

Braxis | IT Services

# Treinamento Abap Objects

## Braxis

***Material elaborado pela Braxis IT Services***

***Data : (26/09/2006)***








## Histórico de Alterações

| Data | Versão | Descrição | Autor |
|---|---|---|---|
| *26/09/2006* | *1.0* | *Treinamento Abap Objects* | *Fábio Ferri* |
| *01/12/2006* | *2.0* | *Alterações e correções* | *Fábio Ferri* |

## Sumário


1. **OBJETIVO**
2. **INTRODUÇÃO AO ABAP OBJECTS**
3. **ANÁLISE E DESENVOLVIMENTO**
4. **PRINCÍPIOS DA PROGRAMAÇÃO ORIENTADA A OBJETO**
5. **HERENÇA (INHARITANCE)**
6. **CASTING**
7. **INTERFACES**
8. **EVENTOS**
9. **CLASSES GLOBAIS E INTERFACES**
10. **TÉCNICAS ESPECIAIS**
11. **MANIPULANDO EXCEÇÕES**
12. **EXERCÍCIO EXTRA**
13. **TESTES**




*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 1. OBJETIVO

Este material tem o objetivo de demonstrar as técnicas em ABAP na versão 4.7 com programação orientada a objeto. Demonstrando conceitos e exercícios práticos, com o objetivo de iniciar os participantes ao ABAP OBJECTS.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Introdução a programação orientada a objetos

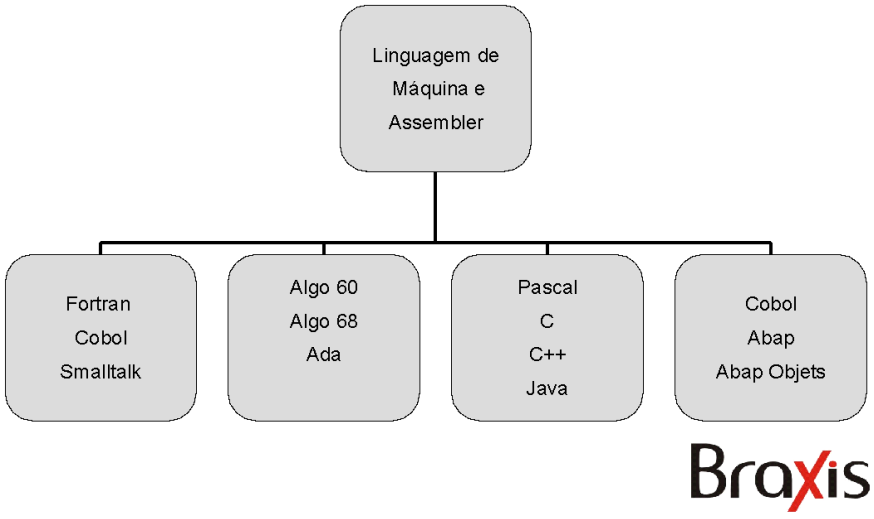
- Conteúdo
  - Programação Procedural
  - Programação Orientada a Objetos
  - Alvos da linguagem Orientada e Objetos

Braxis

Data: 10/05/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## História das Linguagens de Programação




Data: 10/05/06

- Antes ABAP, SAP utilizavam macro Assembler.
- **ABAP** foi criado com a intenção de desenvolver customizações e enhacements e relatórios. A maioria dos desenvolvedores ABAP, foram influenciados por outras linguagens como PASCAL e COBOL.
- **ABAO Objects** é uma extensão do ABAP. Abap objects unifica a maiorias dos aspectos de outras linguagens de programação orientada a objeto, como JAVA, C++ e Smalltalk.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Características Programação Procedural

- **Características:**
  - **Separação de funções e dados**
  - **Possibilidade de encapsulamento de funções usando moduralização**
  - **Acesso direto aos dados possibilitando visibilidade**

Braxis

Data: 10/05/06

- A informação  dos sistemas eram previamente definidas pelas funções. Então dados e funções foram armazenados separadamente e ligados utilizando parâmetros.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Típico programa ABAP

```
REPORT ZABAP_DEMO
*---------------------------------------
TYPES: -----

DATA: ------

PERFORM form1...

CALL FUNCTION 'FB1'

CALL FUNCTION 'FB2'
```

- Definições de Tipos
- Declaração de Dados
- Programa principal
  - Chamar Subrotinas
  - Chamar funções
- Definição de Subrotinas

Braxis

Data: 10/05/06

- Uma típica programação ABAP consiste de declarações de tipos e dados e lógica de processamento.
- Para fazer seu programa mais legível e melhor programação estruturada, é recomendado que você trabalhe em módulos (unidades encapsuladas com funções) como forms e funções. **Estes componentes podem ser reutilizáveis**.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Exemplo de Grupo de Função

```
FUNCTION-POOL s_vehicle
 * Speed   é uma variável global
 * usada no function-pool

DATA:  speed TYPE I.

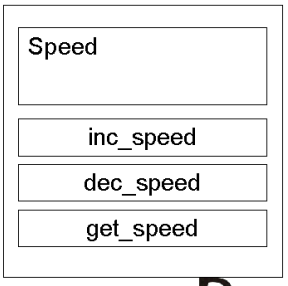
FUNCTION INC_SPEED
 add imp_speed TO speed.
ENDFUNCTION

FUNCTION DEC_SPEED
 subtract imp_speed TO speed.
ENDFUNCTION

FUNCTION GET_SPEED
  exp_speed = speed.
ENDFUNCTION
```

**Grupo de Funções com funções para controlar a velocidade do carro**

**S_VEHICLE**

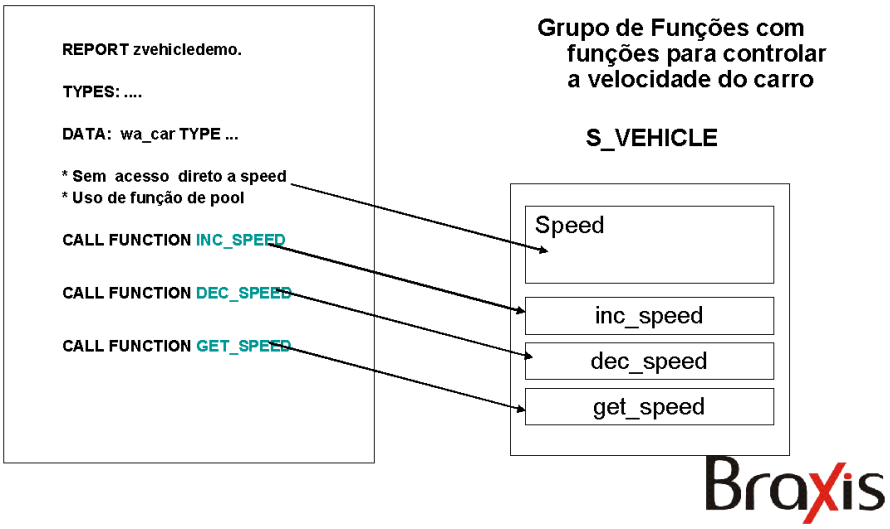



Data: 10/05/06

- O grupo de função S_VEHICLE provê um usuário ou client com serviços *inc_speed, dec_speed, e get_speed.*
- O grupo de função faz uma interface para acessar internamente um componente chamado *speed*.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

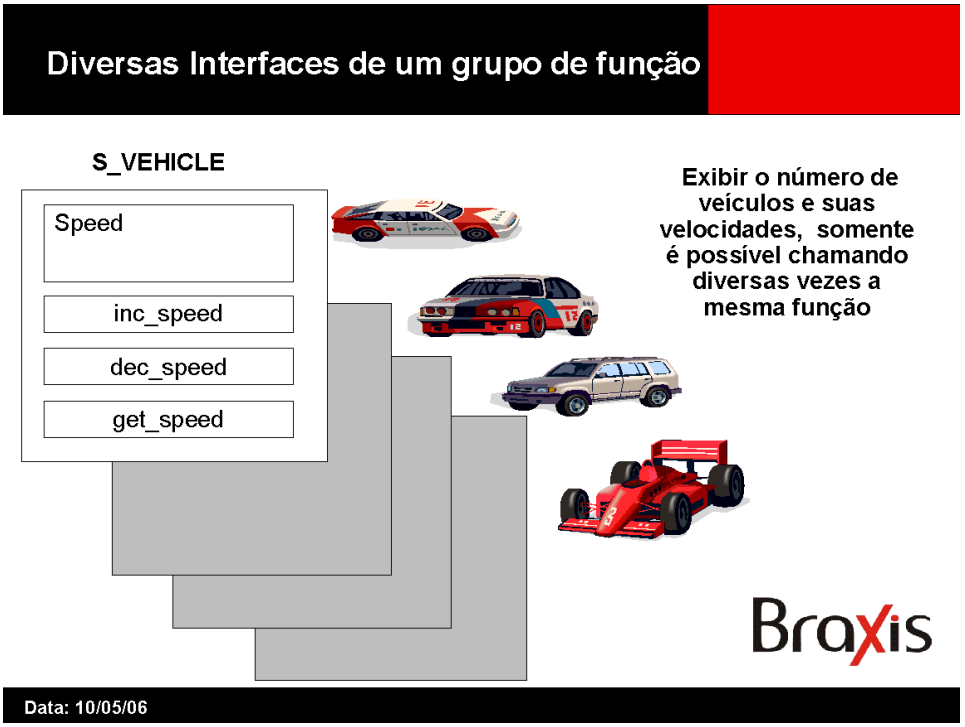
## Uso do Grupo de Função




Data: 10/05/06

- O programa principal não pode acessar *speed* diretamente.

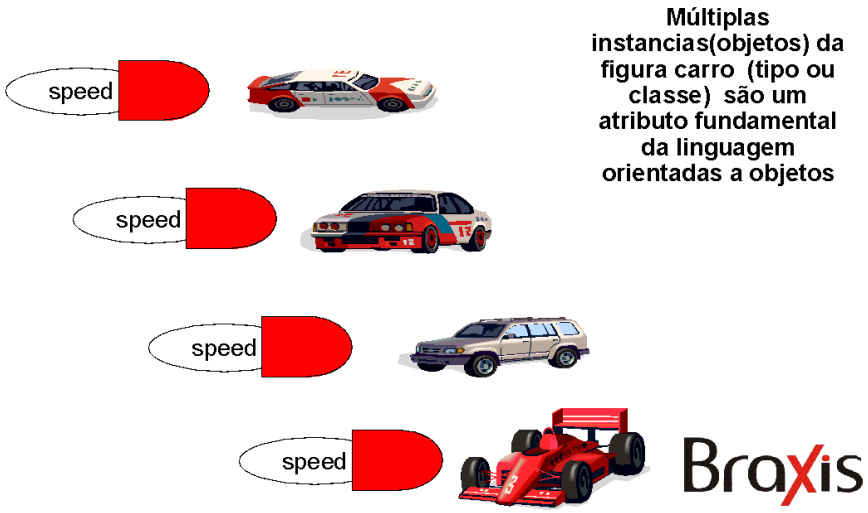

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*




- Se o programa principal está trabalhando com diversos veículos, isto não é possível sem programação extra.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

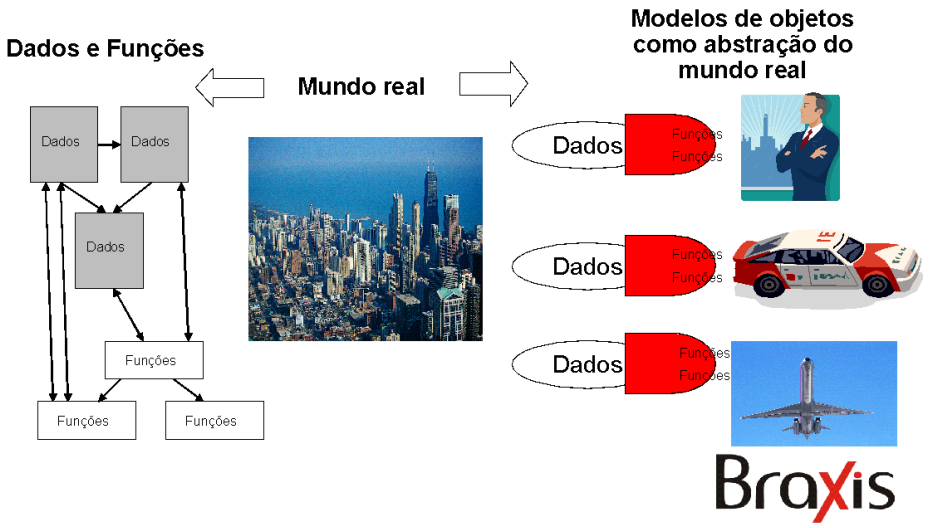
## Instancias e Linguagens Orientadas a Objetos




- A habilidade de criar múltiplas instancias de uma classe, como a vehicle, é um dos principais características de linguagens orientadas a objeto.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Modelos de Programação




- A parte esquerda do slide mostra que, com a forma procedural, dados e funções são:
  - Criados separadamente
  - Armazenados separadamente
  - Ligados com parâmetros

- As cápsulas a direita em vermelho e branco contém os dados e seus comportamentos na própria cápsula. Objetos permitem um melhor desenho no modelo de software, refletindo melhor o mundo real.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Características da Orientação a Objeto

**Mundo real**



- **Objetos são abstrações do mundo real**
- **Objetos são unidades que representam dados e funções**
- **Melhora a estruturas de software**
- **Redução de custo de manutenção**

Braxis

Data: 26/09/06

- Na programação OO dados e funções são desenvolvidas juntamente. Onde o foco de desenvolvimento é representar coisas do mundo real. A principais características são a organizações dos dados(Atributos) e métodos(funções) .
- Consistência através processos de desenvolvimento de software:

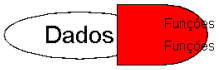
  A linguagem é usada em várias fazes do desenvolvimento ( Analises, Especificação, Desenvolvimento e Implementação) é uniforme.
- O poder desse conceito:

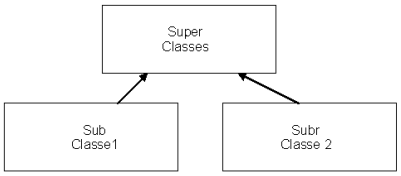
  - Implementação dos processos próximos ao negócio, melhor envolvimento entre modelador e desenvolvedor.
  - Melhora a estrutura e manutenção de software e reduz o trabalho requerido.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Modelos de Programação

- **Encapsulamento de funções e dados.**
- **Polimorfismo ( programação genérica)**
- **Herança**




Data: 26/09/06

- **Encapsulamento:**

  Encapsulamento é uma implementação de um objeto que está oculto para outros objetos. Somente permitindo alguns funções(ações) que estão permitidas para os objetos externos. Dois bons exemplos de objetos encapsulados são (BOR – Bussiness Objetc Repository, utilizados em Workflow, LSMW) e o outro são as BAPI's – Bussiness Aplpication Program Interfaces).

- **Polimorfismo:**

  Polimorfismo é habilidade de programação em múltiplas formas em OO. Isto significa que diferentes (Classes e Métodos) reagem de diferentes formas de acordo com definição de sua classe(implementação) .

- **Herança:**

  Herança define a implementação de relacionamento das classes. Onde serão definidas as hierarquias de SuperClasses e SubClasses.
  OBS: Em Abap Objetcs somente é permitido simples herança. Não é possível fazer herança múltipla.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Desenho e Compatibilidade

```
* Classe ABAP

DATA:  cont  TYPE n.

CLASS  lcl_car
 ....
ENDCLASS

CREATE OBJECT ....
CONT = CONT + 1.
```

- **Compatibilidade:**
  - **È uma extensão do ABAP**
  - **Declarações ABAP Objetcs são usadas em programas ABAP.**
  - **Classes ABAP contém declarações de OO e ABAP**
- **Desenho:**
  - **Simples e Fácil**
  - **Conceitos de OO tem um forte poder de utilização**
  - **Aumento de verificações e validações e código.**

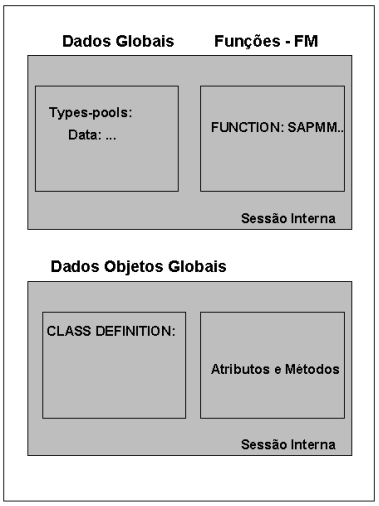
Braxis

Data: 26/09/06

- Abap Objects é apenas uma extensão do atual ABAP.
- Em Abap Objects  as declarações dos tipos são mais criteriosas do que o ABAP. Por que, quando definimos, por exemplos: parâmetros de métodos de  interfaces, você deve declarar os parâmetros de forma correta , para que as implementações das classes possam utilizar está mesma interface.
- Abap Objects tem um código mais limpo. Pelo fato do código ser mais limpo, em tempo de execução somente serão executados as verificações e ações anteriormente definidas.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Memória ABAP e Encapsulmento



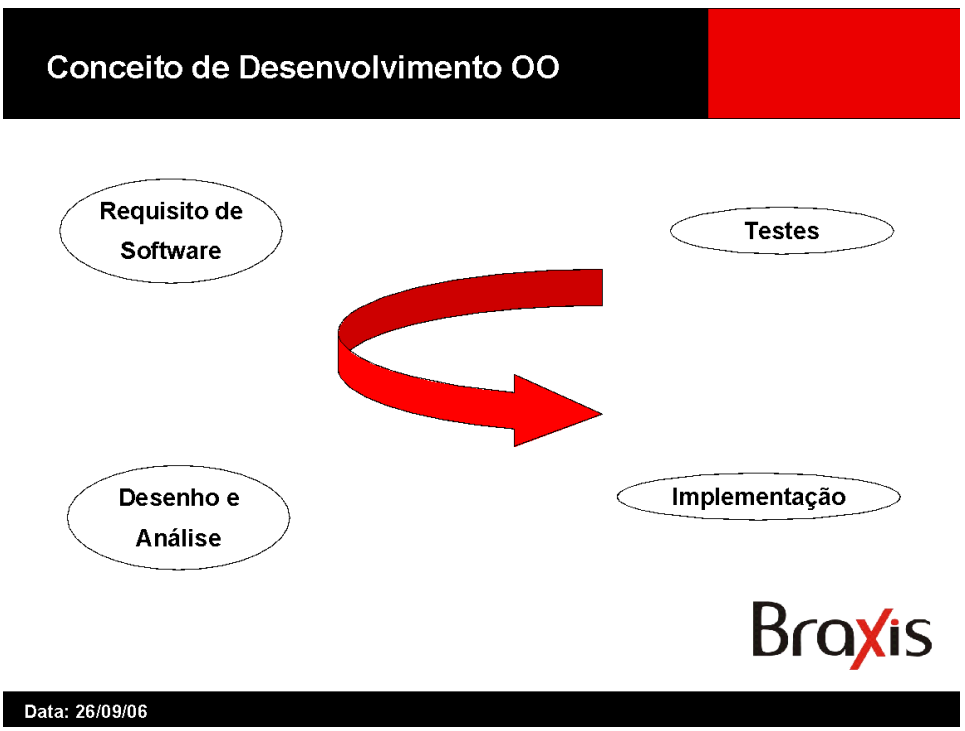

- **Compatibilidade:**
  - **È uma extensão do ABAP**
  - **Declarações ABAP Objetcs são usadas em programas ABAP.**
  - **Classes ABAP contém declarações de OO e ABAP**
- **Desenho:**
  - **Simples e Fácil**
  - **Conceitos de OO tem um forte poder de utilização**
  - **Aumento de verificações e validações e código.**

Braxis

Data: 26/09/06

- Uma das principais características da Orientação a Objetos é unificação de códigos e utilização separadamente.
- No caso o client(o programa) utiliza endereços de objetos(instancias) que acessam as funções encapsuladas. Este conceito prove melhor estrutura de código, reutilização e melhor manutenção do software.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*




- Em linguagem de programação orientada a objetos a fase de Análise e Desenho é mais importante, pois as decisões devem ser realizadas nesta fase, porque as alterações na fase implementação poderão ser muito críticas ao projeto. Ao contrário para a metodologia procedural que pode permitir algumas alterações, sem muito impacto.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Análise e Desenho

- Conteúdo:
  - Características dos Objetos
  - Definição de UML
  - Diagrama de Classes
  - Diagrama de Sequências

Braxis

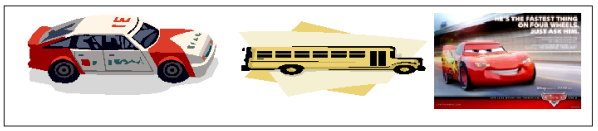
Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Características dos Objetos

Mundo real





lcl_vehicle



lcl_people

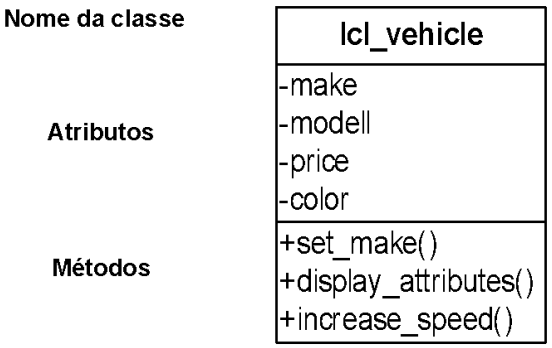
Braxis

Data: 26/09/06

- Os objetos no slide acima podem representam diversos tipos de objetos do mundo real. Isto pode ser realizado, descrevendo suas características (atributos) e suas ações (métodos). Alguns desses objetos podem ter características muito similares, exemplos: tipos de carros:  carro de passeio, carro de corridas.
- Objetos similares podem ser agrupados juntamente em classes. Onde cada classe é diferenciada com suas características.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Características dos Objetos



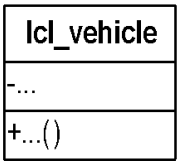

Data: 26/09/06

- **UML – Unified Model Language**
- Para a UML uma classe é representada com um nome, atributos e métodos. Com a UML, você pode ter a opção de omitir atributos ou métodos.
- Atributos descrevem os dados e características de um objeto que pode ser armazenados em uma classe. Eles definem o status do objeto.
- Métodos descrevem as funções de um objeto.
- Abap Objetcs Events não estão incluídos no diagrama de classes.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Classes e Objetos

- **Classes:**
  - Descrição das características de um objeto de forma geral
  - Determina os status dos dados através dos atributos e métodos




- **Objetos(Instancias) :**
  - Representação do mundo real
  - Representa a Instancia de uma classe

