

Perdeu preiboi, agora é OO – Parte II

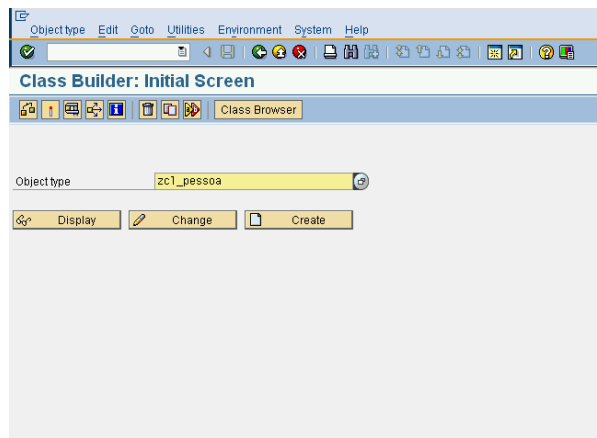
27/06/2012 10:42

Teste.. testando.. 123... F8... Ainda existe esperança, a PARTE II do “Perdeu preiboi, agora é OO”. Antes de ler este post leia [Perdeu preiboi, agora é OO – Parte I](#). Sem churumelas, vamos a parte técnica sobre a criação de uma classe.

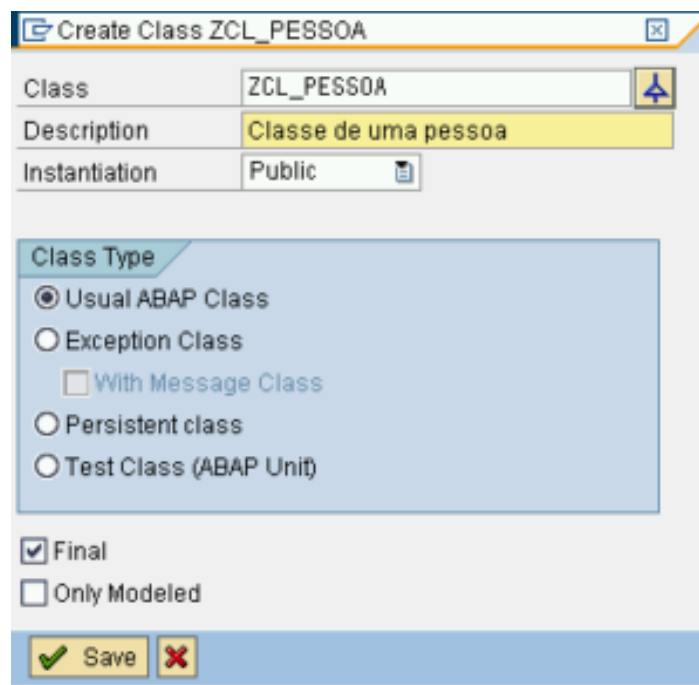
Objetivos:

- Criar um modelo de classe de uma pessoa;
- Atributos: nome, idade, sexo;
- Ações: fazer aniversário, comer, acordar e dormir;
- Regra: a idade de uma pessoa nunca pode retroceder e sempre acrescentar 1 quando o a pessoa fazer aniversário;

Transação SE24, para criar a classe global.



Coloque a descrição da classe, existe um flag “Final”, ele quer dizer que esta classe não pode ser herdada, nos próximos posts, vamos ver o que isso quer dizer.



Os atributos são as características do nosso objeto. Na aba atributos, vamos criar os atributos, de proposito coloquei a visibilidade do atributo IDADE como publico, apenas para vermos a diferença.

The screenshot shows the SAP Class Builder interface for class **ZCL_PESSOA**. The **Attributes** tab is selected, displaying a table of class attributes.

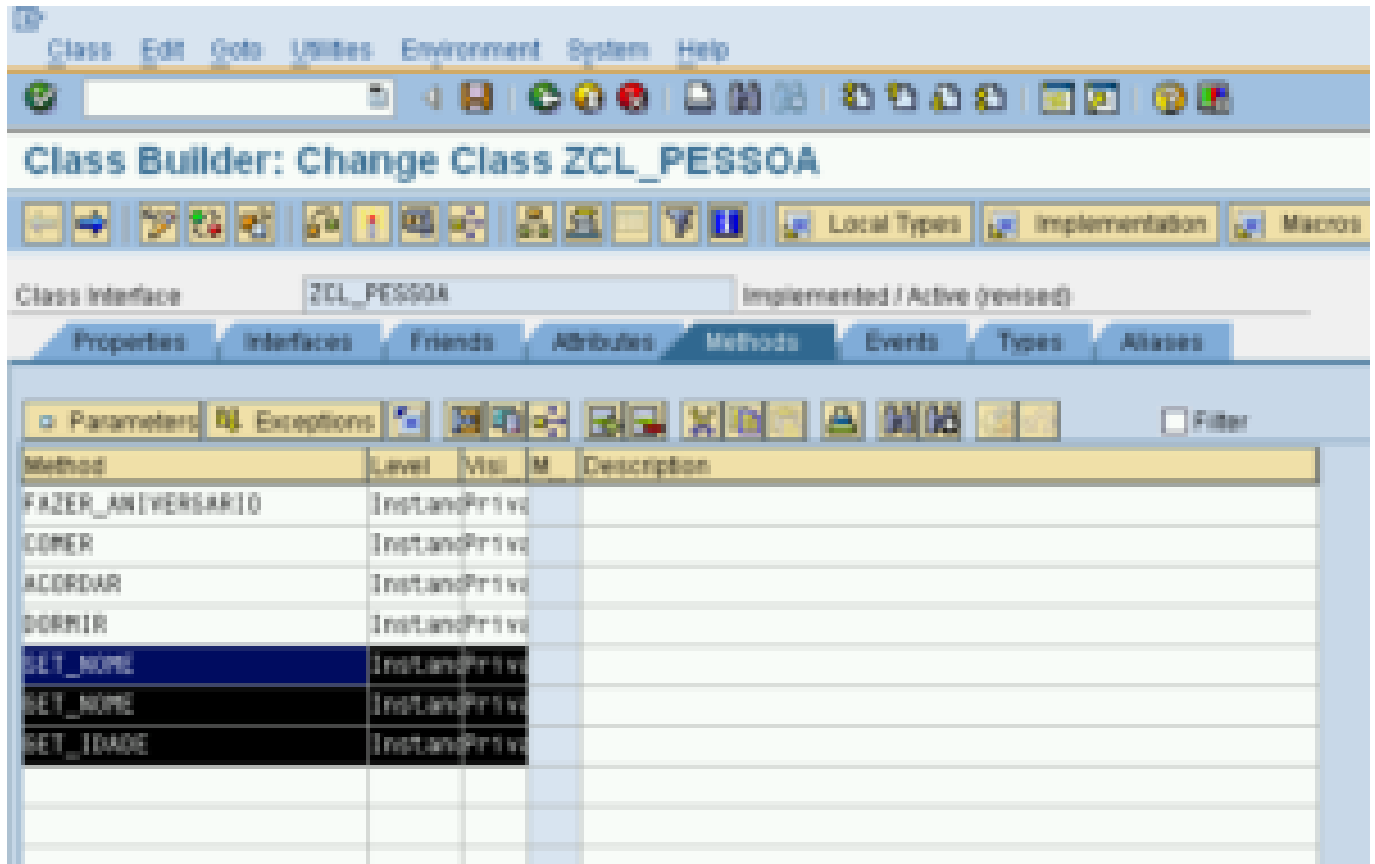
Attribute	Level	Visibility	Re...	Typing	Associated Type	Description	Initial value
NOME	Instance Attribut	Private	<input type="checkbox"/>	Type	CHAR30	30 Characters	
IDADE	Instance Attribut	Public	<input type="checkbox"/>	Type	NUMC2	Two digit number	
SEXO	Instance Attribut	Private	<input type="checkbox"/>	Type	CHAR1	Single-Character Flag	
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			
			<input type="checkbox"/>	Type			

Na aba de métodos, criamos as ações do nosso objetos. Nesse exemplo criei todos os métodos com a visibilidade privada.

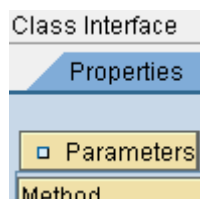
The screenshot shows the SAP Class Builder interface for class **ZCL_PESSOA**, with the **Methods** tab selected. The table lists several methods, all with private visibility.

Method	Level	Visibility	M...	Description
FAZER_ANIVERSARIO	Instance/Private			
COMER	Instance/Private			
ACORDAR	Instance/Private			
DORMIR	Instance/Private			

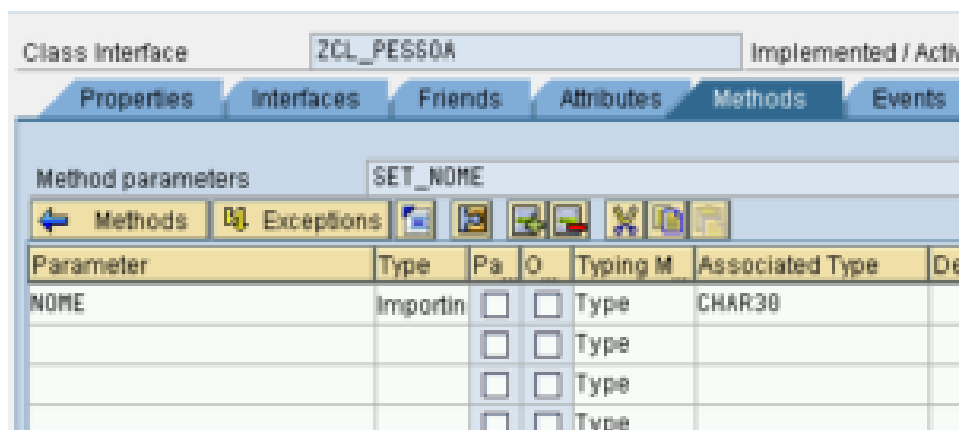
Criação dos métodos GET e SET. Mas o que são estes métodos?! Como um padrão em varias linguagens de programação, usamos o métodos GET para ler a informação de um atributo. Utilizamos o método SET para inserir uma informação em um atributo. O atributo IDADE tem uma regra, acrescentar um quando o método FAZER_ANIVERSÁRIO é chamado, por este motivo não criei o método SET_IDADE.



