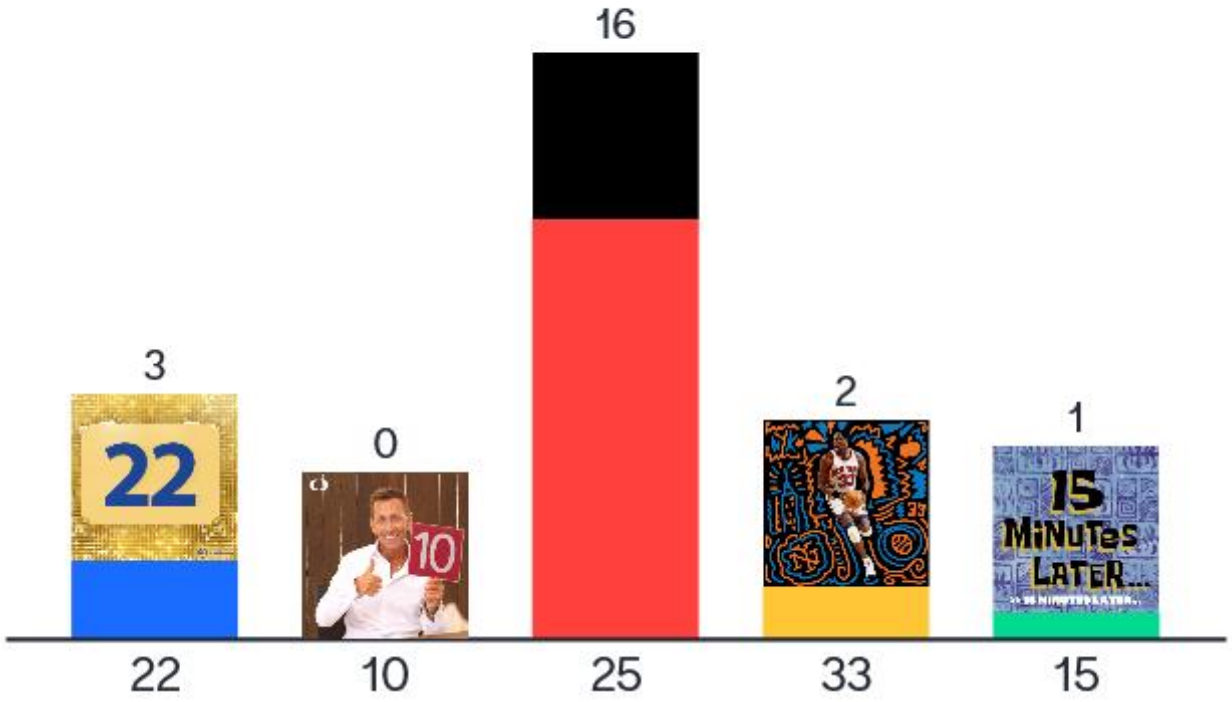
# Apibūdinkite savo nuotaiką šiandien


20

Go to www.menti.com and use the code 42 46 40 2

# Kiek Java kalbai metų?

Mentimeter

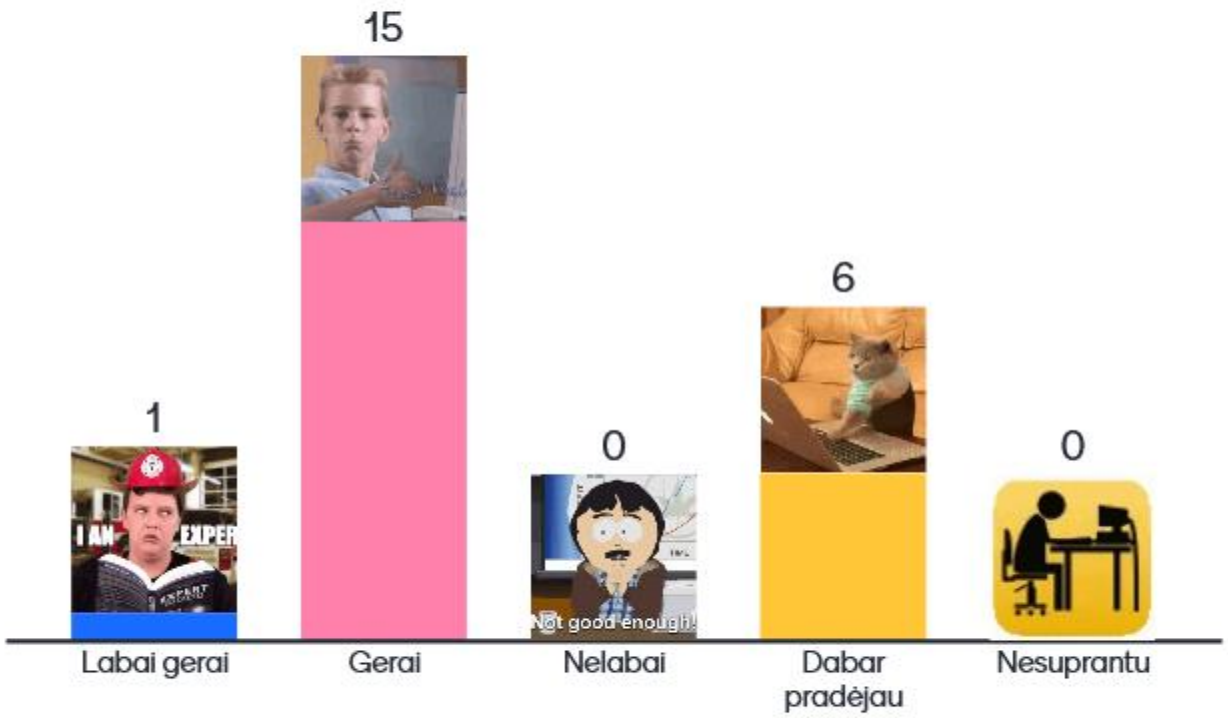



22

Go to www.menti.com and use the code 42 46 40 2

Mentimeter

# Java kalbos mokėjimas




22

Go to www.menti.com and use the code 42 46 40 2

# Ko tikiuosi išmokti Spring mokymuose ?

Mentimeter

pradėti darbą
skaityti koda
patirties
sužinoti su kuo dirba
pritaikyti koda
susipažinti su spring
realūs žmonės
how deep rabbit hole goes
suzinot kuom tai pagelbes
suzinot kas tai yra
suprasti springo fw
backend
nulauzt pentagona
išmokti efektyviai naudot
sužinot kažką naujo
igyti patirties su framew
ismokti spring
išmokti pirmą framework
suprasti koda
patobulinti oop skillz
test spring web app
išmokt spring'ą
patobulinti java