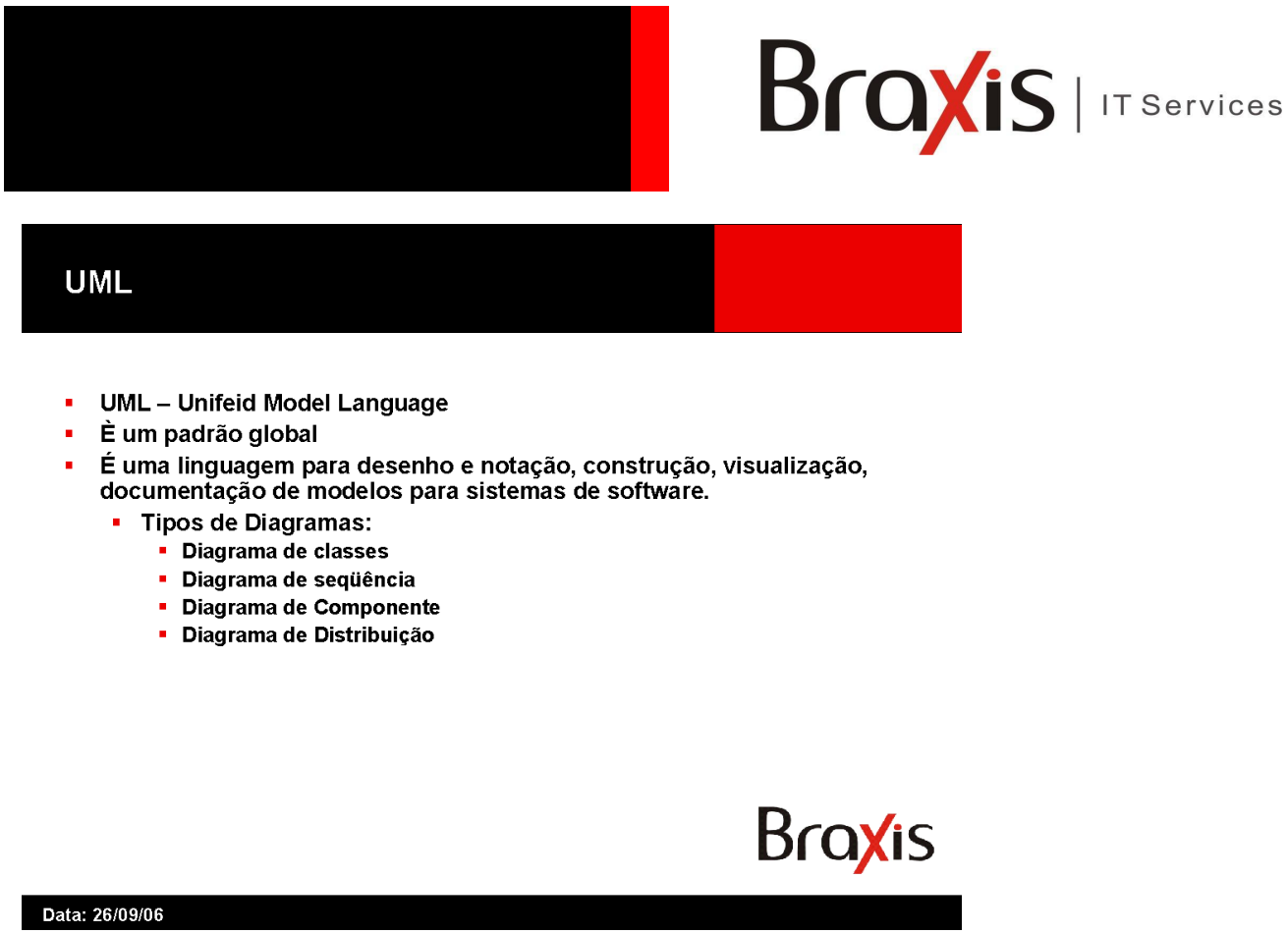



Braxis

Data: 26/09/06

- A classe é modelo de um objeto( exemplo carro) que tem a mesma estrutura.
- Cada objeto tem um identificador ( referencia = instancia) que identifica o comportamento para aquele objeto. Cada objeto tem seus específicos valores de forma única para cada objeto. Exemplo: Dois objetos podem ter os mesmos valores, mas nunca terão a mesma instancia ou referencia.

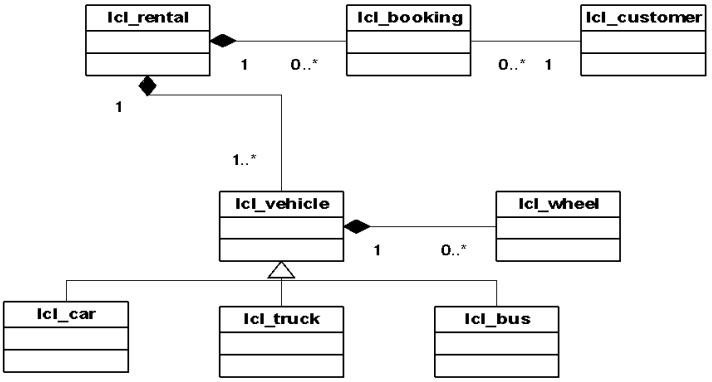

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*




- A UML Unified Model Language) é um padrão aberto modelagem de linguagem. Ela é usada para especificação, construção, visualização e documentação de modelos de softwares, habilitando a comunicação uniforme entre os vários usuários.
- UML é um padrão que é padronizado pela OMG ( Object Management Group) desde de setembro de 1997.
- Você pode encontrar especificação de UML em: http://www.omg.org
- UML descreve um número de diferentes tipos de diagramas em ordem de diferentes visões de um sistema. **Diagrama de Classe** demonstra visão de estado do modelo.
- **Diagrama de Seqüência** demonstra o relacionamento e chamadas de métodos entre objetos. Ele enfatiza e seqüência de tempo entre as chamadas de métodos.
- **Diagrama de Componente** mostra organização e dependências de componentes.
- **Diagrama de Distribuição** representa as dependências de software e hardware.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Diagrama de Classe



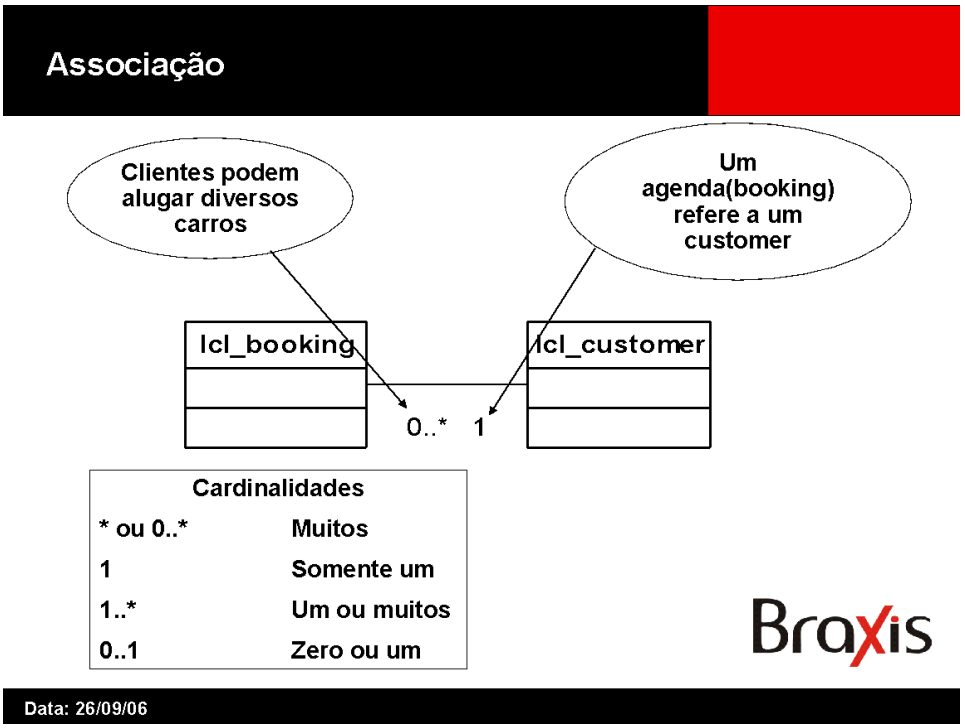

Braxis

Data: 26/09/06

- O diagrama de classe descreve os elementos contidos no modelo e os seus vários relacionamentos estáticos.
  Existem duas formas básicas de relacionamentos estáticos:
  - Associação ( por exemplo, um carro de aluguel(rental) e customer e booking.
  - Generalização / Especialização ( for exemple, um carro e um ônibus são ambas veículos)

- Em diagrama de classes, classes também podem ser exibidas com atributos e métodos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*




- Uma associação descreve um relacionamento semântico entre classes. O relacionamento entre objetos para estas classes é conhecido como um object link. Objetc link são consequentemente instancias de uma associação.
- A associação é usualmente o relacionamento entre diferentes classes. Entretanto, uma associação pode ser recursiva, neste caso, a classe poderia ter um relacionamento com ela mesma. Na maioria dos casos, associação de recursividade, são usadas ´para ligar dois diferentes objetos em uma única classe.
- Cada associação tem dois papéis. Um para cada direção de associação ( booking -> customer , customer -> booking ). Papéis podem ter nomes ( por exemplo: a associação car->reserva pode se chamar reserva)
- Cada papel tem sua cardinalidade que mostra como muitas instancias que participam neste relacionamento. A multiplicidade é o número de objetos participando em uma classe que tem um relacionamento com outras classes.

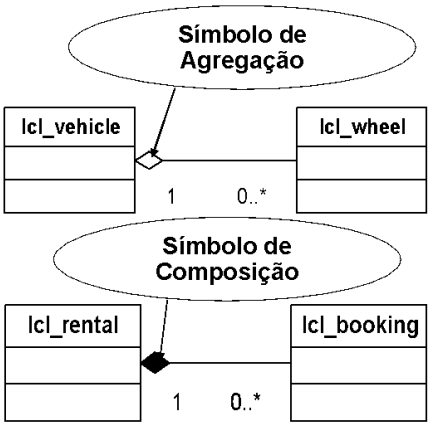
- UML:

  - A associação é representada pela linha entre os símbolos de classes
  - A cardinalidade é o relacionamento que pode ser exibido para cada fim de linha.
  - Associação pode ser especificada por uma fácil identificação ( um verbo ou texto curto ) . Este nome é escrito em itálico acima da linha e muitos tem um seta para ler a direção. Ambos são opcionais.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Agregação e Composição

- **Agregação:**
  - **È um especial caso de associação, a whole-part.**
- **Composição:**
  - **È um especial caso de agregação, um existência dependência whole-part.**




Braxis

Data: 26/09/06

- Agregação é um tipo especial de associação. Agregação descreve um objeto que contem outros ou consiste de outros objetos (whole-part) . Um veículo consiste de rodas. O relacionamento pode ser descrito pelas palavras “consiste de” ou “é parte de”.
- UML.:
- Uma agregação, como uma associação, é representada por uma linha entre classes, o qual adicionalmente tem um pequeno losango no fim da linha. O losango é sempre agregado ao fim, que é, o enquanto objeto fim. Senão a convenções seriam iguais como as associações.
- Composição é um especial tipo de agregação. Composição descreve o fato que o objeto contido não pode conter uma agregação ( por exemplo, uma reserva de carro não pode existir sem um aluguel( rental )
- Diferenças para Agregação:

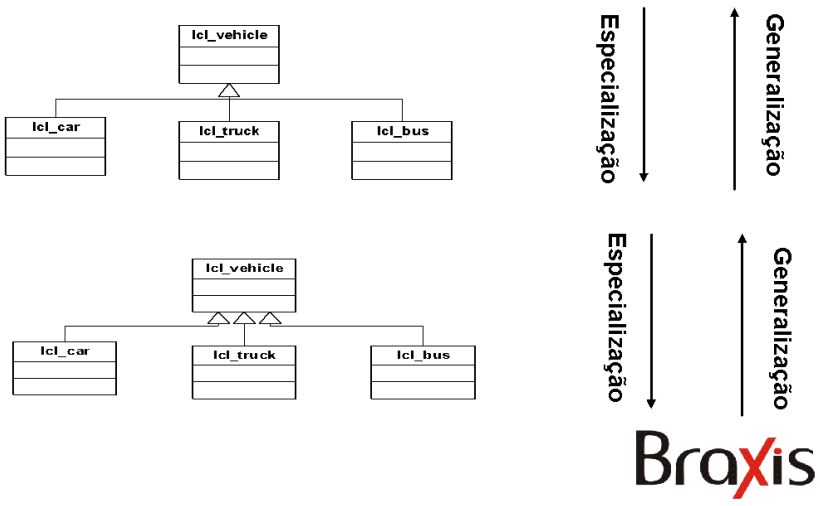
  A cardinalidade no lado da agregação somente pode ser uma. Cada parte é somente parte de um objeto composto, senão as dependências existentes poderiam ser contraditórias. A linha do tempo, de partes individuais é ligada no agregado: Partes são criadas para qualquer um ou imediatamente depois de um agregado, e eles são destruídos para qualquer um ou imediatamente antes da agregação.

- UML. Composição:

  Como agregação, composição é exibida como uma linha entre duas classes e marcadas com um pequeno losango no lado do agregado. Entretanto, em contraste para a agregação, o losango é preenchido.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Agregação e Composição

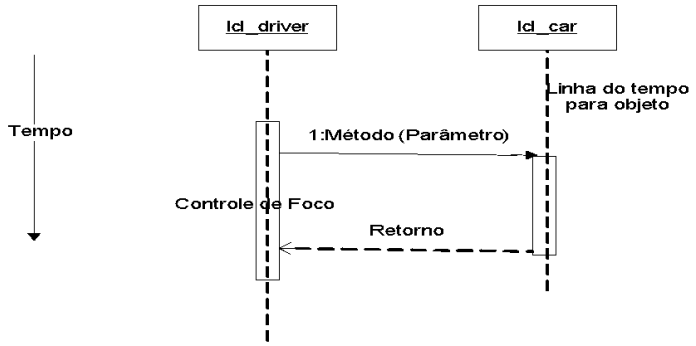



Data: 26/09/06

- UML.:
- Generalização e Especialização são denotadas por linhas com setas que apontam as classes subordinadas para as superclasses.
- Diversas linhas podem se combinadas para um arvore.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Diagrama de Seqüência




Data: 26/09/06

- Seqüência de Diagrama, diferente de diagrama de classes, mostra a dinâmica entre os objetos. Eles representam um processo, ou situação. Diagrama de seqüência tem foco principal na seqüência de tempo da mudança da informação:
  a) Criação e Deleção de Objetos
  b) Mudança de Mensagens entre os objetos.

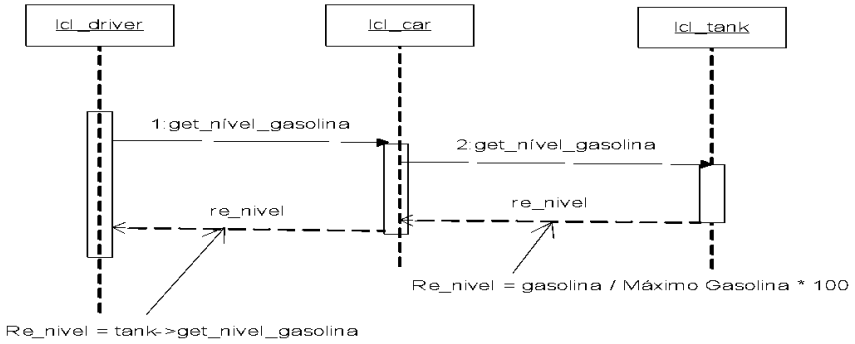
- Diagrama de Seqüência não tem notações para representações de métodos estáticos.
- A linha de vida do objeto é representada por linhas pontilhadas na vertical
- O Controle de Fluxo é exibido na vertical como um retângulo na linha de vida do objeto. O controle de fluxo mostra o período dos objetos ativos:
  - Um objeto é ativo quando as ações são executadas
  - Um objeto é indiretamente ativo se ele esta esperando por uma procedure subordinada para terminar.

- Mensagens são exibidas como setas horizontais entre linhas de objetos. A mensagem é escrita acima na seta no form “Método (parameters) . Existem várias formas de representar resposta (replay). Para este caso, na seta é exibido como um seta de retorno.

Você pode também incluir a descrição de processos a adicionar comentários para as linhas de vida dos objetos requeridos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Diagrama de Seqüência: Delegação




Data: 26/09/06

- Em delegação, dois objetos estão envolvidos na manipulação de uma request. No exemplo acima, a primeira classe delega a execução de uma request a outra classe.
- Exemplo:
- O motorista ( lcl_driver) chama o método get_nivel_gasolina para a classe car (lcl_car) . A classe car não pode executar esta tarefa sozinha.Entretanto, car chama o método get_nivel_gasolina para a classe (lcl_tank), que esta **delega** a execução a um método de classe tank.
- A delegação habilita o car para ser equipado com um novo tank, sem a chamada para o client ou para a classe car.
- Bons encapsulamentos oferecem o uso forçado de delegação: Se a classe tank acima, por exemplo: tivesse um atributo privado da classe car, não seria possível acessar a classe tank diretamente, somente através de carro.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

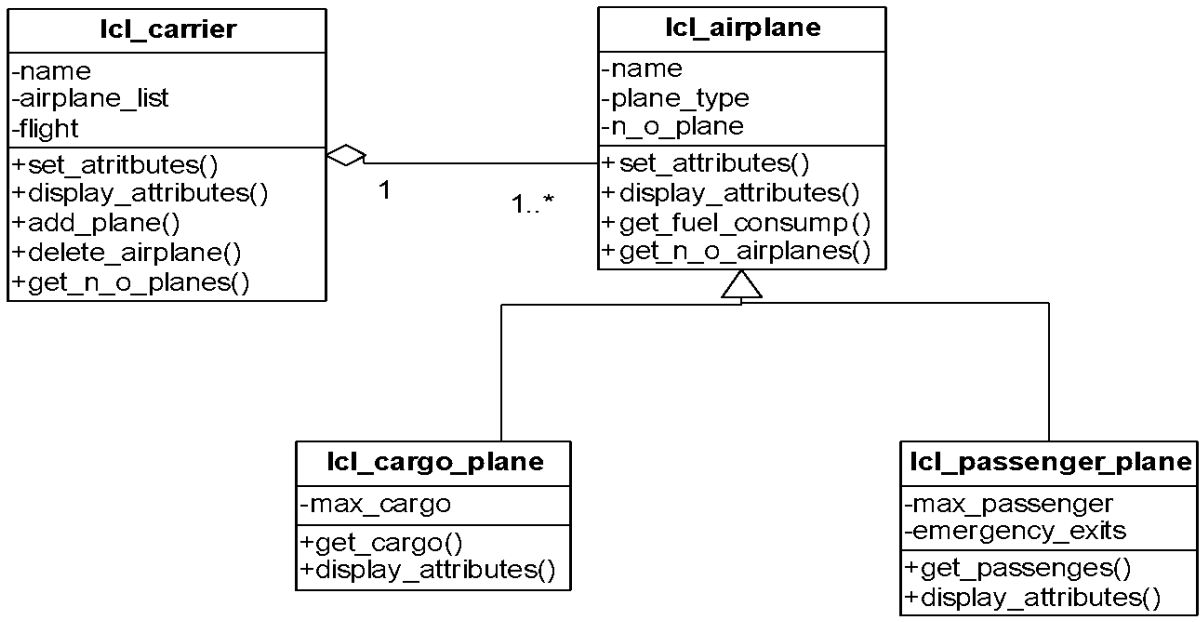
**EXERCÍCIOS:**

1.1 Use um lápis para criar um Diagrama de Classes que contenha as seguintes classes:

| | |
|---|---|
| - Ariplane | lcl_carrier |
| - Airplane(general) | lcl_airplane |
| - Passenger airplane: | lcl_passenger_plane |
| - Cargo plane: | lcl_cargo_plane |

1.2 Incluir alguns atributos e métodos para cada classe.
1.3 Desenhar as linhas de representação entre as classes e indicar possíveis cardinalidades.

**RESPOSTA:**





*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 4. PRINCÍPIOS DA PROGRAMAÇÃO ORIENTADA A OBJETO

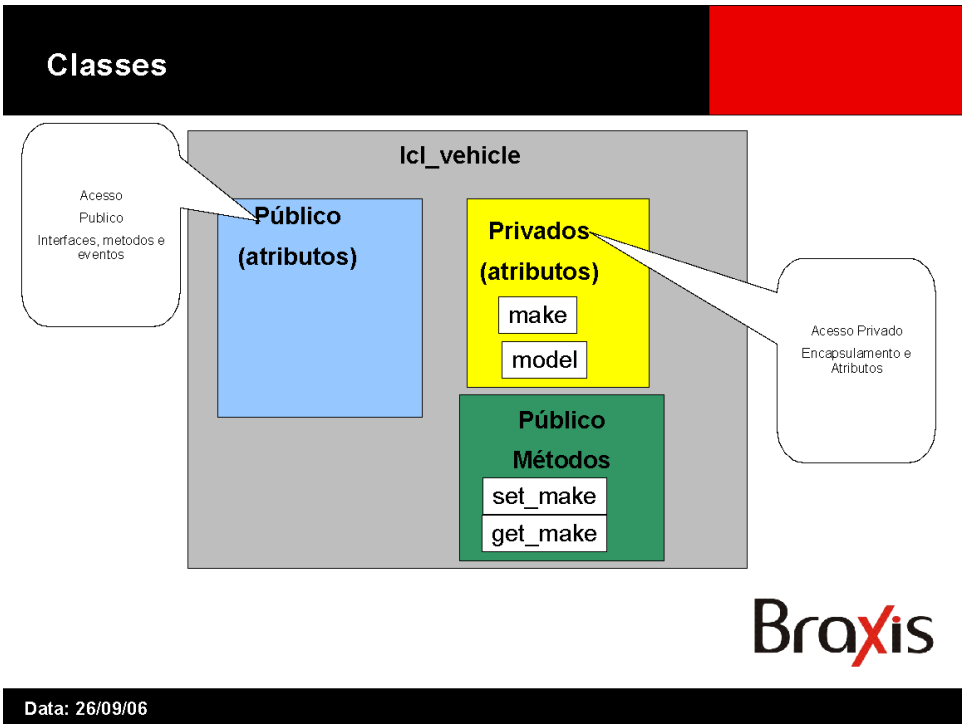
### Princípios da Programação Orientada a Objetos

- **Conteúdo:**
  - Classes
  - Objetos
  - Atributos
  - Métodos
  - Visibilidade/encapsulamento
  - Instancias
  - Construtor
  - Garbage Collector (Coletor de lixo)

Braxis

Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*




- O slide mostra atributos públicos (em azul) de uma classe que podem ser acessados por qualquer outro objeto. E ao lado direto (em amarelo) os atributos privados em amarelo, mostrando que estes estão encapsulados e não podem ser acessados diretamente.
- Porque os atributos privados das classes são ocultos?