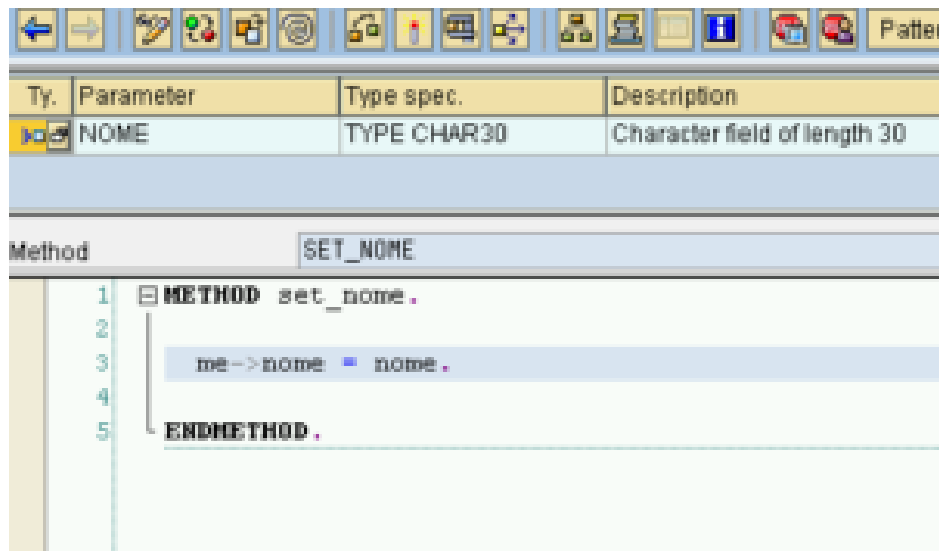
Selecionando um método podemos clicar no botão “Parameters” para inserir quais os parâmetros de entrada e saída do método.



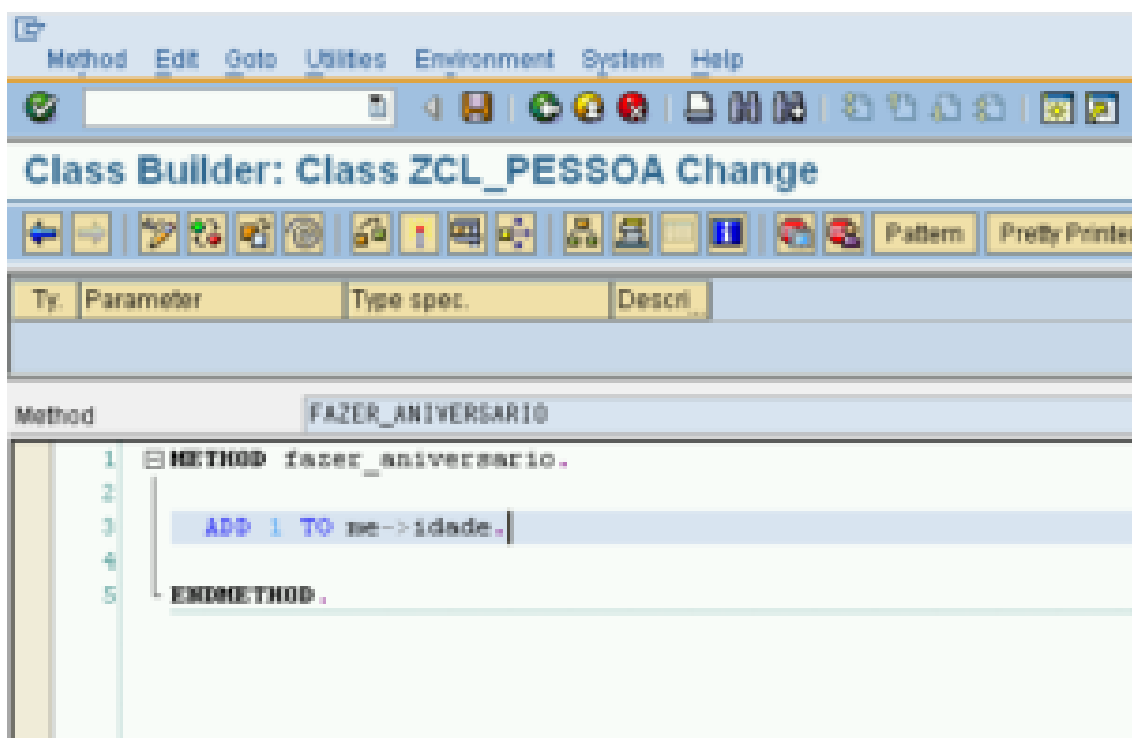
Existem 4 tipo de parâmetros para os métodos: Importing – Entrada de um ou vários valores; Exporting – Saída de um ou vários valores; Changing – Alteração por referencia, exatamente igual nos performs; Returning – Saída de apenas um valor; Neste exemplo estamos implementando o método SET_NOME, este método irá receber um valor através do parâmetro NOME que é do tipo IMPORTING.



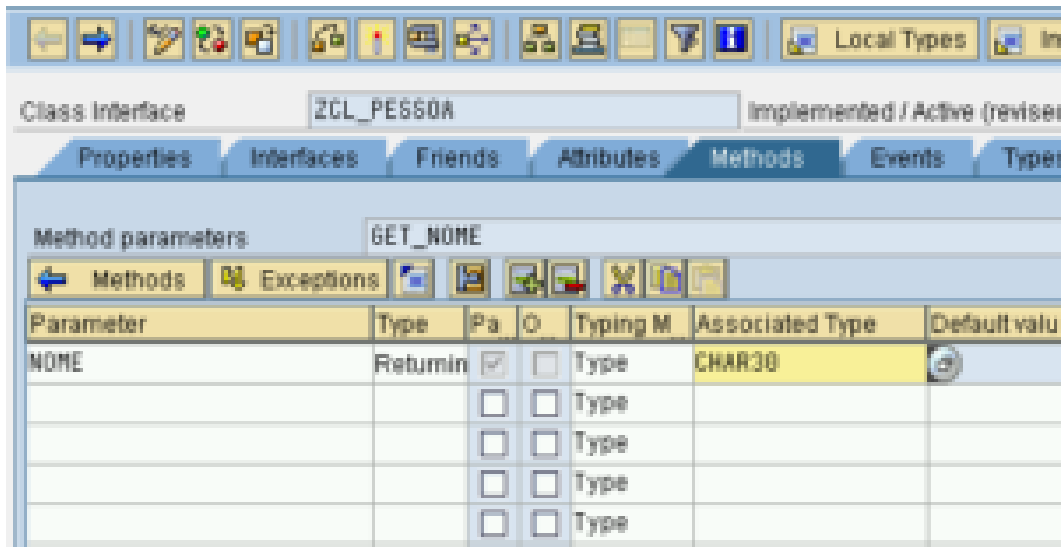
O valor recebido pelo parâmetro NOME irá alimentar o atributo NOME. A referencia “me”, quer dizer que irei atualizar o atributo da instancia atual, ou seja, irei atualizar o atributo NOME do objeto a qual estarei trabalhando. Neste caso ME->NOME (atributo da classe) recebe o valor que estiver em NOME (atributo do método).



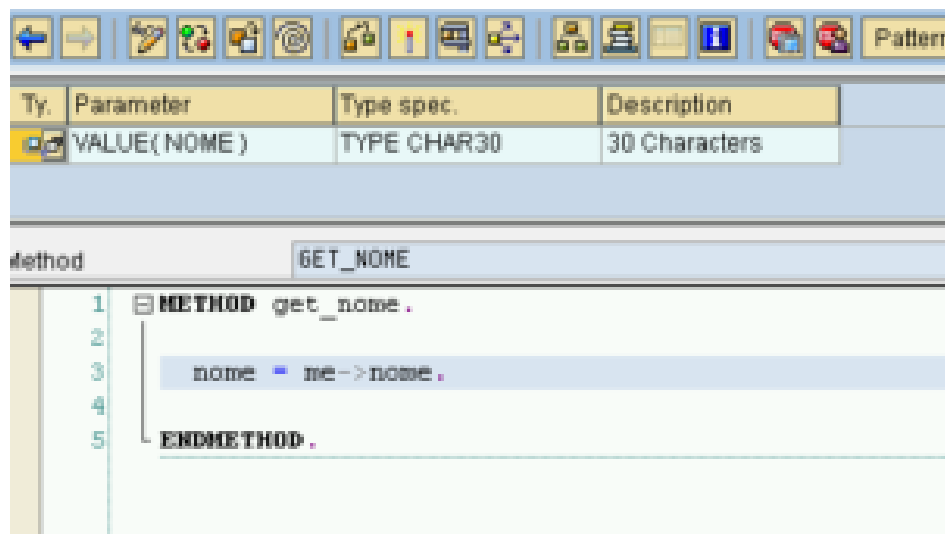
O método FAZER_ANIVERSARIO apenas incrementa 1 para o atributo IDADE.



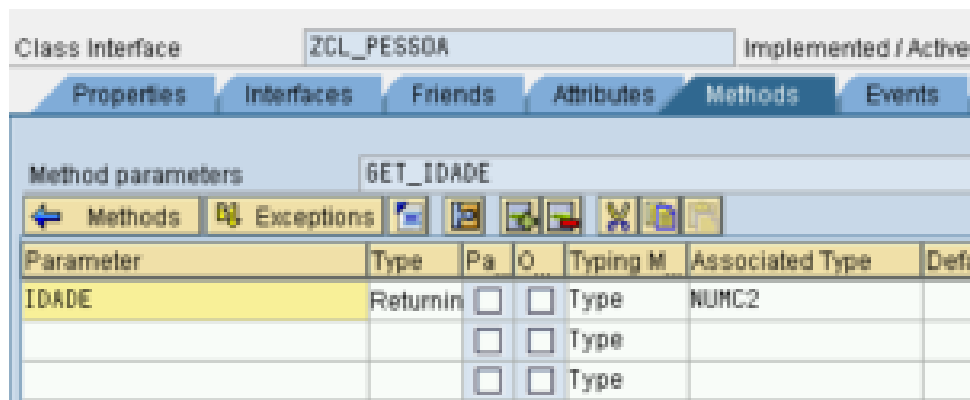
Método GET_NOME retorna o valor do atributo NOME. Como será apenas o retorno de um parâmetro estou usando o Returning.

