SOLID &
Design Patterns

Gabriel Guimarães de Almeida
Squad Omega

Sumário

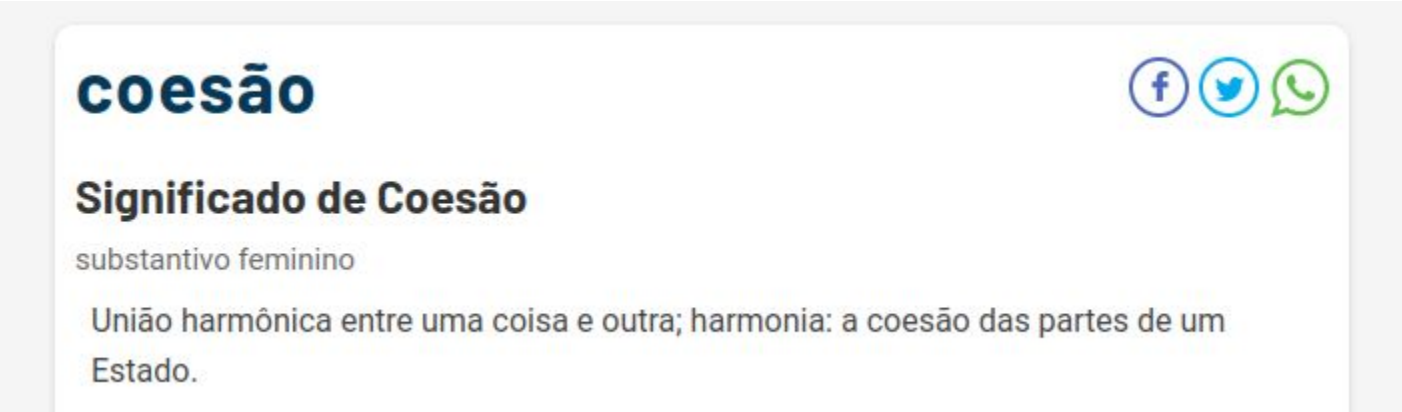
- Revisão de POO (Programação Orientada à Objetos)
  - Coesão
  - Encapsulamento
  - Acoplamento
- SOLID
  - O que é?
  - Siglas

Revisão de POO

- Coesão
- Encapsulamento
- Acoplamento

Coesão

Revisão de POO
Coesão



https://www.dicio.com.br/coesao/

Revisão de POO
Coesão > Sintomas

- Má organização
- Difícil manutenção
- Arquivos com um número grande de linhas

Revisão de POO
Coesão > Exemplos

- Método X Funcionalidade
- Classe X Conteúdo
- *Package* X Classes

Revisão de POO
Coesão > Exemplos > Método X Funcionalidade

Ruim =[

```
12      public User saveUser(User user) {
13          if (verifyFields(user))
14              throw new RuntimeException("Incorrect values!");
15          var queryRunner = new QueryRunner();
16          var query = "INSERT INTO user VALUES (....)";
17          var savedUser = (User) queryRunner.runQuery(query);
18          user.setId(savedUser.getId());
19          return user;
20      }
```

Bom =]

```
10      public User saveUser(User user) {
11          if (verifyFields(user))
12              throw new RuntimeException(ErrorConstants.INCORRECT_VALUES_ERROR_MESSAGE);
13          var userDao = new UserDao();
14          var savedUser :User = userDao.save(user);
15          user.setId(savedUser.getId());
16          return user;
17      }
```

```
5       public User save(User user){
6           var queryRunner = new QueryRunner();
7           var query = "INSERT INTO user VALUES (....)";
8           var savedUser = (User) queryRunner.runQuery(query);
9           return savedUser;
10      }
```

Revisão de POO
Coesão > Exemplos > Classe X Conteúdo

Ruim =[

```java
@Getter
@Setter
public class User {

    private Long id;
    private String name;
    private String cpf;
    private LocalDateTime dateOfBirth;

    public String getFormatedCpf() {
        /*TODO all formating logic here */
        return "xxx.xxx.xxx-xx";
    }

    public String getDateOfBirthAsString() {
        /*TODO all formating logic here */
        return "xx/xx/xxxx";
    }
}
```