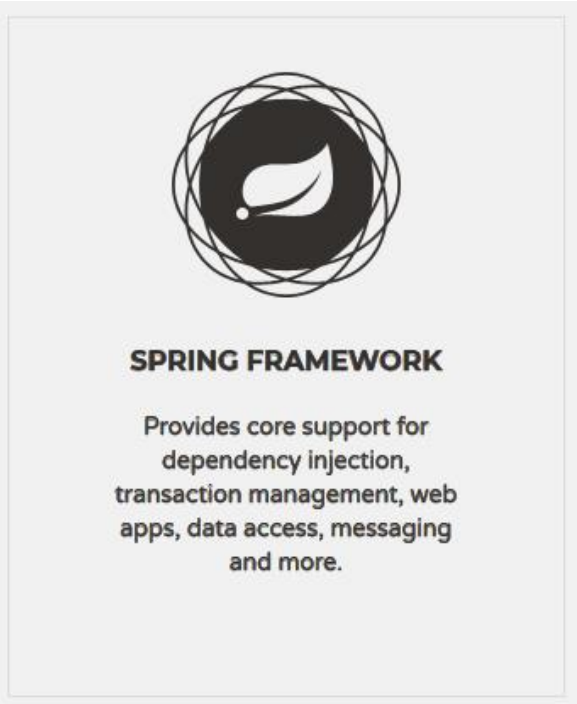
18

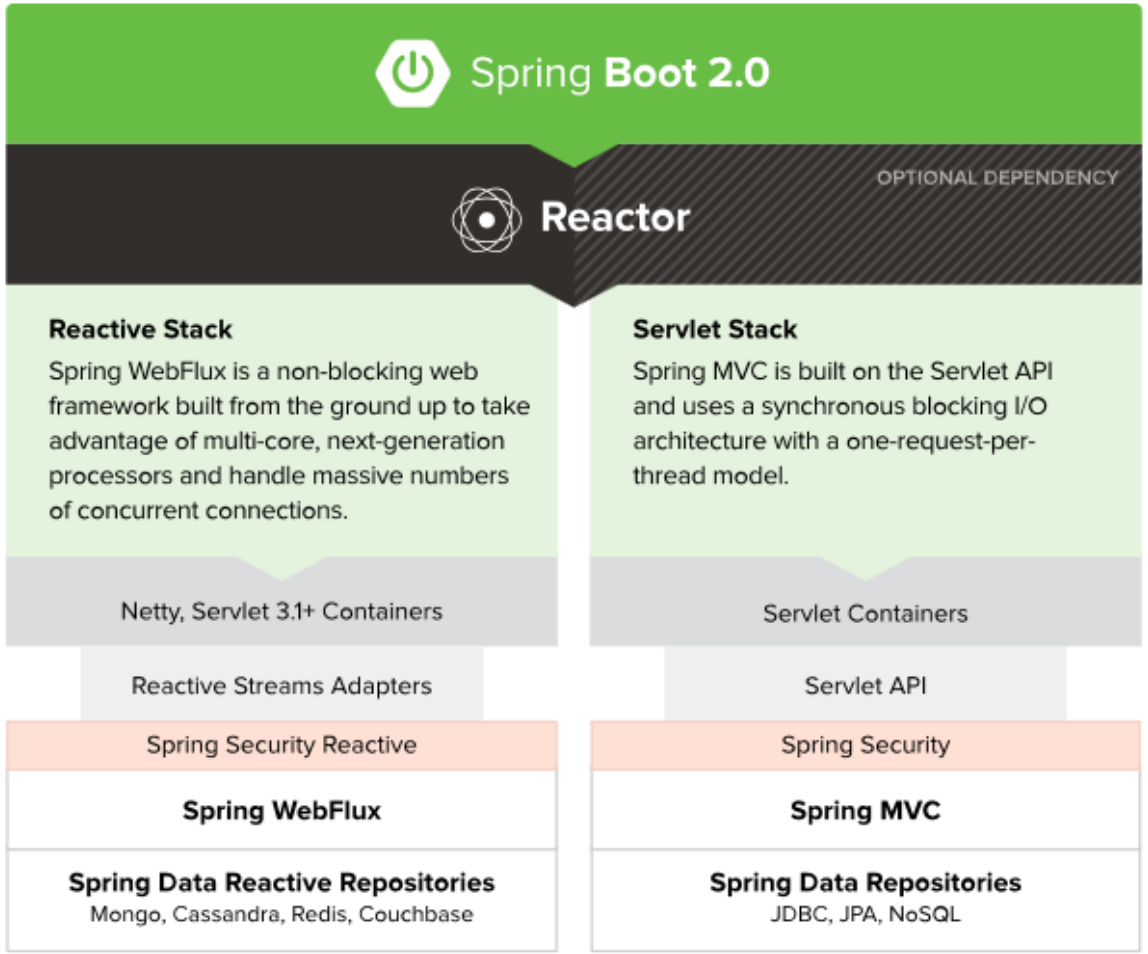
Spring Framework 5

# Overview
# Spring Data
# Spring Rest

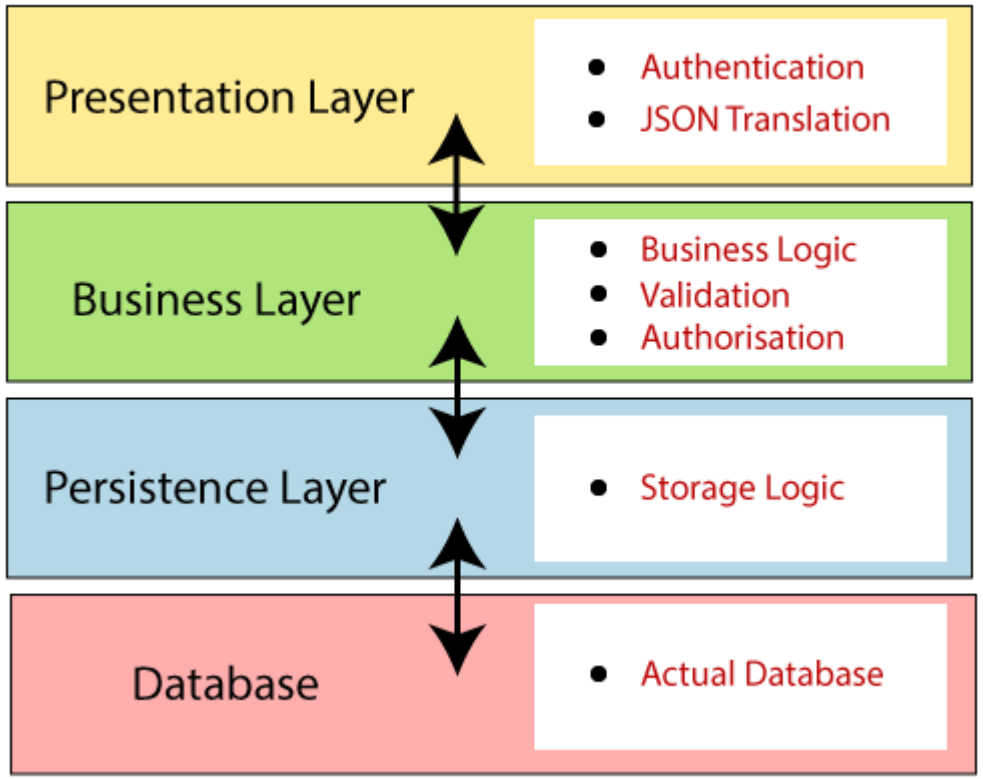
# Spring overview

Spring Framework is one of the most popular Java EE frameworks. It is an open source and light weight framework created by Rod Johnson in 1 October 2002 20 years ago



