DR1

Disciplina Regular 1 Fundamentos do Desenvolvimento Java

Graduação em Engenharia de Software - 2020

DR1

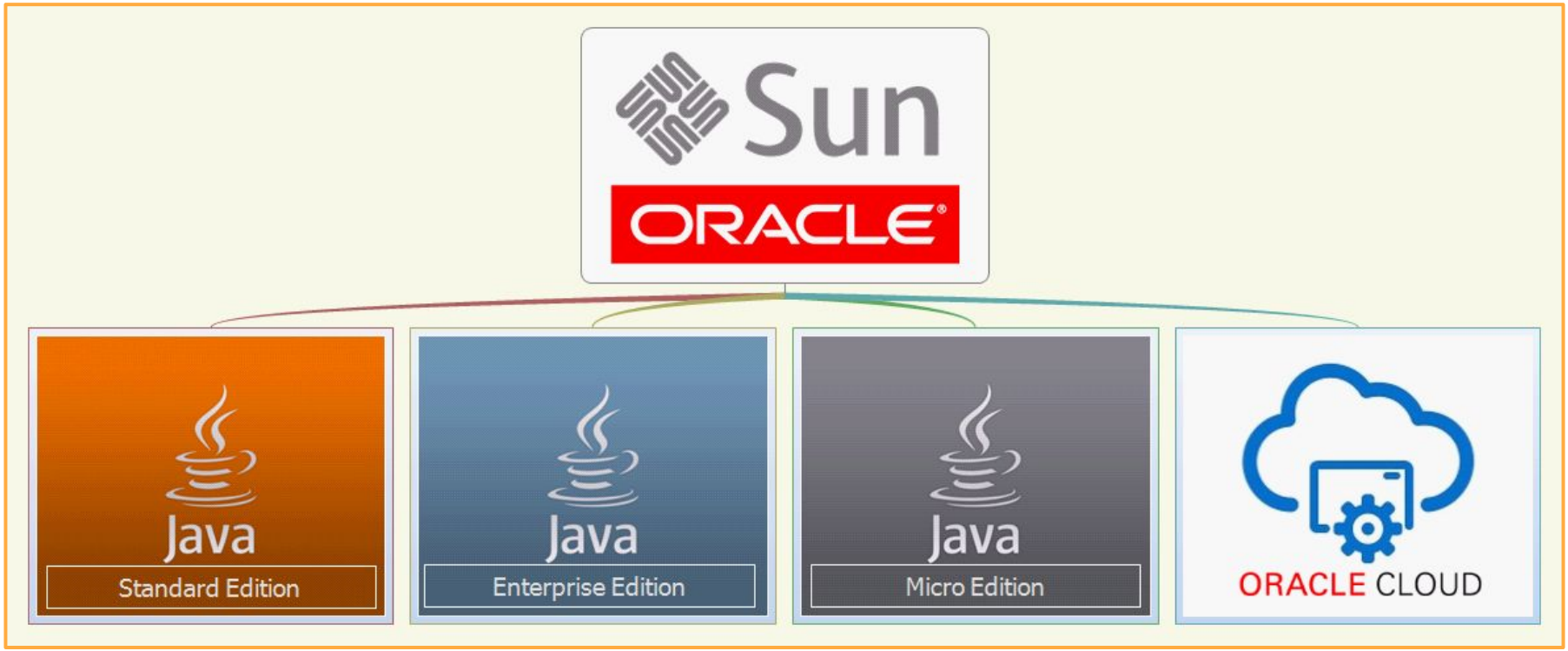
Etapa 9 Aula 2

Próximos Passos - 2º Trimestre 2020

DR1

Java Enterprise Edition

Plataforma Java e suas Edições



Specification	Java EE 6[11]	Java EE 7[3]	Java EE 8[5]
Servlet	3.0	3.1	4.0
JavaServer Pages (*JSP*)	2.2	2.3	2.3
Unified Expression Language (*EL*)	2.2	3.0	3.0
Debugging Support for Other Languages (JSR-45)	1.0	1.0	1.0
JavaServer Pages Standard Tag Library (*JSTL*)	1.2	1.2	1.2
JavaServer Faces (*JSF*)	2.0	2.2	2.3
Java API for RESTful Web Services (*JAX-RS*)	1.1	2.0	2.1
Java API for WebSocket (*WebSocket*)	n/a	1.0	1.1
Java API for JSON Processing (*JSON-P*)	n/a	1.0	1.1
Common Annotations for the Java Platform (JSR-250)	1.1	1.2	1.3
Enterprise JavaBeans (*EJB*)	3.1 Lite	3.2 Lite	3.2
Java Transaction API (*JTA*)	1.1	1.2	1.2
Java Persistence API (*JPA*)	2.0	2.1	2.2
Bean Validation	1.0	1.1	2.0
Managed Beans	1.0	1.0	1.0
Interceptors	1.1	1.2	1.2
Contexts and Dependency Injection for the Java EE Platform	1.0	1.1	2.0
Dependency Injection for Java	1.0	1.0	1.0