Bom =]

```java
@Getter
@Setter
public class User {

    private Long id;
    private String name;
    private String cpf;
    private LocalDateTime dateOfBirth;
}
```

```java
public class CpfUtils {
    public static String formatCpf(String cpfToFormat) {
        /* TODO all formating logic here */
        return "xxx.xxx.xxx-xx";
    }
}
```

```java
public class DateUtils {
    public static String parseLocalDateTimeToApplicationPattern(LocalDateTime dateToFormat) {
        /* TODO all formating logic here */
        return "dd/MM/aaaa";
    }
}
```

Revisão de POO
Coesão > Exemplos > Classe X Conteúdo

Ruim =[

```java
@Getter
@Setter
public class UserService {

    public User saveUser(User user) {
        if (verifyFields(user))
            throw new RuntimeException("Incorrect values!");
        var queryRunner = new QueryRunner();
        var query = "INSERT INTO user VALUES (....)";
        var savedUser = (User) queryRunner.runQuery(query);
        user.setId(savedUser.getId());
        return user;
    }

    private boolean verifyFields(User user) {
        /* TODO verifying logic here */
        return true;
    }
}
```

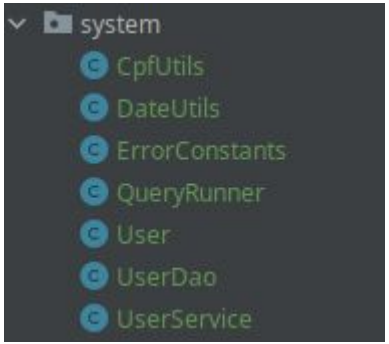
Bom =]

```java
@Getter
@Setter
public class UserService {

    public User saveUser(User user) {
        if (verifyFields(user))
            throw new RuntimeException(ErrorConstants.INCORRECT_VALUES_ERROR_MESSAGE);
        var userDao = new UserDao();
        var savedUser :User = userDao.save(user);
        user.setId(savedUser.getId());
        return user;
    }

    private boolean verifyFields(User user) {
        /* TODO verifying logic here */
        return true;
    }
}
```

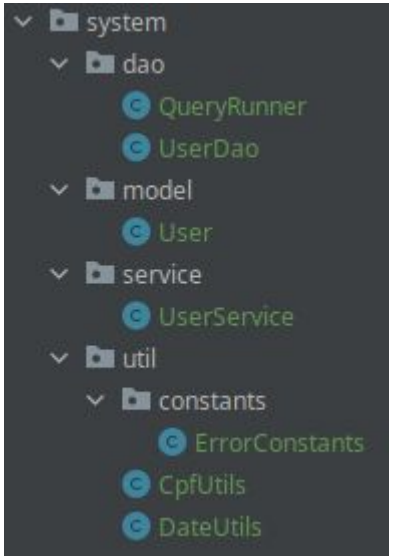
```java
public class UserDao {

    public User save(User user){
        var queryRunner = new QueryRunner();
        var query = "INSERT INTO user VALUES (....)";
        var savedUser = (User) queryRunner.runQuery(query);
        return savedUser;
    }
}
```

Revisão de POO
Coesão > Exemplos > *Package* X Classe

Ruim =[



Bom =]



Encapsulamento

Revisão de POO
Encapsulamento

Proteger e restringir algo dentro de uma cápsula

Link para imagem

Revisão de POO
Encapsulamento > Sintomas

- Risco de segurança dos dados
- Mal acoplamento

Revisão de POO
Encapsulamento > Exemplos

Ruim =[

```
@Getter
@Setter
public class TwitchChannel {

    private Long id;
    private String name;
}
```

Bom =]

```
@Getter
@Setter
public class TwitchChannel {

    private Long id;
    private String name;

    public void setName(String name) {
        if (!this.name.equalsIgnoreCase(name)){
            throw new RuntimeException("Invalid name!");
        }
        this.name = name;
    }
}
```

Revisão de POO
Encapsulamento > Exemplos

Ruim =[

```java
@Getter
@Setter
public class SpotifyAccount {

    private Long id;
    private String name;
    private List<Song> likedSongs;

    public void addLikedSong(Song likedSong) {
        this.likedSongs.add(likedSong);
    }
}
```