# Spring Boot layers



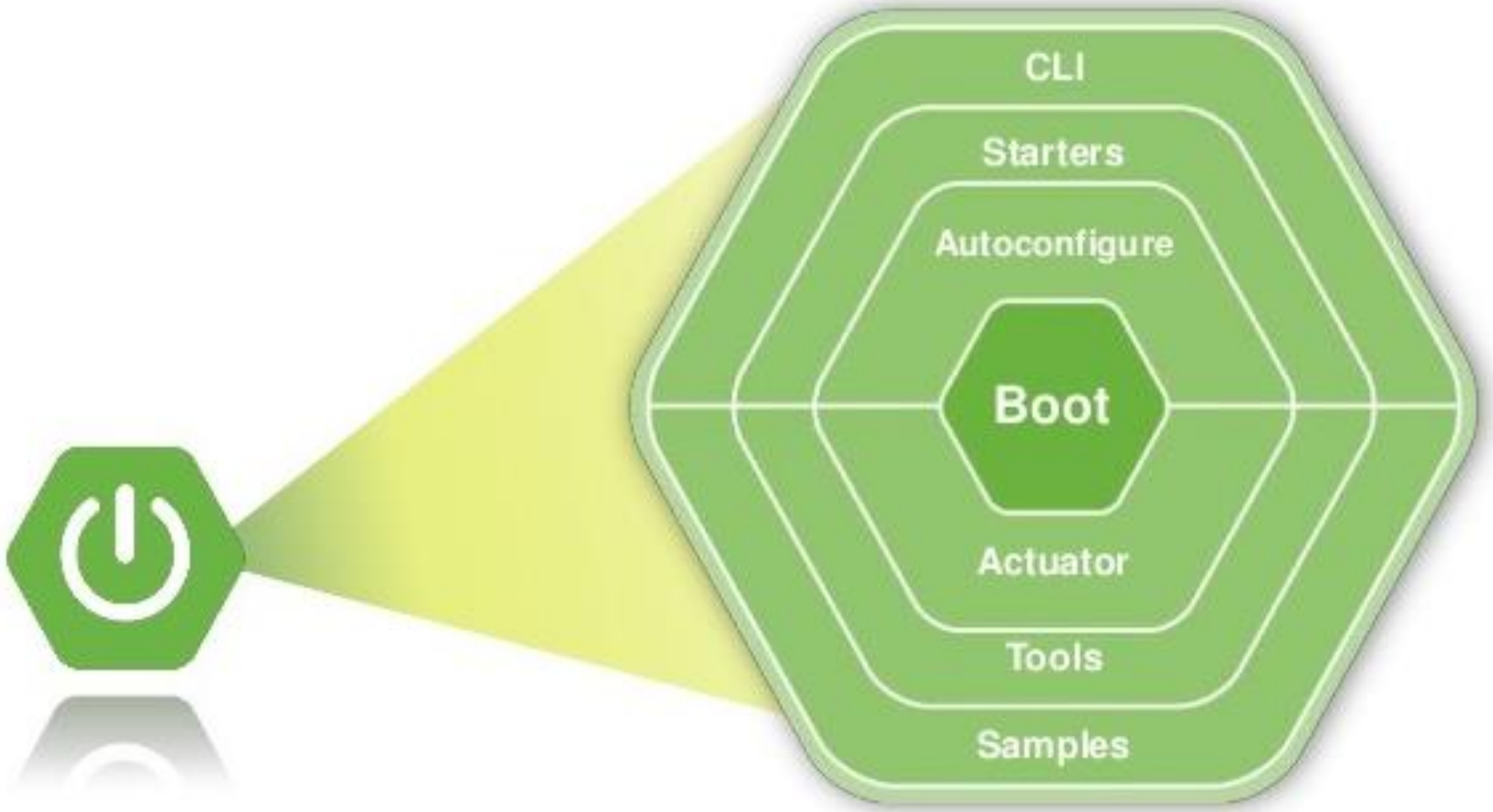
# Spring Boot

- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
- We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.
- Latest version 3.0.0

# Spring – Boot – Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

# Spring boot architecture diagram



# *org.springframework.boot.autoconfigure* and conditions packages – 1

- **@SpringBootApplication** - We use this annotation to mark the main class of a Spring Boot application
- ***@EnableAutoConfiguration* -** as its name says, enables auto-configuration. It means that **Spring Boot looks for auto-configuration beans** on its classpath and automatically applies them.
- ***@Configuration*** - Usually, when we write our **custom auto-configurations**, we want Spring to **use them conditionally**. We can achieve this with the annotations in this section.
- ***@ConditionalOnClass* and *@ConditionalOnMissingClass* -** Using these conditions, Spring will only use the marked auto-configuration bean if the class in the annotation's **argument is present/absent**

