Springboot

Maven

- Maven é uma ferramenta de gerenciamento e automação de construção de projetos
- Maven realiza a gestão de dependências nos projetos
- Ferramenta indispensável em vários frameworks para JAVA
- Estimula a adoção de boas práticas, utiliza boas práticas para reduzir o esforço de configuração do projeto é a utilização do conceito de programação por convenção (do inglês convention over configuration)
- Assume que o seu usuário fará as coisas da forma como ela preconiza como ideais (estrutura de diretórios padrão, por exemplo), e o livra de ter que declarar algo que se repetirá em todo projeto. o incorporar as práticas aceitas pela comunidade Java como as mais indicadas para projetos Java EE
- Maven padroniza os projetos em que ele é empregado

Gerenciamento de Dependências

- Praticamente todo projeto utiliza bibliotecas para executar algum recurso específico
- Maven administra as bibliotecas a partir do arquivo pom.xml no qual contém a lista de dependências que um projeto utiliza.
- O Maven analisa e tenta localizá-las para disponibilizar para o projeto. Os lugares onde o Maven procura por dependências chamam-se repositórios.
- Na máquina local fica armazenado m2/repository normalmente no home do usuário
- O repositório normalmente é configurado para o endereço http://repo.maven.apache.org/maven2/

Ciclo de Vida de Construção

- Os três ciclos de vida de construção que são nativos da ferramenta são
  - Padrão (default)
  - Clean
  - Site
- O primeiro gerencia a implementação do projeto, o segundo a limpeza do projeto e o terceiro compreende a criação automatizada da documentação do projeto

Ciclo de Vida de Construção - Padrão

- validate: valida se todas as informações obrigatórias do projeto estão preenchidas e são válidas;
- compile: compila o código-fonte;
- test: executa os testes unitários presentes no projeto, desde que o framework de testes utilizado seja compatível com o Maven. Por padrão, é usado o jUnit;
- package: empacota o código compilado no formato definido pela tag packaging do pom.xml;
- integration-test: caso exista, o pacote gerado no estágio anterior é instalado em um ambiente de teste de integração;
- verify: executa checagens para verificar se o pacote é válido e se atende aos critérios de qualidade;
- install: instala o pacote no repositório local, de modo que outros projetos locais possam utilizá-lo como dependência;
- deploy: instala o pacote em repositórios externos, efetivamente disponibilizando-o em ambientes remotos.

Plugins

- As funcionalidades do Maven são propiciadas através de plugins
- Uma invocação a um plugin diferente é executada dependendo do comando (clean, install, site, etc)
- Na página http://maven.apache.org/plugins/, temos a listagem dos plugins oficiais existentes.
- Maven permite a adição transparente de funcionalidades conforme a evolução do projeto
- Permite que qualquer um crie um plugin para atender os requisitos que não estejam cobertos pelos plugins existentes.
- Por exemplo:
  - O Maven publique a documentação gerada automaticamente em um servidor web da empresa, de acordo com certos padrões;
  - O Maven automatize procedimentos de projetos legados, substituindo trabalho manual. Como pode ser observado, as possibilidades são infinitas.

Maven

- Pode ser instalado e utilizado por linha de comando
- Pode ser utilizando em conjunto com a maioria das IDEs
- Criando um projeto

SPRING BOOT

Spring Framework

- Objetivo de aumentar a produtividade na escrita de aplicações corporativas;
- Spring Framework Java criado com o objetivo de facilitar o desenvolvimento de aplicações
- Caracteriza-se pela Inversão de Controle e Injeção de Dependências
- Possui módulos para persistência de dados, integração, segurança, testes, desenvolvimento web
- Não precisa de um servidor de aplicação.
- Faz uso apenas da JVM;
- A plataforma J2EE e os EJBs propiciam a implementação de comportamentos que não eram necessários. Esse diferencial do Spring torna a arquitetura mais leve, fácil de compreender, manter e evoluir;
- A inversão de controle e injeção de dependência, fornecendo um container, que representa o núcleo do framework e que é responsável por criar e gerenciar os componentes da aplicação, os quais são comumente chamados de beans.