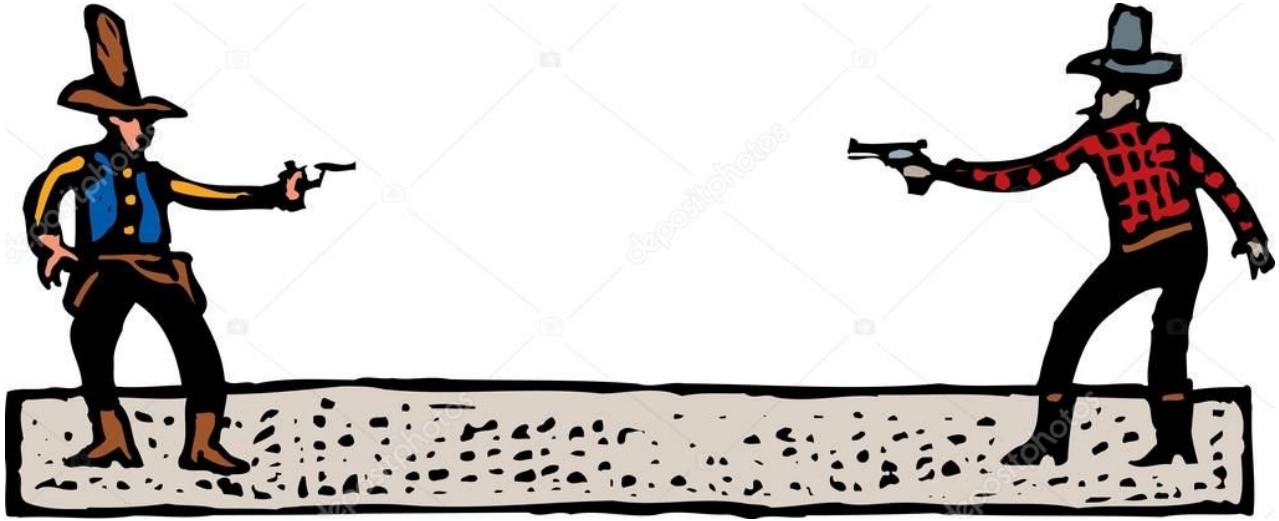
Bom =]

```java
@Getter
@Setter
public class SpotifyAccount {

    private Long id;
    private String name;
    2 usages
    private List<Song> likedSongs;

    public void addLikedSong(Song likedSong){
        if(likedSongs.parallelStream().anyMatch(
                song -> Objects.equals(song.getId(), likedSong.getId())
        )) {
            throw new RuntimeException("Song already liked!");
        }
        this.likedSongs.add(likedSong);
    }
}
```

Acoplamento

Revisão de POO
Acoplamento

Bom ou ruim?



Link da imagem

Revisão de POO
Acoplamento

Bom ou ruim?

Link da imagem

Revisão de POO
Acoplamento

“Nenhuma classe precisa saber muito sobre a outra”

Revisão de POO
Acoplamento > Sintomas

- Implementação de lógicas nos lugares errados
- Difícil manutenção do código

Revisão de POO
Acoplamento > Exemplos

Ruim =[

```java
public class ColaboradorService {
    public Float gerarMediaSalarialGlobal(Colaborador colaborador){
        Float somaGlobal = 0F;
        for (Salario salario : colaborador.getSalarios()) {
            somaGlobal += salario.getValor();
        }

        return somaGlobal / colaborador.getSalarios().size();
    }
}
```

Bom =]

```java
public class ColaboradorService {
    public Float gerarMediaSalarialGlobal(Colaborador colaborador){
        return colaborador.getMediaSalarialGlobal();
    }
}
```

Revisão de POO
Acoplamento > Exemplos

Ruim =[

```
3   public class VendaController {
4   @   public void cadastraVenda(Venda venda){
5           var estoqueService = new EstoqueService();
6           var quantidadeAtual :int = estoqueService.getQuantidadeAtual(venda.getIdProduto());
7           var novaQuantidade = quantidadeAtual - venda.getQuantidade();
8           estoqueService.setQuantidade(venda.getIdProduto(), novaQuantidade);
9       }
10  }
```

