



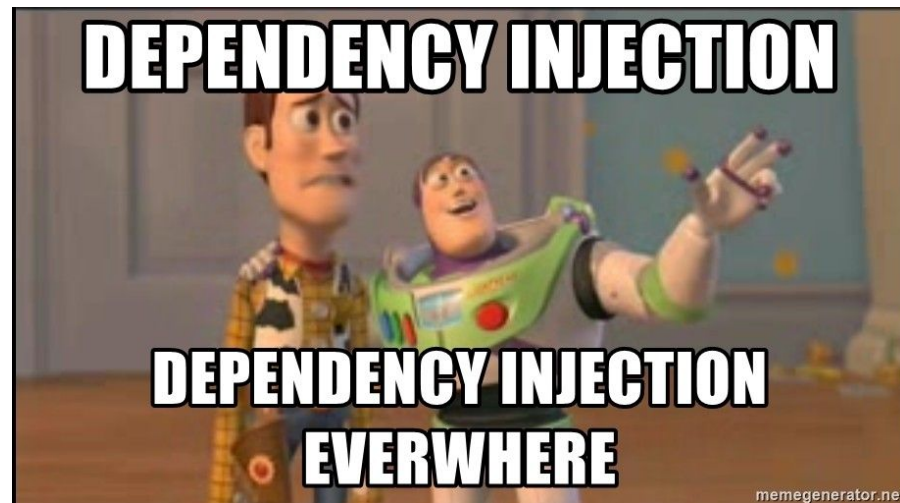
Tecnologias para Back-end I

Aula 03 - Gerenciamento de Dependências e Start no Projeto em Spring

Prof. Kelson Almeida

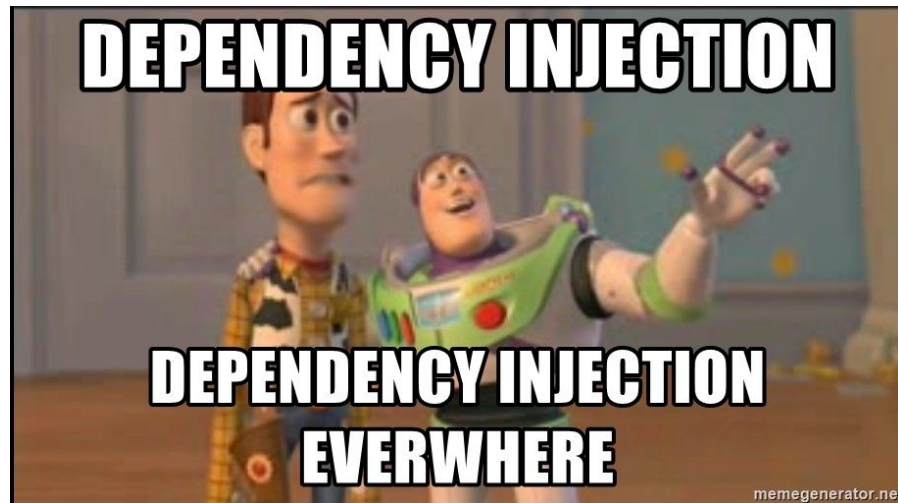
Depedências?

- **POO** é um paradigma de programação mais difundido na atualidade;
- Traz muita **produtividade** no desenvolvimento de software;
- A capacidade de **reuso** de componentes é um dos excelentes fatores deste paradigma.