Spring Framework

- Spring Framework ainda é alvo de críticas a respeito do tempo necessário para se iniciar o desenvolvimento de novos projetos.
- A principal crítica feita ao Spring é sobre o modo como se configura o seu container de injeção de dependências e inversão de controle usando arquivos de configuração.
- Estes artefatos, conforme aumentam de tamanho, se tornam cada vez mais difíceis de serem mantidos, muitas vezes se transformando em um gargalo para a equipe de desenvolvimento.
- A complexidade na gestão de dependências é outro ponto que desfavorece o framework.
- Conforme os projetos precisam interagir com outras bibliotecas e frameworks como, por exemplo, bibliotecas de persistência, mensageria, frameworks de segurança e tantos outros, garantir que todas as bibliotecas estejam presentes no classpath da aplicação acaba se tornando um pesadelo.

Spring Framework

- Desta forma, foi introduzido no Spring 4.0 um novo projeto com o objetivo de responder a estas críticas e também mudar bastante a percepção acerca do desenvolvimento de aplicações para a plataforma Java EE. Este projeto é o Spring Boot.
- O projeto Spring Boot resolve estas questões e ainda nos apresenta um novo modelo de desenvolvimento, mais simples e direto, sem propor novas soluções para problemas já resolvidos, mas sim alavancando as tecnologias existentes presentes no ecossistema Spring de modo a aumentar significativamente a produtividade do desenvolvedor.