Bom =]

```
3   public class VendaController {
4   @   public void cadastraVenda(Venda venda){
5           var estoqueService = new EstoqueService();
6           estoqueService.abaterQuantidade(venda.getIdProduto(), venda.getQuantidade());
7       }
8   }
```

S.O.L.I.D

S.O.L.I.D

- **S:** Single Responsibility Principle
- **O:** Open-Closed Principle
- **L:** Liskov Substitution Principle
- **I:** Interface Segregation Principle
- **D:** Dependency Inversion Principle

S.O.L.I.D

[Link do repositório git]

Single Responsibility Principle (S.R.P.)

S.O.L.I.D
Single Responsibility Principle

- Palavra-chave: **COESÃO**
- **UMA** responsabilidade por classe

S.O.L.I.D
Single Responsibility Principle > Exemplos

```java

@Getter
@Setter
public class UserModel {

    private Long id;
    private String name;
    private LocalDateTime birthDate;

    public boolean isFieldsValid(){
        boolean isFieldsValid = Objects.nonNull(id);
        isFieldsValid = isFieldsValid && Objects.nonNull(name);
        isFieldsValid = isFieldsValid && Objects.nonNull(birthDate);
        isFieldsValid = isFieldsValid && (name.length() >= 3) && (name.length() <= 50);
        isFieldsValid = isFieldsValid && (birthDate.compareTo(LocalDateTime.now()) <= 0);

        return isFieldsValid;
    }
}
```

Ruim =[

S.O.L.I.D
Single Responsibility Principle > Exemplos

```java
@Getter
@Setter
public class UserModel {

    private Long id;
    private String name;
    private LocalDateTime birthDate;
}
```

```java
public class UserModelValidator implements ModelValidator<UserModel>{
    @Override
    public boolean isFieldsValid(UserModel toVerify) {
        boolean isFieldsValid = Objects.nonNull(toVerify.getId());
        isFieldsValid = isFieldsValid && Objects.nonNull(toVerify.getNa
        isFieldsValid = isFieldsValid && Objects.nonNull(toVerify.getBi
        isFieldsValid = isFieldsValid && (toVerify.getName().length() >
        isFieldsValid = isFieldsValid && (toVerify.getBirthDate().compa

        return isFieldsValid;
    }
}
```

```java
public interface ModelValidator <T>{
    boolean isFieldsValid(T toVerify);
}
```

Bom =]

S.O.L.I.D
Single Responsibility Principle > Exemplos

```java
@RequiredArgsConstructor
public class UserService {
    1 usage
    private final UserRepository repository;
    1 usage
    private final UserModelValidator validator;

    public void save(UserModel toSave) {
        if(!validator.isFieldsValid(toSave)) {
            throw new RuntimeException("Some fields has incorrect values.");
        }

        try {
            repository.save(toSave);
        }catch (Exception e){
            throw new RuntimeException("Ops! Some error appeared when we're saving the data.");
        }
    }
}
```

Ruim =[

S.O.L.I.D
Single Responsibility Principle > Exemplos

```java
@RequiredArgsConstructor
public class UserService {
    private final UserRepository repository;
    private final UserModelValidator validator;

    public static final String NOT_VALID_FIELDS = "Some fields has incorrect values.";
    public static final String ERROR_ON_SAVING = "Ops! Some error appeared when we're saving the data.";

    public void save(UserModel toSave) {
        if(!validator.isFieldsValid(toSave)){
            throw new RuntimeException(NOT_VALID_FIELDS);
        }

        try {
            repository.save(toSave);
        }catch (Exception e){
            throw new RuntimeException(ERROR_ON_SAVING + " - " + e.getMessage());
        }
    }
}
```

Ainda
Ruim =[

S.O.L.I.D
Single Responsibility Principle > Exemplos

```java
@RequiredArgsConstructor
public class UserService {
    private final UserRepository repository;
    private final UserModelValidator validator;

    public void save(UserModel toSave) {
        if(!validator.isFieldsValid(toSave)){
            throw new RuntimeException(ErrorConstants.NOT_VALID_FIELDS);
        }

        try {
            repository.save(toSave);
        }catch (Exception e){
            throw new RuntimeException(ErrorConstants.ERROR_ON_SAVING + " - " + e.getMessage());
        }
    }
}
```

Agora tá bão =]

Open Closed Principle

S.O.L.I.D
Open Closed Principle

- Fazer com que as **entidades** estejam:
  - abertas para extensão;
  - fechadas para modificação.