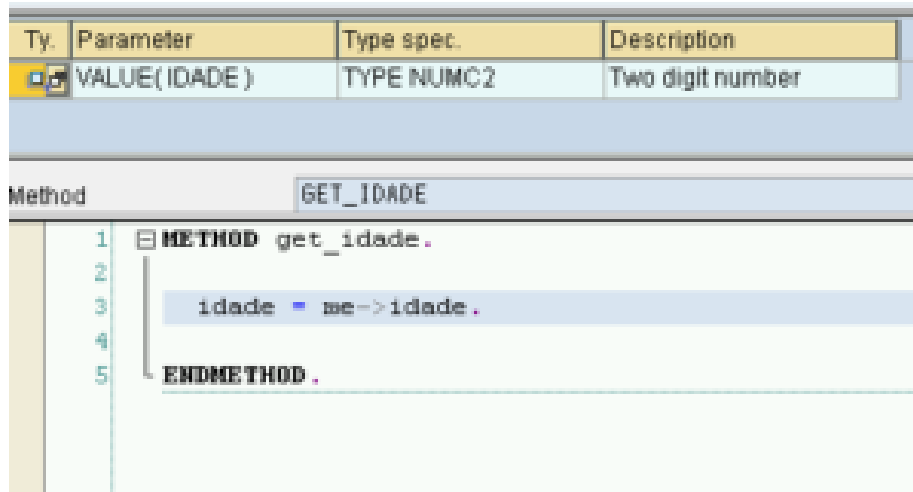


O método, “pega” o valor da instancia atual utilizando o “me” e “joga” no parâmetro “NOME”.

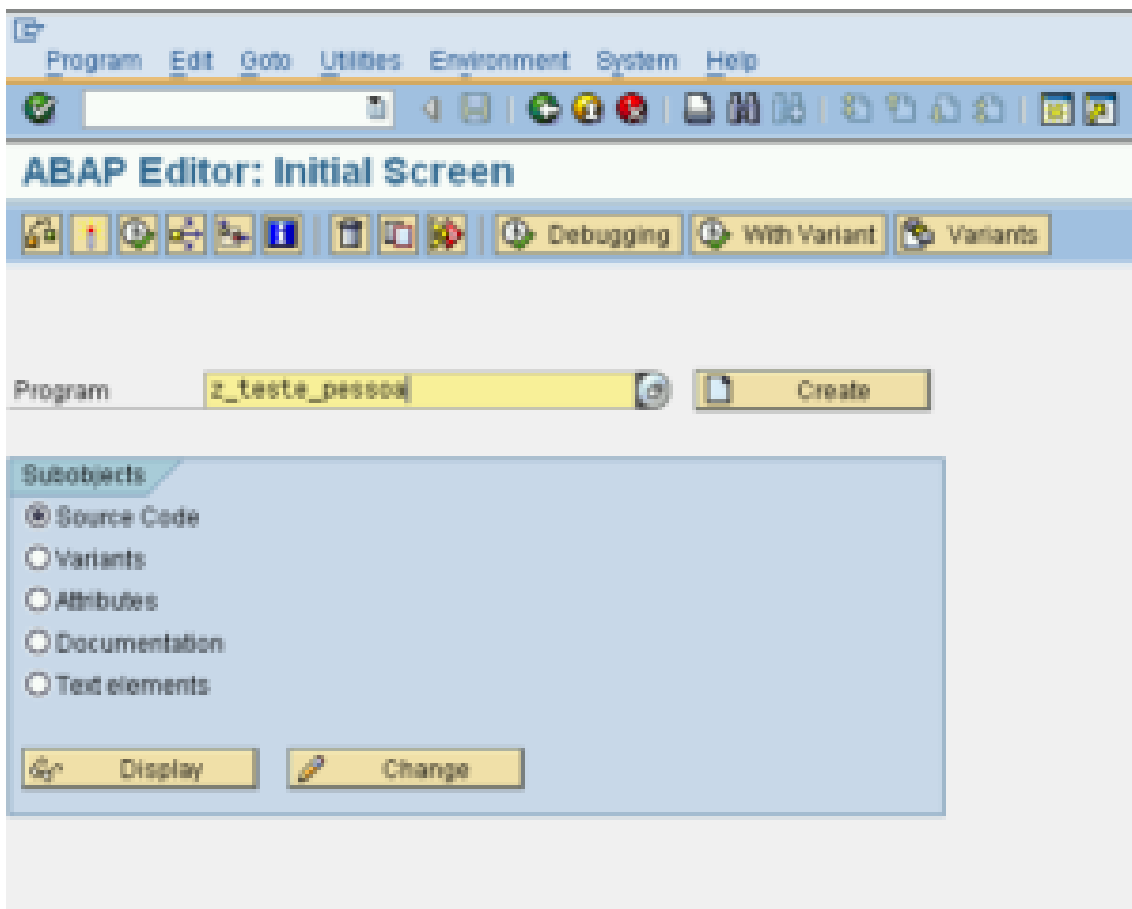


Método GET_IDADE, segue a mesma linha do método GET_NOME, “pega” o valor da instancia atual utilizando o “me”, e retorna para o parâmetro IDADE.



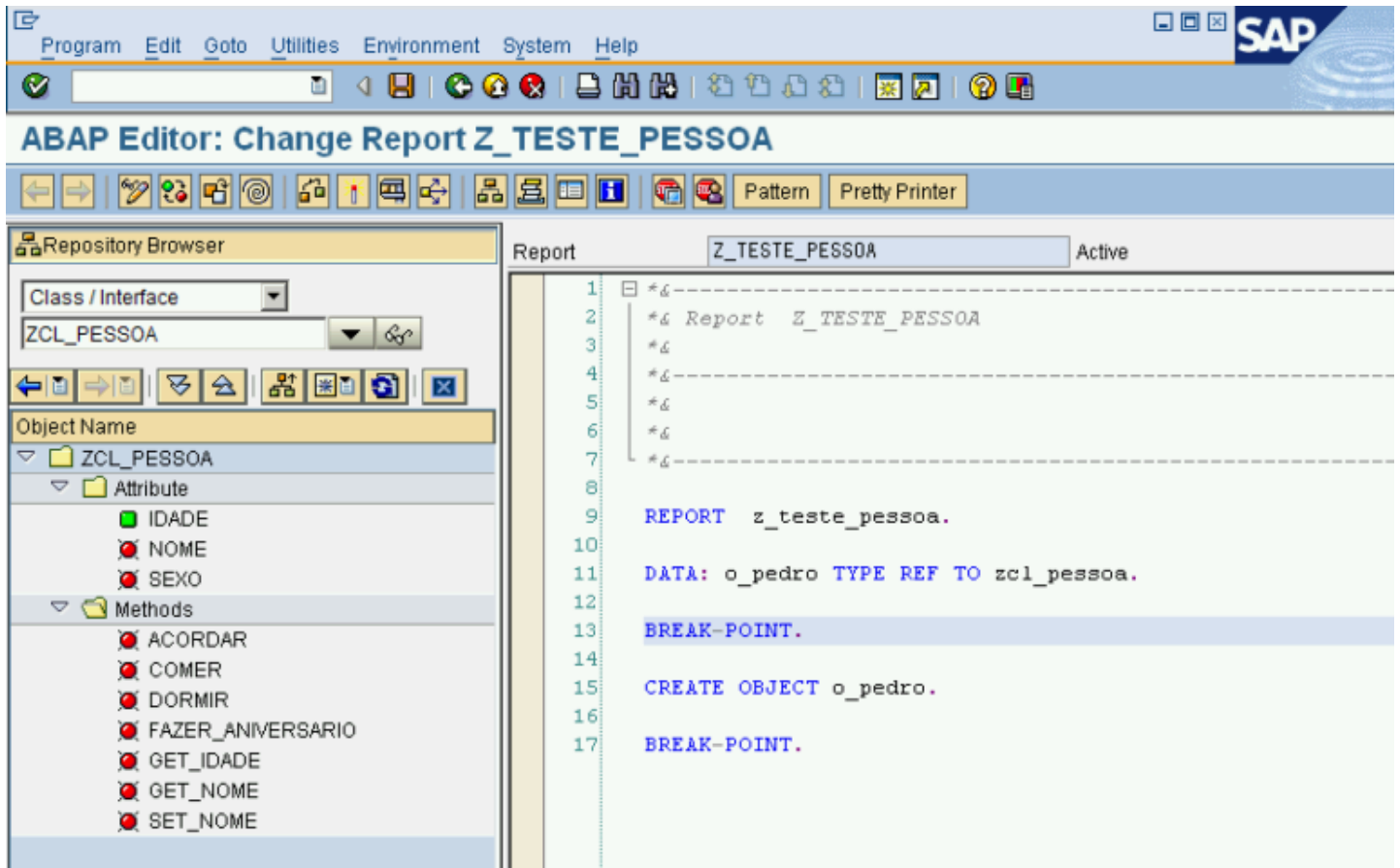


Com a classe feita, vamos fazer a criação do programa na SE38.