# *org.springframework.boot.autoconfigure* and conditions packages – 2

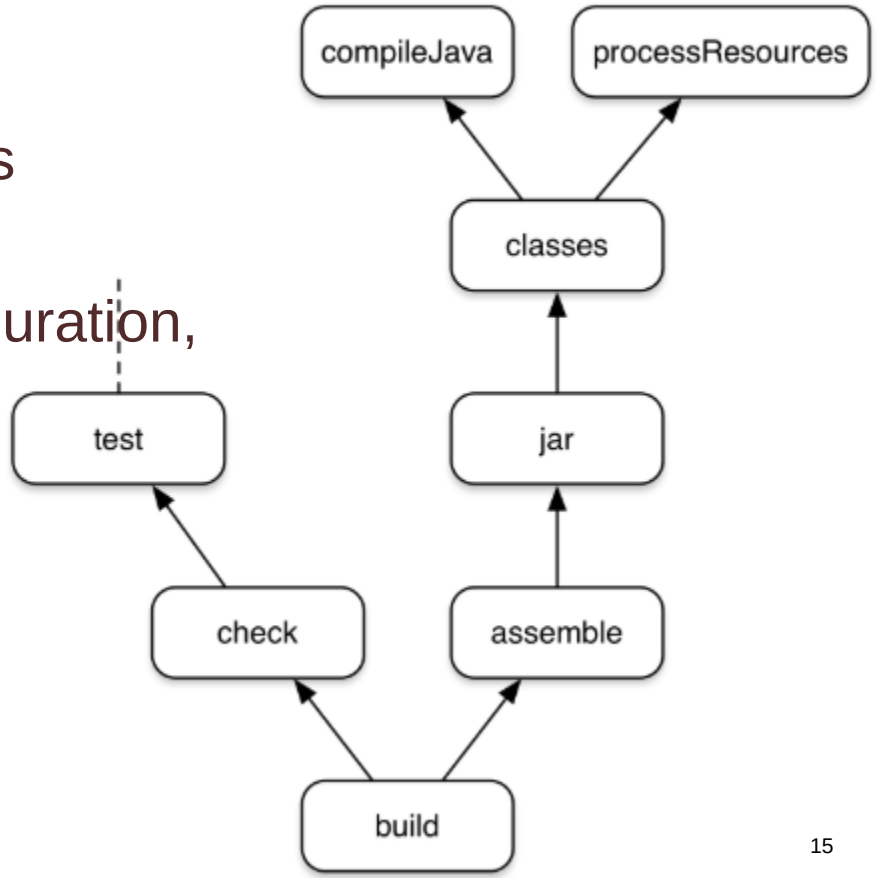
- ***@ConditionalOnBean* and *@ConditionalOnMissingBean* -** We can use these annotations when we want to define conditions based on the **presence or absence of a specific bean**
- ***@ConditionalOnProperty* -** With this annotation, we can make conditions on the **values of properties**
- ***@ConditionalOnResource* -** We can make Spring to use a definition only when a specific **resource is present**
- ***@ConditionalOnWebApplication* and *@ConditionalOnNotWebApplication* -** With these annotations, we can create conditions based on if the current **application is or isn't a web application**
- ***@ConditionalExpression* -** use this annotation in more complex situations
- ***@Conditional* -** even more complex conditions, we can create a class evaluating the **custom condition**

```java
@SpringBootApplication
class DemoApplication {

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


@Configuration
@EnableAutoConfiguration
class Demo DemoFactoryConfig {}
```

# Gradle

- Gradle is an open-source build automation tool that is designed to be flexible enough to build almost any type of software.
- Gradle is a general-purpose build tool
- The core model is based on tasks (Directed Acyclic Graphs (DAGs))
- Gradle has several fixed build phases (Initialization, Configuration, Execution)
- Gradle is extensible in more ways than one
- Build scripts operate against an API (Groovy, Kotlin)

Partial task graph for a standard Java build



# Spring basic annotations

- @Component - generic stereotype for any Spring-managed component indicates that auto scan compoent.
- @Repository - stereotype for persistence layer
- @Service - stereotype for service layer
- @Controller - stereotype for presentation layer (spring-mvc)
- @RestController – stereotype for REST API layer (@Controller + @ResponseBody = @RestController

# Spring basics 1

- “Application context” container

```
@Configuration
@ComponentScan("com.autowire.sample")
public class AppConfig {}
```

- ***@Autowired*** - Once annotation injection is enabled, autowiring can be used on properties, setters, and constructors.

```
@Component("fooFormatter")
public class FooFormatter {

    public String format() {
        return "foo";
    }
}
```

```
@Component
public class FooService {

    @Autowired
    private FooFormatter fooFormatter;

}
```

# Spring basics 2

```
public class FooService {

    private FooFormatter fooFormatter;

    @Autowired
    public void setFooFormatter(FooFormatter fooFormatter) {
            this.fooFormatter = fooFormatter;
    }
}
```

```
public class FooService {

    private FooFormatter fooFormatter;

    @Autowired
    public FooService(FooFormatter fooFormatter) {
        this.fooFormatter = fooFormatter;
    }
}
```

# Spring basics 3

```java
@Component("fooFormatter")
public class FooFormatter implements Formatter {

    public String format() {
        return "foo";
    }
}
```

```java
@Component("barFormatter")
public class BarFormatter implements Formatter {

    public String format() {
        return "bar";
    }
}
```

```java
public class FooService {

    @Autowired
    private Formatter formatter;

}
```

```java
public class FooService {

    @Autowired
    @Qualifier("fooFormatter")
    private Formatter formatter;

}
```