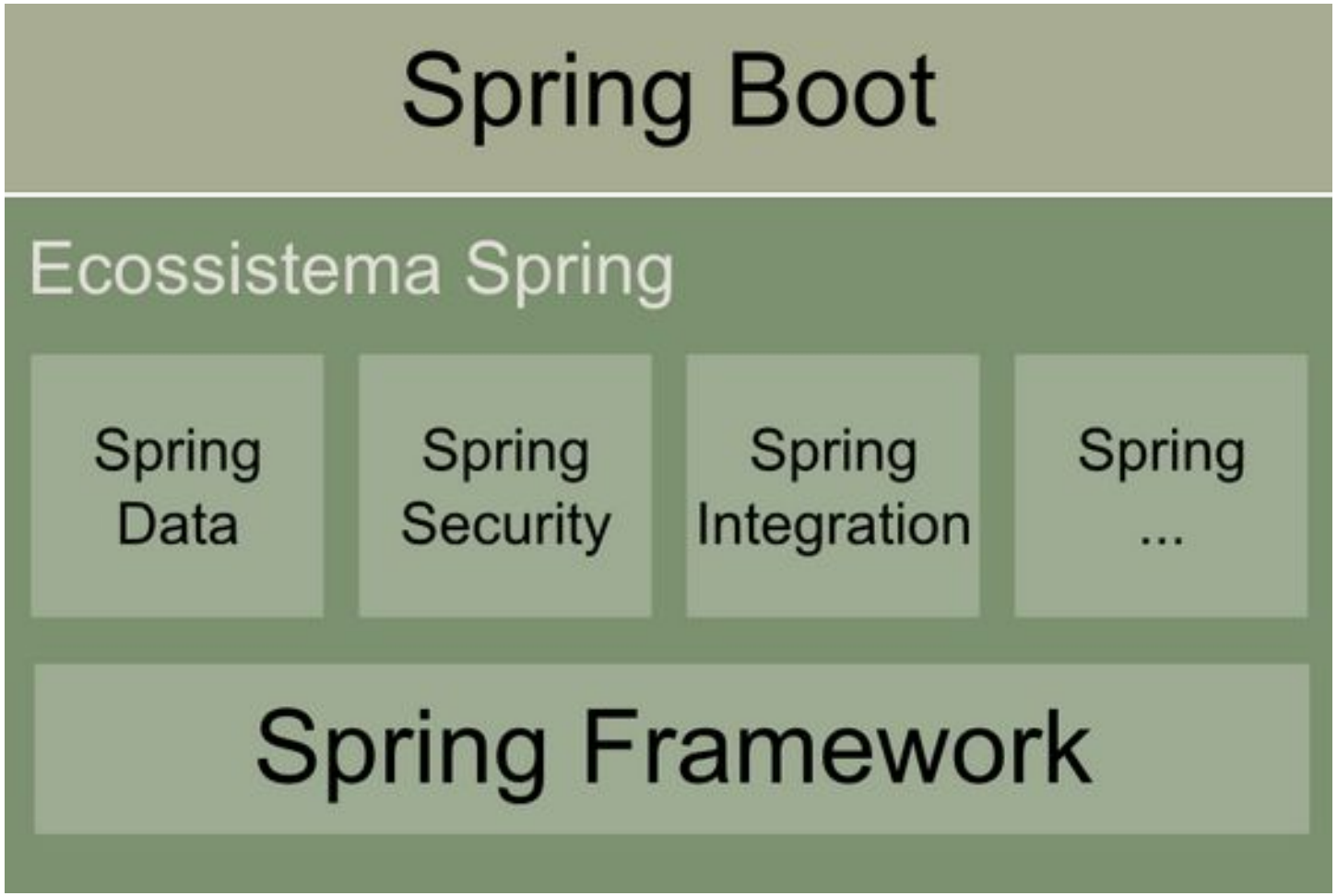
Conceitos fundamentais

- Inversão de Controle (IoC – Inversion of Control)
  - Delega a outro elemento o controle quanto a criação do objeto, chamada de métodos, entre outros
  - Controle da execução de alguns comportamentos passa a ser gerenciado pelo próprio elemento e não pelo programador.
  - No Spring, o elemento que controla e gerencia os comportamentos é o container.
- Injeção de Dependência (DI – Dependency Injection)
  - Classe deixa de se preocupar em como resolver as suas dependências
  - Mantem o foco apenas no uso dos recursos das dependências
  - Característica mais marcante é não necessitar utilizar o “new” para criar objetos por ele gerenciados.

Spring Boot

- É um framework baseado no Spring
- Também conhecido com micro framework
- Objetivo não é trazer novas soluções para problemas que já foram resolvidos, mas sim reaproveitar estas tecnologias e aumentar a produtividade do desenvolvedor.
- É uma excelente ferramenta que podemos adotar na escrita de aplicações que fazem uso da arquitetura de micro serviços.

Spring Boot


Spring Boot
Ecossistema Spring
Spring Data
Spring Security
Spring Integration
Spring ...
Spring Framework

Princípios do Spring Boot

- Prover uma experiência de início de projeto (getting started experience) extremamente rápida e direta;
- Apresentar uma visão bastante opinativa (opinionated) sobre o modo como devemos configurar nossos projetos Spring, mas ao mesmo tempo flexível o suficiente para que possa ser facilmente substituída de acordo com os requisitos do projeto;
- Fornecer uma série de requisitos não funcionais já pré-configurados para o desenvolvedor como, por exemplo, métricas, segurança, acesso a base de dados, servidor de aplicações/servlet embarcado, etc.;
- Não prover nenhuma geração de código e minimizar a zero a necessidade de arquivos XML.

Visão Opinativa

- Conceito de convenção sobre configuração, porém levemente modificado
- É o grande motor por trás do ganho de produtividade do Spring Boot
- Frameworks como Ruby on Rails e Grails já o aplicam há bastante tempo.
- Dado que a maior parte das configurações que o desenvolvedor precisa escrever no início de um projeto são sempre as mesmas, por que já não iniciar um novo projeto com todas estas configurações já definidas?
- Não temos uma imposição aqui, mas sim sugestões
- Se seu projeto requer um aspecto diferente daquele definido pelas convenções do framework, o programador precisa alterar apenas aqueles locais nos quais a customização se aplica

Spring Boot
Spring Scripts

- Uma mudança interessante a se observar é a aparente ausência do Servlet container.
- Ele está presente, porém é executado de uma forma ligeiramente diferente.
- Ao invés de empacotarmos nosso projeto em um arquivo WAR para ser implantado no servidor, agora o próprio Spring Boot se encarrega de embutir o servidor entre as dependências do projeto, iniciá-lo e, de uma forma completamente transparente, implantar nossa aplicação neste.
- Spring Scripts são uma solução interessante quando precisamos escrever algum serviço REST simples e direto como, por exemplo, aqueles que consistem em meros CRUDs a bases de dados pré-existentes.
- Muitos recomendam a utilização da linguagem Goovy, que é uma linguagem projetada para roda na JVM e é cada vez mais importância da esta linguagem dentro do ecossistema Spring.
  - Não faz parte do escopo da disciplinas.

Serviços Rest

- O modelo REST (Representational State Transfer) representa uma possibilidade para a criação de web services
- Utiliza as semântica dos métodos HTTP (GET, POST, PUT e DELETE)
- Pacotes leves dos pacotes de dados transmitidos na rede e na simplicidade, fazendo desnecessária a criação de camadas intermediárias para encapsular os dados.