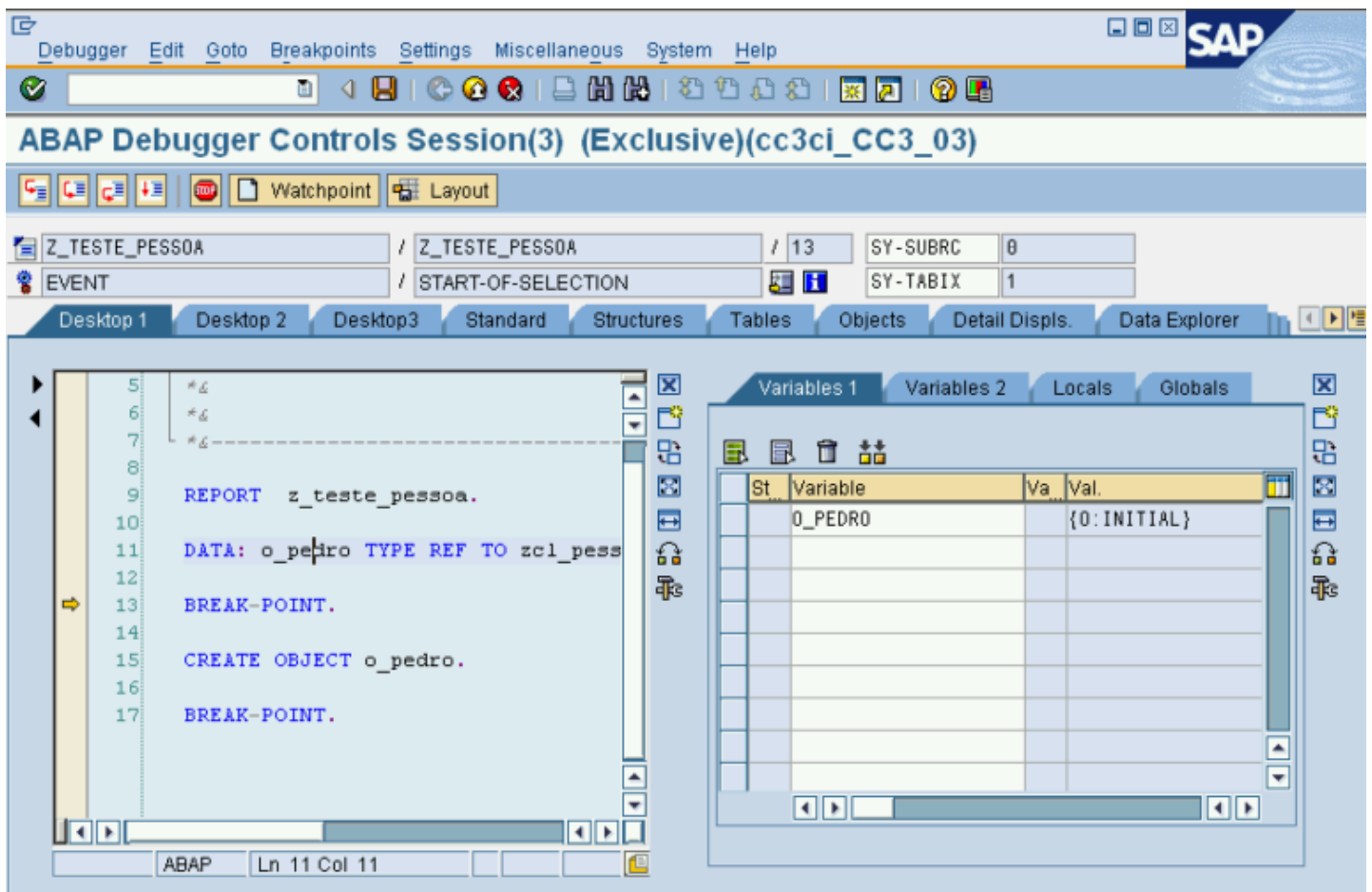


Criei uma variável (O_PEDRO) com referencia da classe ZCL_PESSOA.

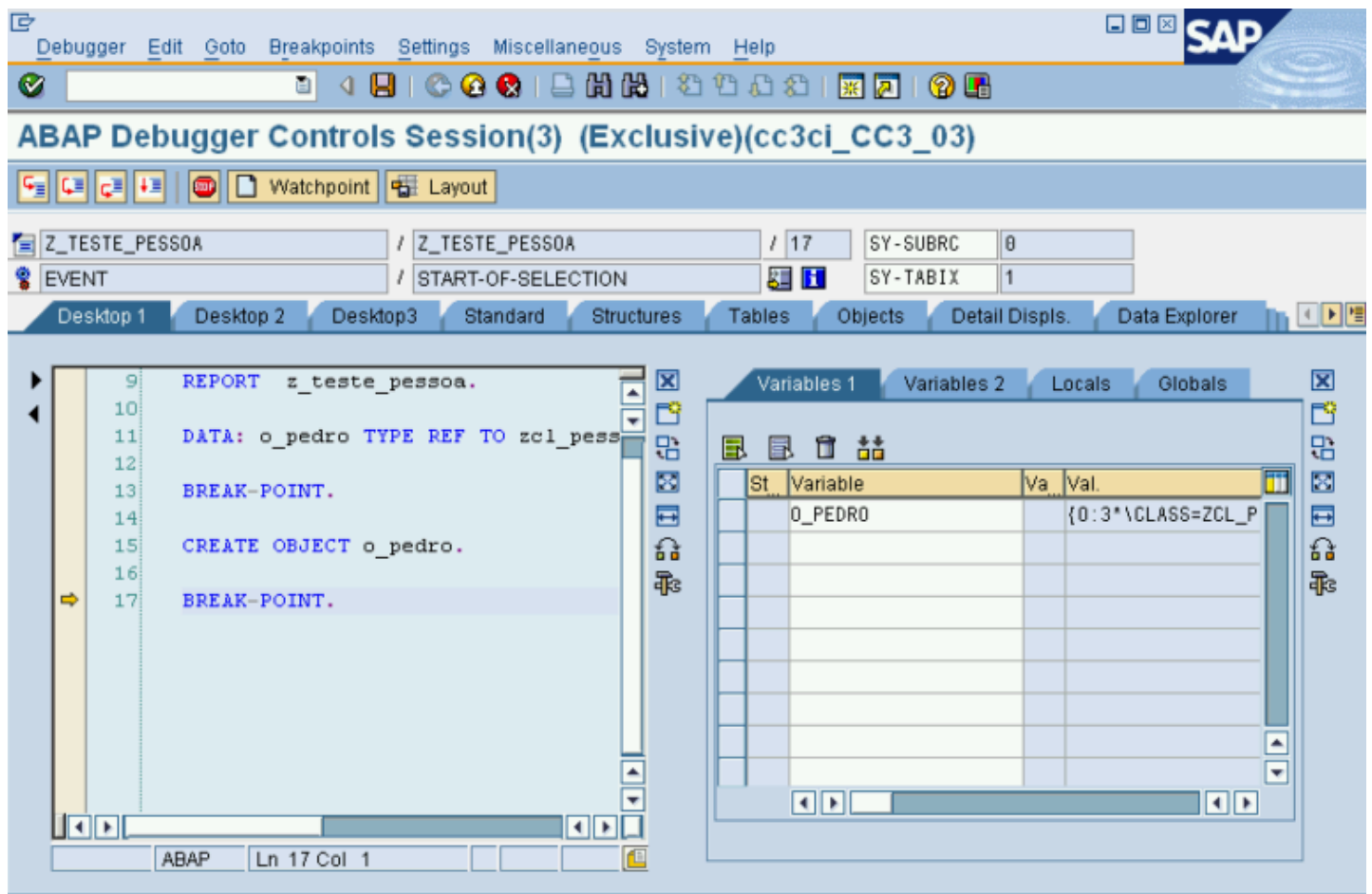
Quando dou o comando CREATE OBJECT, é criada a instancia da classe ZCL_PESSOA, seguindo o conceito do [post anterior](#) a classe ZCL_PESSOA passa de ser apenas um modelo para ser uma instancia, ou seja, um objeto.



Antes do CREATE OBJECT a variável O_PEDRO não tem uma instancia, ainda não é um objeto.



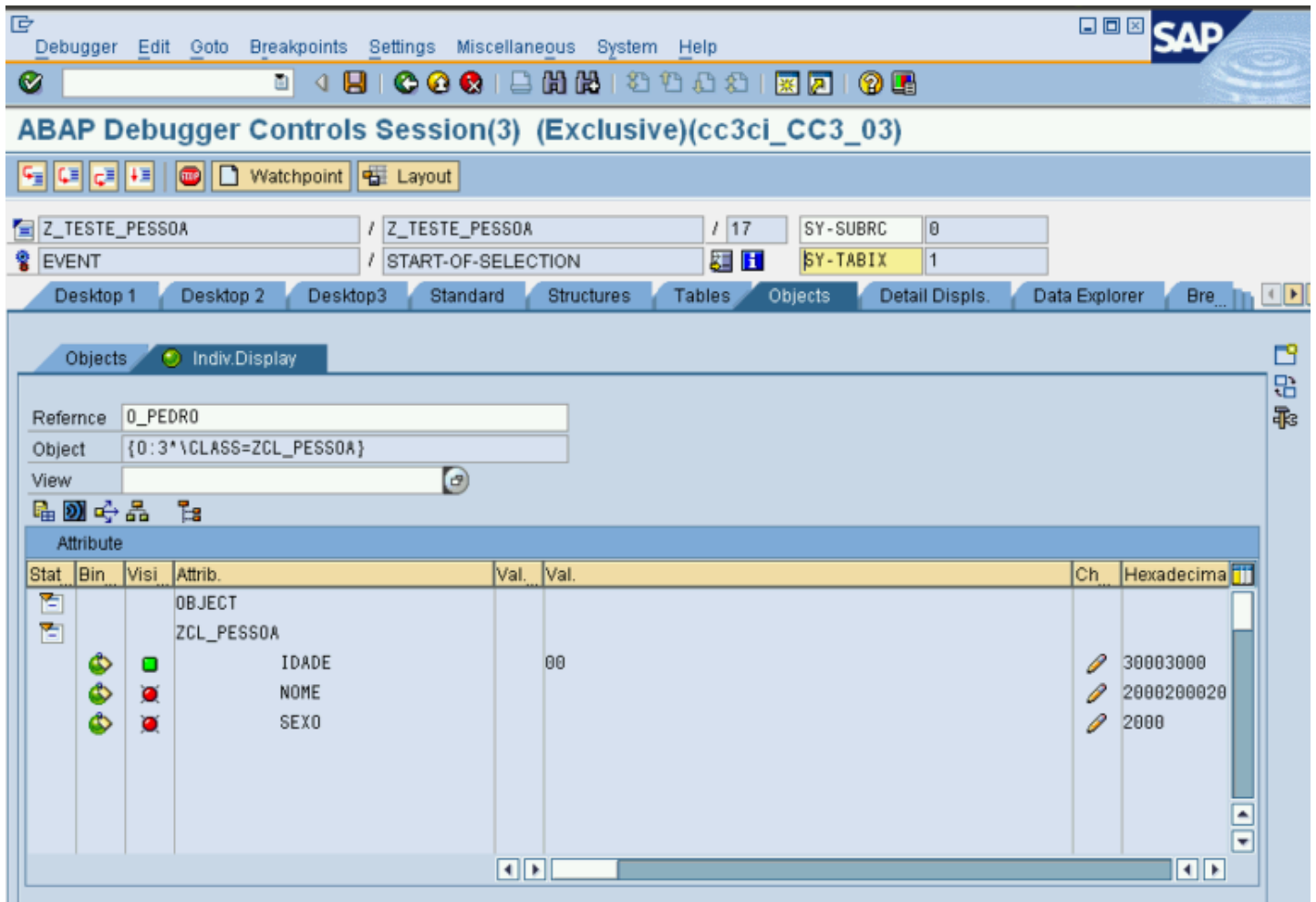
Após o comando CREATE OBJECT a variável O_PEDRO passa a ter uma instancia, começa a ser manipulada como um objeto.