Usaremos o Spring Framework no lugar desse recurso

DR1

Sistemas Corporativos

DR1

Requisitos Não Funcionais

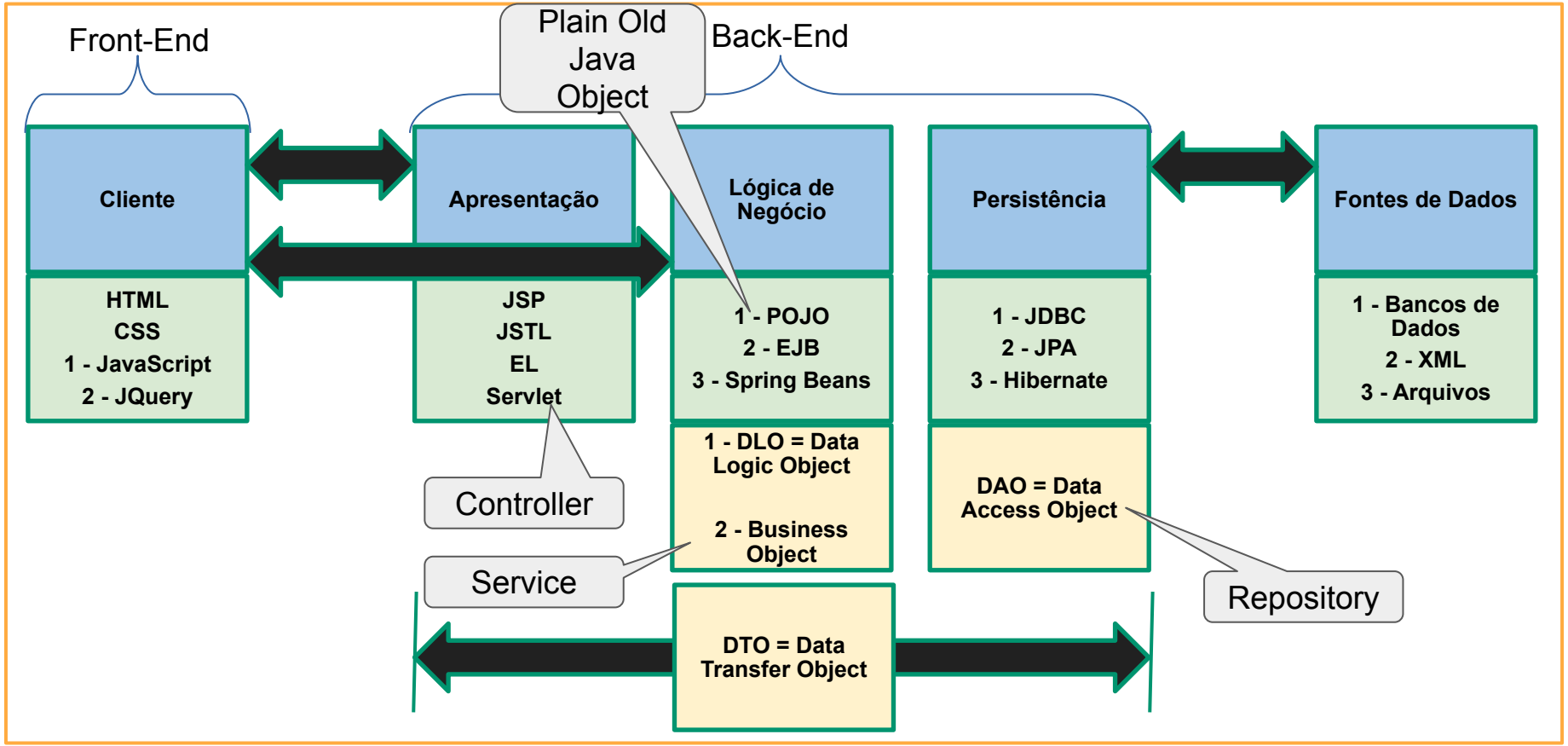
- Disponibilidade.
- Capacidade.
- Extensibilidade.
- Flexibilidade.
- Desempenho.
- Confiabilidade.
- Escalabilidade.
- Segurança.