S.O.L.I.D
Open Closed Principle

- Como detectar o uso desta técnica?

Link da imagem

S.O.L.I.D
Open Closed Principle

- ***Feeling*!**

- “Processos imposto a um mesmo objeto que variam de acordo com um valor (geralmente)”

S.O.L.I.D
Open Closed Principle > Exemplos

- Validações;
- Tratamentos;
- *Update* de uma model.

S.O.L.I.D
Open Closed Principle > Exemplo (validações)

- Validações:
  - o operador deve estar habilitado para realizar uma transação;
  - clientes não *prime* não podem ter desconto;
  - clientes prime podem ter até 35% de desconto;
  - caixas rápidos podem ter até 10 itens.

S.O.L.I.D
Open Closed Principle > Exemplo (validações)

```java
@Getter
@Setter
public class TransactionDTO {
    private Long id;
    private BigDecimal rawValue;
    private BigDecimal discountValue;
    private BigDecimal finalValue;

    private OperatorDTO operator;
    private ClientDTO client;
    private CashierDTO cashier;

    private List<ProductTransactionDTO> items;
}
```

```java
@Getter
@Setter
public class OperatorDTO {
    private Long id;
    private String name;
    private boolean ableToMakeSale;
}
```

```java
@Getter
@Setter
public class ClientDTO {
    private Long id;
    private String name;
    private boolean isPrime;
}
```

```java
@Getter
@Setter
public class ProductTransactionDTO {
    private BigDecimal quantity;
    private ProductDTO product;
}
```

```java
@Getter
@Setter
public class ProductDTO {
    private Long id;
    private String name;
    private BigDecimal price;
}
```

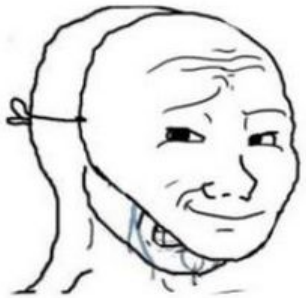
```java
@Getter
@Setter
public class CashierDTO {
    private Long id;
    private Integer number;
    private String descricao;
    private boolean isQuickCashier;
}
```

S.O.L.I.D
Open Closed Principle > Exemplo (validações)

- Como aplicar essas validações?

S.O.L.I.D
Open Closed Principle > Exemplo (validações)

Validações em
um método só



Validações
em factory

Link da imagem base

S.O.L.I.D
Open Closed Principle > Exemplo (validações)

- Validações em um método só

Ver a classe
"br.com.solid.aula.openclosedprinciple.service.TransactionService"

Liskov Substitution Principle

S.O.L.I.D
Liskov Substitution Principle

- Ponto bem focado na herança;

"Se q(x) é uma propriedade demonstrável do objetos x de tipo T, então q(y) deve ser verdadeiro para objetos y de tipo S, onde S é um subtipo de T."
–Barbara Liskov

S.O.L.I.D
Liskov Substitution Principle

Evitar que heranças (e afins) causem efeitos colaterais negativos!

S.O.L.I.D
Liskov Substitution Principle > Exemplo 1

- Transformando aves em classes:
  - Pardal;
  - Galinha;
  - Pombo.

S.O.L.I.D
Liskov Substitution Principle > Exemplo 1

- Transformando aves em classes:

```
3    public class Pombo {
4    }
```

```
3    public class Galinha {
4    }
```

```
3    public class Pardal {
4    }
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 1

- Usando herança:

```
public class Pardal implements Ave
{}
```

```
public class Galinha implements Ave
{}
```

```
public class Pombo implements Ave
{}
```

```
public interface Ave
{
}
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 1

- Usando herança:
  - Criando o método “voar()”

```
public class Pardal implements Ave
{}
```

```
public class Galinha implements Ave
{}
```

```
public class Pombo implements Ave
{}
```

```
public interface Ave {
    void voar();
}
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 1

- Com o método “voar()” a semântica é quebrada, pois uma galinha não voa.