Atributos com o sinal verde significa que são públicos.

Atributos com o sinal vermelho significa que não privados.

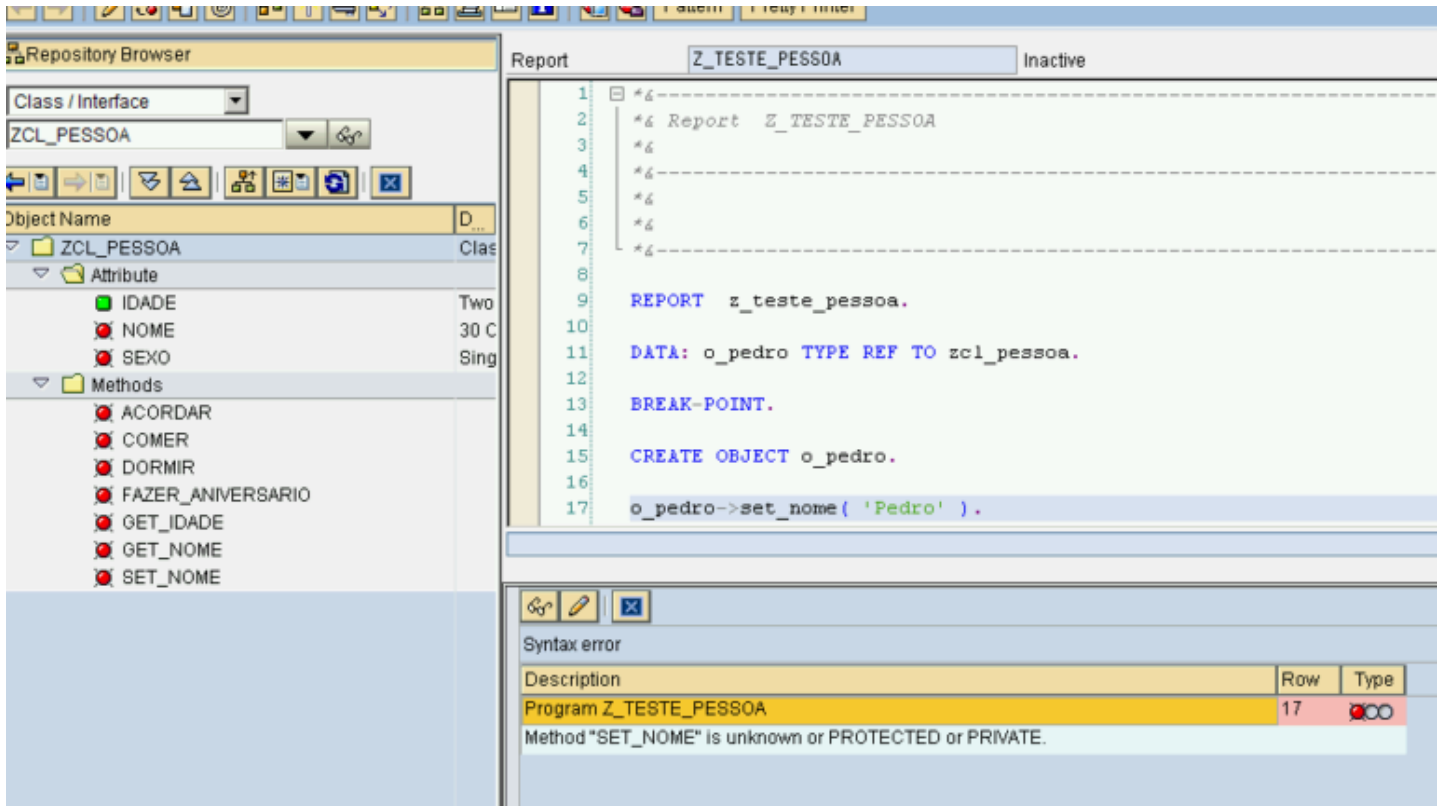
Atributos com sinal amarelo (neste caso não temos nenhum) significa que são protegidos.



Utilizando o método SET_NOME, vamos inserir o nome "Pedro" no objeto O_PEDRO.

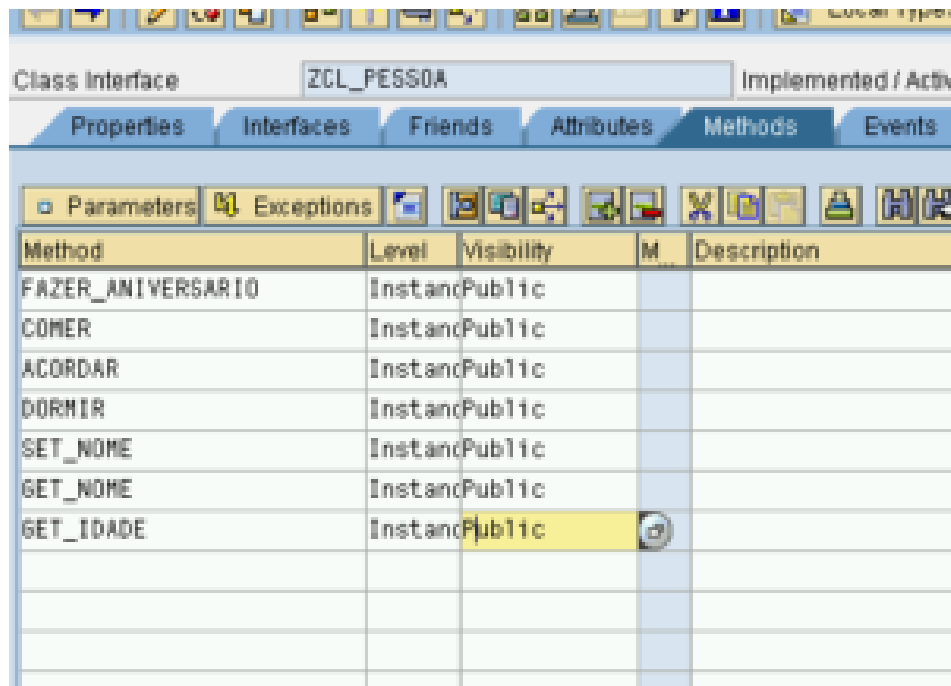
Mas calma, ERROR, NNNNÃÃÃOOO...

É amigo, já começamos com o pé esquerdo, criamos os métodos com visibilidade privada, ou seja, apenas a própria classe poderá utilizá-los. Modelagem é a alma do negócio.



Vamos corrigir estas visibilidade alterando para “public”.

Assim outros objetos também poderão utilizar estes métodos.



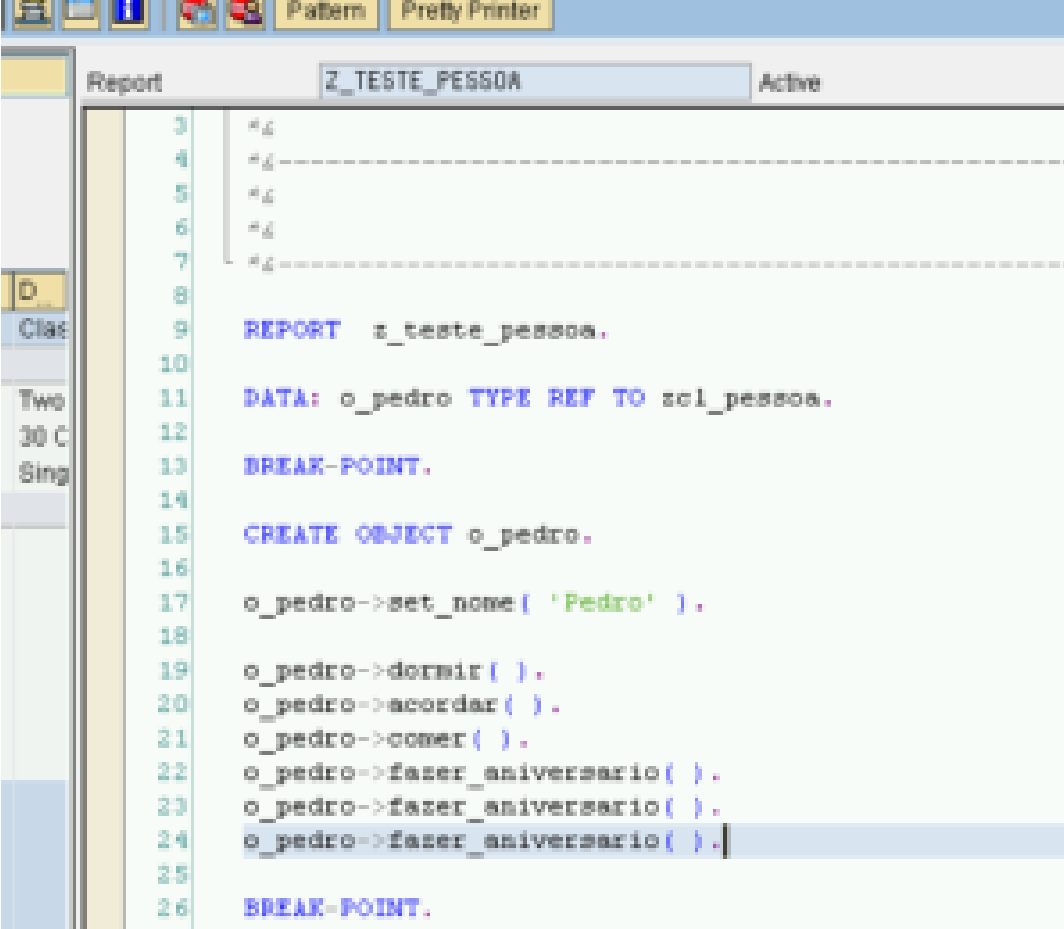
Agora a coisa vai...

Aproveitando, inseri no código outras ações que estão na nossa classe.

Nosso objeto vai “nascer” (ser instanciado), receber um nome, dormir, acordar e fazer aniversário 3 vezes.

Com os 3 aniversários, nosso objeto O_PEDRO terá 3 no atributo IDADE.