Spring Boot
Micro serviços

- Desde 2012, um termo que tem ganhado cada vez mais atenção entre os arquitetos é “micro serviço” (ou micro services).
- Quando lançada a versão 4.0 do Spring, foi feita uma referência a esse estilo arquitetural, visto pela equipe responsável pelo desenvolvimento do framework como uma das principais tendências do mercado.
- Mas, afinal, o que é um micro serviço?

Spring Boot
Micro serviços

- O problema que a arquitetura baseada em micro serviços busca resolver é a eterna questão da componentização.
- Sendo assim, para que se possa definir o que é um micro serviço, primeiro precisa-se entender o que é um componente.
- “uma unidade de software que pode ser aprimorada e substituída de forma independente”1.

1 - James Lewis e Martin Fowler, no artigo chamado Microservices.

Projeto Spring Boot com Maven

- Facilitar a configuração básica foi criado o site:
  - SPRING INITIALIZR
  - https://start.spring.io/
- Assistente no qual o desenvolvedor seleciona quais tecnologias deseja incluir em seu projeto, como Spring Data, Spring Security e tantas outras, preenche os campos fornecidos e, uma vez finalizado o preenchimento do formulário, simplesmente clica sobre o botão Generate Project.
- Feito isso, será gerado um arquivo zip, a ser descompactado no diretório de preferência do usuário e na sequencia importado pela IDE
- A primeira observação é no arquivo pom.xml quanto ao gerenciamento de dependências.
- A modularização das diversas tecnologias que compõem o ecossistema Spring em starter POMs.
- Tratam-se de arquivos POM previamente configurados pela equipe de desenvolvimento do framework que já contêm todas as configurações necessárias para que o programador possa usar imediatamente
  - <parent>
  - <groupId>org.springframework.boot</groupId>
  - <artifactId>spring-boot-starter-parent</artifactId>
  - </parent>

Projeto Spring Boot com Maven

- Com esta configuração o projeto já está pronto para a aplicação desktop
- Acrescentando mais o pacote abaixo está pronto para configurar serviços REST e projeto Web
- <dependencies>
  - <dependency>
    - <groupId>org.springframework.boot</groupId>
    - <artifactId>spring-boot-starter-web</artifactId>
  - </dependency>
- </dependencies>
- Será necessário outras dependências conforme o objetivo da aplicação, como bibliotecas de banco de dados (MySql, Postgre, Oracle, etc), entre outras que será feito o uso conforme sua necessidade e deverão ser configuradas no pom
- As configurações e versão normalmente são baixadas do repositório do maven
  - https://mvnrepository.com/

Guia das annotations do Spring

- @Configuration - É uma annotation que indica que determinada classe possui métodos que expõe novos beans.
- @Controller - Associada com classes que possuem métodos que processam requests numa aplicação web.
- @Repository - Associada com classes que isolam o acesso aos dados da sua aplicação. Comumente associada a DAO's.
- @Service - Associada com classes que representam a ideia do Service do Domain Driven Design. Para ficar menos teórico pense em classes que representam algum fluxo de negócio da sua aplicação. Por exemplo, um fluxo de finalização de compra envolve atualizar manipular o carrinho, enviar email, processar pagamento etc. Este é o típico código que temos dificuldade de saber onde vamos colocar, em geral ele pode ficar num Service.

Posted on July 13, 2016 by Alberto Souza.
https://domineospring.wordpress.com/2016/07/13/guia-das-annotations-do-spring/

Guia das annotations do Spring

- @Component - A annotation básica que indica que uma classe vai ser gerenciada pelo container do Spring. Todas as annotations descritas acima são, na verdade, derivadas de @Component. A ideia é justamente passar mais semântica.
- @ComponentScan - Em geral você a usa em classes de configuração (@Configuration) indicando quais pacotes ou classes devem ser scaneadas pelo Spring para que essa configuração funcione.
- @Bean - Anotação utilizada em cima dos métodos de uma classe, geralmente marcada com @Configuration, indicando que o Spring deve invocar aquele método e gerenciar o objeto retornado por ele. Quando digo gerenciar é que agora este objeto pode ser injetado em qualquer ponto da sua aplicação.
- @Autowired - Anotação utilizada para marcar o ponto de injeção na sua classe. Você pode colocar ela sobre atributos ou sobre o seu construtor com argumentos.

