REFLECTION

Conhecido como *BlackMagic*, **Reflection** possibilita a exposição de quaisquer membro existente dentro uma classe em *RunTime*.

> *É nada mais do que a habilidade de enxergar um *.class*' (no java).* - Garcia

É um pacote que te dá permissão de ver informações do seu *.class*. como:

- **MetaDado**
- **MetaObjeto (meta classe)** Representam informações da classe
- **Classes**
- **Type**
- **Method** Métodos.
- **Field** Atributos.
- **Annotations** Marcador para conversas com o compilador.
- **Modifier** abstract, protected, public, private, default.

Vantagens Criação de aplicativos mais dinâmicos, Redução na quantidade de repetição de código (Boilerplate), Minimização de erros e Facilidade de manutenção.

Desvantagens Domínio mais avançado de lógica de programação, Exigência de um maior nível de atenção ao codificar e Geração de código complexo.

Exemplo de Boilerplate Evitamos repetição trecho de código em várias partes do arquivo mudando apenas pequenas coisas, como por exemplo um toString(): *public String toString(){ return "ClasseNome"; }*

Agregação X Composição

Na Agregação, a existência do Objeto-Parte faz sentido, mesmo não existindo o Objeto-Todo. Vejamos o exemplo Time-Atleta:

Já a Composição é uma agregação mais forte; nela, a existência do Objeto-Parte NÃO faz sentido se o Objeto-Todo não existir. Vejamos o exemplo Pedido-ItemPedido:

Principio do aberto e fechado

O princípio Aberto/Fechado poderia ser entendido como uma implementação que permite adicionar novas funcionalidades sem mexer no código existente. Em outras palavras: *Não precisamos alterar o conteúdo das classes, basta criar novas implementações de interfaces ou sobrescrever os métodos de classes existentes.*

```java
//GenericCreateTableDB/GenericCreate.java
public class GenericCreate {
  public static String create(Object o){
    String sql = "";
    sql = "CREATE TABLE ";
    Class<?> clazz = o.getClass();
    Table t = clazz.getDeclaredAnnotation(Table.class);
    sql += t.nome() + "( ";
    Field[] fs = clazz.getDeclaredFields();
    for( Field f: fs){
      if(f.isAnnotationPresent(Column.class)){
        Column c = f.getDeclaredAnnotation(Column.class);
        String nome = c.nome();
        sql +=  nome;
      }
      if(f.isAnnotationPresent(Varchar.class)){
        Varchar vc = f.getDeclaredAnnotation(Varchar.class);
        int qt = vc.qut();
        sql += " vachar("+String.valueOf(qt)+"),";
      }else{
        String tipo = f.getType().getTypeName();
        sql += " " + tipo + ",";
      }
    }
    sql = trocar(sql);
    return sql;
  }
  public static String trocar(String sql){
    char[] ch = sql.toCharArray();
    ch[ch.length-1]=')';
    return new String(ch);
  }
}
// MAIN
Login l = new Login(1, "teste", "root");
GenericCreate c = new GenericCreate();
System.out.println( c.create(l) );
/* console: CREATE TABLE TB_LOGIN(
 CD_LOGIN int,NM_LOGIN vachar(10),PS_SENHA vachar(60) */
```

```java
// ### Nome simples da classe
Class<?> clazz = o.getClass();
return clazz.getSimpleName();
// ### Pega campos
// console:nome = nome / membro da classe = class ex2.Cliente / modificador = private
Class<? extends Cliente> clazz = new Cliente("Flavia", "Souza").getClass();
Field[] f = clazz.getDeclaredFields();
for(Field g : f){
  System.out.println("nome = " + g.getName());
    System.out.println("membro da classe = " + g.getDeclaringClass());
    System.out.println("modificador = " + Modifier.toString( g.getModifiers() ) + "\n");
}
// ### Pega metodos
// console: nome = getNome / membro da classe = class ex2.Cliente / modificador = public
Class<? extends Cliente> clazz = new Cliente("Flavia", "Souza").getClass();
Method[] m = clazz.getDeclaredMethods();
for(Method n : m){
  System.out.println("nome = " + n.getName());
    System.out.println("membro da classe = " + n.getDeclaringClass());
    System.out.println("modificador = " + Modifier.toString( n.getModifiers()  ) + "\n");
}
// ### pega interfaces
// public class Automovel implements Carro, Moto {
// -- INTERFACES Automovel --
// Interface = ex3.Carro    \n    Interface = ex3.Moto
Class<? extends Automovel> clazz = new Automovel().getClass();
System.out.println("-- INTERFACES Automovel --");
Class[] in = clazz.getInterfaces();
for(Class v : in){
  System.out.println("Interface = " + v.getName());
}
```

```java
// ### pega campos DEFAULT
System.out.println("-- ATTRS (default) --");
Field[] f = clazz.getDeclaredFields();
for(Field n2 : f){
  if (  !(Modifier.isPublic(n2.getModifiers())) &&
      !(Modifier.isProtected(n2.getModifiers())) &&
      !(Modifier.isPrivate(n2.getModifiers())) )
      System.out.println("nome do atributo = " +
        n2.getName() + " -- " + n2.getGenericType());
}
// ### Trocar valor de campo
public static void setAllIntengerFor89Negative(Object o) {
    Class<?> clazz = o.getClass(); // new Cliente("Flavia",
    Field[] fs = clazz.getDeclaredFields();
    for(Field f : fs){
      f.setAccessible(true);
        if(f.getType().equals(int.class)){
          f.set(o, -89); // trocando para -89
        };
    }
}
```

```java
// ### Validar senha (Claass Main)
public static void isValid(Object o) throws Exception{
Class<?> clazz = o.getClass();
Field[] fs = clazz.getDeclaredFields();
for(Field f : fs){
  if(f.isAnnotationPresent(Senha.class)){
    f.setAccessible(true);
    String senha = (String) f.get(o);
    if( !senha.matches(".*\\d+.*") ) // pelo menos 1 numero
      throw new Exception("ERRO! Senha tem que ter numero");
  }
}System.out.println("OK!");
}
public static void main(String[] args) throws Exception {
Cliente c1 = new Cliente("Flavia", "a3b");
isValid(c1); // OK
System.out.println( c1.toString() );
Cliente c2 = new Cliente("Ramon", "aa");
isValid(c2); // ERRO
System.out.println( c2.toString() );
}
```

```java
//Reflection/src/Interceptor/Interceptor.java
@Target(ElementType.METHOD) // pode ser FIELD, CONSTRUCTOR, TYPE (Class)
@Retention(RetentionPolicy.RUNTIME)
public @interface Interceptor {
        String met();
        Class cl(); // default Foo.class
}
// Reflection/src/Interceptor/Bar.java
... @Interceptor( met = "fazAlgo", cl = Foo.class )
public void hello(){ System.out.println("Bem-vindo"); } ...
// Reflection/src/Interceptor/Delegator.java
public class Delegator {
  public void voidExecutor(Object obj, String metodo) {
    Class<?> clazz = obj.getClass();
    Method metFind = clazz.getDeclaredMethod(metodo);
    if(metFind==null){
      System.out.println("ERRO: Metodo não existe!");
    }else{
      //if(metFind.getName()==metodo){
      if (metFind.isAnnotationPresent( Interceptor.class )) {
        Interceptor a = metFind.getDeclaredAnnotation( Interceptor.class );
        String met = a.met();
        Class cl = a.cl();
        Object objx = cl.newInstance();
        Class<?> clazzx = objx.getClass();
        Method[] msx = clazzx.getDeclaredMethods();
        for(Method mx : msx){
          if( mx.getName().equals(met) ){
            mx.setAccessible(true);
            mx.invoke(objx, null);
          }
        }
      }
      metFind.invoke(obj, null); // Bem-vindo
    }
  }
}
// Main
Delegator d = new Delegator();
d.voidExecutor( new Bar(), "hello" );
```

```java
//Annotations/src/ordem/Main.java
public class Main {
  static void ordenar(Object obj) {
    Class<?> clazz = obj.getClass();
    Method[] m = clazz.getDeclaredMethods();
    Method[] aux = new Method[m.length];
    for(Method n : m){
      if(n.isAnnotationPresent(Ordem.class)){
        Ordem s = n.getDeclaredAnnotation(Ordem.class);
        int numero = s.numero();
        aux[numero -1] = n ;
      }
    }
    for(Method n : m){
      if(n!=null)
        // Passa o objeto e parametros
        n.invoke(obj, null);
    }
  }
  public static void main(String[] args) {
    Gato branco = new Gato();
    ordenar(branco);
  }
}
```

```java
// ### FAZ SELECT com reflection
public class Main {
  public static String trocar(String sql){
    char[] ch = sql.toCharArray();
    ch[ch.length-1] = 0;
    return new String(ch);
  }
  public static void doSQL(Object o){
    Class<?> clazz = o.getClass();
    Field[] fs = clazz.getDeclaredFields();
    String sql = "SELECT ";
    for(Field f : fs){
      sql += f.getName() + ",";
    }
    sql = trocar(sql);
    sql += "FROM " + clazz.getSimpleName() + " WHERE " + fs[0].getName() + " = ?";
    System.out.println(sql);
  }
  public static void main(String[] args) throws Exception {
    Cliente c1 = new Cliente();
    doSQL(c1);
    // console:SELECT nome,nome2,nome3,nome4,nome5 FROM Cliente WHERE nome = ?
  }
}
```