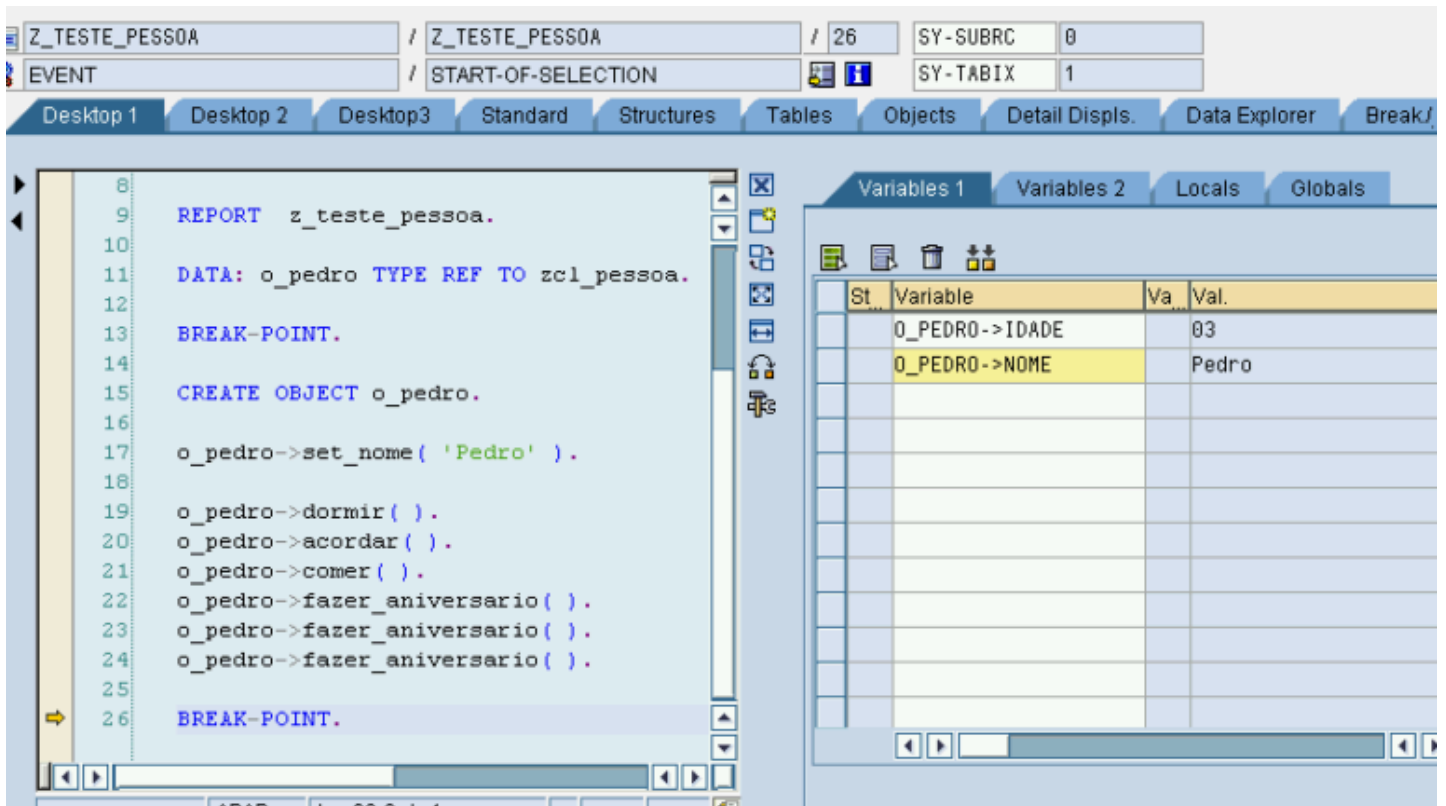
Comments: Entendeu o porque este paradigma de programação chama orientado a objetos? Fizemos nossa classe pensando apenas no objeto pessoa, não pensamos em nenhum momento como a aplicação vai funcionar, o que deve ser executado primeiro. A ideia da Orientação a Objetos é essa, trabalhar apenas com o objeto de uma forma mais genérica, sendo assim, quando outro programa precisar trabalhar com pessoa, terá esta classe. Caso no futuro, a classe pessoa receba alguma melhoria todos os programas que à utilizam, irão estar atualizadas.



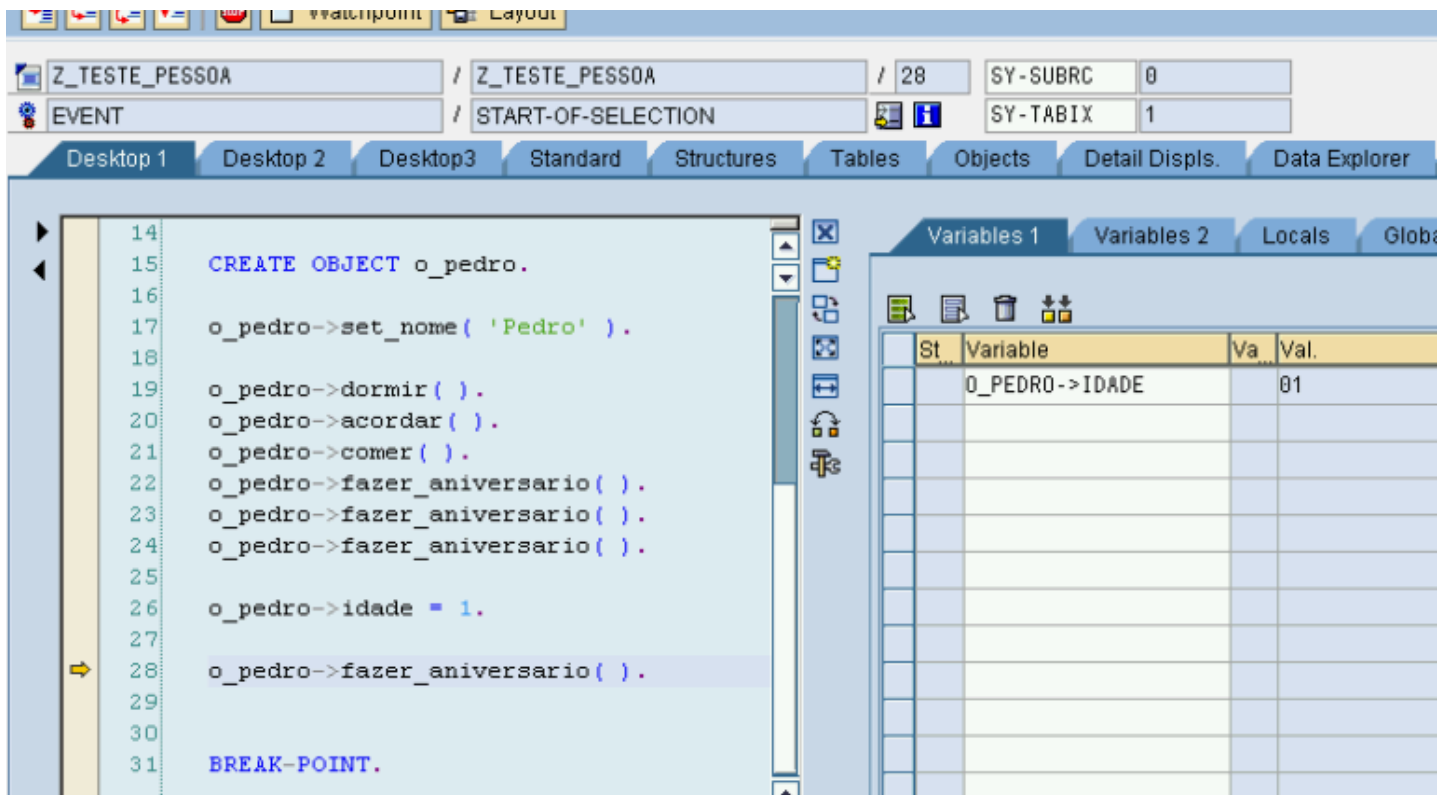
```

3  #c
4  #c-----
5  #c
6  #c
7  #c-----
8
9  REPORT  z_teste_pessoa.
10
11  DATA: o_pedro TYPE REF TO scl_pessoa.
12
13  BREAK-POINT.
14
15  CREATE OBJECT o_pedro.
16
17  o_pedro->set_nome( 'Pedro' ).
18
19  o_pedro->dormir( ).
20  o_pedro->acordar( ).
21  o_pedro->comer( ).
22  o_pedro->fazer_aniversario( ).
23  o_pedro->fazer_aniversario( ).
24  o_pedro->fazer_aniversario( ).
25
26  BREAK-POINT.
  
```

Podemos ver no debugger.



O que?! Outro ERRO, NNNÃÃÃOOOOO.... Mas tinha compilado...

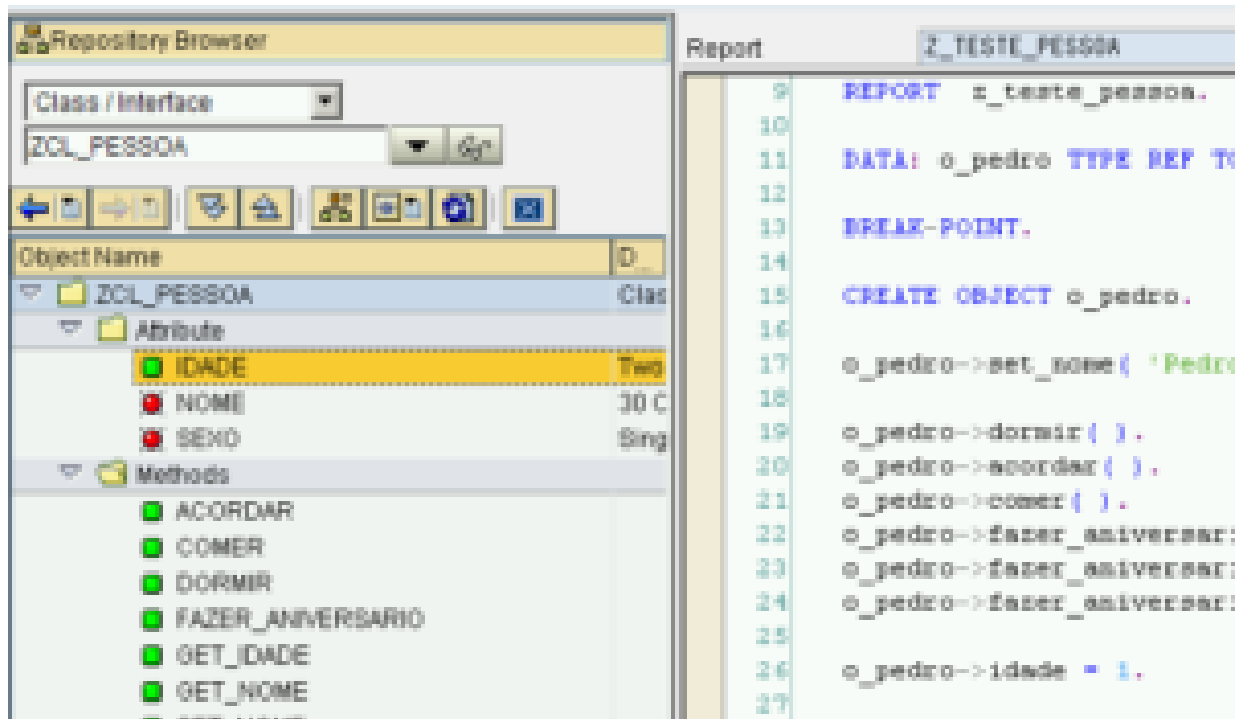


É amigo, temos a regra "a idade de uma pessoa nunca pode retroceder e sempre acrescentar 1 quando o a pessoa fazer aniversário" que por um erro de modelagem pode ser burlada.