  Isto é chamado de encapsulamento e usado para proteger o conteúdo da classe. O objetivo é somente proteger e manter a informação sempre atualizada em um único local e assim, por exemplo: Imagine que os dados são sempre modificados através de atributos privados para a classe car e enquanto isto as outras funções permanecem sem modificações. Então qualquer outro objeto pode acessar a informação da classe trabalhar normalmente, sem acesso a este tipo de informação.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

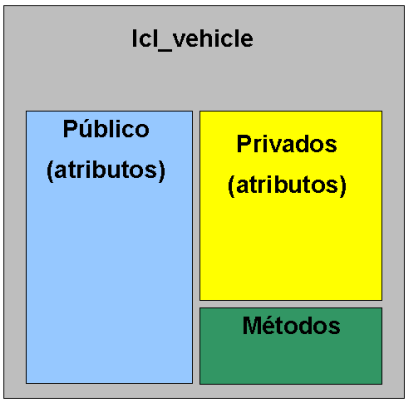
## Definição de Classes

```
CLASS lcl_vehicle DEFINITION.

ENDCLASS.


CLASS lcl_vehicle IMPLEMENTATION.

ENDCLASS.
```




Braxis

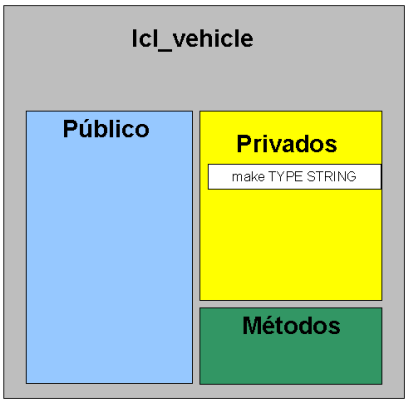
Data: 26/09/06

- A classe é um modelo de objeto que tem a mesma estruturas  e mesmo comportamento. Uma classe é formada por atributos, métodos, eventos, tipos. Em ABAP Objetcs existem duas etapas de declarações : DEFINITION E IMPLEMENTATION.
- Os componentes de uma classe são declarados na parte DEFINITION e somente os métodos podem ser implementados na parte de implementação.
- A classe não pode ser declarada em cascata ou dentro de outra classe.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Atributos

- Atributos podem ter alguns tipos de dados:
  - C,N,I,P, ...., STRING
  - Tipos de Dicionários
  - Tipos de user-define
  - TYPE REF TO define uma referência para um objeto.




Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Declaração no Código ABAP

```
CLASS  carro DEFINITION.
    .....

    TYPES:  ....

    CONSTANTS:   c_x TYPE C    VALUE 'X'

    DATA:          var          TYPE              C,
                   var2         TYPE              MARA-MATNR,
                   var3         TYPE              like var3,
                   o_car        TYPE REF TO       lcl_car,
                   int_car      TYPE REF TO       if_car.]

    CLASS-DATA: modelo      TYPE ...

ENDCLASS.
```

Data: 26/09/06

- Nas declarações de classe somente pode ser usado TYPE para a declaração de tipos de dados.
- Não pode ser utilizado o LIKE para objetos locais.
- A adição READ-ONLY significa que um atributo público declarado com DATA pode ser lido de fora, mas não pode ser modificado pelos métodos de uma classe.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Declarações Públicas e Privadas

- **Atributos Públicos:**
  - Podem ser acessados e alterados por qualquer outro objeto.
  - Acesso direto.
- **Atributos Privados:**
  - Somente podem ser acessados e alterados dentro da própria classe.
  - Sem acesso de fora da classe.

```
CLASS lcl_vehicle DEFINITION.

  PUBLIC SECTION.
    DATA:     cor(10) TYPE   C.

  PRIVATE SECTION.

ENDCLASS.
```



```
CLASS lcl_vehicle DEFINITION.

  PUBLIC SECTION.

  PRIVATE SECTION.
    DATA:     cor(10) TYPE   C.

ENDCLASS.
```



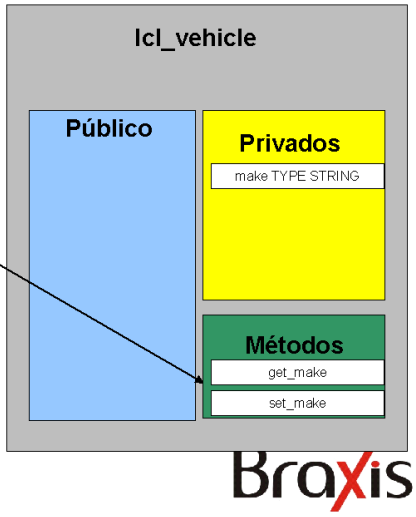
Data: 26/09/06

- Os atributos que não devem ser acessados fora da classe podem ser declarados como private. Na parte PRIVATE SECTION.
- Os atributos que devem ser acessados de fora da classe podem ser declarados como atributos públicos. Na parte PUBLIC SECTION.
- O atributo cor é definido como public para a classe lcl_vehicle.
- Os atributos públicos que pertençam a classes de interfaces, que as implementações poderão publicadas, devem estar declarados como PUBLICOS para que seja possível a implementação.
- De modo geral a declaração de atributos públicos deve ser o menor possível.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Como acessar atributos privados?

- Instancia da Classe
  - o_vehicle
  - Acessos pelos métodos
    - set_make
    - get_make




Data: 26/09/06

- Você pode acessar atributos privados através de métodos , onde eles podem retornar o valor do atributo, ou modificá-lo.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Atributos de Instancia e Estáticos

- **Atributos de Instancias:**
  - Um por instancia
  - Declaração DATA:
- **Atributos Státicos**
  - Somente um por classe
  - Declaração CLASS-DATA:
  - Conhecido também como atributo de classe.

```
CLASS lcl_vehicle  DEFINITION.

    PUBLIC SECTION.
      ....

    PRIVATE SECTION.
      DATA:        cor(10)   TYPE    C.


    CLASS-DATA: numero_veiculos TYPE I.

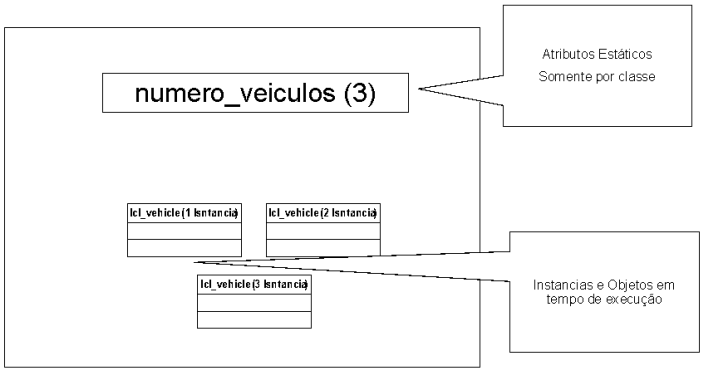
ENDCLASS.
```

Data: 26/09/06

- Os atributos podem ser de Instancia ou Estáticos.
- Atributos de instancia existem separadamente para cada objeto.
- Atributos de Instancias são definidos como declaração **DATA.**
- Atributos Estáticos existem somente um por classe e são visíveis para todas as instancias da classe.
- Atributos Estáticos usualmente contém informações que se aplica para todas as classes, exemplos:
  - Dados que são os mesmos em todas as instancias
  - Informações administrativas da classe, como contadores,...
- Atributos Estáticos são definidos como declaração **CLASS-DATA.**


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Atributos de Instancia e Estáticos



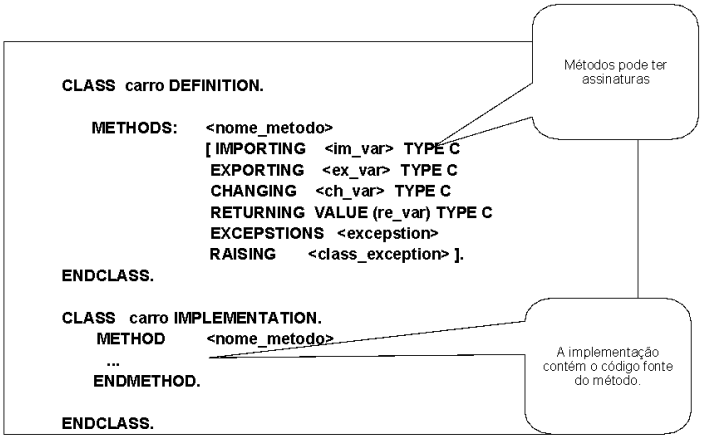

Braxis

Data: 26/09/06

- O slide exibe o atributo estático numero_veiculos contendo o número de instancias.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos




Data: 26/09/06

- Métodos são funções internas determinadas na classe. Eles podem acessar todos os atributos e modificá-los.
- Métodos podem ter parâmetros de interface e são chamados de assinatura que habilitam a recepção de valores, quando passados por outros programas.
- Métodos podem ter os seguintes parâmetros: IMPORTING, EXPORTING, CHANGING e RETURN e parâmetros de exceções. Todos os parâmetros podem ser passados por valor ou referencia. Acima da versão SAP Basis 6.10 é possível utilizar exceptions.
- Você pode definir um código de retorno usando a declaração RETURNING. Este somente pode ser um parâmetro simples, o qual pode ser passado como valor. Você não pode definir parâmetros EXPORTING e CHANGING.
- Todos os parâmetros (IMPORTING, CHANGING) podem ser definidos como opcional, nas declarações usando OPTIONAL ou DEFAULT adições. Estes parâmetros não necessariamente tem que ser passados quando o objeto é criado. Se você usa a opção OPTIONAL, os parâmetros permanecem inicializados de acordo com o tipo, visto que, o DEFAULT entrar com um valor inicia.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos Públicos e Privados

- **Métodos Públicos:**
  - Podem ser acessados e alterados por qualquer outro objeto.
- **Métdos Privados:**
  - Somente podem ser acessados e alterados dentro da própria classe.

```
CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    METHODS: set_cor importing
                im_cor(10) type C

  PRIVATE SECTION.
    METHODS: inicia_cor.
     DATA:      cor(10)  TYPE    C.

ENDCLASS.
```

```
CLASS lcl_vehicle IMPLENTATION.

   METHOD: incia_cor.
    cor = 'sem cor'.
  ENDMETHOD.

  METHOD set_cor.
    cor = im_cor.
  ENDMETHOD.

ENDCLASS.
```

Data: 26/09/06

- O método *incia_cor* é um método privado que somente pode ser executado, por exemplo, pelo constritor. Já o método set_cor pode ser acessado por qualquer outro objeto.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos Públicos e Privados

```
CLASS lcl_vehicle  DEFINITION.
    PUBLIC SECTION.
      METHODS:  set_cor importing
                  im_cor(10)  type C
     CLASS-METHODS:
                  get_count exporting
                   im_make TYPE I.

    PRIVATE SECTION.
      DATA:        cor(10)   TYPE    C.
    CLASS-DATA: numero_veiculos TYPE I
.
ENDCLASS.
```

```
CLASS lcl_vehicle  IMPLENTATION.

   METHOD get_count.
     ex_count  = numero_veiculos
   ENDMETHOD.

ENDCLASS.
```

Braxis

Data: 26/09/06

- No método get_count, você pode somente usar o atributos estático numero_veiculos. Todos os outros atributos da classe são atributos de instancias e somente pode ser usados com métodos de instancia.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos Públicos e Privados

| lcl_vehicle |
| --- |
| -make<br>-model<br>+numero_veiculos |
| +set_make()<br>-inicia_cor()<br>+get_count() |

+ representam componentes públicos

- representam componentes privados

_ componentes estáticos são sublinhados

```
CLASS lcl_vehicle DEFINITION.
    PUBLIC SECTION.
      METHODS: set_cor importing
                  im_cor(10) type C
     CLASS-METHODS:
                  get_count exporting
                   im_make TYPE I.

    PRIVATE SECTION.
      DATA:       cor(10)  TYPE    C.
    CLASS-DATA: numero_veiculos TYPE I
.
ENDCLASS.
```

Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

I

## Criação de Objetos

- **Objetos são criados usando a declaração CREATE OBJECT**
- **Objetos somente podem ser criados e acessados através de variáveis de referências, exemplo: TYPE REF TO.**

| lcl_vehicle |
| --- |
| -make<br>-model<br>+numero_veiculos |
| +set_make()<br>-inicia_cor()<br>+get_count() |

**CREATE OBJECT o_vehicle**

Braxis

Data: 26/09/06

- As classes são as descrições de um objeto. Elas descrevem todas as características que serão comuns em todos os objetos da mesma classe. Em tempo de execução os objetos são criados em memória e essa criação se chama instanciação.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Declarações Públicas e Privadas

```
CLASS lcl_vehicle DEFINITION.

    PUBLIC SECTION.
      DATA:       cor(10)  TYPE    C.

    PRIVATE SECTION.

ENDCLASS.

CLASS lcl_vehicle IMPLEMENTATION.
   ......
ENDCLASS.

---------------------------------------------------------------------------------------------
REPORT.

DATA: o_carro    TYPE REF TO lcl_vehicle,
      o_carro2  TYPE REF TO lcl_vehicle,

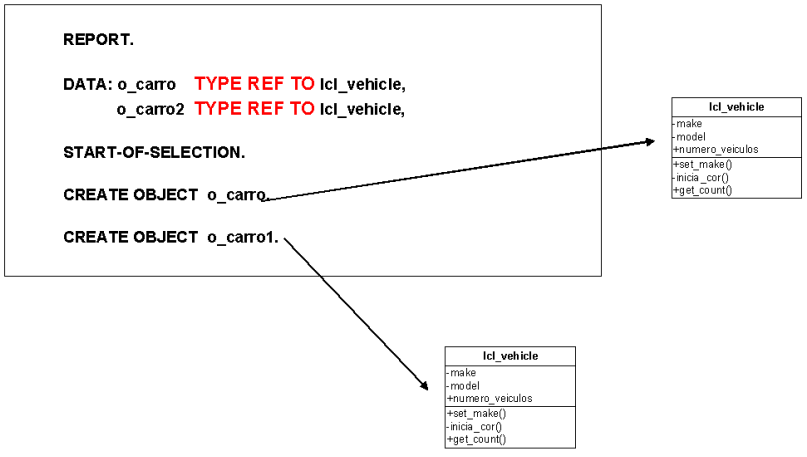
START-OF-SELECTION.
```

Data: 26/09/06

- DATA: o_carro TYPE REF TO lcl_vehicle declarada como um variável de referencia que atua como um ponteiro para um objeto.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Criação de Objetos

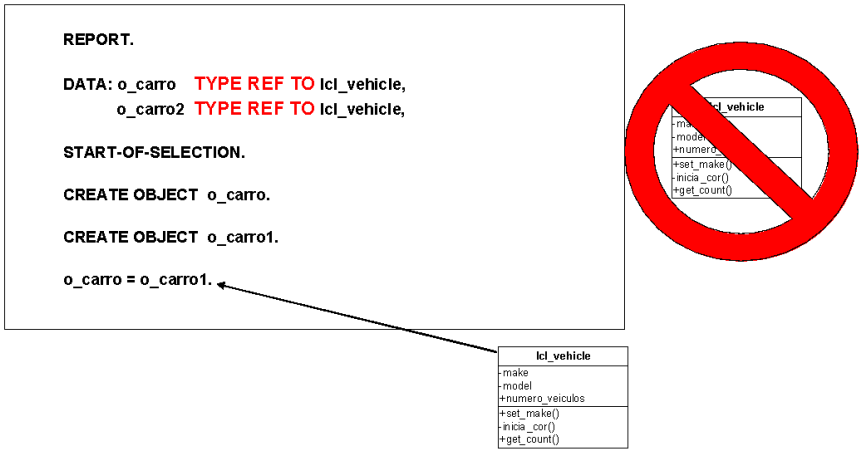



Data: 26/09/06

- O comando CREATE OBJECT cria um objeto em memória. Os valores dos atributos para este objeto são inicialmente criados com os valores iniciais ou entradas iniciais com a declaração VALUE.
- No slide acima os dois objetos o_carro e o_carro1, são duas referencias ou ponteiros da classe lcl_vehicle.
- Variáveis de referencias podem ser atribuídas uma a outra, exemplo (o_carro = o_carro1).


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Garbage Collector



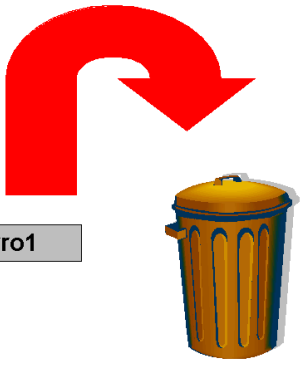

Data: 26/09/06

- O GARBAGE COLLECTOR é uma rotina de sistema que deleta objetos que estão muito sem atividades na memória e libera espaço em memória.
- O Garbage Collector um recurso comum em todas as linguagens orientadas a objeto. Pois com certeza é uma função muito importante para monitorar memória e eliminar objetos que estão com status de inativos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Garbage Collector

- Todos os objetos ativos são marcados com um check.
- Os objetos que não estão marcados são deletados da memória pelo "Garbage Collector"

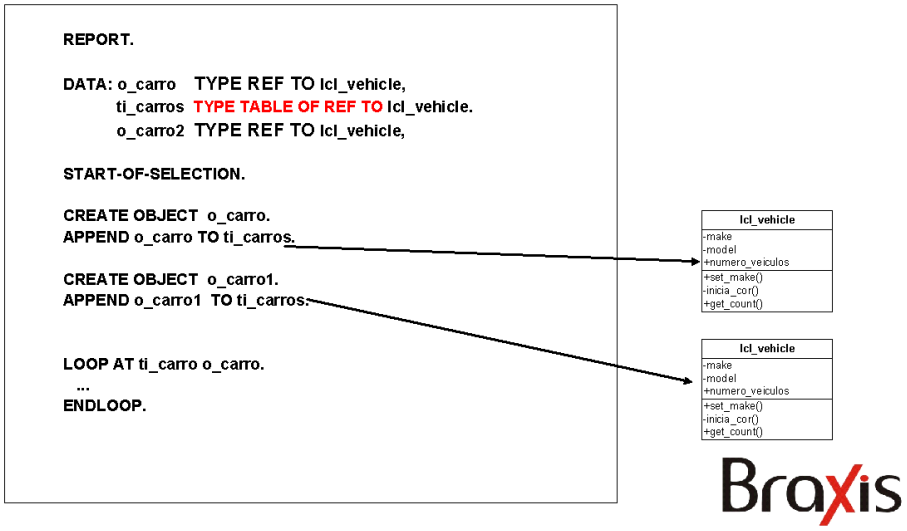



o_carro

Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Referencias de Buffer

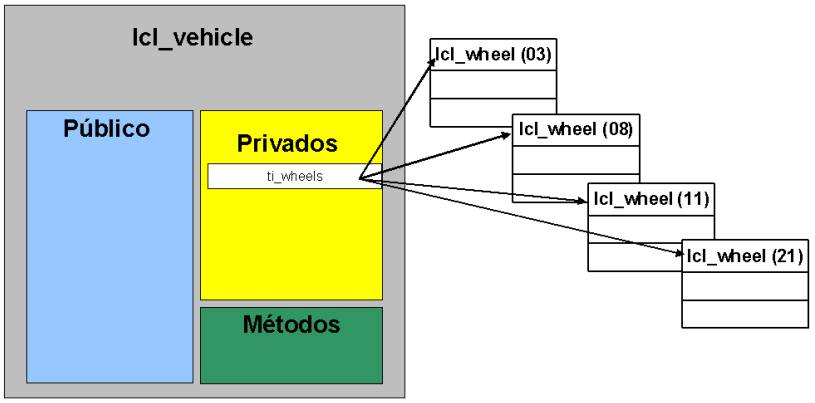



Data: 26/09/06

- È possível guardar diversos objetos em um programa através de uma tabela interna. Onde os campos da tabela interna serão os mesmos da classe que foi referencia no comando TYPE TABLE OF REF TO.
- È possível ler a tabela através de LOOP.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Agregação




Data: 26/09/06

- Se uma classe define referencias de objetos para uma segunda classe como atributos ( no exemplo acima referencias da classe lcl_wheels), somente estas referencias de objetos serão armazenadas em objetos pertencentes a aquela classe.
- Os objetos da segunda classe lcl_wheels têm identidade própria. Eles estão encapsulados na primeira classe lcl_vehicle a podem somente serem acessados para esta classe, usando variáveis de referências.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

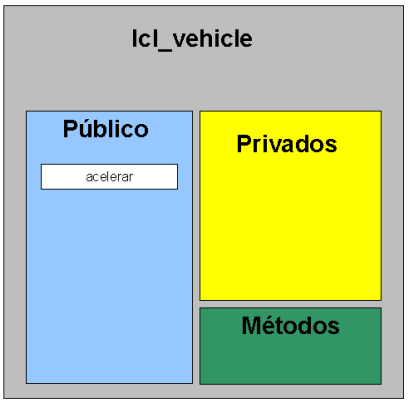
## Chamando Métodos

Anterior 4.7

```
CALL METHOD o_carro->acelerar( )
```

>= 4.7

```
o_carro->acelerar( )
```




Braxis

Data: 26/09/06

- Existem três tipos de métodos:
  -1. São acionados e não retornam valores.
  -2. Métodos que retornam valores.
  -3. Métodos que retornam ou modificam diversos valores.

