Crud de Objetos em Java

CRUD (Create, Read, Update e Delete em Inglês) é uma sigla utilizada para se referir às quatro operações básicas realizadas em banco de dados relacionais que são consulta, inclusão, alteração e exclusão dos registros.

Seguindo a Arquitetura do BackEnd é preciso começar pelo **objeto**, criando assim a entidade que será armazenada no banco de dados, então vamos lá:

Entity(Entidade ou Objeto)

Para a criação do Objeto seguiremos o checklist abaixo:

- ***Checklist para criar entidades***
  - Atributos básicos
  - Associações(inicie as coleções)
  - Construtores (não inclua as associações nos construtores com parâmetros)
  - Getters e Setters
  - HashCode e equals(implementação padrão: somente id)
  - Serializable (padrão 1L)

---

obs:as annotations em vermelho são usadas para criar a entidade no **banco de dados.**

```
@Entity
public class Pessoa {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	protected Integer id;
	protected String nome;
	protected String cpf;
	protected String email;
	protected String senha;

	public Pessoa() {
		super();
	}

	public Pessoa(Integer id, String nome, String cpf, String email, String senha) {
		super();
		this.id = id;
		this.nome = nome;
```

```
        this.cpf = cpf;
        this.email = email;
        this.senha = senha;
}


public Integer getId() {
        return id;
}

public void setId(Integer id) {
        this.id = id;
}

public String getNome() {
        return nome;
}

public void setNome(String nome) {
        this.nome = nome;
}

public String getCpf() {
        return cpf;
}

public void setCpf(String cpf) {
        this.cpf = cpf;
}

public String getEmail() {
        return email;
}

public void setEmail(String email) {
        this.email = email;
}

public String getSenha() {
        return senha;
}

public void setSenha(String senha) {
        this.senha = senha;
}


public LocalDate getDataCriacao() {
```

```
        return dataCriacao;
    }

    public void setDataCriacao(LocalDate dataCriacao) {
        this.dataCriacao = dataCriacao;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((cpf == null) ? 0 : cpf.hashCode());
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Pessoa other = (Pessoa) obj;
        if (cpf == null) {
            if (other.cpf != null)
                return false;
        } else if (!cpf.equals(other.cpf))
            return false;
        if (id == null) {
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        return true;
    }

}
```

Após a criação do objeto começamos o desenvolvimento do CRUD, lembrando que as boas práticas devem ser aplicadas e que uma camada conversa com a adjacente e não pode pular para falar com outra, seguindo assim uma ordem.

- ***Resource:***
- ***Service;***
- ***Repository;***

O Resource é a camada que conversa com o frontend, ele recebe a requisição externa e repassa para as camadas internas do backend.

Leitura(Read)

O método de **leitura**(Read) é feito por duas operações:

- findAll(Buscar Todos)
- findById(Buscar por id)

Vamos começar por eles.

Resource

```
@RestController
@RequestMapping(value = "/categorias")
public class PessoaResource {

	@Autowired
	public PessoaService service;

	@GetMapping(value = "/{id}")
	public ResponseEntity<Pessoa> findById(@PathVariable Integer id){
		Pessoa obj = service.findById(id);
		return ResponseEntity.ok().body(obj);
	}

	@GetMapping
	public ResponseEntity<List<Pessoa>> findAll(){
		List<Pessoa> list = service.findAll();
		return ResponseEntity.ok().body(list);
	}
```

Service

Para criar a camada que recebe requisições do RESOURCES e pergunta ao REPOSITÓRIO (Repository), lembrando que o service é usado para operações de negócio, um Método nessa camada deve ter um significado relacionado ao negócio, podendo executar várias operações.

EX: Registrar pedidos(verificar estoque, salvar pedido, baixar estoque, enviar email.)

```
@Service
public class PessoaService {

	@Autowired
	public PessoaRepository repository;

	public Pessoa findById(Integer id) {
		Optional<Pessoa>obj = repository.findById(id);
		return obj.orElse(null);
	}
	public List<Pessoa> findAll() {
		return repository.findAll();
	}
}
```

Repository

Criar pacote REPOSITORY (REPOSITÓRIO BANCO DE DADOS) aqui é realizado operações individuais de acesso ao banco de dados.

```
package com.rodrigo.Pessoa.repositories;

import org.springframework.data.jpa.repository.JpaRepository;
import com.rodrigo.Pessoa.entities.Worker;

public interface PessoaRepository extends JpaRepository<Pessoa, Integer> {
}
```

Criação(Create)

Após o desenvolvimento da leitura(read) iniciaremos o método **criação**(**Create**).

Resource

```
	@PostMapping
	public ResponseEntity<Pessoa> create(@RequestBody Pessoa obj) {
		Pessoa newObj = service.create(obj);
		URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(newObj.g
etId()).toUri();
		return ResponseEntity.created(uri).build();
	}
```

Service

```
	public Pessoa create(Pessoa  obj) {
		obj.setId(null);
		Pessoa newObj = new Pessoa (obj);
		return repository.save(newObj);
	}
```

Update(Atualizar)

Esse método atualiza um objeto já cadastrado.

Resource

```
	@PutMapping(value = "/{id}")
	public ResponseEntity<Pessoa> update(@PathVariable Integer id, @Valid @RequestBody
Pessoa obj) {
		Pessoa obj = service.update(id, obj);
		return ResponseEntity.ok().body(new Pessoa (obj));
	}
```

Service

```
public Pessoa update(Integer id, @Valid Pessoa obj) {
	obj.setId(id);
	Pessoa oldObj = findById(id);
	oldObj = new Pessoa (obj);
	return repository.save(oldObj);
}
```

Deletar(Delete)

Resource

```
@DeleteMapping(value = "/{id}")
public ResponseEntity<Pessoa> delete(@PathVariable Integer id) {
	service.delete(id);
	return ResponseEntity.noContent().build();
}
```

Service

```
public void delete(Integer id) {
	Pessoa obj = findById(id);
	repository.deleteById(id);
}
```

E Assim criamos um CRUD para a entidade Pessoa.