Posted on July 13, 2016 by Alberto Souza.
https://domineospring.wordpress.com/2016/07/13/guia-das-annotations-do-spring/

Guia das annotations do Spring

- @Scope - Annotation utilizada para marcar o tempo de vida de um objeto gerenciado pelo container. Pode ser utilizada em classes anotadas com @Component, ou alguma de suas derivações. Além disso também pode usada em métodos anotados com @Bean. Quando você não utiliza nenhuma, o escopo default do objeto é o de aplicação, o que significa que vai existir apenas uma instância dele durante a execução do programa. Você alterar isso, anotando o local e usando alguma das constantes que existem na classe ConfigurableBeanFactory ou WebApplicationContext.
- @RequestMapping - Geralmente utilizada em cima dos métodos de uma classe anotada com @Controller. Serve para você colocar os endereços da sua aplicação que, quando acessados por algum cliente, deverão ser direcionados para o determinado método.
- @ResponseBody -Utilizada em métodos anotados com @RequestMapping para indicar que o retorno do método deve ser automaticamente escrito na resposta para o cliente. Muito comum quando queremos retornar JSON ou XML em função de algum objeto da aplicação.

Posted on July 13, 2016 by Alberto Souza.
https://domineospring.wordpress.com/2016/07/13/guia-das-annotations-do-spring/

Guia das annotations do Spring

- @Primary - Caso você tenha dois métodos anotados com @Bean e com ambos retornando o mesmo tipo de objeto, como o Spring vai saber qual dos dois injetar por default em algum ponto da sua aplicação? É para isso que serve a annotation @Primary. Indica qual é a opção padrão de injeção. Caso você não queira usar a padrão, pode recorrer a annotation @Qualifier.
- @Profile - Indica em qual profile tal bean deve ser carregado. Muito comum quando você tem classes que só devem ser carregadas em ambiente de dev ou de produção. Essa eu utilizo bastante e acho uma ideia simples, mas super relevante dentro do Spring.
- @SpringBootApplication - Para quem usa Spring Boot, essa é uma das primeiras que você. Ela engloba a @Component, @ComponentScan e mais uma chamada @EnableAutoConfiguration, que é utilizada pelo Spring Boot para tentar advinhar as configurações necessárias para rodar o seu projeto. Bom, acho que é isso aí. Adoro escrever posts que foram solicitados diretamente pela galera que acompanha o meu trabalho, é uma motivação bem grande.

Posted on July 13, 2016 by Alberto Souza.
https://domineospring.wordpress.com/2016/07/13/guia-das-annotations-do-spring/

Guia das annotations do Spring

- @EnableAsync - Essa aqui não é tão comum, mas muitas vezes você precisa ações no sistema em background(outra thread). Essa annotation deve ser colocada em alguma classe marcada com @Configuration, para que o Spring habilite o suporte a execução assíncrona.
- @Async - Uma vez que seu projeto habilitou o uso de execução de métodos assíncronos com a @EnableAsync, você pode marcar qualquer método de um bean gerenciado do projeto com essa annotation. Quando tal método for invocado, o Spring vai garantir que a execução dele será em outra thread.

Posted on July 13, 2016 by Alberto Souza.
https://domineospring.wordpress.com/2016/07/13/guia-das-annotations-do-spring/

Spring Data JPA

JPA

- O sistema é desenvolvido na grande maioria utilizando somente objetos, sem referências específicas de cada banco de dados.
- Oferece uma linguagem semelhante ao SQL chamada JPA Query, que possibilita escrever consultas e instruções de persistência.
- Principais Características
  - Persistência de POJOs (Plain Old Java Objects)
  - Permite elaboração de domínios ricos (herança, polimorfismo, navegabilidade, multiplicidade, associações e composições, etc)
  - Acessos sob demanda (Lazy)
  - Suporte à annotations - sem arquivos de mapeamento
  - Permite conectar frameworks de persistência (ex: Hibernate)

Entidade

- Entidade não é uma nova abordagem adotada no JPA. Pelo contrário, esse termo já vem sendo utilizado há muito tempo e é utilizado em diversas linguagens de programação.
- Uma entidade continua sendo essencialmente um nome ou um grupo de estados relacionados compondo uma unidade simples, muitas vezes representada - como uma tabela.
- Uma entidade pode participar de relacionamentos com uma ou muitas outras entidades e, esses relacionamentos, podem ser de diversos tipos, conforme os conceitos de multiplicidade e navegabiliade definidos no paradigma de Orientação a Objetos.
- Numa aplicação orientada a objetos (utilizando ou não JPA), qualquer objeto definido pode ser uma entidade.