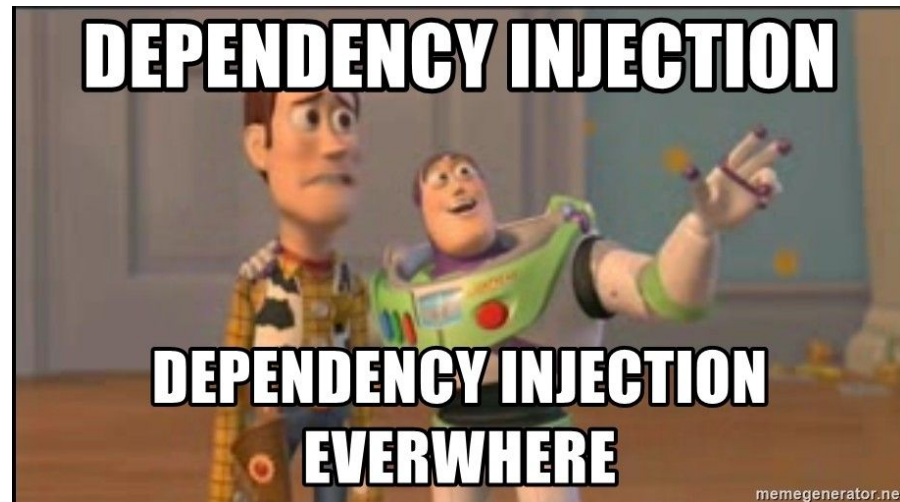
Depedências?

- Permite que os devs criem módulos totalmente **desacoplados**.
- A solução de um **problema maior** pode ser dividido em **soluções menores**, que em conjunto irão resolver o todo.



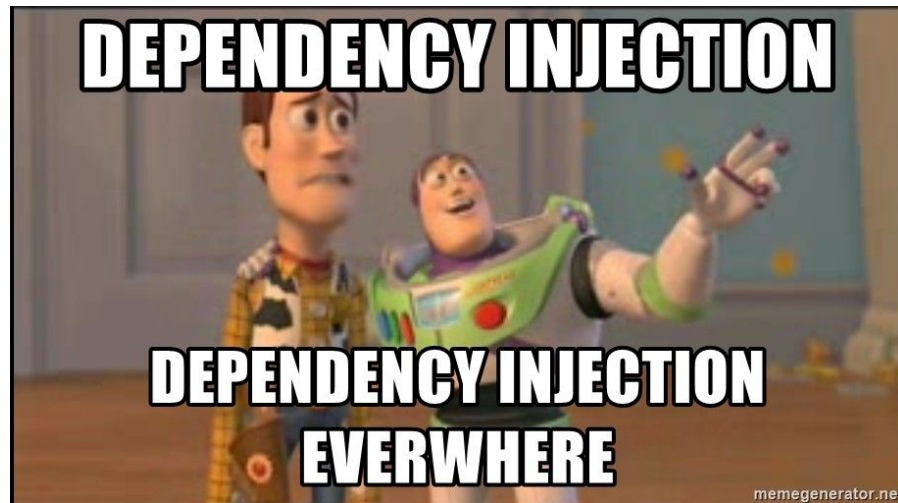
Depedências?

- **Vantagem disso?** Se surgir uma nova necessidade de se utilizar aquele mesmo código... É preciso escrevê-lo todo novamente?
 - Com o reuso não!
- E como configuramos a reutilização desses componentes??



Depedências?

- Imagina ter que add libs novas ao “classpath” do projeto, de forma manual a cada atualização de biblioteca?
- Como levar isso de forma automatizada, sem gerar problemas, para a produção??



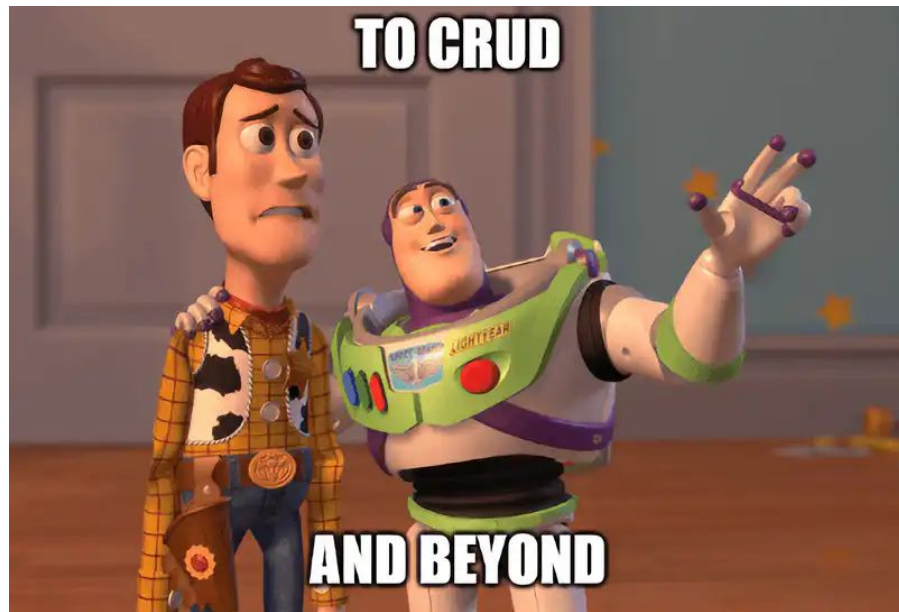
No mundo Java, temos soluções...