DR1

Requisitos Não Funcionais

DR1

Modelo de Referência de Arquitetura



DR1

Spring Framework

Spring Framework

- O Spring Framework foi construído com base no princípio de que Java EE deveria ser mais fácil de implementar.

- O que é o Spring?
  - Integrador de Frameworks.
  - Faz a ligação entre as diversas “camadas” (tier) de uma aplicação: Web, Persistência, Transações, Segurança etc.
  - Gerencia as classes de negócio.
  - Facilita a extensibilidade.

Spring Framework

- Spring é uma plataforma de desenvolvimento bem bolada e bem projetada que incorpora boas práticas de design de aplicação de arquitetos experientes e é **voltada para o desenvolvimento de aplicações com classes Java "puras"** – POJO (Plain Old Java Object).

- Foi introduzido pela primeira vez no livro Expert One-on-One J2EE Design and Development de Rod Johnson.

- Um dos problemas mais comuns com o qual os arquitetos e desenvolvedores de aplicação precisam lidar é obter desacoplamento dos componentes – este é o foco principal do Spring.

Spring Framework

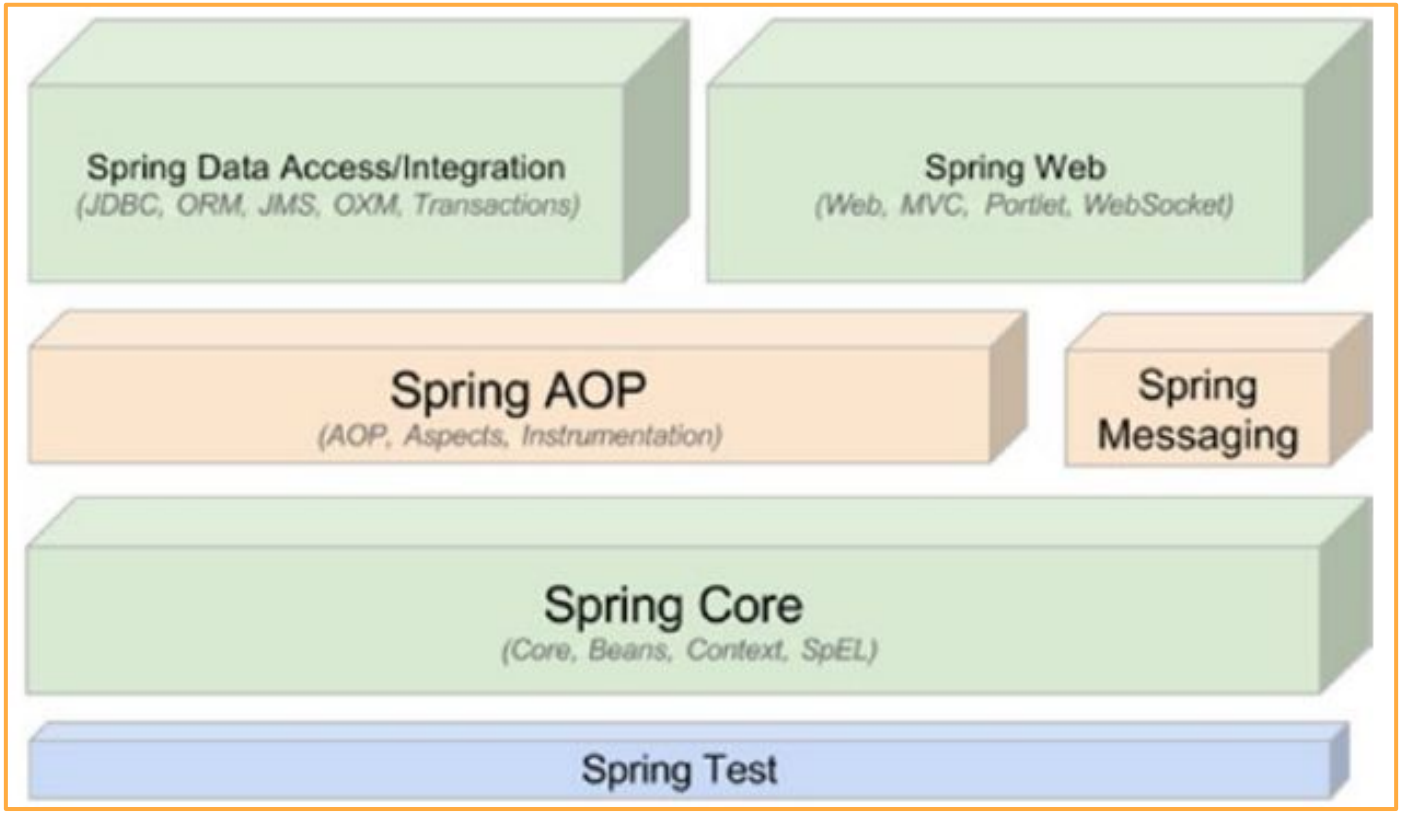
- O Spring cuida todas as camadas de uma aplicação – da Web até a camada de negócios.