- O slide exibe a duas chamadas de métodos: uma anterior a versão do release SAP 4.7 que mostra que deve ser utilizado CALL METHOD e outra, exibindo que o release igual ou maior que 4.7 o comando CALL METHOD pode ser omitido.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Chamando Métodos de Instancia

```
CALL METHOD  <instancia>-><método de  instancia>
                    EXPORTING  <ex_var> = var
                    IMPORTING  <im_var> = var
                    CHANGING   <ch_var> = var
                    RECEVING    <re_var> =  var
                   EXCEPTIONS <exceptions> = <nr>
```

```
DATA: o_carro    TYPE REF TO lcl_vehicle,
      make_name TYPE   STRING.
...
make_name  = 'make a car'.

CALL  METHOD o_carro->set_make EXPORTING ex_make = make_name.

o_carro->set_make  (  make_name ).

o_carro->get_make (  IMPORTING im_make = make_name ).
```

Braxis

Data: 26/09/06

- Métodos de instancia são chamados com o comando CALL METHOD utilizando a sintaxe <objeto>**->**<método de instancia>.
- Os parâmetros para a assinatura do método (IMPORTING, EXPORTING...) são separados por espaços.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Chamando Métodos Estáticos

```
CALL METHOD  <nome classe>=><método de  instancia>
                    EXPORTING  <ex_var> = var
                    IMPORTING  <im_var> = var
                    CHANGING   <ch_var> = var
                    RECEVING   <re_var> =  var
                   EXCEPTIONS <exceptions> = <nr>
```

```
DATA: o_carro    TYPE REF TO lcl_vehicle,
      make_name TYPE   STRING.
      count          TYPE I.
...
make_name  = 'make a car'.
CALL  METHOD lcl_vehicle=>set_count ( IMPORTING im_count  = count ).
```

Braxis

Data: 26/09/06

- Métodos estáticos são chamados pela classe e método <nome classe>**=>**<método da classe estático>
- Métodos estáticos não precisam de instancias.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos de Funcionalidades

- **Definindo:**
  - **Somente um parâmetro RETURNING.**
- **Chamando:**
  - **RECEIVING parâmetros são possíveis.**
  - **Várias formas são possíveis:**
    - **MOVE, CASE, LOOP**
    - **Lógicas (IF,CHECK)**

Braxis

Data: 26/09/06

- Métodos que tem um parâmetro de retorno são classificados como métodos funcionais. Estes métodos não podem ter parâmetros EXPORTING e CHANGING.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos Funcionais

```
CLASS lcl_vehicle  DEFINITION.
    PUBLIC SECTION.
      METHODS: get_avarege_fuel
                  IMPORTING im_distance                type  s_distance,
                            im_fuel                    type  ty_fuel,
                  RETURNIG  VALUE(re_fuel)             tupe  ty_fuel.
    ENDCLASS.
```

```
DATA:  o_carro    TYPE REF TO lcl_vehicle,
       o_carro2   TYPE REF TO lcl_vehicle,
       avg_fuel   TYPE ty_fuel.


avg_fuel =
    o_carro->get_average_fuel ( im_distance = 200   im_fuel = 20 )
+   o_carro->get_average_fuel ( im_distance = 300   im_fuel = 30 ).
```

Braxis

Data: 26/09/06

- Sintaxe de parâmetros:

  - Sem parâmetros IMPORTING:  o_ref->metodo( ).
  - Um parâmetro IMPORTING o_ref->método ( p1 ) ou o_ref->método ( im_fuel = p1 )
  - Mais de uma parâmetro IMPORTING o_ref->método ( im_fuel = p1 im_modell = p2  im_color = p3  )

- Exemplo de sintaxe de um método funcional.

```
CALL METHOD o_carro->get_average_fuel
                    IMPORTING im_distance      = 500
                              Im_fuel          = 50
                    RECEIVING re_fuel          = avg_fuel.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Acessos a atributos Públicos

```
CLASS lcl_vehicle  DEFINITION.
    PUBLIC SECTION.
      DATA:   make  TYPE string READ-ONLY.
      CLASS-DATA: n_o_vehicles   TYPE I READ-ONLY.
      ...
    ENDCLASS.

DATA:   o_carro     TYPE REF TO lcl_vehicle,
        make_name TYPE string,
        count TYPE I.

START-OF-SELECTION.

CREATE OBJECT o_carr.

make_name = o_carro->make.
count = lcl_vehicle=>n_o_vehicles.
```

Data: 26/09/06

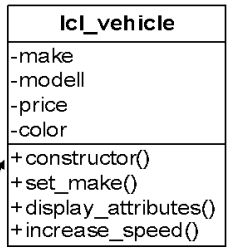
- Existem duas formas de acessar atributos públicos:
  - Através <nome classe>**=>**< atributo da classe>.
  - <instancia>**->**<atributo de instancia>.
  - **=>** Acessa atributos estáticos.
  - **->** Acessa atributo de instancia.

OBS.: A diferença entre como acessar um método e um atributo é parênteses ( ). Os atributos não precisam de parênteses e métodos é necessário informá-los.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Método Constructor

- **È método que pode ser definido para definir um estado inicial de um objeto.**
- **Somente parâmetros IMPORTING e EXCEPTIONS.**
- **Somente é executado na criação do OBJETO.**




```
METHODS constructor IMPORTING
                    im_para1
                    EXCEPTIONS ex_par .
```

Data: 26/09/06

- O constructor é um método que é executado sempre na criação de objeto (instancia) e tem os seguintes papéis:
  - Cada classe somente pode ter definido um método constructor
  - Sempre é criado em tempo de execução CREATE OBJETC.
  - Se o constructor é declarado ele dever ser declaro como PUBLICO.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Constructor

```
CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    METHODS constructor IMPORTING im_make  TYPE string,
                                  im_model TYPE string.
  PRIVATE SECTIONS.
    DATA:   make   TYPE string,
            weight TYPE p.
    CLASS-DATA: n_o_vehicles TYPE I.
  ENDCLASS.
```

```
CLASS lcl_vehicle IMPLENTATION.
    METHODS constructor
              make         = im_make.
              model        = im_model.
              ADD 1 TO n_o_vehicles.
    ENDMETHOD.
  ENDCLASS.
```

```
DATA o_carro TYPE REF TO lcl_vehicle.
...
CREATE OBJECT o_carro
                EXPORTING im _make   = ' BMW'
                          im_model   = 'X5'.
```

Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Constructor Estático

```
CLASS lcl_vehicle DEFINITION.
    PUBLIC SECTION.
      CLASS-METHODS: CLASS_CONSTRUCTOR.

    PRIVATE SECTIONS.
      CLASS-DATA: n_o_vehicles   TYPE I.
    ENDCLASS.
```

```
CLASS lcl_vehicle IMPLENTATION.
      METHODS CLASS_CONSTRUCTOR.
        ...
      ENDMETHOD.
    ENDCLASS.
```

```
DATA o_carro TYPE REF TO lcl_vehicle.
...
CREATE OBJECT o_carro.

Cont = lcl_vehicle=>n_o_vehicles.
```

Braxis

Data: 26/09/06

- O construtor estático é um método especial de uma classe com o nome class_constructor. Ele é executado uma única vez quando a classe é acessada pela primeira vez.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

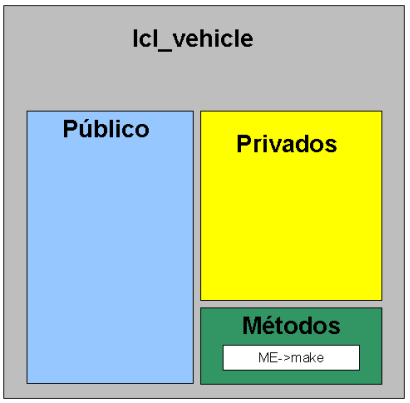
## Recurso ME

```
CLASS lcl_vehicle DEFINITION.
  PUBLIC SECTION.
    METHODS constructor IMPORTING im_make
  TYPE string,
                                  im_model
  TYPE string.
  PRIVATE SECTIONS.
    DATA:   make    TYPE string.
  ENDCLASS.
```

```
CLASS lcl_vehicle IMPLENTATION.

    METHODS constructor
      DATA make TYPE string 'FORD'.
      CONCATENATE im_make make INTO
       ME->make.
    ENDMETHOD.

  ENDCLASS.
```




Data: 26/09/06

- Assim como outras linguagens O.O como JAVA, .NET existe um recurso semelhante no ABAP OO, o comando ME. Como este comando é possível acessar objetos(atributos e métodos) da classe, desde que esta classe seja uma instancia. Exemplo **ME->make** que acessa o atributo make da própria classe.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Criação da Classe**

1.1 Criar um pacote ZBXOO_XX por aluno ou micro.

1.1.1 Criar um programa principal chamado ZBXOO_MAIN_AIRPLANE_XX e um include chamado ZBXOO_AIRPLANE_XX.

1.1.2 Cria uma classe local chamada lcl_ariplane no include ZBXOO_AIRPLANE_XX.

1.1.3 Criar dois atributos de instancia privados
- name tipo string
- planetype tipo saplane-planetype.

1.1.4 Criar um atributo público estático n_o_airplanes do tipo i.

1.1.5 Criar um método público de instancia set_attributes para atribuir valores aos atributos privados name planetype e adicione 1 no atributo estático n_o_airplanes.

1.1.6 Criar um método público de instancia display_attributes para exibir o atributo n_o_airplanes com o comando WRITE, você pode exibir através dos ícones icon_ws_plane, onde estão no TYPE-POOLS.

**2. Instanciando Objetos**

2.1 No programa ZBXOO_MAIN_AIRPLANE_XX declarar uma referencia (TYPE REF TO) para a classe lcl_airplane com nome exemplo r_plane.

2.2 Definir uma tabela interna TYPE TABLE OF REF TO para armazenar as instancias criadas com seus valores da classe lcl_airplane.

2.3 Criar o objeto CREATE OBJECT para r_plane.

2.4 Executar o método set_attributes três vezes passando os parâmetros name e planetype. Exemplos:

2.5 name = ‘AA Dallas’
2.6 planetype = ‘747-400’

2.7 Criar um loop na tabela interna criada e atribuir ao objeto r_plane criado e executar dentro do loop a chamada ao método display_attributes, para exibir os atributos dos objetos criados.

2.8 Debugar o programa verificando como são armazenadas as instancias.

2.9 Executar o método display_n_o_airplanes para exibir o total de instancias.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

### 3. Criando Constructor

3.1.1 Criar um método constructor no include ZBXOO_AIRPLANE_XX. Este constructor deve inicializar os objetos com os valores dos atributos name e planetype e adicionar 1 ao atributo privado n_o_ariplanes, no exercício 2.

3.1.2 Comente as linhas do método set_attributes e inclua os parâmetros no CREATE OBJECT com os novos parâmetros de IMPORTING.

## RESPOSTAS EXERCÍCIOS:

### 1. Criação da Classe

```
*&---------------------------------------------------------------------*
*&  Include          ZBXOO_AIRPLANE_00                                 *
*&---------------------------------------------------------------------*


*---------------------------------------------------------------------*
*   CLASS lcl_airplane DEFINITION.                                    *
*---------------------------------------------------------------------*

CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

    CONSTANTS: pos_1 TYPE i VALUE 30.

     METHODS: set_attributes IMPORTING
               im_name       TYPE string
               im_planetype  TYPE saplane-planetype,

              display_atributes.

      CLASS-METHODS: display_n_o_airplanes.

   PRIVATE SECTION.

   DATA:  name        type     string,
          planetype   type     saplane-planetype.

   CLASS-DATA: n_o_airplanes  type i.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_airplane IMPLEMENTATION.                                *
*---------------------------------------------------------------------*
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
CLASS lcl_airplane  IMPLEMENTATION.


  METHOD set_attributes.
    name       = im_name.
    planetype  = im_planetype.
    n_o_airplanes = n_o_airplanes + 1.
  ENDMETHOD.


  METHOD display_atributes.
    WRITE: / icon_ws_plane as icon,
           / 'Name of airplane: '(001), AT pos_1 name,
           / 'Airplane type '(002), AT pos_1 planetype.

  ENDMETHOD.


  METHOD display_n_o_airplanes.
    WRITE: /,/ 'Total nember of planes'(ca1),
           AT pos_1 n_o_airplanes LEFT-JUSTIFIED,/.
  ENDMETHOD.


ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 2. Instanciando Objetos

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_MAIN_AIRPLANE_00                                      *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_MAIN_AIRPLANE_00                                        .


TYPE-POOLS icon.

INCLUDE ZBXOO_AIRPLANE_00.

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane.

DATA: count TYPE i.

START-OF-SELECTION.

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).

CREATE OBJECT r_plane.

r_plane->set_attributes( im_name = 'LH Berlin'
                         im_planetype = '747-400' ).


APPEND r_plane TO plane_list.


CREATE OBJECT r_plane.

r_plane->set_attributes( im_name = 'AA Dallas'
                         im_planetype = '747-400' ).



APPEND r_plane TO plane_list.


CREATE OBJECT r_plane.

r_plane->set_attributes( im_name = 'UA Chigago'
                         im_planetype = '747-400' ).


APPEND r_plane TO plane_list.



LOOP AT plane_list INTO r_plane.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
  CALL METHOD r_plane->display_atributes.
ENDLOOP.

CALL METHOD r_plane->display_n_o_airplanes.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

### 3. Criando Constructor

```
*&---------------------------------------------------------------------*
*&  Include          ZBXOO_AIRPLANE_00_B                               *
*&---------------------------------------------------------------------*

*---------------------------------------------------------------------*
*   CLASS lcl_airplane DEFINITION.                                    *
*---------------------------------------------------------------------*

CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

    CONSTANTS: pos_1 TYPE i VALUE 30.

    METHODS: constructor IMPORTING
               im_name       TYPE string
               im_planetype  TYPE saplane-planetype,

             display_atributes.

     CLASS-METHODS: display_n_o_airplanes.

  PRIVATE SECTION.

  DATA:  name       type     string,
         planetype  type     saplane-planetype.

  CLASS-DATA: n_o_airplanes  type i.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_airplane IMPLEMENTATION.                                *
*---------------------------------------------------------------------*

CLASS lcl_airplane  IMPLEMENTATION.


  METHOD constructor.
    name       = im_name.
    planetype  = im_planetype.
    n_o_airplanes = n_o_airplanes + 1.
  ENDMETHOD.


  METHOD display_atributes.
    WRITE: / icon_ws_plane as icon,
           / 'Name of airplane: '(001), AT pos_1 name,
           / 'Airplane type '(002), AT pos_1 planetype.

  ENDMETHOD.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
  METHOD display_n_o_airplanes.
    WRITE: /,/ 'Total nember of planes'(ca1),
           AT pos_1 n_o_airplanes LEFT-JUSTIFIED,/.
  ENDMETHOD.


ENDCLASS.




*&---------------------------------------------------------------------*
*& Report  ZBXOO_MAIN_AIRPLANE_00_B                                    *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_MAIN_AIRPLANE_00_B                .


TYPE-POOLS icon.

INCLUDE ZBXOO_AIRPLANE_00_B.

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane.

DATA: count TYPE i.

START-OF-SELECTION.

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).

CREATE OBJECT r_plane EXPORTING
        im_name = 'LH Berlin'
        im_planetype = '747-400'.

APPEND r_plane TO plane_list.


CREATE OBJECT r_plane EXPORTING
        im_name = 'AA New York'
          im_planetype = '747-300'.

APPEND r_plane TO plane_list.


CREATE OBJECT r_plane EXPORTING
        im_name = 'UA Chigago'
         im_planetype = '747-400'.

APPEND r_plane TO plane_list.
```

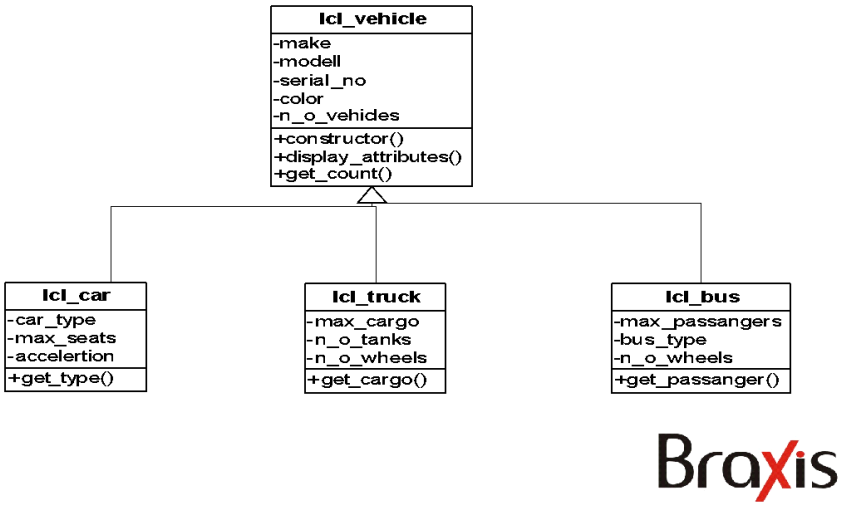

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
LOOP AT plane_list INTO r_plane.
  CALL METHOD r_plane->display_atributes.
ENDLOOP.
CALL METHOD r_plane->display_n_o_airplanes.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 5. HERANÇA (INHARITANCE)

### Herança



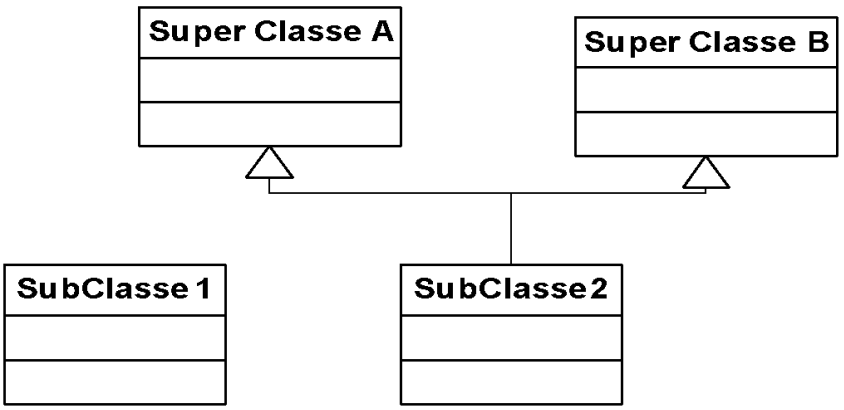

Braxis

Data: 26/09/06

- Herança é o relacionamento entre Super Classes e SubClasses, onde as subclasse herdam as características da superclasses. Nas subclasses também é possível adicionar novos componentes como atributos e métodos.
- No slide acima a classe vehilce tem atributos como make, model..., e os métodos display_atributes e get_count e estes componentes são herdados pelas subclasses (lcl_car, lcl_truck e lcl_bus) automaticamente.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Herança Múltipla




Braxis

Data: 26/09/06

- Herança deve ser utilizada para implementação de GENERALIZAÇÃO e ESPECIALIZAÇÃO. A superclasse é a generalização das suas subclasses. As subclasses é a especialização da superclasses. Em ABAP Objects não é possível Herança Múltipla(herdar características de duas classe ao mesmo tempo) , somente herança simples.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Herança (Superclasses e Subclasses)

- **Superclasses**
  - **Contém componentes de forma genérica para possível reutilização de códigos.**
- **Subclasse**
  - **Contém os componentes herdados da Superclasse e possibilita implementação de novos componentes.**

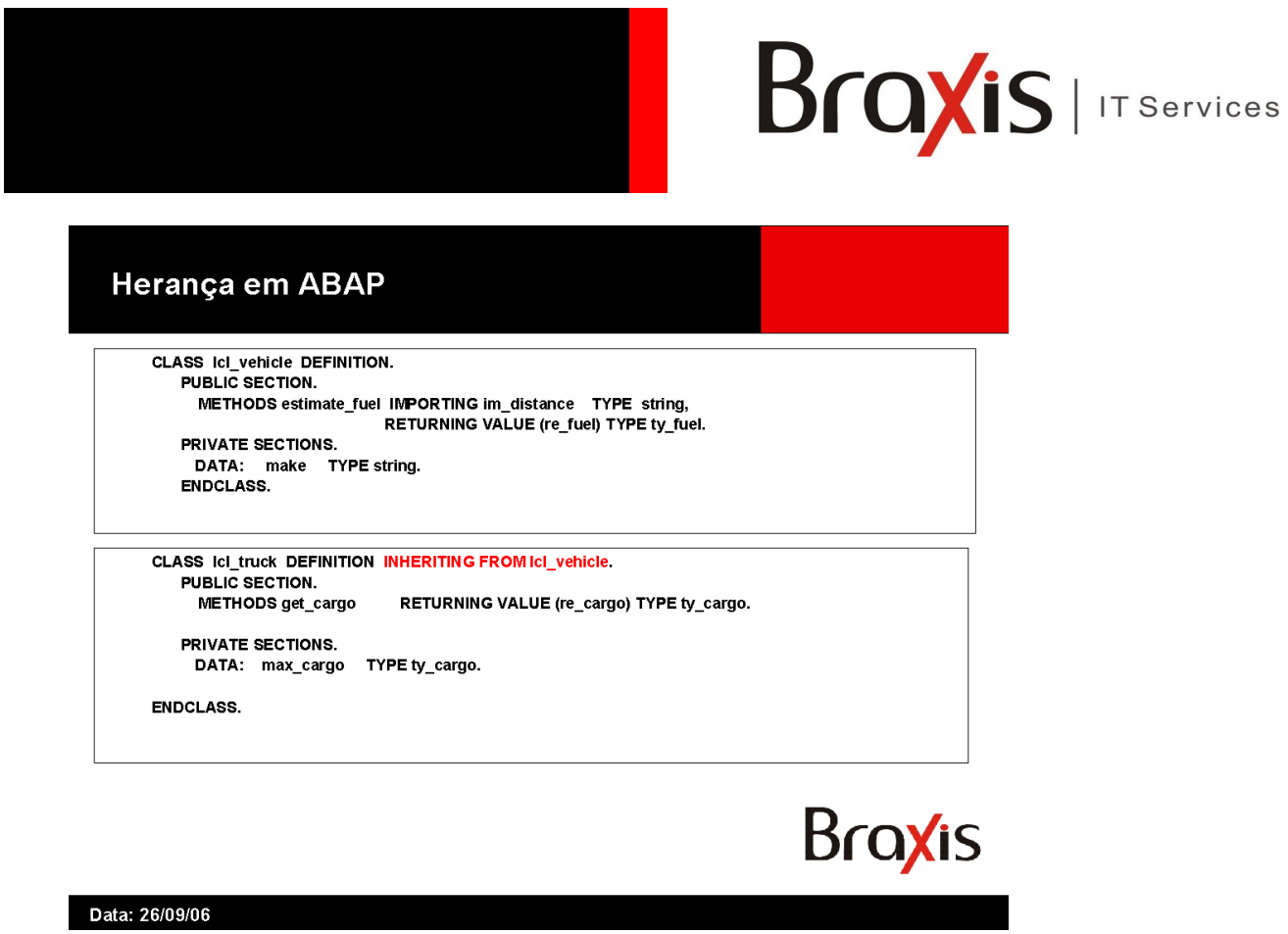
Braxis

Data: 26/09/06

- A Herança é excelente forma de centralizar alguns componentes comuns e poderão ser utilizados por demais subclasses. E ainda permite implementações nas subclasses, não invalidando o que este definido na superclasse.
- Segue abaixo alguns pontos fortes na utilização de herança:

- Centralização de código nas SuperClasses
- Reuso de código pelas subclasses
- É possível fazer redefinições( alterações de métodos)
- As alterações realizadas nas Superclasses são automaticamente herdadas pelas Subclasses.