Entidade - Características

- Persistenciabilidade (persistabiliity):
  - Esta é a mais importante característica para considerar um objeto como uma Entidade: ele deve ser persistido/armazenado ou o ciclo de vida é apenas volátil? Além disso, o objeto deve ser capaz de ser persistido, ou seja, o seu estado (conteúdo de seus atributos) devem ser capazes de serem persistidos em um banco de dados, por exemplo.
- Identidade (identity):
  - Um objeto deve ser identificado no sistema, seja por um código ou por um estado. Quando um objeto é persistido em um banco de dados ele deve possuir uma forma de identificação, chamada de identifier. Este identificador é a chave (key) que identifica unicamente uma instância de um objeto no sistema, ou seja, com ela é possível recuperá-la.

Entidade - Características

- Transacionalidade (transactionality):
  - é a possibilidade de realizar transações (inserção, alteração, consulta e deleção) destes objetos em qualquer contexto.
- Granularidade (granularity):
  - capacidade de detalhar um objeto em uma série de atributos (um tipo primitivo nunca será uma entidade), a fim de que cada objeto representará uma linha de uma tabela e cada atributo, uma coluna, por exemplo. Um objeto muito grande, pode ser detalhado, talvez, através de outros objetos, e o JPA oferece suporte à isto.

Metadados das Entidades

- Uma entidade, entretanto, possui um metadados. Ou seja, cada entidade no sistema deve ter informações sobre os dados que estão armazenando, com o objetivo de guiar o JPA no mapeamento adequado dos Objetos vs. Banco de Dados Relacional.
- As annotations vem para simplificar a implementação, reduzindo a quantidade de arquivos de configuração e facilitando a visualização e leitura do código.

Configuração Inicial

- A configuração inicial do Spring Data JPA consiste basicamente em:
- Spring Boot possibilita configurar os atributos de conexão no application.propertiesarquivo.
- O arquivo de de configuração do recurso deve ser criado em src/main/resources/application.properties, e pode ter uma configuração como:
  - spring.jpa.hibernate.ddl-auto=update
  - spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_example
  - spring.datasource.username=springuser
  - spring.datasource.password=ThePassword
  - spring.datasource.driver-class-name =com.mysql.jdbc.Driver

Construindo uma entidade

- Qualquer classe Java (não abstrata) pode ser uma Entidade. Porém, somente as classes que geram objetos com as características citadas acima realmente fazem sentidos serem implementadas como Entidades.
- Considerando este requisito fundamental, basta utilizar uma série de annotations para construir uma Entidade a partir de uma classe Java.

```
@Entity
 public class Employee {
        @Id
        private String name;
        private long salary;


        public Employee() {}
        public Employee(int id) { this.id = id; }
    public gets();
    public sets();
}
```

Annotations do JPA

- Mapeamento de uma Tabela
- Annotation @Entity, informa ao JPA que uma determinada classe poderá ser manipulada por uma EntityManager,
  - Poderá ser persistida em um banco de dados definido pela PersistenceUnit.
- Por padrão, o JPA define o nome da tabela que será gerada no banco de dados em função do nome da classe.
  - Exemplo: se a classe se chamar Usuario, a tabela ser a "usuario". Muitas vezes, a base de dados já existe, ou ainda, o DBA prefere utilizar nomes personalizados para as tabelas, o que desconfiguraria o nome dos objetos em java, se o desenvolvedor optasse em seguir o padrão do DBA.

```
@Entity
@Table(name="TB_PESSOA")
public class Pessoa {
(...)
}
```

```
@Entity
@Table(name="TB_PESSOA",uniqueConstraints={@Un
iqueConstraint(columnNames="NAME"),
@UniqueConstraint(columnNames="CPF")})
public class Pessoa {
(...)
}
```

Annotations do JPA

- Annotation @Column
- É possível mapear o nome da coluna no banco de dados com um nome diferente do nome do atributo

```
(...)
@Column(name="str_nome"
private String nomeDoFuncionario
(...)
```

- Definição da coluna
- Permite que o desenvolvedor faça a definição da coluna de forma manual, de acordo com seu banco de dados. Isso deve ser evitado, visto que o JPA vem para ser um framework independetemente do banco de dados utilizado.

```
(...)
@Column(name="START_DATE", columnDefinition="DATE
DEFAULT SYSDATE")
private java.sql.Date startDate;
(...)
```

- Define a coluna no banco de dados com o tipo DATE e que o valor default será através do SYSDATE.