# Rest API basics 1

- REST is acronym for **RE**presentational **S**tate **T**ransfer. It is architectural style for **distributed hypermedia systems** and was first presented by Roy Fielding in 2000

Principles:

- Client-server
- Statless
- Cacheable (client cache)
- Layered system

# Rest API basics 2

| HTTP Verb | CRUD | Entire Collection (e.g. /customers) | Specific Item (e.g. /customers/{id}) |
|---|---|---|---|
| POST | Create | 201 (Created), 'Location' header with link to /customers/{id} containing new ID. | 404 (Not Found), 409 (Conflict) if resource already exists.. |
| GET | Read | 200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists. | 200 (OK), single customer. 404 (Not Found), if ID not found or invalid. |
| PUT | Update/Replace | 405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| PATCH | Update/Modify | 405 (Method Not Allowed), unless you want to modify the collection itself. | 200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid. |
| DELETE | Delete | 405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable. | 200 (OK). 404 (Not Found), if ID not found or invalid. |

# Useful configuration

- spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false
- logging.level.org.hibernate.SQL=DEBUG
- logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

# Spring Data 1.0.4

- Spring Data’s mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.
- It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services. This is an umbrella project which contains many subprojects that are specific to a given database. The projects are developed by working together with many of the companies and developers that are behind these exciting technologies.

# Spring Data – Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

# Spring Data – Main modules

- Spring Data Commons - Core Spring concepts underpinning every Spring Data module.
- Spring Data JDBC - Spring Data repository support for JDBC.
- Spring Data JDBC Ext - Support for database specific extensions to standard JDBC including support for Oracle RAC fast connection failover, AQ JMS support and support for using advanced data types.
- Spring Data JPA - Spring Data repository support for JPA.
- Spring Data KeyValue - Map based repositories and SPIs to easily build a Spring Data module for key-value stores.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.

# Spring Data - JDBC

- Spring Data JDBC, part of the larger Spring Data family, makes it easy to implement JDBC based repositories. This module deals with enhanced support for JDBC based data access layers. It makes it easier to build Spring powered applications that use data access technologies.
- Spring Data JDBC aims at being conceptually easy. In order to achieve this it does NOT offer caching, lazy loading, write behind or many other features of JPA. This makes Spring Data JDBC a simple, limited, opinionated ORM.

# Spring Data – JDBC – Fetures

- CRUD operations for simple aggregates with customizable NamingStrategy.
- Support for @Query annotations.
- Events.
- JavaConfig based repository configuration by introducing @EnableJdbcRepositories.

# Spring Data – JDBC - @Query

```
public interface StuadentRepository extends CrudRepository<Student, Long> {

    List<Student> findByName(String title);

    List<Student> findByNameAndSurname(String name, String surname);

    @Query("SELECT name, surname FROM Student s WHERE upper(name) like '%' ||
upper(:name || '%' )"))
    List<Staudent> fetchStudens(@Param("title)) String category);
}
```

# Spring Data – JDBC – Event

```
public class Student {
 ...
 private long inserted;
 public void timeStamp() {
  if (inserted == 0) {
   inserted = System.currentTimeMillis();
  }
 }
}

@Configuration
@EnableJdbcRepositories
@Import(JdbcConfiguration.class)
public class StudentConfiguration {

 @Bean
 public ApplicationListener < BeforeSaveEvent > timeStampingSaveTime() {

  return event -> {

   Object entity = event.getEntity();

   if (entity instanceof Student) {
    Student category = (Student) entity;
    Student.timeStamp();
   }
  };
 }
}
```

# Spring Data – JPA

- Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.
- Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that’s actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

# Spring Data – JPA – Features

- Sophisticated support to build repositories based on Spring and JPA
- Support for Querydsl predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Pagination support, dynamic query execution, ability to integrate custom data access code
- Validation of @Query annotated queries at bootstrap time
- Support for XML based entity mapping
- JavaConfig based repository configuration by introducing @EnableJpaRepositories.

# Spring Data – Repository

JpaRepository extends PagingAndSortingRepository which in turn extends CrudRepository.