```
public class Galinha implements Ave
{}
```

```
public interface Ave {
    public void voar();
}
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 1

- Para contornar este problema, podemos criar uma classe intermediária:

```
public class Pardal implements
AveAveQueVoa
{
    @Override
    public void voar() {}
}
```

```
public class Galinha implements Ave
{}
```

```
public class Pombo implements AveAveQueVoa
{
    @Override
    public void voar() {}
}
```

```
public interface AveQueVoa implements Ave{
    void voar();
}
```

```
public interface Ave
{}
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- Sistema de promoção/rebaixamento de membros de uma guilda:
  - apenas membros podem ser promovidos ou rebaixados;
  - membros temporários não podem ser promovidos ou rebaixados.

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- Implementação com problemas:

```java
public class MemberModel {
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDateTime birthDate;
    private GuildRank rank;
}
```

```java
public class TemporaryMemberModel extends MemberModel {
    private LocalDateTime startTrial;
    private LocalDateTime finalTrial;
}
```

```java
public void promote(MemberModel member) {
public void demote(MemberModel member) {
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- Heranças podem causar problemas de lógica:

```java
public class MemberModel {
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDateTime birthDate;
    private GuildRank rank;
}
```

```java
public class TemporaryMemberModel extends MemberModel {
    private LocalDateTime startTrial;
    private LocalDateTime finalTrial;
}
```

```java
public void promote(MemberModel member) {
public void demote(MemberModel member) {
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- O “extends MemberModel” faz com que tanto o “MemberModel” e “TemporaryMemberModel” possam ser promovidos e rebaixados.

```
memberService.promote(temporaryMember);// NÃO DEVERIA PERMITIR
memberService.promote(member);// PERMITE

memberService.demote(temporaryMember);// NÃO DEVERIA PERMITIR
memberService.demote(member);// PERMITE
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- O que fazer?

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- O que fazer?

Modificar o “fluxo” de herança ou usar a **composição**.

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- Modificando o “fluxo” de herança:
  - Extrair os dados comuns para outra classe

```java
public class BaseMemberModel {
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDateTime birthDate;
}
```

```java
public class MemberModel extends BaseMemberModel {
    private GuildRank rank;
}
```

```java
public class TemporaryMemberModel extends BaseMemberModel {
    private LocalDateTime startTrial;
    private LocalDateTime finalTrial;
}
```

S.O.L.I.D
Liskov Substitution Principle > Exemplo 2

- Usando a composição:
  - Extrair os dados comuns para um atributo

```java
public class BaseMemberInformation {
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDateTime birthDate;
}
```

```java
public class MemberModel {
    private BaseMemberInformation baseMemberInformation;
    private GuildRank rank;
}
```

```java
public class TemporaryMemberModel {
    private BaseMemberInformation baseMemberInformation;
    private LocalDateTime startTrial;
    private LocalDateTime finalTrial;
}
```

Interface Segregation Principle

S.O.L.I.D
Interface Segregation Principle

- Modificando o “fluxo” de herança:
    - Extrair os dados comuns para outra classe

S.O.L.I.D
Interface Segregation Principle > Exemplo 1

- Heróis:
  - heróis que voam;
  - heróis que ficam invisíveis;
  - heróis que solta laser pelos olhos;
  - heróis com super força;
  - heróis que são ricos.

S.O.L.I.D
Interface Segregation Principle > Exemplo 1

- Heróis:
  - Batman;
  - Homem Aranha;
  - Mulher Invisível;
  - Super Homem.

S.O.L.I.D
Interface Segregation Principle > Exemplo 1

Vamos ao *package*

"br.com.solid.aula.interfacesegregationprinciple.exemplo1.implementacaoruim"

S.O.L.I.D
Interface Segregation Principle > Exemplo 1

- Alguns heróis precisam implementar poderes que eles não têm!

- Como contornar isso?