- **Maven:** Ferramenta para automação e gerenciamento de projetos Java.
- **Gradle:** Sistema avançado de automatização de builds. Une o melhor da flexibilidade do Ant e o gerenciamento de dependências do Maven.



CRUD?

- **Create:** Criar um novo registro;
- **Read:** Ler um registro, ou uma lista de registros;
- **Update:** Atualizar um ou mais registros;
- **Delete:** Excluir registro(s).



Exemplo de **CRUD** - Sistema Aluno Online

Cadastro de Aluno



Nome:

Email:

Curso:

Salvar

Operações de READ, UPDATE e DELETE

Nome	Email	Curso	Ação
Kelson Almeida	kelson.almeida@uniesp.edu.br	Sistemas para Internet	 

**CRUD
é o que?**

bit.ly/crud-com-qualidade

**Legal fazer
um CRUD**

bit.ly/crud-com-qualidade

**CRUD é
coisa de junior**

bit.ly/crud-com-qualidade

Tudo é CRUD!



Vamos ver isso na prática?

- Lembra do nosso projeto exemplo Hello World da primeira aula?
- Vamos repetir o acesso ao `start.spring.io`

Pacotes - Estrutura

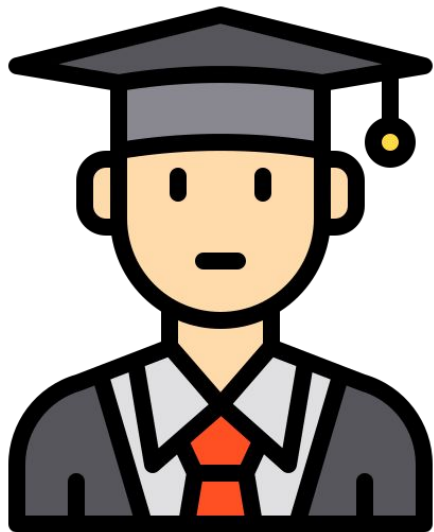
- **Model:** Responsável pelas classes de entidade, conexão com o BD;
- **Services:** Separar as regras de negócio, regras da aplicação e regras de aplicação para que possam ser testadas e reutilizadas por outras partes;
- **Repository:** Responsável pela comunicação com os dados que o service precisa;
- **Controller:** Orquestrador. Recebe chamadas e retorna dados.