- A arquitetura em camadas do Spring Framework permite decidir qual de seus componentes você deseja implementar. Também fornece a flexibilidade de usar o Spring em fases, onde é possível usar um componente do Spring, colocá-lo em funcionamento e então selecionar outro.

- Os módulos do Spring são projetados sobre o módulo principal “Core”, que funciona como container para criar, gerenciar e configurar beans no tempo de execução.

DR1

Spring Framework



DR1

IoC Design Pattern

- Inversion of Control: frameworks executam diversas tarefas independentes da aplicação que está chamando, ou seja, o controle não retorna ao programa chamador – quem controla é o framework.

- IoC é um padrão que ajuda a eliminar o acoplamento entre componentes OOP. Faz isso removendo a dependência de um objeto (X) com outro objeto (Y).

- Há diferentes maneiras de implementar IoC, mas, em termos básicos, é obtido introduzindo-se uma interface entre X e Y para que interajam um com o outro.

- O container do Spring Framework cuida da resolução de dependência e do ciclo de vida dos objetos em runtime.

DR1

Dependency Injection

- Dependency Injection é um padrão de desenvolvimento utilizado quando é necessário manter baixo o nível de acoplamento entre diferentes módulos de um sistema.

- Nesta solução as dependências entre os módulos não são definidas programaticamente, mas sim pela configuração de uma infraestrutura de software responsável por "injetar" em cada componente suas dependências declaradas.

- Dependency Injection se relaciona com o padrão IoC mas não pode ser considerada um sinônimo deste – são conceitos complementares.

DR1

Dependency Injection

```
1  public class ClienteServico {
2
3    private ClienteRepositorio repositorio = new ClienteRepositorioJPA();
4
5    public void salvar(Cliente cliente) {
6      this.repositorio.salvar(cliente);
7    }
8
9    ...
10 }
```

```
1  public class ClienteServico {
2
3    @Autowired
4    private ClienteRepositorio repositorio;
5
6    ...
7  }
```

Dependency Injection

- Para que uma instância do tipo ClienteRepositorio possa ser injetada em algum dos pontos de injeção é preciso que ela se torne um bean Spring.

- Fazemos isso anotando ela com a anotação **@Component** ou com qualquer uma de suas especializações:
  - **@Repository**
  - **@Service**
  - **@Controller**.

- O Spring chama essas quatro anotações de estereótipos, sendo que as três últimas são como anotações “filhas” da anotação @Component.

Dependency Injection

- A anotação @Component é a mais geral e as outras, que são @Repository, @Service e @Controller, são para usos mais específicos em componentes de persistência, serviço e controlador, respectivamente.

- Apesar de termos esses quatro estereótipos, eles não tem especificidades, com exceção da anotação @Repository, que pode adicionar um pequeno comportamento relacionado a tradução das exceções de persistência.