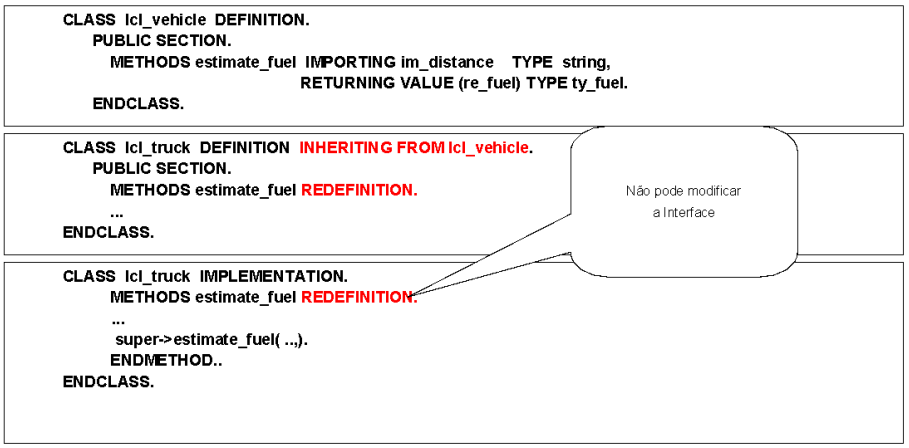

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*




- No exemplo acima a subclasse lcl_truck herda todos os componentes da superclasse lcl_vehicle com a declaração INHERINTING FROM. Então a subclasse já possui o método estimate_fuel e o atributo make.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Redefinição



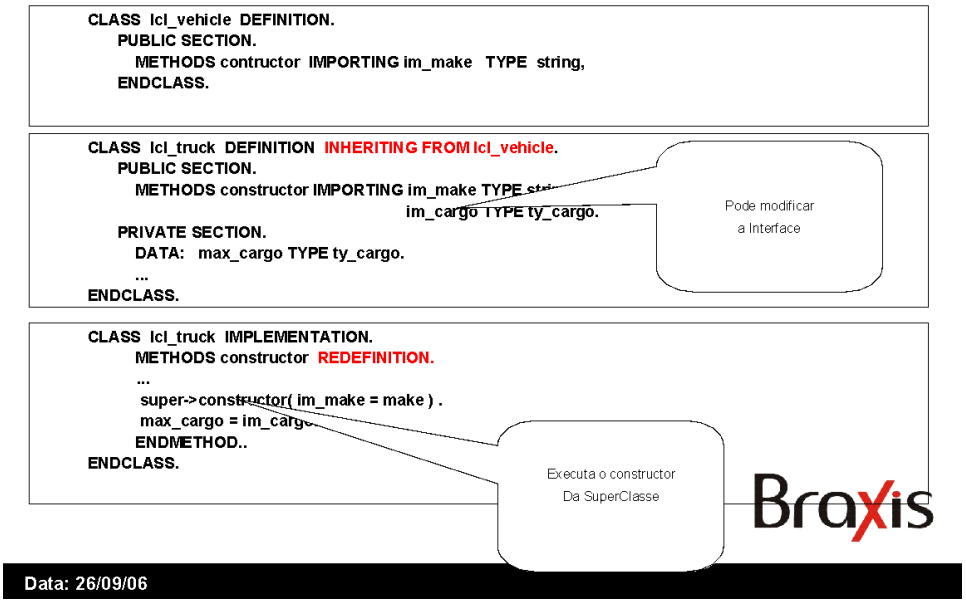

Braxis

Data: 26/09/06

- A REDEFINIÇÃO é uma redefinição do código do método na subclasse herdado da superclasse. No caso do slide acima o método estimate_fuel está sendo redefinido e sendo executado novamente.
- Note que o método estimate_fuel está sendo executado com a declaração super->, onde esta declaração executa o método da superclasse.
- Não é possível redefinir métodos privados.
- ABAP Objetc não suporta overloading, somente no método constructor. Isto é valido para o caso de redefinição sem interface.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Redefinição do Construtor

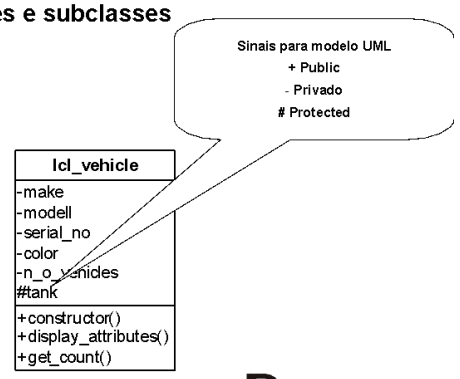



- O constructor da superclasse **sempre deve ser executado,** para iniciar os componentes da superclasse corretamente para garantir a integridade dos componentes.
- No caso o constructor na Subclasse adicionou o atributo max_cargo que pertence a REDEFINIÇÂO da Subclasse.
- Construtores estáticos são chamados automaticamente.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Protect Componentes

- Protect componentes
  - Somente visíveis  para as classes e subclasses
- Public compoentes
  - Qualquer objeto  pode acessar
- Private compoentes
  - Somente a classe pode acessar



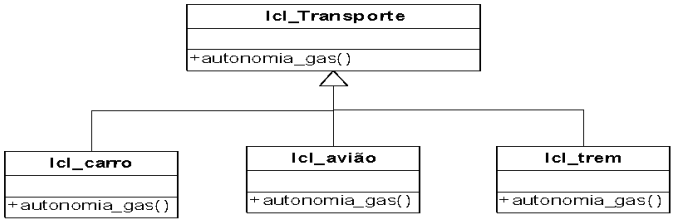

Data: 26/09/06

- Assim como declarações Públicas (  todos acessam ) , Private ( somente as classes acessam) , existe também PROTECTED, onde este permite que componentes sejam visualizados de superclasses para subclasses.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Regras para Redefinição

- **Métodos herdados podem ser redefinidos com as seguintes regras nas subclasses.**
  - **Sempre devem ser implementados na subclasse**
  - **Os parâmetros não podem ser modificados**
  - **Métodos estáticos não podem ser redefinidos**
  - **Em herança, atributos estáticos são compartilhados, elas compartilham PUBLIC e PROTECT atributos com subclasses.**

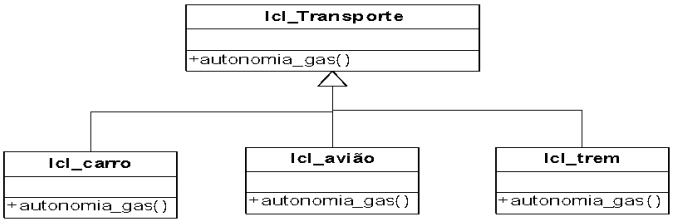



Data: 26/09/06

- Ao utilizar o conceito de redefinição, o ideal é utilizar a notação correta de redefinição. Não somente adicionar novos componentes para as subclasses, e sim, utilizar a redefinição. Redefinindo somente (métodos de instancia Públicos e Protect) , os demais componentes como ( métodos e atributos estáticos) não pode ser redefinidos.
- Não confundir REDEFINITION com OVERLOADING: Onde overloading é um forma de ter classes com métodos com mesmo nome, mas com assinaturas diferentes, isto não é possível em ABAP Objects.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Exemplos de Redefinição




```
CLASS lcl_carro IMPLEMENTATION.
      METHODS autonomia_gas REDEFINITION.

      total = tank * 50.

      ENDMETHOD.
ENDCLASS.
```

```
CLASS lcl_trem IMPLEMENTATION.
      METHODS autonomia_gas REDEFINITION.

      total = tank * 150.

      ENDMETHOD.
ENDCLASS.
```

Data: 26/09/06

- Note que os dois métodos foram redefinidos utilizando regras diferentes para calcular o método autonomia_gas para as classes lcl_carro e lcl_trem.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Hierarquia de Classes**

1.1 Criar uma subclasse lcl_passenger_plane para classe lcl_airplane, no mesmo include ZBXOO_AIRPLANE_XX.

1.2 Implementar também um método Construtor(name e planetype) Público para receber os valores iniciais para os atributos de todas as instancias criadas.

1.3 Criar um atributo privado de instancia max_seats do tipo sflight-seatsmax e incluir como parâmetro do método constructor.

1.4 No método constructor execute o método constructor da superclasse (importando os atributos name e planetype) e depois a atribua algum valor a atributo seats.

1.5 Redefina o método display_attributes da classe lcl_airplane, usando a declaração REDEFINITION e com o comando WRITE exibir o comando max_seats.

1.6 Criar uma subclasse lcl_cargo_plane da classe lcl_airplane, no mesmo include.

1.7 Criar um atributo privado de instancia max_cargo com o mesmo tipo saclane-cargomax e incluir como parâmetro do método constructor.

1.8 Criar um construtor público para receber todas as instancias.

1.9 No método constructor execute o método constructor da superclasse (importando os atributos name e planetype) e depois a atribua algum valor a atributo max_cargo.

1.10 Redefina o método display_attributes da classe lcl_airplane, usando a declaração REDEFINITION e com o comando WRITE exibir o max_cargo.

1.11 No programa principal, utilizando a declaração CREATE OBJETC criar novas referencias das classes lcl_passenger_palne e lcl_cargo_plane.

1.12 Crie um método estático display_n_o_airplanes depois de instanciar os objetos.

1.13 Preencher os atributos na criação das novas classes, através do constructor.

1.14 Chamar o método display_attributes para todas as instancias.

1.15 Chamar o método display_n_o_attributes do objeto lcl_airplane ao final do programa.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**RESPOSTA EXERCÍCIOS:**

## 1. Hierarquia de Classes

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_INHS_MAIN_00                                          *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_INHS_MAIN_00              .


TYPE-POOLS icon.

INCLUDE ZBXOO_INHS_00 .

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane.

DATA: count TYPE i.

START-OF-SELECTION.

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).

CREATE OBJECT r_passenger EXPORTING
        im_name = 'LH Berlin'
        im_planetype = '747-400'
        im_seats = 345.


CREATE OBJECT r_cargo EXPORTING
        im_name = 'AA New York'
          im_planetype = '747-300'
          im_maxcargo = 533.


r_cargo->display_atributes( ).

r_passenger->display_atributes( ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*&---------------------------------------------------------------------*
*&  Include          ZBXOO_AIRPLANE_00_B                               *
*&---------------------------------------------------------------------*


*---------------------------------------------------------------------*
*   CLASS lcl_airplane DEFINITION.                                    *
*---------------------------------------------------------------------*

CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

    CONSTANTS: pos_1 TYPE i VALUE 30.

     METHODS: constructor IMPORTING
               im_name       TYPE string
               im_planetype  TYPE saplane-planetype,

              display_atributes.

      CLASS-METHODS: display_n_o_airplanes.

   PRIVATE SECTION.

   DATA:  name        type     string,
          planetype   type     saplane-planetype.

   CLASS-DATA: n_o_airplanes  type i.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_airplane IMPLEMENTATION.                                *
*---------------------------------------------------------------------*

CLASS lcl_airplane  IMPLEMENTATION.


   METHOD constructor.
     name       = im_name.
     planetype  = im_planetype.
     n_o_airplanes = n_o_airplanes + 1.
   ENDMETHOD.


   METHOD display_atributes.
     WRITE: / icon_ws_plane as icon,
            / 'Name of airplane: '(001), AT pos_1 name,
            / 'Airplane type '(002), AT pos_1 planetype.

   ENDMETHOD.


   METHOD display_n_o_airplanes.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
    WRITE: /,/ 'Total member of planes'(ca1),
           AT pos_1 n_o_airplanes LEFT-JUSTIFIED,/.
  ENDMETHOD.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane DEFINITION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
             im_name       TYPE string
             im_planetype  TYPE saplane-planetype
             im_maxcargo   TYPE scplane-cargomax,

             display_atributes REDEFINITION.

  PRIVATE SECTION.

    DATA: max_cargo TYPE scplane-cargomax.

ENDCLASS.



*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane IMPLEMENTATION.

    METHOD: constructor.

     CALL METHOD super->constructor( im_name = im_name
              im_planetype  = im_planetype ).
              max_cargo = im_maxcargo.

    ENDMETHOD.

    METHOD: display_atributes.

      CALL METHOD super->display_atributes( ).

      WRITE: / 'Max Cargo = ', max_cargo.

    ENDMETHOD.


ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane DEFINITION.
*---------------------------------------------------------------------*
CLASS lcl_passenger_plane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
             im_name       TYPE string
             im_planetype  TYPE saplane-planetype
             im_seats      TYPE sflight-seatsmax,

           display_atributes REDEFINITION.


  PRIVATE SECTION.

    DATA: max_seats TYPE sflight-seatsmax.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_passenger_plane IMPLEMENTATION.

  METHOD constructor.

    CALL METHOD super->constructor( im_name = im_name
             im_planetype  = im_planetype ).
             max_seats = im_seats.

  ENDMETHOD.

  METHOD display_atributes.

   CALL METHOD super->display_atributes( ).

    WRITE: / 'Max Seats = ', max_seats.

  ENDMETHOD.


ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 6. CASTING

### O que é CAST?

```
DATA:  o_veículo  TYPE REF TO lcl_vehicle,
       o_truck    TYPE REF TO lcl_truck,

CREATE OBJECT o_veículo.

CREATE OBJECT o_truck.

* ---Narrow cast
o_veículo = o_truck.
```

Braxis

Data: 26/09/06

- Cast é um conceito que existe nas linguagens orientadas a objeto de generalização, ou seja, atribuir referencias de subclasse a superclasses. No exemplo acima verificamos a classe truck ( subclasse ) a classe veiculo ( superclasse ).


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Conceitos de Casting

- **Casting:**
  - **Instancias de subclasses podem ser usadas em qualquer contexto, depois das referencias de subclasses serem atribuídas as superclasses.**
  - **Os componentes herdados de superclasses podem ser acessados.**
  - **Permite acesso genérico de hierarquia de classes.**

Data: 26/09/06

- Um dos princípios da herança é a instancia de subclasses que podem ser usadas de qualquer forma, então também é possível atribuir subclasses a superclasses, dessa forma, é possível acessar todas as subclasses de uma mesma forma e com uma visão genérica.
- O motivo de se utilizar este conceito é devido a necessidade de visualização das informações de forma genérica, ou seja, se o programa que for desenvolvido não tem necessidade visualizar as classes de forma mais específica, então é utilizado o CAST.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## O que é CAST?

```
DATA:  o_veículo   TYPE REF TO lcl_vehicle,
       o_truck     TYPE REF TO lcl_truck,
       o_carro     TYPE REF TO lcl_carro,
       o_onibus    TYPE REF TO lcl_onibus,

       ti_vehiche TYPE TABLE REF TO lcl_vehicle.


CREATE OBJECT o_carro.
APPENDE o_carro TO ti_vehicle.

CREATE OBJECT o_truck.
APPENDE o_truck TO ti_vehicle.

CREATE OBJECT o_onibus.
APPENDE o_onibus TO ti_vehicle.
```

Data: 26/09/06

- O exemplo acima todas as referencias criadas de subclasses são atribuídas a tabela interna ti_vehicle do tipo lcl_vehicle que é uma superclasse. Dessa forma, é possível ter acesso genérico as informações das subclasses.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Acesso Genérico e Polimorfismo

```
CLASS lcl_rental DEFINITION.
    PUBLIC SECTION
      METHOD add_vehicle IMPORTING im_vehicle TYPE REF TO lcl_vehicle.
      METHOD calc_estimated_fuel RETURNIG VALUE ( re_fuel )

    PROTECTED SECTION
      DATA: vehicle_list TYPE TABLE OF REF TO lcl_vehicle
ENDCLASS.
```

```
CLASS lcl_rental DEFINITION.
    METHOD add_vehicle.
      ADD im_vehicle TO vehicle_list
    ENDMETHOD.

    METHOD calc_estimated_fuel.
      DATA: o_vehicle TYPE REF TO lcl_vehicle.
        LOOP AT vehilce_list INTO o_vehicle.
          re_fuel = re_fuel + o_vehicle->estimated_fuel (  im_distance ).
        ENDLOOP.
    ENDMETHOD.
ENDCLASS.
```

Braxis

Data: 26/09/06

- Polimorfismo é o comportamento de métodos de objetos de diferentes classes com o mesmo nome, ou seja, de acordo com o tipo de classe o método terá um comportamento de acordo com a sua implementação. Isto é possível de ser feito através de herança, redefinindo um método de superclasse na subclasse e implementando de formas diferentes de acordo com especialidade da classe. Polimorfismo é um dos pontos fortes de herança, onde você pode trabalhar da mesma forma com diferentes classes, com a garantia da implementação dos tipos de classes diferentes e o runtime que será encarregado de executar a implementação correta de acordo com a classe.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Polimorfismo

```
CLASS lcl_rental DEFINITION.
  ...
  METHOD calc_estimated_fuel.
    DATA: o_vehicle TYPE REF TO lcl_vehicle.
      LOOP AT vehilce_list INTO o_vehicle.
        re_fuel = re_fuel + o_vehicle->estimated_fuel (  im_distance ).
      ENDLOOP.
  ENDMETHOD.
ENDCLASS.
```

Objeto para a classe ônibus com a implementação com o cálculo especifico para lcl_bus

```
METHOD calc_estimated_fuel.
  total_weight = max_passenger * average_weight
+ weight.
  re_fuel = total_weight * im_distantce * factor.
ENDMETHOD.
```

Objeto para a classe caminhão com a implementação com o cálculo especifico para lcl_trucx

```
METHOD calc_estimated_fuel.
  total_weight = max_cargo  + weight.
  re_fuel = total_weight * im_distantce * factor.
ENDMETHOD.
```

Braxis

Data: 26/09/06

- No LOOP acima na tabela interna vehicle_list que contém de forma genérica todas as referencias das subclasses(lcl_bus, lcl_truck e lcl_car), então de acordo com cada implementação da subclasse no método estimate_fuel, o próprio runtime executará o método da respectiva classe, como mostra o exemplo acima. Para os tipos de classes lcl_bus existe um tipo de cálculo total_weight e para truck ele será outro cálculo. Mas quando executado o método dentro do  loop,  ele será calculado de acordo com a reference criada.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Comparação Linguagem Procedural X OO

- **Nas linguagens de programação orientadas a objetos, não é necessário modificar o código, se você adiciona subclasses com polimorfismo.**
- **Em linguagens procedurais, você utilizaria um CASE para implementar o mesmo código.**

```
LOOP AT vehilce_list INTO o_vehicle.
  CASE vehicle-tipo.
    WHEN 'TRUCK'
      perform estimate_fuel_truck USING .....
    WHEN 'BUS'
      perform estimate_fuel_bus  USING .....
ENDLOOP.
```

Braxis

Data: 26/09/06

- Polimorfismo torna a programação mais fácil, e para caso de ser necessário implementar novos cálculos por exemplo, uma classe trem. No caso de utilização de polimorfismo, somente será necessário incluir na classe e atribuí-la a classe vehicle. No caso de um programa normal, seria necessário, no exemplo acima, criar um novo perform estimate_fuel_trem e alterar o case incluindo a clausula 'TREM'.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Conceitos de Widening

- **Widening:**
  - **È o oposto de Cast**
  - **Instancias de subclasses que já foram alteradas, ou o programa principal precisa de um detalhe que esta nas subclasses, então o processo inverso é feito. É atribuído a subclasse a superclasse.**

```
METHOD calc_estimated_fuel.
        LOOP AT vehilce_list INTO o_vehicle.
         TRY
           o_truck ?=  o_vehicle.
         CATCH CX_SY_MOVE_CAST_ERROR.

         ENDTRY.
        ENDLOOP.
ENDMETHOD.
```

Data: 26/09/06

- O Widening é utilizado quando existe a necessidade de especialização, ter acesso a informação em nível mais especifico. Conforme o exemplo acima foi necessário obter uma informação na subclasse truck, onde não existia no nível da classe vehicle(superclasse)
- O operador utilizado para widening é **?=,** equivalente ao MOVE, TO.
- Outro comando muito importante utilizado é TRY CATCH ( com a classe de exceção CX_SY_MOVE_CAST_ERRO ) que é um comando muito comum é linguagens OO, assim como JAVA, .NET , DELPHI e outras. (Este será detalhado no capitulo de Exceptions).


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Polimorfismo**

1.1 Copiar o programa ZBXOO_MAIN_AIRPLANE_XX com nome de ZBXOO_CASS_MAIN_XX.

1.2 Para os dois objetos que estão sendo criados incluí-los na tabela interna plane_list.

1.3 No final do programa comentar as duas chamadas das instancias r_plane e r_passenger e montar um loop atribuindo ao objeto r_plane (lcl_airplane) e executar o método display_atributes.

1.4 Debugar o programa e verificar as instancias no loop.Você perceberá que são objetos diferentes, executando o mesmo método.

**2. Polimorfismo 2**

2.1 No programa principal criado ZBXOO_CASS_MAIN_XX na declaração data, criar um referência para classe lcl_carrier. ( Copiar a definição e implementação da classe lcl_carrier do include de ZBX00_INHS_OO)

2.2 Crie o objeto r_carrier e informar um nome de uma linha aérea no constructor.

2.3 Adicionar aeronaves já criadas noS objetos r_passenger e r_ cargo no método add_airplane da classe lcl_carrier.

2.4 Chamar o método display_atributes da classe lcl_carrier.

**RESPOSTA EXERCÍCIOS:**

**1. Polimorfismo**

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_CASS_MAIN_00                                          *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_CASS_MAIN_00                .


TYPE-POOLS icon.

INCLUDE ZBXOO_INHS_00 .

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane.

DATA: count TYPE i.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
START-OF-SELECTION.

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).

CREATE OBJECT r_passenger EXPORTING
        im_name = 'LH Berlin'
        im_planetype = '747-400'
        im_seats = 345.
APPEND r_passenger TO plane_list.


CREATE OBJECT r_cargo EXPORTING
        im_name = 'AA New York'
          im_planetype = '747-300'
          im_maxcargo = 533.
APPEND r_cargo TO plane_list.


LOOP AT plane_list INTO r_plane.

  r_plane->display_atributes( ).

