

# Melhores práticas de nomenclatura de métodos e variáveis.

*20 de Agosto de 2018*

<b>Objeto</b>	<b>3</b>
<b>Comentários</b>	<b>3</b>
Classe (criação)	3
Classe (alteração)	4
Método (criação)	4
<b>Variáveis</b>	<b>5</b>
Nomenclatura	5
ToAdd	6
ToFilter	6
ToReturn	6
ToInsert	6
ToUpdate	6
ToUpsert	6
ToDelete	7
for	7
page	7
map	7
list	7
set	7
<b>Métodos</b>	<b>8</b>

# Objeto

Etapas para criação do nome de um objeto personalizado:

Os objetos devem ser criados com o mesmo nome da classe, com a separação das palavras feita com “\_”, e sem acentuação. Ex:

*Classe: Composição de produto*

*Nome do objeto: Composicao\_produto*

## Comentários

### Classe (criação)

Para qualquer classe ou trigger criada, o seguinte comentário de cabeçalho deve ser inserido:

- 1) @testClass: nome da classe de testes
- 2) @description: descrição breve
- 3) @author: nome completo do desenvolvedor que está criando
- 4) @date (**N08**): data da criação

Ex:

```
/*
    @testClass Teste_Trigger_Account

    @description Trigger for object: Account

    @author Joao Silva
    @date 27/03/2018
*/
trigger Trigger_Account on Account(before insert){
}

/*
    @testClass Teste_DAO_Composicao_Produto

    @description DAO class for object: Composicao_produto__c

    @author Joao Silva
    @date 27/03/2018
*/
public class DAO_Composicao_Produto{
}
```

## Classe (alteração)

Quando uma classe ou trigger for alterada, o seguinte comentário de cabeçalho deve ser inserido:

- 1) @author: nome completo do desenvolvedor que está alterado
- 2) @date (**N08**): data da alteração
- 3) @description: descrição breve do que foi alterado

Ex:

```
/*
-----
  @author Joao Silva
  @date 27/03/2018
  @description new method: setFields
*/
```

## Método (criação)

Para todo método criado, o seguinte comentário de cabeçalho deve ser inserido:

```
/*
  @description return the sum of params
  @param Integer pValueA
  @param Integer pValueB
  @return Integer

  @author Joao Silva
  @date 28/03/2018
*/
public static Integer getSum(Integer pValueA, Integer pValueB){
    return pValueA + pValueB;
}
```

As variáveis sempre devem ser declaradas no início do método e nunca no meio, desta forma fica claro quais variáveis o método possui, ex:

```
public void getMethod(){
    String newText = "";
    String oldText = "";
    Decimal valueToSum;

    for(){
    }
```

```

        if(){
        }else{
        }
    }

```

## Variáveis

### Nomenclatura

Etapas para criação do nome de uma variável:

As variáveis devem ser criadas contendo uma indicação do tipo no nome, em minúsculo, e ter sempre nomes representativos para que assim seja possível identificar o objetivo dela durante a leitura do código.. Ex:

Iniciais por tipo de variável:

double	= d
int	= i
string	= s
char	= c
boolean	= b
byte	= by
short	= sh
float	= f

Utilizando um padrão de nomenclatura para as variáveis, o desenvolvedor que irá analisar o código irá se identificar do que se trata a variável assim que olhar o nome:

```

ToAdd
ToFilter
ToReturn
ToInsert
ToDelete
ToUpdate
ToUpsert
for
page
map
list
set

```

### ToAdd

Utilizada quando a variável será apenas inserida em uma lista, ex:

```
List<Account> listAccountToReturn = new List<Account>();
Account accountToAdd = new Account();
listAccountToReturn.add(accountToAdd);
return listAccountToReturn;
```

## ToFilter

Utilizada quando a variável será enviada apenas como filtro para outro método, ex:

```
List<Id> listIdAccountToFilter = new List<Id>();
listIdAccountToFilter.add('0013D00000SvUjF');
```

```
List<Account> listAccount = DAO_Account.getById(listIdAccountToFilter);
```

## ToReturn

Utilizada quando a variável for o retorno do método, ex:

```
List<Date> listDateToReturn = new List<Date>();
listDateToReturn.add(System.today());
listDateToReturn.add(System.today().addDays(1));
return listDateToReturn;
```

## ToInsert

Utilizada para DML insert, ex:

```
Account accountToInsert = new Account();
Insert accountToInsert;
```

## ToUpdate

Utilizada para DML update, ex:

```
Account accountToUpdate = new Account();
accountToUpdate.Id = 'XXX';
update accountToUpdate;
```

## ToUpsert

Utilizada para DML upsert, ex:

```
Account accountToUpsert = new Account();
accountToUpsert.CNPJ__c = '123';
upsert accountToUpsert CNPJ__c;
```

## ToDelete

Utilizada para DML delete, ex:

```
Account accountToDelete = new Account();
accountToDelete.Id = 'XXX';
delete accountToDelete;
```

## for

Utilizada para laços de repetição, o nome deve ser **for** seguido do nome do objeto, ex:

```
for(Account forConta : accounts){
}
```

## page

Utilizada para variáveis que são utilizadas em visualforce page ou componente, devem ter o nome de **page** seguido do tipo da variável, ex:

```
public String pageListContas {get; set;}
public String pageTotalPaginas {get; set;}
```

## map

Utilizada para Map, devem ter o nome de **map** seguido do tipo da variável, ex:

```
public Map<Id, Account> mapAccount {get; set;}
public Map<String, Decimal> mapDiscountType {get; set;}
```

## list

Utilizada para List, devem ter o nome de **list** seguido do nome da variável, ex:

```
public List<Id> listIdAccount {get; set;}
public List<String> listTypes {get; set;}
```

## set

Utilizada para Set, devem ter o nome de **set** seguido do nome da variável, ex:

```
public Set<Id> setIdOpportunity {get; set;}
public Set<Decimal> setDiscounts {get; set;}
```

# Métodos

O nome do método deve significar a sua ação, se um método retorna uma soma o nome do método pode ser ***getSum***, se o método instancia uma variável o nome pode ser ***newVar***.

Os métodos devem ser construídos de uma forma simples que facilite a manutenção, desta forma podemos criar outros métodos para auxiliar o método atual, repartindo o código o entendimento poderá ficar mais simples.

Alguns exemplos:

```
getValores()  
getListaByAccountId()  
buscarContatosByName()  
atualizarRegistros()
```

Se o desenvolvimento se tornar muito extenso a lógica deve ser quebrada para que execute pequenos métodos.