Annotations do JPA

- length / Tamanho
- Define o tamanho da coluna. O número de caracteres ou dígitos que serão utilizados para armazenar o valor.

```
(...)
@Column(length=10)
private int codigo;

@Column(length=60)
private String nome;
(...)
```

Annotations do JPA

- nullable / pode ser nulo?
- Informa se o valor pode (ou não) ser nulo (vazio). Caso nullable=true, então o valor pode ser nulo. Caso contrário, é de preenchimento obrigatório.

```
(...)
@Column(nullable=true)
private int idade;

@Column(length=60, nullable=false)
private String nome;
(...)
```

Annotations do JPA

- Tipos Enum
- É comum que seja armazenado campos do tipo "Status", "Fase do projeto", etc, que utilizam em seu conteúdo um tipo de Enum.
- Poderia haver uma tabela no banco de dados com este tipo, porém, muitas vezes não se faz necessário e a melhor estratégia é realmente utilizar uma coluna simples contendo esta informação.
- Neste segundo caso, utilizamos o recurso do java chamado Enumeration.

```
public enum UsuarioStatus {
ATIVO, AGUARDANDO_CONFIRMACAO, INATIVO
}
```

Annotations do JPA

- Esta enum será utilizada no atributo status da entidade Usuario. A única preocupação que se deve ter é como esse enum será persistido na base de dados:
  - ORDINAL: será um numero sequencial, ou seja ATIVO = 0 e INATIVO = 2; ou,
  - String: será armazenado como String "ATIVO", "AGUARDANDO_CONFIRMACAO" e "INATIVO".

```
@Entity
public class Usuario {
    @Id
    private int id;

    @Enumerated(EnumType.STRING)
    private UsuarioStatus status;
}
```

Visão geral de mapeamento com JPA

- Relacionamento com valores "Únicos"
- Um relacionamento onde um dos lados (roles) possui a cardinalidade (multiplicidade) 1(um) é chamado de relacionamento com valor único (single-valued association). Estes relacionamentos podem ser:
  - Muitos para um (many-to-one); e,
  - Um para um (one-to-one).
- Muitos para um
- O relacionamento Muitos para Um (many-to-one) indica que, dependendo da direcionalidade, em A teremos uma lista de B e em B, pode-se ter uma referência única para A.
- O JPA estabelece este relacionamento utilizando a annotation @ManyToOne e, de acordo com a direcionalidade, também a @OneToMany.
- A annotation @ManyToOne deverá ser utilizada na Entidade que terá apenas uma instância da outra relacionada a ela.

```
public class Aluno {
(...)
        @ManyToOne
        private Turma turma;
(...)
}
```

Visão geral de mapeamento com JPA

- Utilizando colunas de Junção
- É possível informar ao JPA qual é o nome da coluna que será gerada para armazenar o código (ID) da outra Entidade. No caso de não informar, por exemplo, seria criado uma coluna turma_id dentro da tabela Aluno.

```
public class Aluno {
(...)
        @OneToOne
        @JoinColumn(name="armario_id")
        private Armario armario;
(...)
}

public class Armario {
(...)
        @OneToOne
        @JoinColumn(mappedBy="armario")
        private Aluno aluno;
(...)
}
```

```
public class Aluno {
(...)
        @ManyToOne
        @JoinColumn(name="cod_turma")
        private Turma turma;
(...)
}
```

```
public class Aluno {
(...)
        @OneToOne
        @JoinColumn(name="armario_id")
        private Armario armario;
(...)
}
```

Visão geral de mapeamento com JPA

- Utilizando colunas de Junção

```
public class Aluno {
(...)
        @OneToOne
        @JoinColumn(name="armario_id")
        private Armario armario;
(...)
}
```

```
public class Armario {
(...)
        @OneToOne
        @JoinColumn(mappedBy="armario")
        private Aluno aluno;
(...)
}
```

Visão geral de mapeamento com JPA

- Associação Muitos para Muitos
- Já é conhecido que um relacionamento muitos-para-muitos (many-to-many) necessita de tabelas entidades no banco de dados para fazer as associações de duas entidades que podem ter muitas instâncias da outra, ou vice-versa.
- O JPA oferece este tipo de mapeamento e gera automaticamente esta entidade responsável pela junção das entidades Origem e Destino.
- A annotation utilizada é a @ManyToMany. Esta annotation deverá ser colocada primeiramente na entidade "dona" do relacionamento e, em seguida - e se for necessário -, na entidade que faz o relacionamento invesrso, definindo o parâmetro mappedBy.

```
@Entity
public class Aluno {
(...)
    @ManyToMany
    private List<Curso> cursos;
(...)
}
```

```
@Entity
public class Curso {
(...)
    @ManyToMany(mappedBy="cursos")
    private List<Aluno> alunos;
(...)
}
```