- CrudRepository mainly provides CRUD (**C**reate, **R**ead, **U**pdate, **D**elete ) functions.
- PagingAndSortingRepository provides methods to do pagination and sorting records.
- JpaRepository provides some JPA-related methods such as flushing the persistence context and deleting records in a batch.

# Spring Data – CrudRepository

```
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);
}
```

# Spring Data – PagingAndSortRepository

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

    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}

class SomeClass {

    Sort sort = new Sort(new Sort.Order(Direction.ASC, "name"));

    Pageable pageable = new PageRequest(0, 5, sort);
}
```

# Spring Data - JpaRepository

```
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);
}
```

# Spring Data – Rest

- Spring Data REST is part of the umbrella Spring Data project and makes it easy to build hypermedia-driven REST web services on top of Spring Data repositories.
- Spring Data REST builds on top of Spring Data repositories, analyzes your application’s domain model and exposes hypermedia-driven HTTP resources for aggregates contained in the model.

# Spring Data – Rest – Features

- Exposes a discoverable REST API for your domain model using HAL as media type.
- Exposes collection, item and association resources representing your model.
- Supports pagination via navigational links.
- Allows to dynamically filter collection resources.
- Exposes dedicated search resources for query methods defined in your repositories.
- Allows to hook into the handling of REST requests by handling Spring ApplicationEvents.
- Exposes metadata about the model discovered as ALPS and JSON Schema.
- Allows to define client specific representations through projections.
- Ships a customized variant of the HAL Browser to leverage the exposed metadata.
- Currently supports JPA, MongoDB, Neo4j, Solr, Cassandra, Gemfire.
- Allows advanced customizations of the default resources exposed.

# Spring exception handling 1

- ***@ExceptionHandler*** controller level
- ***HandlerExceptionResolver***
- ***@ControllerAdvice*** since v3
- ***ResponseStatusException*** since v5

# Spring exception handling 2

```
@ControllerAdvice
public class RestResponseEntityExceptionHandler
  extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value
      = { IllegalArgumentException.class, IllegalStateException.class })
    protected ResponseEntity<Object> handleConflict(
      RuntimeException ex, WebRequest request) {
        String bodyOfResponse = "This should be application specific";
        return handleExceptionInternal(ex, bodyOfResponse,
          new HttpHeaders(), HttpStatus.CONFLICT, request);
    }
}
```

# Spring exception handling 3

```
@GetMapping("/student/{id}")
public String getStudent(@PathVariable("id") int id) {
    try {
        return studentService.getStudent(id);
    } catch (StudentNotFoundException ex) {
        throw new ResponseStatusException(
          HttpStatus.NOT_FOUND, "Student Not Found", ex);
    }
}
```

# Swagger REST documentation 1

```
implementation 'io.springfox:springfox-swagger2:2.9.2'

implementation 'io.springfox:springfox-swagger-ui:2.9.2'
```

```
/**
 * The Class SwaggerConfig.
 */
@Configuration
@EnableSwagger2
public class SwaggerConfig {
        @Bean
        public Docket api() {
                //TODO remove framework api documentation leave only project api.
                 //RequestHandlerSelectors.basePackage("org.swedbank.student")
                return new
Docket(DocumentationType.SWAGGER_2).select().apis(RequestHandlerSelectors.any())
                                .paths(PathSelectors.any()).build();
        }
}
```

# Swager REST documentation 2

swagger

Select a spec default

## Api Documentation 1.0

[ Base URL: swedbank-demo.herokuapp.com/ ]

https://swedbank-demo.herokuapp.com/v2/api-docs

Api Documentation

Terms of service

Apache 2.0

**basic-error-controller** Basic Error Controller

**operation-handler** Operation Handler

**person-controller** Person Controller

| | | |
|---|---|---|
| POST | /api/person | addPerson |
| PUT | /api/person | updateArticle |
| GET | /api/person/{pid} | getPersonByPid |
| DELETE | /api/person/{pid} | deleteArticle |

# Data base management tools Flyway

- Add dependency  in gradle build -> implementation 'org.flywaydb:flyway-core‘
- Create this folder structure under src directory src/main/resources/db/migration
- Create file V1.0__init.sql