Model

Service

Repository

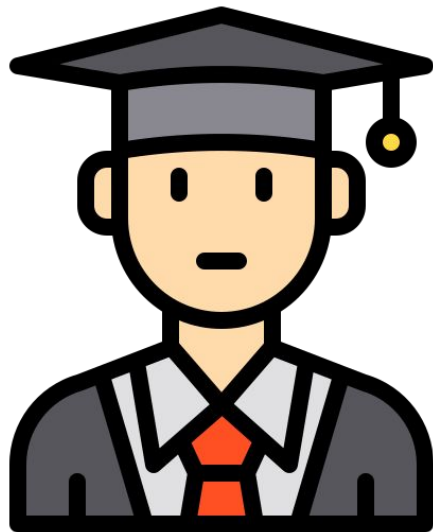
Controller



Objeto Aluno

**Pacote para
“model”ar os
atributos de
Aluno?**

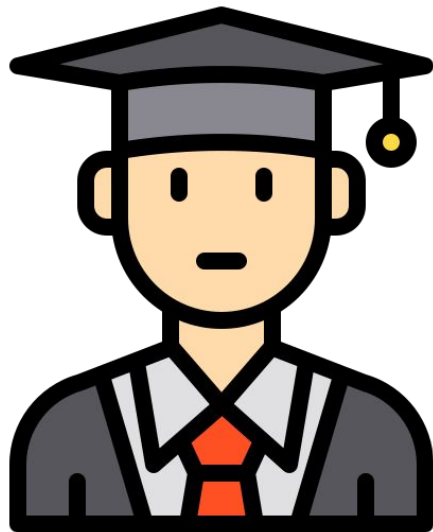
Model
Classe: Aluno



Objeto Aluno

**Pacote para fazer a
ponte entre o Java
o Banco de Dados?**

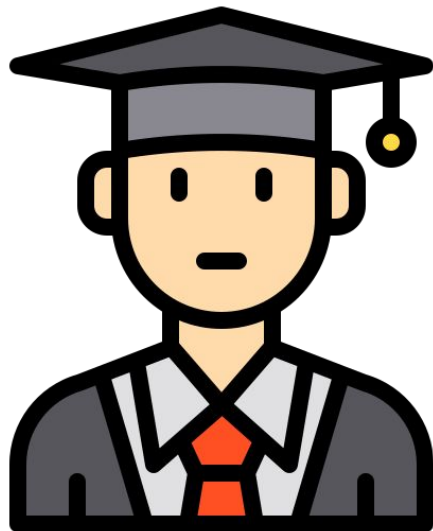
**Repository
Interface:
AlunoRepository**



Objeto Aluno

**Pacote para aplicar
as regras de
negócio?**

Service
Classe: AlunoService



Objeto Aluno

**Pacote para
implementar os
end-points?**

Controller
Classe: AlunoController

Qual vai ser nosso projeto na disciplina?

- **Aluno Online**
- Funcionalidades iniciais do nosso sistema:
 - Cadastro e Gerenciamento de Aluno
 - Primeira etapa (Aula de Hoje): Cadastro de Aluno
 - Apenas informações básicas neste início (nome, email e curso)

Algumas anotações utilizadas...

- Classe Aluno:
 - **@Entity**: A classe pode ser representada por uma tabela do banco de dados. Cada instância representa uma linha da tabela.
 - **@Id**: Especifica que o atributo é uma chave primária da entidade.
 - **@GeneratedValue(strategy = GenerationType.IDENTITY)**: A estratégia utilizada na persistência do Id vai ser de chave de auto-incremento, quando um novo aluno for criado.

Lombok:

- **@Data**: Cria o toString, equals, hashCode, getters e setters para nossa classe.
- **@AllArgsConstructor**: Cria o construtor da classe com todos os argumentos.
- **@NoArgsConstructor**: Cria um construtor vazio da classe com todos os argumentos.

Algumas anotações utilizadas...

- Interface AlunoRepository
 - **@Repository**: Aquela interface vai prover um mecanismo de armazenamento, recuperação, busca, atualização e remoção nos objetos.
- Classe AlunoService
 - **@Service**: Informa que a classe será um “Service” do sistema. (As regras e lógicas de validações estarão ali).
 - **@Autowired**: Injeção de Dependência.
- Classe AlunoController
 - **@RestController**: Será uma controller do sistema, a partir de requisições REST.
 - **@RequestMapping**: Mapeamento da URL da nossa requisição HTTP.

Algumas anotações utilizadas...

- Classe `AlunoController`
 - **@PostMapping**: Este método será responsável por um POST request.
 - **@ResponseStatus**: Código HTTP de retorno da nossa requisição.
 - **@RequestBody**: Retorna o “corpo” da nossa requisição, geralmente o *JSON* que estamos levando como parâmetro no *body*.

Injetando Dependências do
Service no Controller

Injetando Dependências do
Repository do Service