Visão geral de mapeamento com JPA

- Ajustando a Tabela de Junção
- Um relacionamento muitos para muitos resulta numa terceira tabela responsável pelo relacionamento das chaves primárias das entidades Origem e Destino.
- Na tabela resultante deste relacionamento serão encontrados somente duas colunas: KEY de origem e KEY de destino.
- No caso de desejar fazer ajustes nesta tabela resultante, é possível utilizando a annotation @JoinTable, na classe "dona" do relacionamento.

```
@Entity
public class Aluno{
(...)
    @ManyToMany
    @JoinTable(name="alunos_curso",
       joinColumns=@JoinColumn(name="ALUNO_ID"),
       inverseJoinColumns=@JoinColumn(name="CURSO_ID"))
    private List<Curso> cursos;
(...)
```

Repositório

- A interface central na abstração do repositório Spring Data é Repository.
- É necessária a classe de domínio, notada como @Entity para gerenciar, bem como o tipo de ID da classe de domínio como argumentos de tipo.
- Essa interface atua principalmente como uma interface de marcador para capturar os tipos com os quais trabalhar.
- É estendidada da CrudRepository interface, no qual fornece funcionalidade CRUD para a classe de entidade que está sendo gerenciada.

Repositório

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {

  <S extends T> S save(S entity);      //Salva a entidade fornecida.

  Optional<T> findById(ID primaryKey);  //Retorna a entidade identificada por um determinado ID.

  Iterable<T> findAll();     //Retorna todas as entidades.

  long count();   //Retorna o número de entidades.

  void delete(T entity);  //Exclui a entidade fornecida.

  boolean existsById(ID primaryKey);   //Indica se existe uma entidade com o ID fornecido.

  // … more functionality omitted.
}
```

Consultas

- Uma consulta pode utilizar o critério JPA API a partir da construção do método, que será traduzido na seguinte consulta: select u from User u where u.emailAddress = ?1 and u.lastname = ?2. O Spring Data JPA faz uma verificação de propriedade e percorre as propriedades aninhadas. Veja a construção do método:

```
interface pública UserRepository extends Repository <User, Long> {

  List <User> findByEmailAddressAndLastname (String emailAddress, String lastname);
}
```

NamedQuery

- A anotação @NamedQuery devem ser definidas na linguagem de consulta JPA. Permite definir a consulta em SQL nativo, perdendo a independência da plataforma do banco de dados.

```
@Entity
@NamedQuery(name = "User.findByEmailAddress",
  query = "select u from User u where u.emailAddress = ?1")
public class User {

}
```

Query

- Outra alternativa é vincular ao método Java que as executa, pode vinculá-las diretamente usando a @Query do Spring Data JPA diretamente na interface em vez de anotá-las na classe de domínio.
- Isso libera a classe de domínio de informações específicas de persistência e coloca a consulta na interface do repositório.

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query("select u from User u where u.emailAddress = ?1")
  User findByEmailAddress(String emailAddress);
}
```

Consultas nativas

- A @Queryanotação permite a execução de consultas nativas definindo o parâmetro nativeQuery como verdadeiro

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query(value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1", nativeQuery = true)
  User findByEmailAddress(String emailAddress);
}
```

Repositórios

- https://github.com/edson-lessa-jr/primeiro-spring-exemplo
- https://github.com/edson-lessa-jr/micro-service-exemplo-spring
- https://spring.io/

Atividades criadas

- https://github.com/edson-lessa-jr/unisul-sisdist-pb-proj-prof
- https://github.com/edson-lessa-jr/unisul-sisdist-pb-proj-aluno
- https://github.com/edson-lessa-jr/unisul-sisdist-pb-proj-turma
- https://github.com/edson-lessa-jr/unisul-sisdist-pb-proj-config-server
- https://github.com/edson-lessa-jr/unisul-sisdist-pb-proj-eureka-server
- https://github.com/edson-lessa-jr/unisul-sisdist-pb-atividade-cliente-endereco