ENDLOOP.
```

## 2. Polimorfismo 2

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_CASS_MAIN_00                                          *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_CASS_MAINB_00                .


TYPE-POOLS icon.

INCLUDE ZBXOO_INHS_00 .

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane,
      r_carrier type ref to lcl_carrier.

DATA: count TYPE i.

START-OF-SELECTION.

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
CREATE OBJECT r_carrier EXPORTING im_name = 'Smile$Fly-Travel'.

CREATE OBJECT r_passenger EXPORTING
       im_name = 'LH Berlin'
       im_planetype = '747-400'
       im_seats = 345.
APPEND r_passenger TO plane_list.


CREATE OBJECT r_cargo EXPORTING
       im_name = 'AA New York'
        im_planetype = '747-300'
        im_maxcargo = 533.
APPEND r_cargo TO plane_list.

r_carrier->add_airplane( r_passenger ).

r_carrier->add_airplane( r_cargo ).


r_carrier->display_atributes( ).




*&---------------------------------------------------------------------*
*&  Include          ZBXOO_AIRPLANE_00_B                               *
*&---------------------------------------------------------------------*


*----------------------------------------------------------------------*
*   CLASS lcl_airplane DEFINITION.                                     *
*----------------------------------------------------------------------*

CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

    CONSTANTS: pos_1 TYPE i VALUE 30.

     METHODS: constructor IMPORTING
               im_name       TYPE string
               im_planetype  TYPE saplane-planetype,

             display_atributes.

      CLASS-METHODS: display_n_o_airplanes.

   PRIVATE SECTION.

   DATA:  name        type     string,
          planetype   type     saplane-planetype.

   CLASS-DATA: n_o_airplanes  type i.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_airplane IMPLEMENTATION.                                *
*---------------------------------------------------------------------*

CLASS lcl_airplane  IMPLEMENTATION.


  METHOD constructor.
    name       = im_name.
    planetype  = im_planetype.
    n_o_airplanes = n_o_airplanes + 1.
  ENDMETHOD.


  METHOD display_atributes.
    WRITE: / icon_ws_plane as icon,
           / 'Name of airplane: '(001), AT pos_1 name,
           / 'Airplane type '(002), AT pos_1 planetype.

  ENDMETHOD.


  METHOD display_n_o_airplanes.
    WRITE: /,/ 'Total nember of planes'(ca1),
           AT pos_1 n_o_airplanes LEFT-JUSTIFIED,/.
  ENDMETHOD.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane DEFINITION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
             im_name       TYPE string
             im_planetype  TYPE saplane-planetype
             im_maxcargo   TYPE scplane-cargomax,

             display_atributes REDEFINITION.

  PRIVATE SECTION.

    DATA: max_cargo TYPE scplane-cargomax.

ENDCLASS.



*---------------------------------------------------------------------*
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*   CLASS lcl_cargo_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane IMPLEMENTATION.

    METHOD: constructor.

     CALL METHOD super->constructor( im_name = im_name
               im_planetype  = im_planetype ).
               max_cargo = im_maxcargo.

    ENDMETHOD.

    METHOD: display_atributes.

      CALL METHOD super->display_atributes( ).

      WRITE: / 'Max Cargo = ', max_cargo.

    ENDMETHOD.


ENDCLASS.



*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane DEFINITION.
*---------------------------------------------------------------------*
CLASS lcl_passenger_plane DEFINITION INHERITING FROM lcl_airplane.

   PUBLIC SECTION.

     METHODS: constructor IMPORTING
              im_name       TYPE string
              im_planetype  TYPE saplane-planetype
              im_seats      TYPE sflight-seatsmax,

             display_atributes REDEFINITION.


   PRIVATE SECTION.

     DATA: max_seats TYPE sflight-seatsmax.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_passenger_plane IMPLEMENTATION.

  METHOD constructor.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
    CALL METHOD super->constructor( im_name = im_name
              im_planetype  = im_planetype ).
              max_seats = im_seats.

  ENDMETHOD.

  METHOD display_atributes.

   CALL METHOD super->display_atributes( ).

    WRITE: / 'Max Seats = ', max_seats.

  ENDMETHOD.


ENDCLASS.



*---------------------------------------------------------------------*
*   CLASS lcl_carrier DEFINITION.
*---------------------------------------------------------------------*
CLASS lcl_carrier DEFINITION .

   PUBLIC SECTION.

     METHODS: constructor IMPORTING
              im_name       TYPE string,
              get_name      RETURNING value(ex_name) TYPE string,
              add_airplane  IMPORTING im_plane TYPE REF TO lcl_airplane,
              display_airplanes,
              display_atributes.

   PRIVATE SECTION.

     DATA: name TYPE string,
          airplane_list TYPE TABLE OF REF TO lcl_airplane.

ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_carrier IMPLEMENTATION.

  METHOD constructor.
    name = im_name.
  ENDMETHOD.

  METHOD get_name.
    ex_name = name.
  ENDMETHOD.

  METHOD add_airplane.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
    APPEND im_plane TO airplane_list.
  ENDMETHOD.


  METHOD display_atributes.
    display_airplanes( ).
  ENDMETHOD.


  METHOD display_airplanes.
    DATA: r_plane TYPE REF TO lcl_airplane.
      LOOP AT airplane_list INTO r_plane.
        r_plane->display_atributes( ).
      ENDLOOP.
  ENDMETHOD.

ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Interfaces

- **Conteúdo:**
  - **O que são interfaces?**
  - **Polimorfismo**

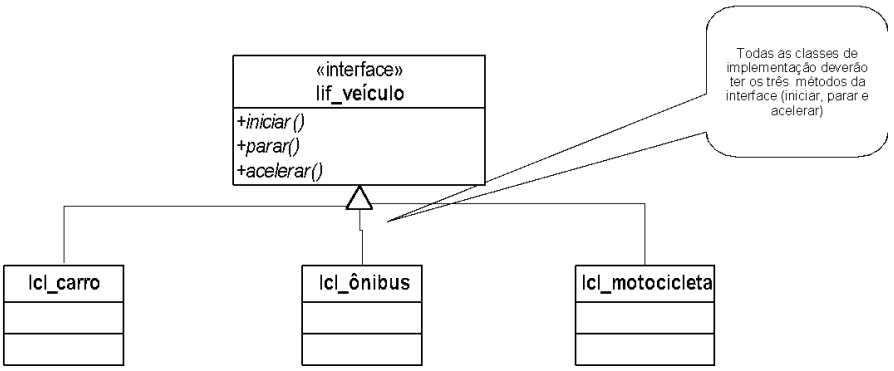
Braxis

Data: 26/09/06

- Interfaces são implementações e adições em classes. Derivadas de linguagens OO, como JAVA, .NET interfaces em ABAP Objects tem a mesma função, manter um padrão de implementação de classes. As interfaces mantém um padrão de desenvolvimento com os serviços nelas definidos como ( métodos) , assim qualquer classe que implementar dessa interface deverá, implementar todos os métodos da interface, mesmo que não utilize, obedecendo a nomenclatura e o padrão. Isto também ajuda na comunicação entre a equipe de desenvolvimento e isto é chamado também de polimorfismo com interfaces.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Interfaces ( Como Implementar )



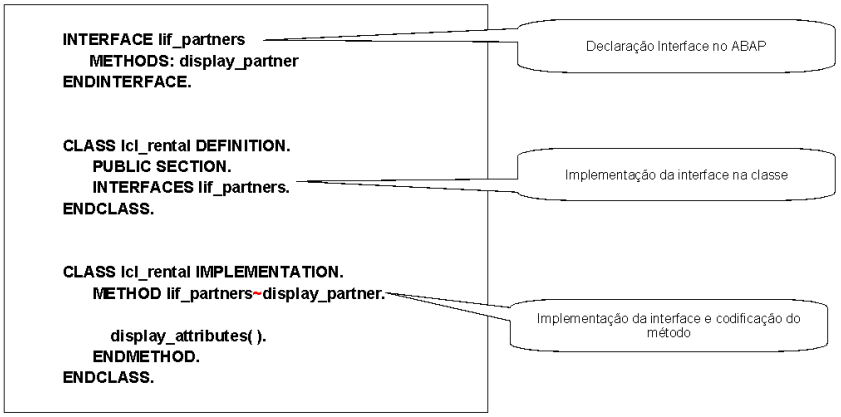

Braxis

Data: 26/09/06

- As interfaces são representadas da mesma forma que classes em UML, exceto pelo nome interfaces.
- Uma classe pode implementar qualquer número de interfaces e uma interface pode ser implementada por qualquer número de classes. Por implementação pode ser simulando múltipla herança.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Interface (ABAP)




Data: 26/09/06

- Em Interfaces componentes como ( atributos, métodos, constantes, tipos, etc.. ) podem ser declarados como em interfaces e implementados em classes, mas estes não visibilidade. (Público , Privado e Protect)

- As interfaces têm as seguintes regras:

  - Somente podem ser declarados métodos públicos
  - As operações são definidas nas interfaces e são implementadas nos métodos das classes.
  - Atributos, events, constants, e tipos definidos nas interfaces estão automaticamente disponíveis na classe de implementação.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Componentes de Interface

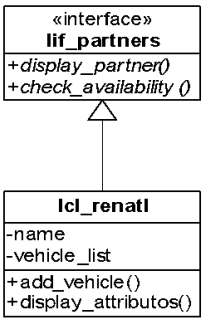
```
CLASS lcl_rental IMPLEMENTATION.
    METHOD lif_partners~display_partner.

        display_attributes( ).
    ENDMETHOD.
ENDCLASS.

DATA: r_rental TYPE REF TO lcl_rental.

CREATE OBJECT r_rental.

r_renatl->lif_partners~display_partner( ) .
```




Data: 26/09/06

- È possível acessar componentes de interface usando objetos reference, das classes implementadas de interfaces.
- Isto permite você diferenciar é feito com operador de interface, somente como as definições de métodos na parte da implementação da classe.

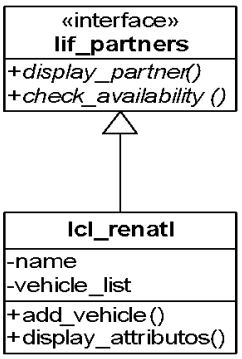

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Referencias de Interfaces

```
DATA: o_rental TYPE REF TO lcl_rental.
      o_li f    TYPE REF TO lif_partners.

CREATE OBJECT r_rental.

o_lif = o_rental.
```

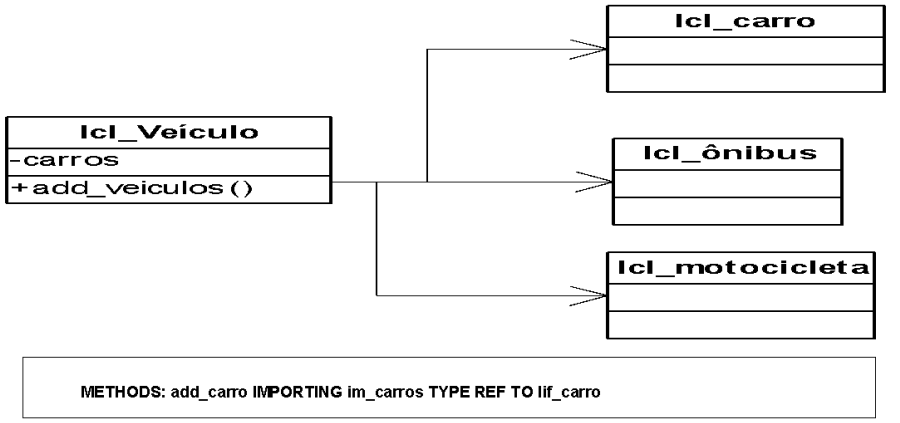



Data: 26/09/06

- Interfaces são acessadas usando referencias de interfaces. Interfaces de referencias sempre referem para instancias de classes.
- A atribuição de um objeto de referencia para uma referencia de interface é conhecido como narrowing cast, por uma herança, somente uma parte de objetos de interfaces é visível somente se você tem atribuído a referencia.
- Com object reference, você pode acessar todos os componentes na classe carregada na implementação, mas somente os componentes definidos na interface. Estes componentes acessam usando exclusividade referencia de interfaces com nomes curtos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Usando Interfaces



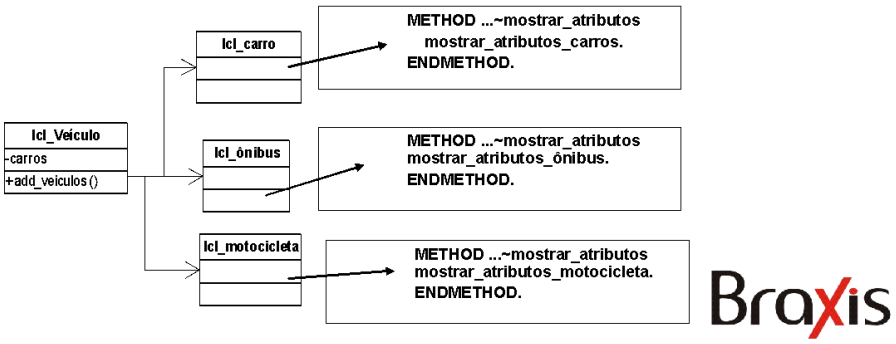

Data: 26/09/06

- A classes implementadas através da classe lcl_veículo recebem todas as outras referencias dos objetos (lcl_carro, lcl_ônibus e lcl_motocicleta) através do método add_veiculos e são gravados, em tabela interna do tipo lcl_veículols.
- A principal vantagem como forma de implementação é o polimorfismo. Pois somente estão disponíveis acessos genéricos e as alterações serão feitas nas implementações.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Polimorfismo e Interfaces

```
METHOD lif_veiculos.
  DATA: o_veiculos TYPE REF TO lif_veiculos.
    LOOP AT veiculos_list INTO o_veicluos.
      o_veiculo->mostrar_atributos( ).
    ENDLOOP.
ENDMETHOD.
```

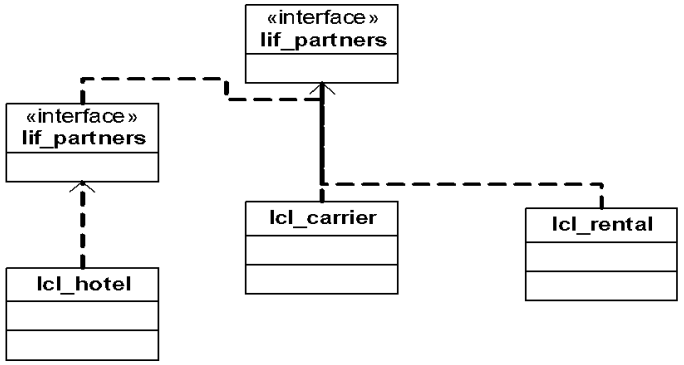



Data: 26/09/06

- È possível utilizar polimorfismo através de interfaces utilizando referencias de interfaces chamando métodos que podem ter diferentes implementações.
- Através de uma tipo dinâmico de através do método o_veiculo->mostrar_atributos( ) é possível executar implementações de forma diferentes para o mesmo nome de método.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Interfaces Compostas




Data: 26/09/06

- ABAP Objects contém modelo de composição para interfaces. Composição de interfaces contém outras interfaces como componentes (componentes de interfaces) a sumarizam a extensão dos componentes de interfaces.
- Uma interface pode ser usada como um componente interface em diversas interfaces compostas.
- Em UML o relacionamento de interfaces compostas (Especialização e Generalização) é definido com linhas pontilhadas.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Interface (ABAP)

```
INTERFACE lif_partners
    METHODS: display_partner
ENDINTERFACE.

INTERFACE lif_room_booking.
    INTERFACES lif_partners.
    METHODS: book_room.
ENDINTERFACE.
```

```
CLASS lcl_hotel DEFINITION.
    PUBLIC SECTION.
    INTERFACES lif_room_booking.
ENDCLASS.

CLASS lcl_hotel  IMPLEMENTATION.
    METHOD lif_partners~display_partner.
    ENDMETHOD.

    METHOD lif_room_booking~book_room.
    ENDMETHOD.
ENDCLASS.
```

```
DATA:  i_partner          TYPE REF TO lif_partners,
       i_room_book        TYPE REF TO lif_room_booking.

i_partner = i_room_book.

i_room_book->lif_partner~display_partner( ).
i_room_book ?= i_partner.  "Widening
```

Data: 26/09/06

- A sintaxe do ABAP OO para composição de interfaces é <componente-nomedainterface>~<nomecomponente>.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Interfaces**

1.1.1 Incluir no programa principal (ZBXOO_CASS_MAIN_XX) o include ZBXOO_INT_XX (copiar do include ZBXOO_INT_00) com a implementação da interface lif_partners e a classe lcl_travel_agency.
1.1.2 Criar os objetos com os seguintes tipos:
1.1.2.1.1 r_carrier TYPE REF TO lcl_carrier
1.1.2.1.2 r_agency TYPE REF TO lcl_travel_agency
1.1.2.1.3 r_rental TYPE REF TO lcl_rental
1.1.2.1.4 r_truck TYPE REF TO lcl_truck
1.1.2.1.5 r_bus TYPE REF TO lcl_bus

1.1.3 Criar um objeto r_carrier
1.1.4 Utilizando a instancia r_carrier chamar o método add_airplane passando como parâmetro as instancias r_passenger e r_cargo.
1.1.5 Chamar o método display_atributes da instancia r_carrier.
1.1.6 Criar uma instancia r_rental
1.1.7 Criar uma instancia r_truck (incluir name(fabricante) e cargo(modelo).
1.1.8 Através da instancia r_rental chamar o método add_vehicle e passar como parâmetro (r_truck)
1.1.9 Criar uma instancia r_-bus.
1.1.10 Através da instancia r_rental chamar o método add_vehicle e passar como parâmetro (r_bus)
1.1.11 Criar uma instancia r_truck (incluir name(fabricante) e cargo(modelo).
1.1.12 Através da instancia r_rental chamar o método add_vehicle e passar como parâmetro (r_truck)
1.1.13 Executar o método r_agency->display_agency_partners( ).

**RESPOSTA EXERCÍCIOS:**

**1. Interfaces**

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_INT_MAIN_00                                           *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_INT_MAIN_00                .


TYPE-POOLS icon.

INCLUDE ZBXOO_INT_00.
INCLUDE ZBXOO_INHS_00.
*INCLUDE ZBXOO_INT_AIR_00.

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane,
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
      r_carrier  type ref to lcl_carrier,
      r_agency TYPE REF TO lcl_travel_agency,
      r_rental TYPE REF TO lcl_rental,
      r_truck TYPE REF TO lcl_truck,
      r_bus TYPE REF TO lcl_bus.


DATA: count TYPE i.

START-OF-SELECTION.


CREATE OBJECT r_carrier EXPORTING im_name = 'Smile$Fly-Travel'.

CREATE OBJECT r_agency EXPORTING im_name = 'CVC - Turismo'.

*r_agency->add_partner( r_carrier ).


CREATE OBJECT r_passenger EXPORTING
       im_name = 'LH Berlin'
       im_planetype = '747-400'
       im_seats = 345.

CREATE OBJECT r_cargo EXPORTING
       im_name = 'AA New York'
         im_planetype = '747-300'
         im_maxcargo = 533.


r_carrier->add_airplane( r_passenger ).

r_carrier->add_airplane( r_cargo ).

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).


CREATE OBJECT r_agency EXPORTING im_name = 'CVC'.


CREATE OBJECT r_rental EXPORTING im_name = 'RENT A CAR'.


CREATE OBJECT r_truck  EXPORTING im_name = 'MACK'
                         im_cargo = '458'.

r_rental->add_vehicle( r_truck ).


CREATE OBJECT r_bus  EXPORTING   im_name = 'MERCEDES'
                         im_passengers = '80'.

r_rental->add_vehicle( r_bus ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
CREATE OBJECT r_truck  EXPORTING im_name = 'VOLVO'
                                 im_cargo = '48'.

r_rental->add_vehicle( r_truck ).


r_agency->add_partner( r_rental ).


r_carrier->display_atributes( ).

r_agency->display_agency_partners( ).




*&---------------------------------------------------------------------*
*&  Include          ZBXOO_INT_00                                      *
*&---------------------------------------------------------------------*

TYPES ty_fuel TYPE P DECIMALS 2.
TYPES ty_cargo TYPE P DECIMALS 2.

*---------------------------------------------------------------------*
*   INTERFACES lcl_vehicle
*---------------------------------------------------------------------*

INTERFACE lif_partners.
  METHODS display_partner.
ENDINTERFACE.


*---------------------------------------------------------------------*
*   CLASS lcl_vehicle DEFINITION.                                     *
*---------------------------------------------------------------------*

CLASS lcl_vehicle DEFINITION.

  PUBLIC SECTION.

    METHODS: get_average_fuel
             IMPORTING
               im_distance TYPE s_distance
               im_fuel     TYPE ty_fuel
             RETURNING
               VALUE(re_avgfuel) TYPE ty_fuel.
    METHODS   constructor
              IMPORTING
               im_make       TYPE string.
    METHODS  display_atributes.
    METHODS  set_make
              IMPORTING
                im_make   TYPE string.
    METHODS  get_make
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
            EXPORTING
            ex_make TYPE string.

      CLASS-METHODS: get_count EXPORTING re_count TYPE i.


    PRIVATE SECTION.

    DATA:  make     type     string.

    METHODS: init_make.

    CLASS-DATA: n_o_vehicles TYPE i.

  ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_vehicle IMPLEMENTATION.                                 *
*---------------------------------------------------------------------*

CLASS lcl_vehicle  IMPLEMENTATION.

    METHOD get_average_fuel.
      re_avgfuel = im_distance / im_fuel.
    ENDMETHOD.

    METHOD constructor.
      make    = im_make.
      n_o_vehicles = n_o_vehicles + 1.
    ENDMETHOD.

    METHOD display_atributes.
     WRITE: make.
    ENDMETHOD.


    METHOD set_make.
     IF im_make IS INITIAL.
        me->init_make( ).
       ELSE.
        make = im_make.
     ENDIF.
    ENDMETHOD.

    METHOD init_make.
       make = 'default make'.
    ENDMETHOD.

    METHOD get_make.
       ex_make = make.
    ENDMETHOD.

    METHOD get_count.
       re_count = n_o_vehicles.
    ENDMETHOD.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_truck   DEFINITION.                                     *
*---------------------------------------------------------------------*

CLASS lcl_truck DEFINITION INHERITING FROM lcl_vehicle.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING im_name  TYPE string
                                   im_cargo TYPE ty_cargo.

    METHODS     display_atributes REDEFINITION.

    METHODS  get_cargo RETURNING value(re_cargo) TYPE ty_cargo.


  PRIVATE SECTION.

    DATA: max_cargo TYPE ty_cargo.

ENDCLASS.

CLASS lcl_truck IMPLEMENTATION.

    METHOD constructor.
      super->constructor( im_name ).
      max_cargo = im_cargo.
    ENDMETHOD.

    METHOD display_atributes.
      super->display_atributes( ).
      WRITE: 20 ' Cargo = ', max_cargo.
      ULINE.
    ENDMETHOD.

    METHOD get_cargo.
      re_cargo = max_cargo.
    ENDMETHOD.

ENDCLASS.


*---------------------------------------------------------------------*
*   CLASS lcl_bus   DEFINITION.                                       *
*---------------------------------------------------------------------*

CLASS lcl_bus DEFINITION INHERITING FROM lcl_vehicle.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING im_name  TYPE string
                                   im_passengers TYPE i.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
    METHODS     display_atributes REDEFINITION.

  PRIVATE SECTION.

    DATA: max_passengers TYPE ty_cargo.

ENDCLASS.

CLASS lcl_bus IMPLEMENTATION.

    METHOD constructor.
      super->constructor( im_name ).
      max_passengers = im_passengers.
    ENDMETHOD.

    METHOD display_atributes.
      super->display_atributes( ).
      WRITE: 20 ' Passengers = ', max_passengers.
      ULINE.
    ENDMETHOD.

ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_rental  DEFINITION.                                     *
*---------------------------------------------------------------------*

CLASS lcl_rental  DEFINITION.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING im_name  TYPE string.

    METHODS  add_vehicle   IMPORTING im_vehicle
                        TYPE REF TO lcl_vehicle.



    METHODS  display_atributes.

    INTERFACES: lif_partners.


  PRIVATE SECTION.

    DATA: name TYPE string,
          vehicle_list TYPE TABLE OF REF TO lcl_vehicle.


ENDCLASS.

CLASS lcl_rental IMPLEMENTATION.

    METHOD lif_partners~display_partner.
      display_atributes( ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
    ENDMETHOD.

    METHOD constructor.
     name = im_name.
    ENDMETHOD.

    METHOD add_vehicle.
      APPEND im_vehicle TO vehicle_list.
    ENDMETHOD.

    METHOD display_atributes.
      DATA: r_vehicle TYPE REF TO lcl_vehicle.
      WRITE: 'Lista de Veículos '. ULINE. ULINE.
      LOOP AT vehicle_list INTO r_vehicle.
        r_vehicle->display_atributes( ).
      ENDLOOP.
      ULINE.
    ENDMETHOD.

ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_travel_agency  DEFINITION.                              *
*---------------------------------------------------------------------*

CLASS lcl_travel_agency  DEFINITION.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING im_name  TYPE string.

    METHODS  add_partner IMPORTING im_partner
                         TYPE REF TO lif_partners.


    METHODS  display_agency_partners.


  PRIVATE SECTION.

    DATA: name TYPE string,
          partner_list TYPE TABLE OF REF TO lif_partners.


ENDCLASS.

CLASS lcl_travel_agency IMPLEMENTATION.

    METHOD constructor.
     name = im_name.
    ENDMETHOD.

    METHOD add_partner.
      APPEND im_partner TO partner_list.
    ENDMETHOD.
```