S.O.L.I.D
Interface Segregation Principle > Exemplo 1

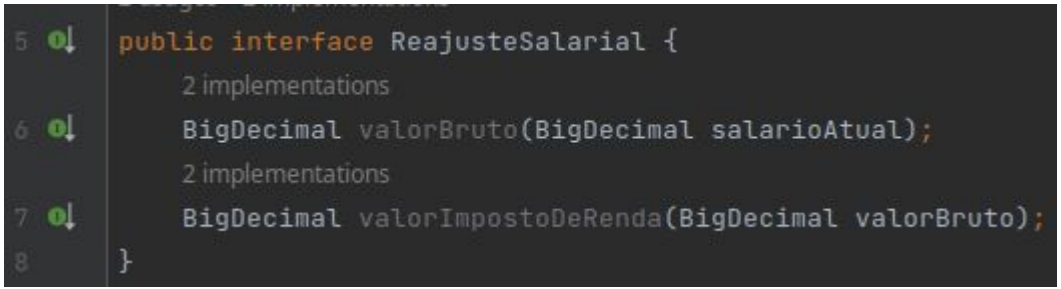
- Como contornar isso?
    - Segregando as interfaces!
    - Heróis não precisam implementar os poderes que não têm!

S.O.L.I.D
Interface Segregation Principle > Exemplo 2

- Reajustes salariais:
  - Reajuste de promoção
    - Consta imposto de renda
  - Reajuste anual
    - Não consta imposto de renda

S.O.L.I.D
Interface Segregation Principle > Exemplo 2

- Possível implementação:



S.O.L.I.D
Interface Segregation Principle > Exemplo 2

- Mas isso força ambos os reajustes a implementarem o imposto de renda:

```java
public class ReajustePromocao implements ReajusteSalarial {
    @Override
    public BigDecimal valorBruto(BigDecimal salarioAtual) {
        return salarioAtual.multiply(BigDecimal.valueOf(0.1));
    }

    @Override
    public BigDecimal valorImpostoDeRenda(BigDecimal valorBruto) {
        return valorBruto.multiply(BigDecimal.valueOf(0.09));
    }
}
```

```java
public class ReajusteAnual implements ReajusteSalarial {
    @Override
    public BigDecimal valorBruto(BigDecimal salarioAtual) {
        return salarioAtual.multiply(BigDecimal.valueOf(0.09));
    }

    @Override
    public BigDecimal valorImpostoDeRenda(BigDecimal valorBruto) {
        return BigDecimal.ZERO;
    }
}
```

S.O.L.I.D
Interface Segregation Principle > Exemplo 2

- Para contornar isso podemos criar uma nova interface:

```
public interface ReajusteSalarial {
    2 implementations
    BigDecimal valorBruto(BigDecimal salarioAtual);
}
```

```
public interface ReajusteSalarialComImpostoDeRenda extends ReajusteSalarial {
    1 implementation
    BigDecimal valorImpostoDeRenda(BigDecimal valorBruto);
}
```

S.O.L.I.D
Interface Segregation Principle > Exemplo 2

- Assim, apenas o reajuste que possui imposto de renda precisará implementá-lo

```java
public class ReajusteAnual implements ReajusteSalarial {
    @Override
    public BigDecimal valorBruto(BigDecimal salarioAtual) {
        return salarioAtual.multiply(BigDecimal.valueOf(0.09));
    }
}
```

```java
public class ReajustePromocao implements ReajusteSalarialComImpostoDeRenda {
    @Override
    public BigDecimal valorBruto(BigDecimal salarioAtual) {
        return salarioAtual.multiply(BigDecimal.valueOf(0.1));
    }

    @Override
    public BigDecimal valorImpostoDeRenda(BigDecimal valorBruto) {
        return valorBruto.multiply(BigDecimal.valueOf(0.09));
    }
}
```

Dependency Inversion Principle

S.O.L.I.D
Dependency Inversion Principle

Implementação **deve** depender da **abstração**, não o contrário!

S.O.L.I.D
Dependency Inversion Principle > Exemplo

Para o *package*:

"br.com.solid.aula.dependencyinversionprinciple.exemplo1"

Perguntas?

Referências Bibliográficas

Curso de SOLID com Java: princípios da programação orientada a objetos

SOLID Design Principles Explained: Dependency Inversion Principle with Code Examples

Obrigado!!!

Link da imagem

Foi um prazer valeu!