Vamos supor que um outro programador está usando a sua classe, ele não conhece as regras para se trabalhar com pessoa.

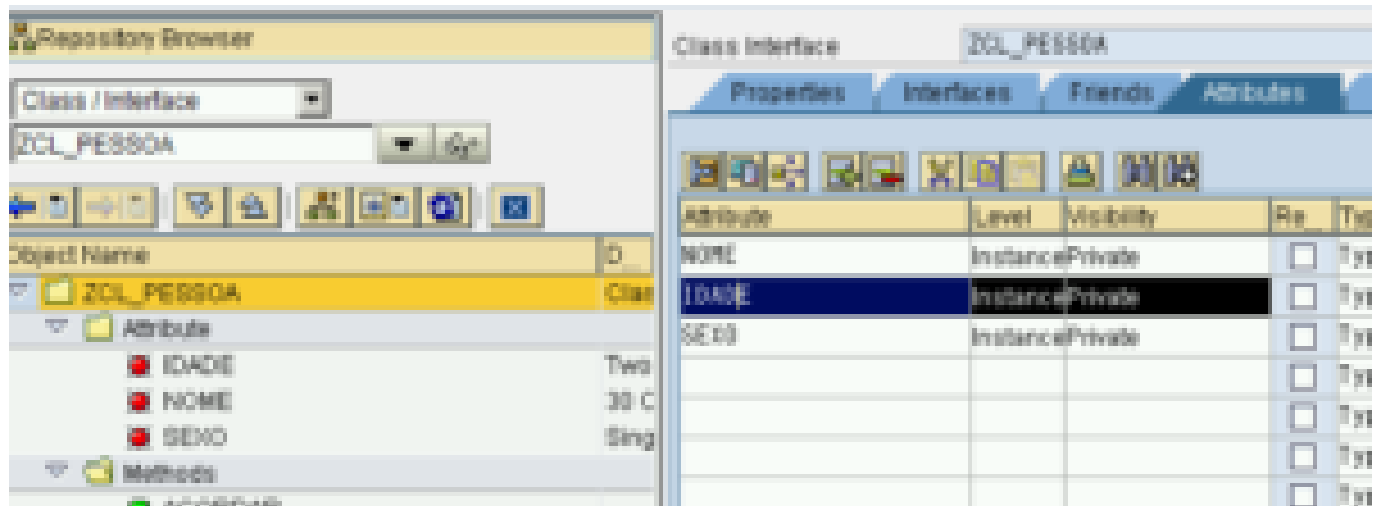
Mas o outro programador viu que o atributo IDADE é publico, sendo assim, não será necessário chamar o método FAZER_ANIVERSARIO para alterar o valor do atributo IDADE.

Sua regra acabou de ir para o espaço. #FAIL

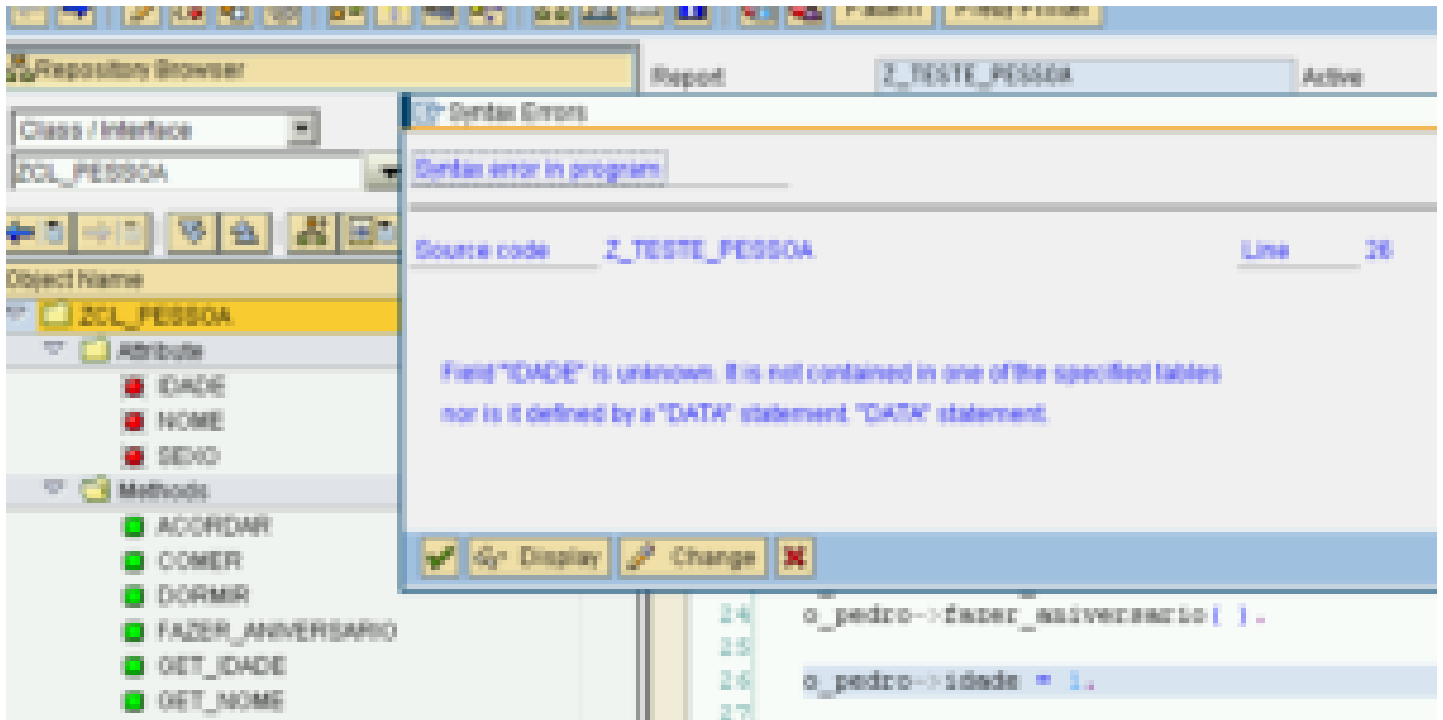


Porem, errando que se aprende...

Vamos alterar a visibilidade do atributo IDADE, sendo assim não será possível alterar o valor da IDADE sem usar o método FAZER_ANIVERSARIO.



Agora temos que corrigir o programa, não será mais possível um outro objeto acessar este parâmetro.



Com a nossa classe melhor modelada para atender as nossas regras, podemos utilizar os métodos SETs e GETs para entender sua real necessidade.

```
REPORT z_teste_pessoa.

DATA: o_pedro TYPE REF TO zcl_pessoa.

DATA: idade TYPE numc2,
      nome TYPE char30.

BREAK-POINT.

CREATE OBJECT o_pedro.

o_pedro->set_nome( 'Pedro' ).

o_pedro->dormir( ).
o_pedro->acordar( ).
o_pedro->comer( ).
o_pedro->fazer_aniversario( ).
o_pedro->fazer_aniversario( ).
o_pedro->fazer_aniversario( ).
o_pedro->fazer_aniversario( ).

nome = o_pedro->get_nome( ).
idade = o_pedro->get_idade( ).

WRITE: nome, idade.
```

Olhe esse exemplo para esse código, podemos utilizar o mesmo “modelo” para trabalhar com N objetos.

Vai lá Abap Espertinho, faz isso utilizando função sem duplicar variáveis ou fazer logica tosca 😊

```
REPORT z_teste_pessoa.

DATA: o_pedro TYPE REF TO zcl_pessoa,
      o_ana TYPE REF TO zcl_pessoa.
```

```

"Nasci Pedro
CREATE OBJECT o_pedro.

"Nasci Ana
CREATE OBJECT o_ana.

"Pedro recebe o nome
o_pedro->set_nome( 'Pedro' ).

"Ana recebe o nome
o_ana->set_nome( 'Ana' ).

"Pedro faz 4 aniversários
DO 4 TIMES.
    o_pedro->fazer_aniversario( ).
ENDDO.

"Ana faz 2 aniversários
DO 2 TIMES.
    o_ana->fazer_aniversario( ).
ENDDO.

```

Galera, por hoje é só...

Qualquer duvida, sugestão ou reclamação só comentar. So não vale ficar Zombizando por aee...

Abraços a todos que aceitaram o desafio de utilizar apenas OO.

Comentários

Renan Arrieiro — 23/06/2016 16:11

Tem como criar os getters e setters automático de acordo com os atributos pelo sapgui ? abraço

Mauro Laranjeira — 23/06/2016 16:22

Renan,

Com grande pesar no coração que te respondo, nem com reza braba =(

Ouvi dizer que no ABAP 7.5 sera possível, o eclipse ja tem isso há anos.

Custodio de Oliveira — 23/06/2016 20:26

So usando persintent class. Mas voce nao quer fazer isso, acredite.