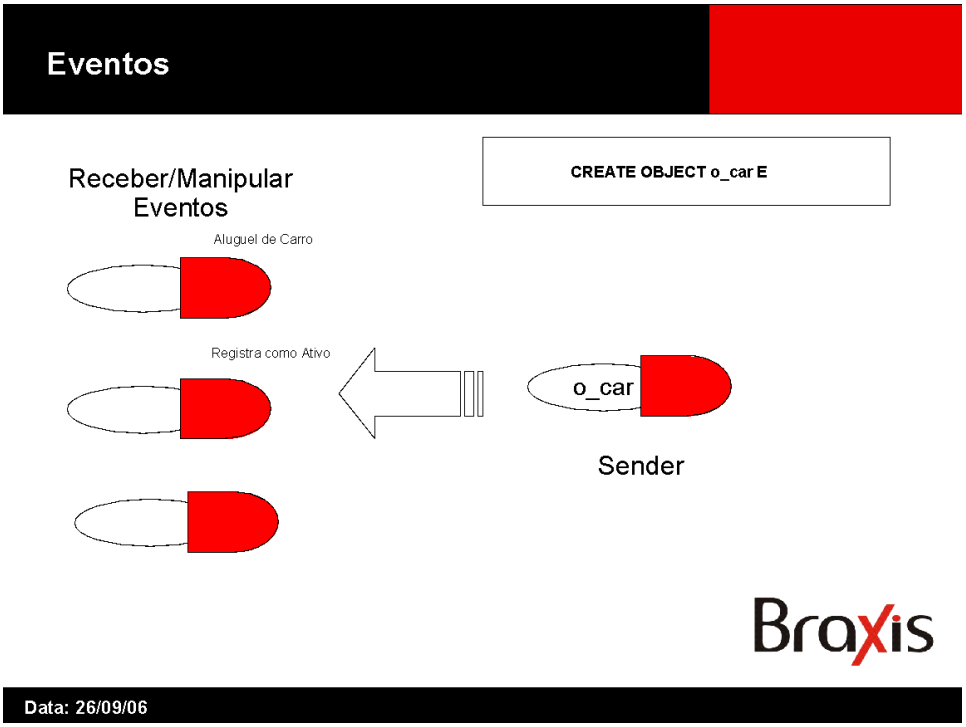

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
  METHOD display_agency_partners.
    DATA: r_partner TYPE REF TO lif_partners.
    WRITE: 'Lista de partners'. ULINE. ULINE.
    LOOP AT partner_list INTO r_partner.
      r_partner->display_partner( ).
    ENDLOOP.
  ENDMETHOD.

ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 8. EVENTOS




- Eventos podem ser iniciados (triggers) e podem, por exemplo, iniciar uma ação. No exemplo acima, a classe carro, após a criação da instancia o_car, aciona um evento para as outras classes “Aluguel de Carro” e “Registro de Ativo”.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Acionando e Manipulando Eventos

- **Triggering Events**
  - Definição Classes de Eventos
    - (EVENTS, CLASS-EVENTS)
  - Objeto ou Classe com trigger
    - (RAISE EVENT)
- **Handling Events**
  - Métodos e Classes de manipulação de eventos
  - Objeto de manipulação de eventos
    - (CLASS- METHODS ... FOR EVENT... OF....)
  - Objetos manipuladores de eventos ou classes registradas com eventos próprios.
    - (SET HANDLER)

Data: 26/09/06

- Tipos de EVENTOS

  - Eventos de Instancias ( utilizando a declaração EVENTS )
  - Eventos Estáticos ( utilizando declaração CLASS-EVENTS )
- Classes ou instancias de classes que recebem uma mensagem quando um evento é acionado em runtime:
  CLASS-METHODS <handler_method> FOR EVENT <event> OF <classname>
- Estas classes ou instancias dessas classes são registradas para uma, ou mais de um evento em runtime:

  - SET HANDLER <handler_method> FOR <reference>. Para métodos de instancias.
  - SET HANDLER <handler_method>. Para métodos estáticos
- Uma classe ou instancia pode acionar um evento em runtime usando a declaração RAISE EVENT.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

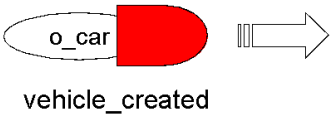
## Sintaxe ABAP para EVENTOS

```
CLASS <classname> DEFINITION.
    EVENTS: <events> EXPORTING VALUE (ex_par) TYPE <type>
ENDCLASS.

CLASS <classname>I  IMPLEMENTATION.
    METHOD <nome_método>
     RAISE EVENT <event> EXPORTING <ex_par> = <act_par>
    ENDMETHOD.
ENDCLASS.
```

```
CLASS lcl_vehicle  DEFINITION.
    PUBLIC SECTION.
    METHODS constructor IMPORTING ...
    EVENTS vehicle_created.
ENDCLASS.

CLASS lcl_vehicle  IMPLEMENTATION.
    METHOD constructor.
     RAISE EVENT vehicle_created.
    ENDMETHOD.
ENDCLASS.
```

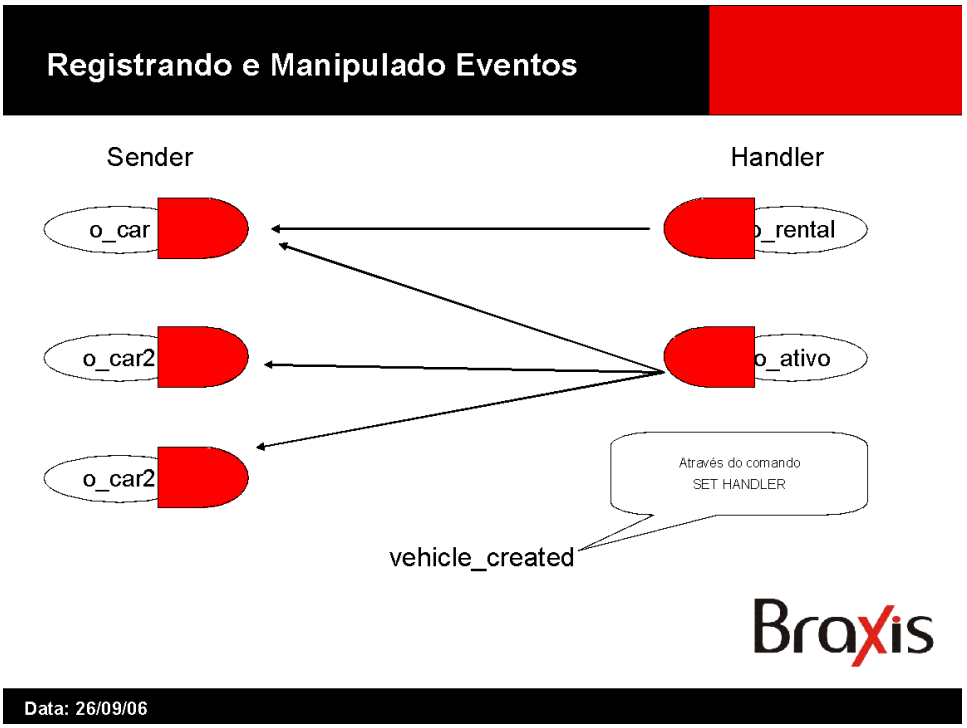



vehicle_created

Data: 26/09/06

- Ambos os eventos de instancia e estáticos podem ser acionados em métodos de instancia.
- Somente eventos estáticos podem ser acionados em estáticos métodos.
- Eventos podem somente ter o parâmetro EXPORTING, onde deve ser passado como valor.
- Acionando um Evento usando a declaração RAISE EVENT tem o seguinte efeito:
  - O fluxo do programa é interrompido naquele ponto.
  - Uma vez que os métodos de event handler foram executados, o fluxo do programa continua.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*



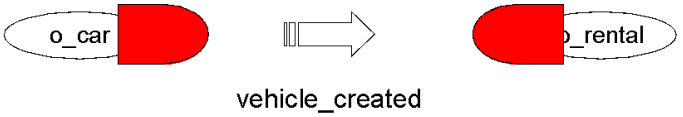

- Eventos são registrados usando  a declaração SET HANDLER. Este registro somente é ativo em runtime.
- É possível também registrar um objeto para um evento através de outro objeto. A declaração SET HANDLER entra todos os registros de eventos em uma lista de objetos. Todos manipuladores( HANDLER)  são entrados na lista.
- Quando um evento é acionado, a lista exibe quais métodos serão acionados.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Métodos de Manipular Eventos

```
CLASS <class_handle> DEFINITION.
    METHODS: <on_events> FOR EVENT <event>
                         OF <classname> | <interface>
                         IMPORTING <ex_par1> ...<ex_parN> [sender].
ENDCLASS.

CLASS lcl_rental  IMPLEMENTATION.
    PRIVATE SECTION.
      METHODS: add_vehicle FOR EVENT vehicle_create OF lcl_vehicle IMPORTING sender.
ENDCLASS.
```




vehicle_created

Data: 26/09/06

- Métodos de Manipulação de Eventos são acionados por RAISE EVENT, uma vez que eles podem ser chamados como métodos. (CALL METHOD)
- A interface de um método de manipulação de evento consiste somente de parâmetros de IMPORTING.

  Você pode somente usar parâmetros de definições aos eventos correspondentes. Um evento de interface, o qual somente tem parâmetros EXPORTING, é definido usando a declaração EVENTOS na declaração de evento. Os parâmetros são do tipo de definição de evento e o tipo é passado método manipulador de evento, que é, uma interface de parâmetros do método manipulador de evento não pode ser tipado na definição método de manipulação de evento.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Registrando Eventos

```
SET HANDLER <ref_handle>-><on_event>
                FOR <ref_sender> | FOR ALL INSTANCES
                [ACTIVATION <var>].

CLASS lcl_rental  DEFINITION.
    PRIVATE SECTION.
      METHODS: add_vehicle FOR EVENT vehicle_create OF lcl_vehicle IMPORTING sender.
ENDCLASS.

CLASS lcl_rental  IMPLENTATION.
       METHOD constructor.
        SET HANDLER add_vehicle FOR ALL INSTANCES.
       ENDMETHOD.
ENDCLASS.
```




vehicle_created

Braxis

Data: 26/09/06

- Quando um evento é acionado, somente estes métodos manipuladores de eventos são executados, que tem o ponto de registro usando SET HANDLER.
- É possível registrar um evento usando ACTIVATION ‘X’, e desregistrar ele usando ACTIVATION space. Se você fazer a especificação ACTIVATION, então os eventos ficam com o comportamento default.
- Você pode registrar diversos métodos em único SET HANDLER:

```
SET HANDLER     <ref_handler1>-><handler_method1>
                <ref_handler2>-><handler_method2>

                .....
                <ref_handlerN>-><handler_methodN>
          FOR   <ref_sender> | FOR ALL INSTANCES.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Manipulando Eventos (Características)

- **Manipulação de Eventos são seqüenciais**
- **Garbage Collector tem a mesma integridade dos objetos registrados.**
- **Objetos registrados nunca são deletados**
- **A visibilidade de um evento define a autorização para evento de manipulação.**
- **A visibilidade de um método manipulador de evento define a autorização usando SET HANDLER.**

Data: 26/09/06

- Se diversos objetos estão registrados para um evento, então a seqüência nos quais os métodos manipuladores de eventos são chamados, e não estão definidos, isto é, não existe garantia de seqüência nos métodos de manipulações de eventos que são chamados.
- Se um novo evento de manipulação é registrado em um método de manipulação de evento para um único evento que tinha sido acionado, então este evento é adicionado no fim da seqüência, e é então executado quando eles estão retornando.
- Se um event handler é desregistrado em um método manipulador de evento, então este manipulador é deletado de uma seqüência de método manipulador de evento.
- Eventos tem o conceito de visibilidade podem ser públicos, privados e protected, onde:
  - PUBLIC Todos podem acessar.
  - PRIVATE Somente acesso dentro da classe.
  - PROTECTED Somente Classes e SubClasses
- Métodos manipuladores de eventos tem visibilidade de atributos. Métodos manipulação de eventos, entretanto, somente podem ter mesma visibilidade ou mais restrições de visibilidade que os eventos a que eles se referem. A visibilidade de método de manipulação estabelece autorização por SET HANDLER. Esta declaração pode ser usada.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Eventos**

1.1 Copiar o include ZBXOO_INT_00 para o include ZBXOO_EVE_XX incluir um evento chamado de **airplane_create** no método add_airplane
Importando o tipo de classe lcl_airplane.
1.2 Atribuir o SET HANDLER no constructor para o evento.
1.3 Implementar no método add_airplane a chamado do evento.
1.4 Comentar no programa principal os métodos r_carrier->add_airplane.

**RESPOSTAS EXERCÍCIOS:**

**1. Eventos**

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_INT_MAIN_00                                           *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_EVE_MAIN_00               .


TYPE-POOLS icon.

INCLUDE ZBXOO_INT_00.
INCLUDE ZBXOO_EVE_00.


DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane,
      r_carrier  type ref to lcl_carrier,
      r_agency TYPE REF TO lcl_travel_agency,
      r_rental TYPE REF TO lcl_rental,
      r_truck TYPE REF TO lcl_truck,
      r_bus TYPE REF TO lcl_bus.



DATA: count TYPE i.

START-OF-SELECTION.

CREATE OBJECT r_carrier EXPORTING im_name = 'Smile$Fly-Travel'.

CREATE OBJECT r_agency EXPORTING im_name = 'CVC - Turismo'.

*r_agency->add_partner( r_carrier ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
CREATE OBJECT r_passenger EXPORTING
        im_name = 'LH Berlin'
        im_planetype = '747-400'
        im_seats = 345.

CREATE OBJECT r_cargo EXPORTING
        im_name = 'AA New York'
          im_planetype = '747-300'
          im_maxcargo = 533.

*r_carrier->add_airplane( r_passenger ).
*
*r_carrier->add_airplane( r_cargo ).

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).


CREATE OBJECT r_agency EXPORTING im_name = 'CVC'.


CREATE OBJECT r_rental EXPORTING im_name = 'RENT A CAR'.


CREATE OBJECT r_truck  EXPORTING im_name = 'MACK'
                          im_cargo = '458'.

r_rental->add_vehicle( r_truck ).


CREATE OBJECT r_bus  EXPORTING   im_name = 'MERCEDES'
                          im_passengers = '80'.

r_rental->add_vehicle( r_bus ).


CREATE OBJECT r_truck  EXPORTING im_name = 'VOLVO'
                          im_cargo = '48'.

r_rental->add_vehicle( r_truck ).


r_agency->add_partner( r_rental ).


r_carrier->display_atributes( ).

r_agency->display_agency_partners( ).



*&---------------------------------------------------------------------*
*&  Include          ZBXOO_EVE_00                                      *
*&---------------------------------------------------------------------*

*----------------------------------------------------------------------*
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*   CLASS lcl_airplane DEFINITION.                            *
*---------------------------------------------------------------------*

CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

    CONSTANTS: pos_1 TYPE i VALUE 30.

    METHODS: constructor IMPORTING
               im_name       TYPE string
               im_planetype  TYPE saplane-planetype,

             display_atributes.

     CLASS-METHODS: display_n_o_airplanes.

     EVENTS airplane_created.

  PRIVATE SECTION.

  DATA:  name       type     string,
         planetype  type     saplane-planetype.

  CLASS-DATA: n_o_airplanes  type i.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_airplane IMPLEMENTATION.                        *
*---------------------------------------------------------------------*

CLASS lcl_airplane  IMPLEMENTATION.


  METHOD constructor.
    name       = im_name.
    planetype  = im_planetype.
    n_o_airplanes = n_o_airplanes + 1.
    raise event airplane_created.
  ENDMETHOD.


  METHOD display_atributes.
    WRITE: / icon_ws_plane as icon,
           / 'Name of airplane: '(001), AT pos_1 name,
           / 'Airplane type '(002), AT pos_1 planetype.

  ENDMETHOD.


  METHOD display_n_o_airplanes.
    WRITE: /,/ 'Total nember of planes'(ca1),
           AT pos_1 n_o_airplanes LEFT-JUSTIFIED,/.
  ENDMETHOD.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane DEFINITION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
          im_name       TYPE string
          im_planetype  TYPE saplane-planetype
          im_maxcargo   TYPE scplane-cargomax,

          display_atributes REDEFINITION.

  PRIVATE SECTION.

    DATA: max_cargo TYPE scplane-cargomax.

ENDCLASS.



*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane IMPLEMENTATION.

   METHOD: constructor.

    CALL METHOD super->constructor( im_name = im_name
          im_planetype  = im_planetype ).
          max_cargo = im_maxcargo.

   ENDMETHOD.

   METHOD: display_atributes.

     CALL METHOD super->display_atributes( ).

     WRITE: / 'Max Cargo = ', max_cargo.

   ENDMETHOD.


ENDCLASS.



*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane DEFINITION.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*---------------------------------------------------------------------*
CLASS lcl_passenger_plane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
             im_name       TYPE string
             im_planetype  TYPE saplane-planetype
             im_seats      TYPE sflight-seatsmax,

            display_atributes REDEFINITION.


  PRIVATE SECTION.

    DATA: max_seats TYPE sflight-seatsmax.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_passenger_plane IMPLEMENTATION.

  METHOD constructor.

    CALL METHOD super->constructor( im_name = im_name
              im_planetype  = im_planetype ).
              max_seats = im_seats.

  ENDMETHOD.

  METHOD display_atributes.

   CALL METHOD super->display_atributes( ).

    WRITE: / 'Max Seats = ', max_seats.

  ENDMETHOD.


ENDCLASS.


*---------------------------------------------------------------------*
*   CLASS lcl_carrier DEFINITION.
*---------------------------------------------------------------------*
CLASS lcl_carrier DEFINITION .

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
            im_name       TYPE string,
            get_name      RETURNING value(ex_name) TYPE string,
            add_airplane  FOR EVENT airplane_created of lcl_airplane
                    IMPORTING sender,
            display_airplanes,
            display_atributes.

    PRIVATE SECTION.

      DATA: name TYPE string,
           airplane_list TYPE TABLE OF REF TO lcl_airplane.

ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_carrier IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_carrier IMPLEMENTATION.

  METHOD constructor.
    name = im_name.
    SET HANDLER add_airplane for all instances.
  ENDMETHOD.

  METHOD get_name.
    ex_name = name.
  ENDMETHOD.

  METHOD add_airplane.
   APPEND sender TO airplane_list.
  ENDMETHOD.


  METHOD display_atributes.
   display_airplanes( ).
  ENDMETHOD.


  METHOD display_airplanes.
    DATA: r_plane TYPE REF TO lcl_airplane.
      LOOP AT airplane_list INTO r_plane.
        r_plane->display_atributes( ).
      ENDLOOP.
  ENDMETHOD.

ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Conceitos: Classes Locais e Interfaces

```
INCLUDE  class_xxx.

CLASS lcl_airplane DEFINITION. ]
...
ENDCLASS.
```

```
REPORT programa1.