DR1

Spring Boot

Spring Boot

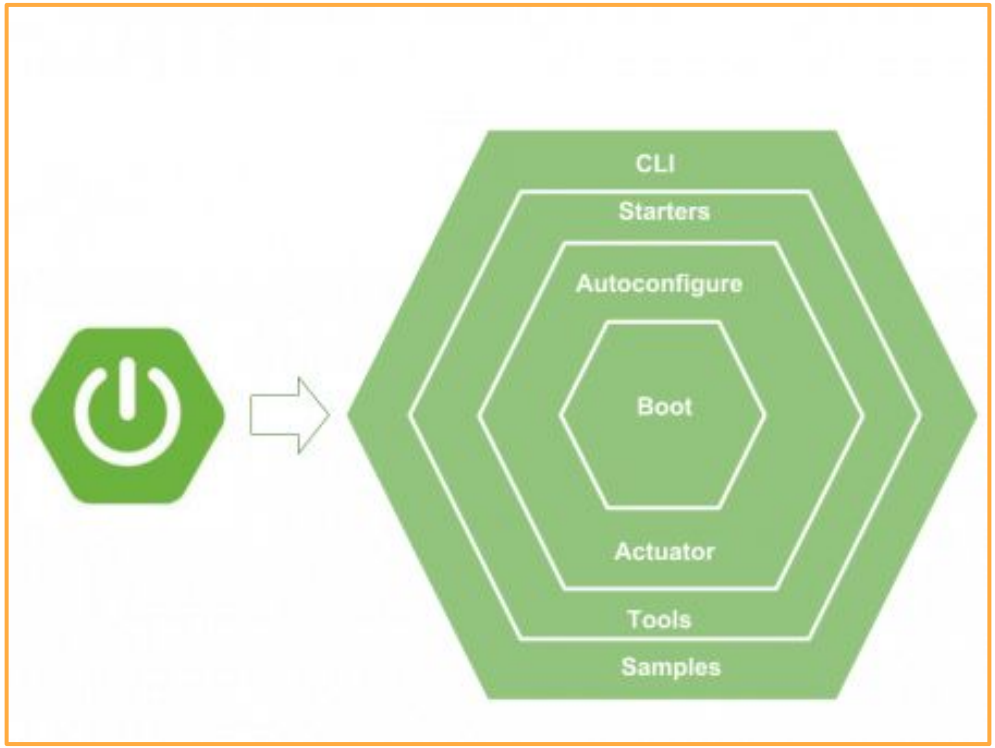
- O Spring Boot **facilita a criação de aplicativos** autônomos baseados em Spring e com grau de produção onde pode-se "simplesmente executar".

- Tem-se uma visão opinativa da plataforma Spring e de bibliotecas de terceiros para que se possa começar com o mínimo de confusão.

- A maioria dos aplicativos Spring Boot precisa de muito pouca configuração do Spring.

DR1

Spring Boot



DR1

Spring Boot

- Criação de aplicativos Spring independentes e autônomos.
- Tomcat, Jetty ou Undertow incorporados diretamente (não há necessidade de implantar arquivos WAR).
- Fornece dependências iniciais para simplificar sua configuração de construção.
- Configuração automática das bibliotecas Spring e de terceiros sempre que possível.
- Fornecimento de recursos prontos para produção, como métricas, verificações de integridade e configuração externalizada.
- Absolutamente nenhuma geração de código e nenhum requisito para configuração em XML.

DR1

Spring Boot

- Requisitos de Sistema do Spring Boot 2.1.8.RELEASE:
  - Java 8 até Java 12.
  - Spring Framework 5.1.9.RELEASE ou superior.
  - Maven 3.3+ ou Gradle 4.4+.
  - Suporta os containeres:
    - Tomcat 9.0 - Jetty 9.4 - Undertow 2.0 .
- O Apache Maven é uma ferramenta de gerenciamento de projetos de software. Com base no conceito de um project object model POM, o Maven pode gerenciar a construção, as dependências e a documentação de um projeto a partir de um repositório central.

Spring Boot

- Starters:
  - Starters do Spring Boot são modelos que contêm uma coleção de todas as dependências transitivas relevantes necessárias para iniciar uma funcionalidade específica.
  - Por exemplo, se você deseja criar um aplicativo Spring WebMVC em uma configuração tradicional, você deverá incluir todas as dependências necessárias.
  - Spring Boot evita conflitos de versão, que resultam em exceções de tempo de execução.
  - Com o Spring boot, para criar um aplicativo MVC, tudo o que você precisa importar é a dependência do spring-boot-starter-web.

DR1

Spring Boot

- Para executar um aplicativo, precisamos usar a anotação @SpringBootApplication.
- Nos bastidores, isso é equivalente a @Configuration, @EnableAutoConfiguration e @ComponentScan juntos.
  - @Configuration = Indica que existem beans a processar.
  - @EnableAutoConfiguration = Varre o classpath e gera um modelo inteligente de configuração para a aplicação.
  - @ComponentScan = Busca beans nos pacotes da aplicação.
  -
- No exemplo a seguir, a execução começa com o método main (). Ele carrega todos os arquivos de configuração, configura-os e inicializar o aplicativo com base nas propriedades do aplicativo no arquivo application.properties que está na pasta / resources.