Renan Arrieiro — 23/06/2016 16:29

Eu treino no MINI SAP então não da para usar eclipse e na empresa ainda não tenho as manha para trocar o ide, que é o SAPGUI, então zumbizei =(.

Paulo Macedo — 29/10/2014 15:32

Boa tarde mauro...

me aprofundando um poko mais, teria como eu pegar 4 parametros diferentes (entrado pelo usuario) importar os mesmos para a classe (onde serao tratados) e devolver os mesmos diretamente para um range (ct_dt_inicial type range of date)?

Mauro Laranjeira — 29/10/2014 15:40

Olá meu caro amigo.

Tem varias formas de fazer isso.

Vc pode criar uma classe com um set_valor.. e chamar esse set_valor 4 vezes, depois utilizar um get_range para pegar o range montado.

Vc também pode ter um método que recebe 4 parâmetros de importing, monta o seu range e exporta.

Porem para o get_range o parâmetro de saída tem que ser do tipo exporting.

Caso seja uma tabela e não um range, existe a aba Tipos dentro da classe, que vc pode fazer este tipo de declaração.

Espero ter ajudado.

Abs.

Paulo Macedo — 29/10/2014 15:46

Obrigado mauro, ja deu uma clariada boa.

Muito obrigado

Juliana — 22/10/2014 19:56

Fiz este procedimento também, mas recebo o mesmo retorno.

Juliana — 22/10/2014 19:41

Olá Mauro Laranjeira, estou tentando criar a classe seguindo os passos, porém ao ativar sempre recebo o retorno informando que algum método não foi implementado.

Poderia me ajudar?

Grata.

Mauro Laranjeira — 22/10/2014 19:44

Certo Juliana, da 2 cliques no método que esta exibindo mensagem de erro, e ativa.

Fala ae se rolou 🤔

Renan — 18/07/2013 09:39

E ai Pessoal,

Eu nunca dei muita bola pra OO nem UML nem essas “bagacera”, porque herança, encapsulamento, polimorfismo e etc nunca fizeram sentido nem valiam o esforço de programar OO somente por programar OO.

Mas meu pensamento mudou a partir do momento que li esse artigo:

*Especialmente quanto aos “sintomas de apodrecimento de software”

http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

Nesse mesmo link acima, é possível encontrar muitos motivos para pensar e “viver” OO. Especialmente com os princípios introdutórios OCP e DIP.

Nada de OO fez sentido antes de eu ler isso, isso foi primordial e essencial para eu aceitar OO como uma necessidade.

Além disso, recomento o site zevolving.com , tem muito material abap OO avançado, especialmente sobre design patterns.

Renan

Wesley Fernandes — 16/01/2017 17:00

Valeu Mauro pelo post!

Compartilho da sua opinião, e o que tem me ajudado muito a entender isso tudo são os posts do BAr8, eles fizeram uma sequencia bacana sobre S.O.L.I.D

<https://bar8.com.br/abap-oo-solid-a8bfd8c5191b>

Flw[]

Wanderson — 20/10/2013 14:57

Concordo com o Renan, os conceitos de OCP e DIP não só ajudam a entendermos a importância da programação orientada a objetos como nos guiam para realizá-la de forma eficiente.

Gustavo Venceslau — 17/07/2013 11:40

Já tem a parte 3 ?

Camila — 10/09/2012 14:43

Excelente a explicação, Mauro.

Muito obrigada.

Fico em dúvida ainda quanto a por na prática. Como começa, pegar um gap e fazer em OO. Tem alguma restrição quanto a decisão de fazer em OO ou estruturado?

Peguei um programa seu para fazer manutenção há uns 2 anos e fiquei perdida na época . Agora vc me ajudou hehehe...

Valeu

Mauricio Cruz — 11/09/2012 10:14

Só pra constar eu dei risada quando você falou que sofreu com um programa OO do Mauro. Algum dia espero alguém falar isso de um programa meu tb mwahahahaha (pega essa risada maléfica 😊)

Flávio Furlan — 19/07/2012 18:19

Muito boa explicação, gostei também do exemplo do Maurício sobre os métodos de acesso, GET e SET.

No mundo OO, os métodos GET e SET são amplamente usados, justamente para proteger os dados da classe. Podem parecer burocráticos, mas te dá um chance de programar alguma regra de negócio na saída do método.

Por exemplo no Cocoa (programando para iPhone), você tem uma declaração especial que você coloca no equivalente ao CLASS... DEFINITION do Objective-C que faz com os métodos GET e SET sejam declarados automaticamente: @synthesize

Ótimo artigo!

Abraços!

Mauricio Cruz — 23/07/2012 10:24

Em outras linguagens é bem comum mesmo usar GET e SET para praticamente tudo.

É engraçado que, quanto mais eu estudo outras linguagens, mais eu percebo que a galera do ABAP “se isola” porque quer. Dá pra fazer tudo (ou quase tudo), com a mesma qualidade.

Eu só sinto falta de uma IDE melhor para escrever o código, não acho a SE80 suficiente no quesito produtividade. Ela é boa para várias coisas, mas pra escrever o código ainda acho um pouco pobre. Porém, na nova versão do AS o novo editor vem com umas coisas animais para acelerar o desenvolvimento com classes (meu minisap que o diga 😊).

Abraços!

Custodio — 24/07/2012 01:26

Vou insistir no meu ponto: Setters = ok, Getters = superfluo.

Se voce quer olhar outras linguagens, vera que get eh praticamente apenas para retornar o valor do atributo. Implementar business logic no get eu nunca vi. E como o ABAP te proporciona o attributo public read only, por que nao usa-lo? Eh uma vantagem em relacao a outras linguagens. Nao eh porque os outros fazem que temos que fazer tb, ne?

Sobre a IDE, vc já deve ter visto que o ADT/AiE está disponível. Estou com o pé atrás, mas vou tentar na boa vontade. Mas acho que era melhor investir mais na SE80 que arrumar uma outra ferramenta...

Mauricio Cruz — 24/07/2012 08:02

Interessante ver que você não gosta do GET, ele praticou bully com você quando você estava na escola?
😏😏

Sempre usei o GET e vou continuar usando. Usar o READ-ONLY é mais performático, porque vai matar N chamadas que você faria no método GET_blablabla. Porém, isso também diz que o programa que implementa a classe vai ter que saber como e quando utilizar aquele valor. Supondo que eu queira que um valor somente exista quando eu for buscá-lo, posso implementar a lógica do select num método que vai ser executado antes do valor ser retornado no GET, caso o atributo esteja vazio.

Gosto da idéia de ter possibilidades, e, para mim, é exatamente isso que o GET faz.

Mas não acho errado usar READ-ONLY não, se você gosta vai na fé. IO importante é ser feliz 😊

Custodio — 29/06/2012 04:21

Excelente post e boa pergunta do rapazinho ae 😊

Eu costumo não usar os GETs, apenas os SETs (exceto quando uso persistence class, aí não tem jeito). Defino os atributos como públicos e read-only e está tudo certo. Nunca vi em classe nenhuma nenhum GET que não seja:

```
method get_abc returning abc.  
abc = me->abc.  
end_method.
```

Se alguém achar me apresenta por favor 😊

Já os SETs são sim importantes, quase que indispensáveis.

Abracos.

Fawcs — 27/06/2012 19:10

pô, que preconceito é esse com pessoas de mais de 100 anos?

Brincadeiras à parte, eu fico meio na dúvida com esses métodos set e get.. qual é a grande vantagem deles? pq não colocar o atributo logo como público?

A única razão que eu vejo para isso seria colocar regras de negócio no método set, correto?

Mauricio Cruz — 27/06/2012 21:28

Divertida a sua pergunta 😊 Essa é uma dúvida comum para as pessoas que não estão familiarizadas com OO.

O Mauro e outros podem depois complementar minha resposta: Com os métodos get e set, quem controla **como** o atributo vai ser lido/gravado é a classe. Você garante que o valor do atributo sempre vai ser gravado do jeito que a classe quiser, e que o nome retornado vai ser o que a classe quiser.

Imagine que, você pegou essa classe `ZCL_PESSOA` e a utilizou por todo o sistema. Você busca o nome no objeto `O_PESSOA` pelo `GET_NOME`, e faz o print desse nome na tela, em vários programas/telas/forms/métodos diferentes. Um belo dia, alguém vem para você e diz que, a partir de agora, os nomes tem que aparecer sempre em maiúsculo. Como é a classe `ZCL_PESSOA` que controla como o nome vai ser lido, através do método `GET_NOME`, tudo que você precisa fazer é transformar o nome em maiúsculo dentro daquele método, e todo o seu sistema vai estar funcionando.

Porém, se você estivesse acessando diretamente um atributo público, você teria que transformar TODA SANTA VEZ o valor do nome para maiúsculo, o que é tosco demais, não é?

Desculpe se o exemplo estiver confuso, está tarde e estou com sono 😊 hahahaha

Abraços!

Fawcs — 28/06/2012 10:30

Entendi, é quase o que eu falei só que ao contrário(wtf!)?! Entao você pode tanto fazer validações no método `SET` quanto fazer definições de output no `get` (conversão de unidades?).

Ah sim, uma bruxaria que eu aprendi ontem no SAP. Se você declarar o parametro de importação do método `set_nome` como `TYPE c`. ele vai formatar automaticamente o tipo na hora da importação! Lembrei disso olhando os `importing ali` do método `set_nome` =)

Lucas Sales — 17/06/2016 13:31

E aqui estamos basicamente 4 anos depois deste post com este comentário ir ao ar para comentar que sabia que já havia lido sobre esse tema de `GET` e `SET` em algum lugar, lembro que na época a dúvida do fawcs era exatamente a minha, mas agora estamos em 2016 e ainda vejo posts como esse:

<https://bar8.com.br/o-que-%C3%A9-um-objeto-mesmo-7650ce5c59d2#.xptip0q6s>

Neste post é comentado sobre `SET` e `GET` e seu uso indiscriminado. Mas o mais incrível/bizarro é posts como esse parecerem extremamente novo, sofisticado e alguns diriam até de outro mundo,

ABAPer é realmente um ser humaninho muito engraçado, será que daqui a 4 anos ainda vamos ter conceitos tão simples como este de OO sendo tão obscuros?

Mauro Laranjeira — 28/06/2012 11:32

Fawcs,

Fiquei muito feliz que vc chegou nessa conclusão, sério, só por isso, ja acho que o post já valeu a pena

=D

Vlw e Abraços