Spring Boot

DR1

SpringFramework1Application.java

```java
package br.com.infnet;

import org.slf4j.Logger;

@SpringBootApplication
public class SpringFramework1Application implements CommandLineRunner {

    private static Logger LOG = LoggerFactory.getLogger(SpringFramework1Application.class);

    public static void main(String[] args) {

        SpringApplication.run(SpringFramework1Application.class, args);
    }

    @Override
    public void run(String... args) throws Exception {

        LOG.info("Hello Spring Boot");
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> SpringFramework_1 - SpringFramework1Application [Spring Boot App] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (18 de set de 2019 01:13:51)

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.8.RELEASE)

2019-09-18 01:13:51.999  INFO 12087 --- [           main] b.c.infnet.SpringFramework1Application   : Starting SpringFramework1Application on ubuntu18
2019-09-18 01:13:52.001  INFO 12087 --- [           main] b.c.infnet.SpringFramework1Application   : No active profile set, falling back to default pr
2019-09-18 01:13:52.340  INFO 12087 --- [           main] b.c.infnet.SpringFramework1Application   : Started SpringFramework1Application in 0.546 secc
2019-09-18 01:13:52.341  INFO 12087 --- [           main] b.c.infnet.SpringFramework1Application   : Hello Spring Boot
```

DR1

Web Services REST

DR1

Web Service REST

- REST significa Representational State Transfer.
- É um estilo de arquitetura que pode ser usado para projetar Web Services que podem ser consumidos por diversos clientes.
- A idéia principal é que, em vez de usar mecanismos complexos como CORBA, RPC ou SOAP para conectar-se entre máquinas, HTTP simples seja usado para fazer chamadas entre eles.
- No design REST, os recursos são manipulados usando um conjunto comum de verbos mapeados em comandos HTTP.
  - Create → HTTP POST
  - Retrieve → HTTP GET
  - Update → HTTP PUT
  - Delete → HTTP DELETE

Controller REST

- A seguir um exemplo de controlador baseado em REST, implementando uma API – Application Programming Interface:
  - Requisição GET para o path /contacts retorna uma lista de registros.
  - Requisição GET para o path /contacts/1 retorna o registro com ID 1.
  - Requisição POST para o path /contacts com um objeto JSON cria um novo usuário.
  - Requisição PUT para o path /contacts/3 com um objeto JSON atualiza o registro com ID 3.
  - Requisição DELETE para o path /contacts/4 deleta o regsitro com o ID 4.

Controller REST

```
@RestController
@RequestMapping({"/contacts"})
public class ContactController {

   private ContactRepository repository;

   ContactController(ContactRepository contactRepository) {
       this.repository = contactRepository;
   }
   // métodos do CRUD aqui
}
```

```
@GetMapping
public List findAll(){
   return repository.findAll();
}
```

Controller REST

```
@GetMapping(path = {"/{id}"})
public ResponseEntity findById(@PathVariable long id){
   return repository.findById(id)
           .map(record -> ResponseEntity.ok().body(record))
           .orElse(ResponseEntity.notFound().build());
}
```

```
@PostMapping
public Contact create(@RequestBody Contact contact){
   return repository.save(contact);
}
```

Controller REST

```
@PutMapping(value="/{id}")
public ResponseEntity update(@PathVariable("id") long id,
                                    @RequestBody Contact contact) {
   return repository.findById(id)
           .map(record -> {
               record.setName(contact.getName());
               record.setEmail(contact.getEmail());
               record.setPhone(contact.getPhone());
               Contact updated = repository.save(record);
               return ResponseEntity.ok().body(updated);
           }).orElse(ResponseEntity.notFound().build());
}
```

```
@DeleteMapping(path ={"/{id}"})
public ResponseEntity<?> delete(@PathVariable long id) {
   return repository.findById(id)
           .map(record -> {
               repository.deleteById(id);
               return ResponseEntity.ok().build();
           }).orElse(ResponseEntity.notFound().build());
}
```

DR1

Controller REST

- Listar todos os contatos - **@GetMapping**("/contacts")
- Obter um contato específico pelo ID - **@GetMapping**("/contacts/**{id}**")
- Remover um contato pelo ID - **@DeleteMapping**("/contacts/{id}")
- Criar um novo contato - **@PostMapping**("/contacts)
- Atualizar detalhes de um contato - **@PutMapping**("/contacts/{id}")