DATA: r_airplane TYPE REF TO lcl_airplane.
```

- **Classes Locais**
  - **Classes locais são somente visíveis no programa onde elas foram definidas**
  - **Não é possível acesso global**
  - **Não está gravada no Repositório R/3, e não é possível executar where-used list.**

Braxis

Data: 26/09/06

- Classes locais e interfaces são somente conhecidas no programa onde elas foram definidas e implementadas.
- Classes locais e interfaces não estão armazenadas no Repositório R/3 ( na tabela TADIR). Dessa forma não é possível acesso global.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Classes Globais e Interfaces

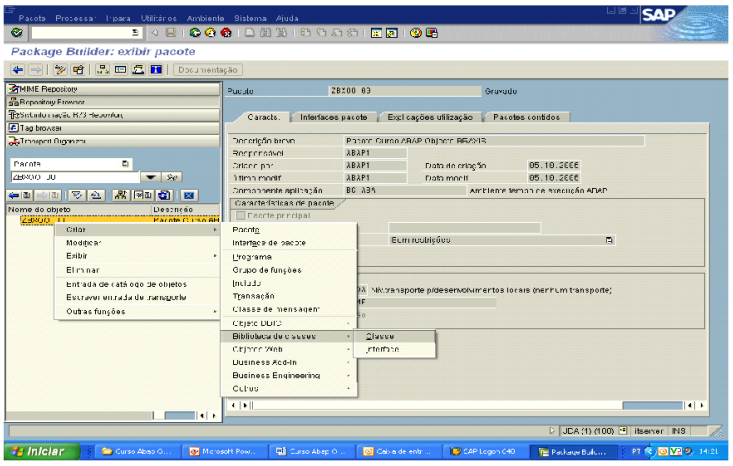
- **ABAP Workbenck development tool:Class Builder**
- **Ferramenta para criação, testes e gerenciamento de classes globais e interfaces**
  - **Armazenamento de Repositório**
  - **Geração de código framework, por exemplo CLASS <nome> DEFINITION.**
- **Namespaces customizáveis para classes e interfaces: Y* ou Z*.**
- **Utilização Where-used list disponível.**

Data: 26/09/06

- Diferentemente de classes e interfaces locais, classes globais e interfaces pode ser criadas e implementadas usando ABAP Workbench tool Class Builder.E estas classes estarão disponíveis em todo ambiente R/3.
- Nomes de classes globais e interfaces são compartilhados com o mesmo namespace.
- Métodos individuais de classes globais podem ser transportados separadamente.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

# Criando Classes Globais no Object Navigator

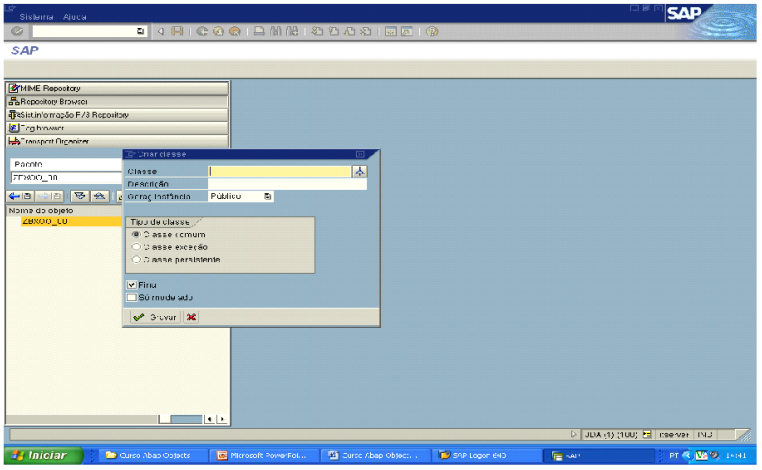



Data: 26/09/06

- As classes e interfaces podem ser visualizadas no Objetc Navigator (SE80).
- È possível criar e visualizar as classes através do Objetc Navigator (SE80). Para criar, selecione por exemplo, a opção pacote e clique com botão direito e selecione a opção biblioteca de classes. Você também pode escolher a na SE80 a “Classe/Interface”.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Criando Classes Globais no Object Navigator



Braxis

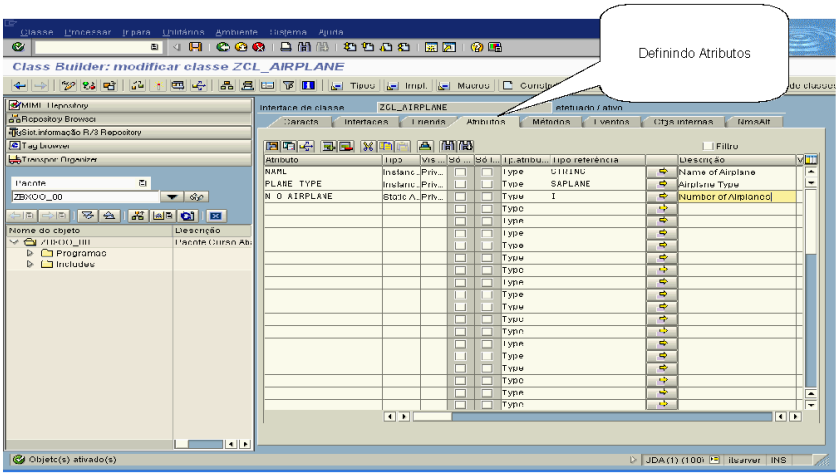
Data: 26/09/06

- Entre com o nome da Classe e sua descrição, em seguida será necessário informar o pacote.
- Será exibida uma caixa de diálogo para informar maiores detalhes:

  - Geração de Instancia: Onde as instancias serão criadas?
  - Por exemplo, se você especificar Privado a classe somente poderá ser instanciado dentro da própria classe, o default é Público.
  - Classe de Exceção: ( Será discutido depois ).
  - Classe Persistente: ( Será discutido depois ).
  - Classe Final ( Representa o fim de uma arvore de herança.
  - Só Modelado ( Definição de Classe por modelo). ]


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Class Builder: Atributos

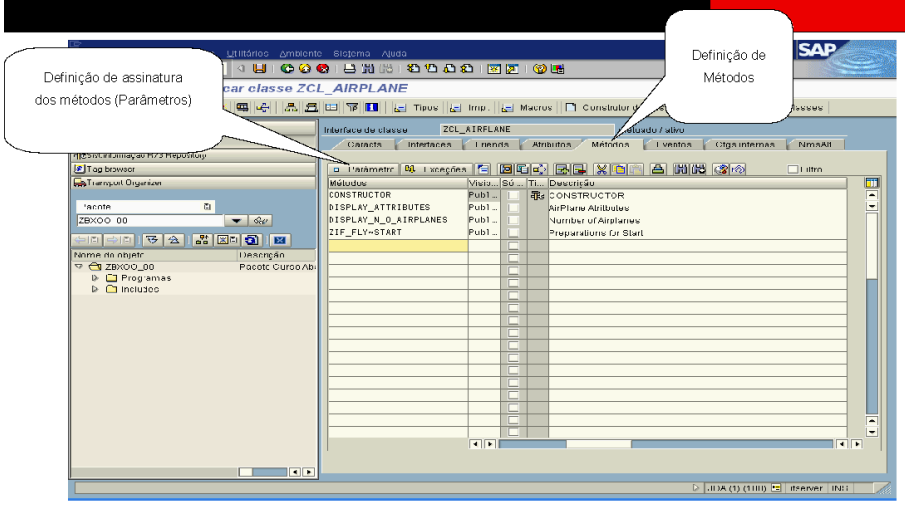



Data: 26/09/06

- No Class Builder, escolhendo a opção TAB Atributo de uma classe é possível visualizar os atributos da classe.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

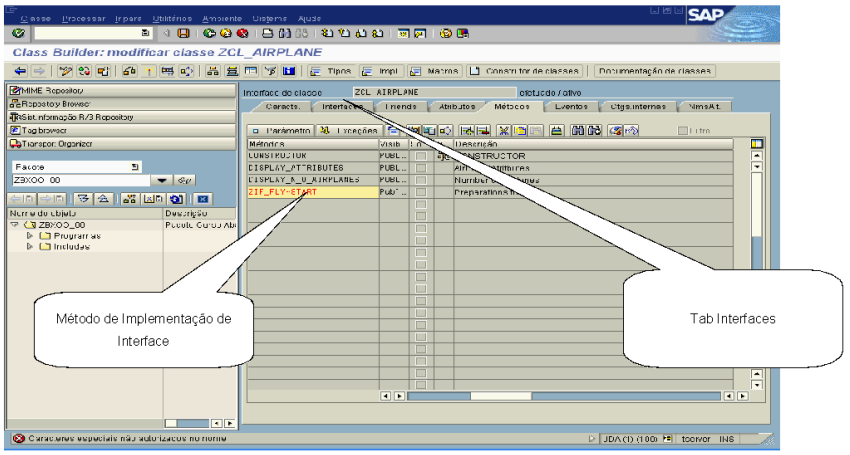
# Class Builder: Métodos




Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

# Class Builder: Métodos de Interfaces

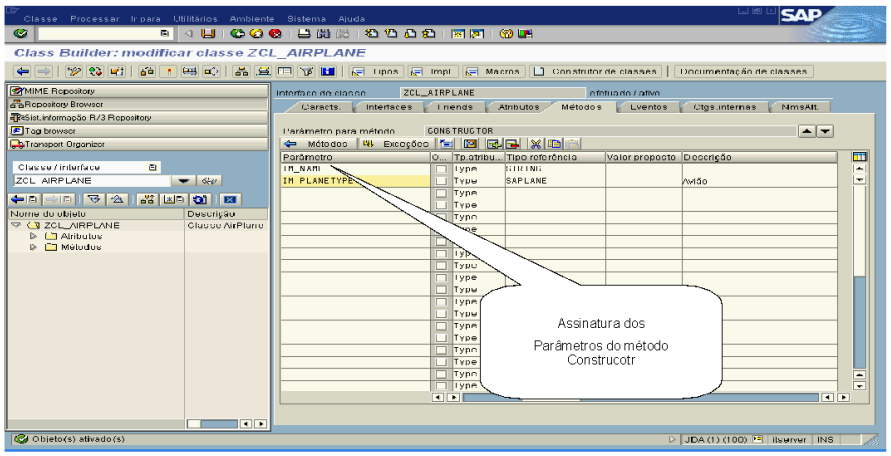



Data: 26/09/06

- Interfaces Globais que são criadas no Class Builder podem ser anunciadas em servidor de classes pela escolha da Interfaces TAB.
- Todos os métodos de interfaces são automaticamente listados na TAB Métodos.
- Todos os métodos devem ser implementados no servidor de classes. Para este exemplo, a interface ZIF_FLY consiste de simples método start.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Class Builder: Parâmetros de Métodos

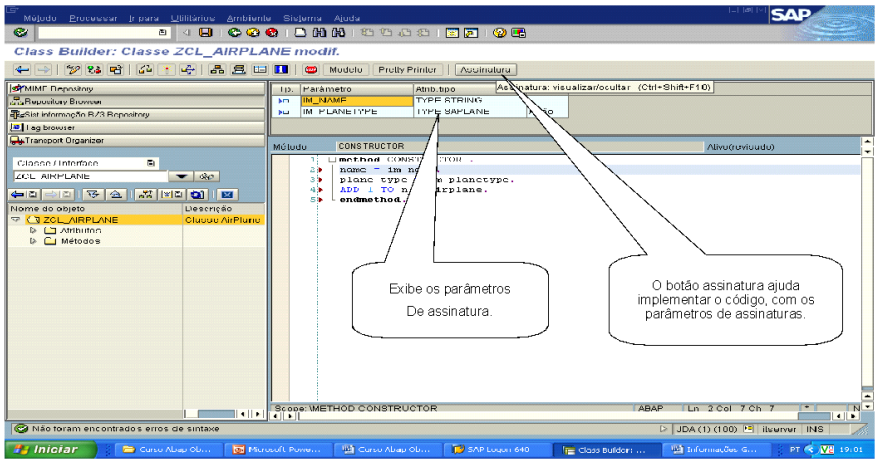



Data: 26/09/06

- Selecionado a opção Parâmetros no Método, serão exibidos os parâmetros de assinatura do método.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

# Class Builder: Implementação de Métodos

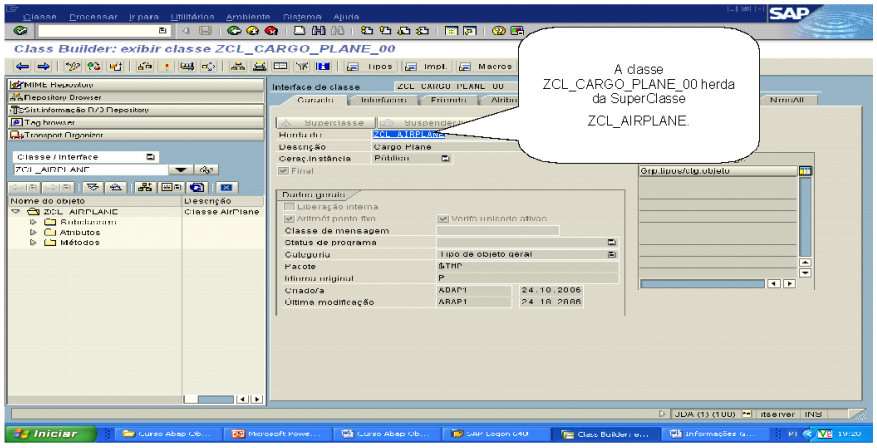



Data: 26/09/06

- Nas implementações dos métodos é possível utilizar o botão assinatura para exibir os parâmetros que devem ser importados para ajudar no desenvolvimento do código. Para este caso, os parâmetros im_name e im_planetype.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Class Builder: Herança

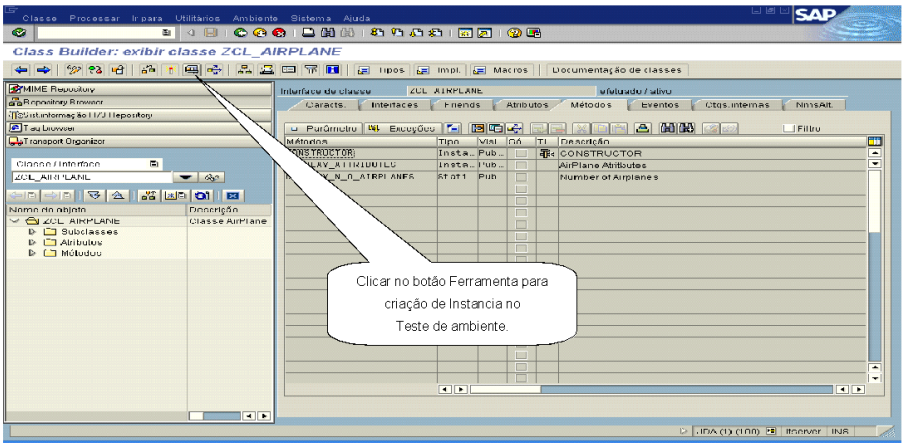



Data: 26/09/06

- È possível visualizar os relacionamentos entre as superclasses e subclasses através da TAB Características.
- No exemplo acima, a classe ZCL_CARGO_PLANE recebe a herança da classe ZCL_AIRPLANE.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

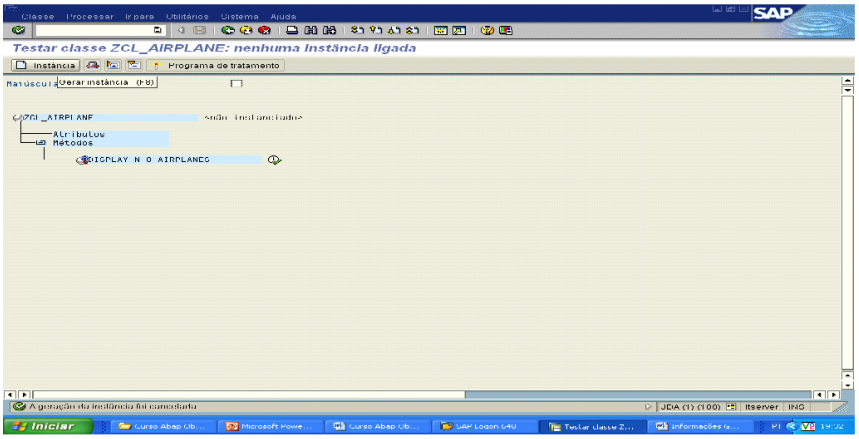
# Class Builder: Teste de Ambiente




Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Class Builder: Teste de Ambiente

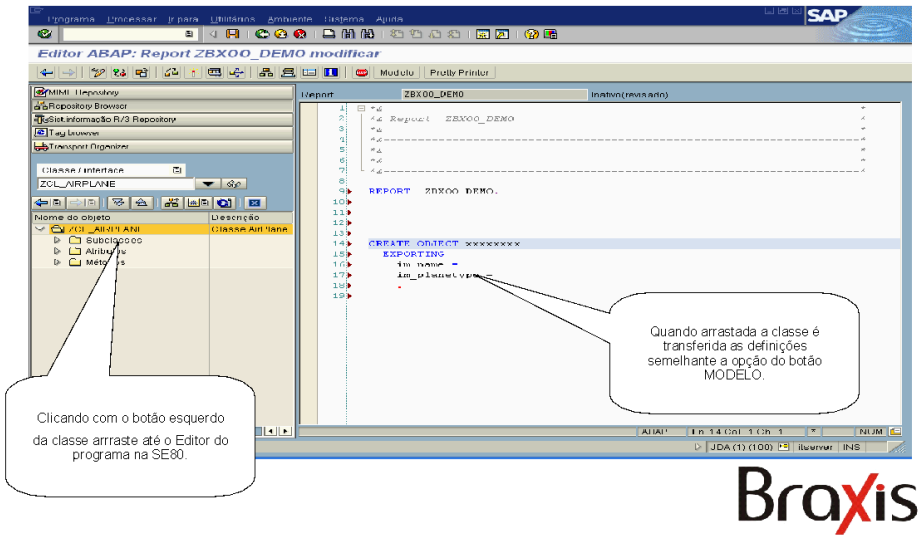



Data: 26/09/06

- Utilizando o a ferramenta de teste é possível criar instancias da classe, preencher os atributos e executar os métodos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Classes Globais em Object Navigator

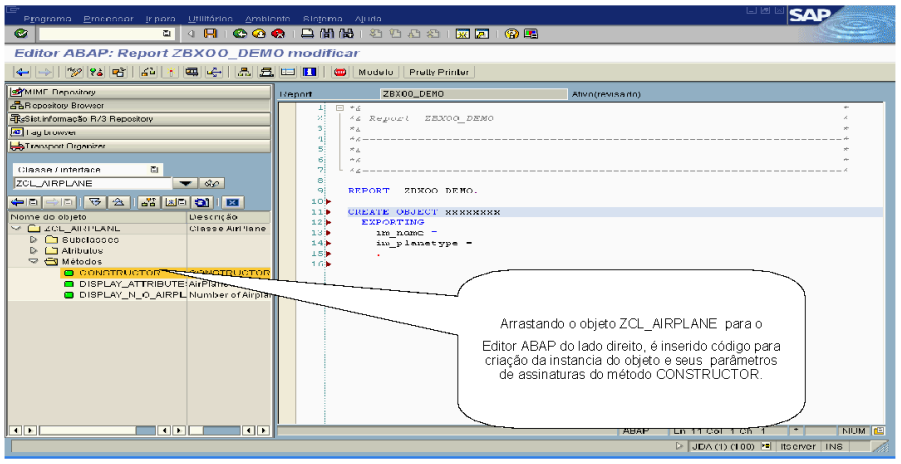



Data: 26/09/06

- Um recurso interessante do Object Navigator (SE80) é a opção e de arrastar objetos, como classes, métodos no editor ABAP e visualizando as classes e métodos, conforme a o slide acima.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Criando Objetos com Objetc Navigator

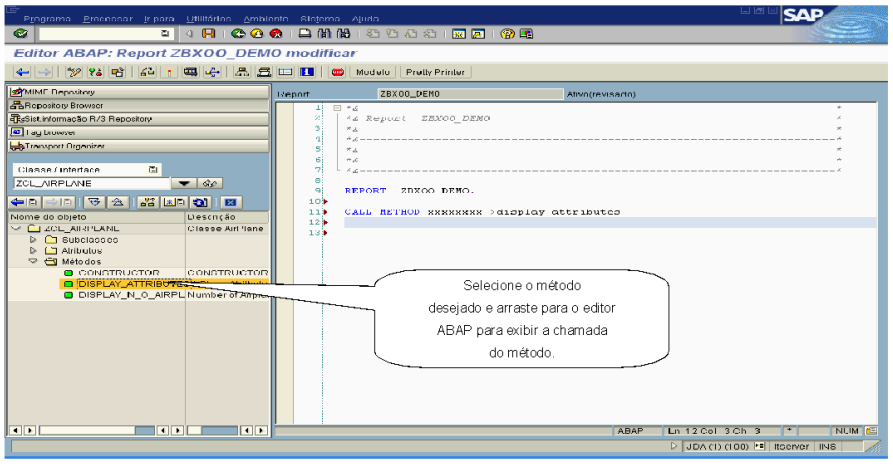



Data: 26/09/06

- O slide acima mostra como é fácil a utilização, por exemplo para criação do comando OBJECT CREATE, através do drag & drop com mouse.No exemplo foi selecionado o objeto(classe) ZCL_AIRPLANE e quando arrastado foi criado um template para instancia.
- OBS: Os caracteres X são onde devem ser preenchidos o nome da instancia.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Chamando Métodos com Objetc Navigator




Data: 26/09/06

- Da mesma forma que é possível a criação de objetos(instancias), também é possível arrastar os métodos. No exemplo acima, foi selecionado o método DISPLAY_ATTRIBUTES do lado esquerdo da tela e quando arrastado para o lado direito, foi inserido o comando CALL METHOD do método selecionado.
- OBS: Para o caso de drag & drop para métodos os caracteres X é a instancia criada com o comando OBJECT CREATE.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Classes Globais**

1.1 Utilizando o Class Bulder SE24 ou SE80 criar uma classe global chamada ZCL_HOTEL_XX.

1.2 Criar os seguintes atributos:

| | | |
|---|---|---|
| - NAME | TYPE STRING | Atributo PRIVADO e Instancia |
| - MAX_BEDS | TYPE I | Atributo PRIVADO e Instancia |
| - N_O_HOTEL | TYPE I | Atributo PRIVADO e Estático |

1.3 Criar os métodos

| | |
|---|---|
| - CONSTRUCTOR | Importando os parâmetros IM_NAME e IM_BEDS e atribuir os parâmetros aos seus atributos de classe privados NAME e MAX_BEDS. |
| – DISPLAY_ATTRIBUTES | Exibir os dois atributos (NAME e MAX_BEDS) com o comando WRITE. |
| - DISPLAY_N_O_HOTELS | Exibir o atributos N_O_HOTEL. |

1.4 Teste a classe na Ferramenta de Teste na SE24 ou SE80.

1.5 Utilizando o programa do exercício anterior ZBXOO_EVE_MAIN_00 implementar o seguinte código:
- Declarar um objeto r_hotel do tipo ZCL_HOTEL_XX
- Criar o objeto importando os parâmetros do constructor (IM_NAME e IM_BEDS), por exemplo: name = HILTOM e max_beds como 320.
- Chamar o método display_attributes para exibir os atributos

**RESPOSTAS EXERCÍCIOS:**

**1. Classes Globais**

1.1 A resposta para este exercício está na classe ZCL_HOTEL_00.

1.2 esposta para o programa principal está abaixo e as alterações em negrito.

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_CLSS_MAIN_00                                          *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_CLSS_MAIN_00                .


TYPE-POOLS icon.

INCLUDE ZBXOO_INT_00.
INCLUDE ZBXOO_EVE_00.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane,
      r_carrier  type ref to lcl_carrier,
      r_agency TYPE REF TO lcl_travel_agency,
      r_rental TYPE REF TO lcl_rental,
      r_truck TYPE REF TO lcl_truck,
      r_bus TYPE REF TO lcl_bus,
      r_hotel TYPE REF TO ZCL_HOTEL_00.


DATA: count TYPE i.

START-OF-SELECTION.

CREATE OBJECT r_carrier EXPORTING im_name = 'Smile$Fly-Travel'.

CREATE OBJECT r_agency EXPORTING im_name = 'CVC - Turismo'.

*r_agency->add_partner( r_carrier ).


CREATE OBJECT r_passenger EXPORTING
        im_name = 'LH Berlin'
        im_planetype = '747-400'
        im_seats = 345.

CREATE OBJECT r_cargo EXPORTING
        im_name = 'AA New York'
          im_planetype = '747-300'
          im_maxcargo = 533.

*r_carrier->add_airplane( r_passenger ).
*
*r_carrier->add_airplane( r_cargo ).

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).


CREATE OBJECT r_agency EXPORTING im_name = 'CVC'.


CREATE OBJECT r_rental EXPORTING im_name = 'RENT A CAR'.


CREATE OBJECT r_truck  EXPORTING im_name = 'MACK'
                          im_cargo = '458'.

r_rental->add_vehicle( r_truck ).


CREATE OBJECT r_bus  EXPORTING   im_name = 'MERCEDES'
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
                    im_passengers = '80'.

r_rental->add_vehicle( r_bus ).


CREATE OBJECT r_truck  EXPORTING im_name = 'VOLVO'
                    im_cargo = '48'.


CREATE OBJECT r_hotel EXPORTING
                    im_name = 'HILTOM HOTEL'
                    im_beds = 320.
      .


r_rental->add_vehicle( r_truck ).


r_agency->add_partner( r_rental ).


r_carrier->display_atributes( ).

r_agency->display_agency_partners( ).


r_hotel->display_attributes( ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Técnicas Especiais (Classes Abstratas)

- **Classes Abstratas**
  - **Classes Abstratas não podem ser instanciadas, mas podem ser instanciadas pelas Subclasses.**
  - **Classes Abstratas são semelhantes as Interfaces**
  - **Métodos de Instancias de classes abstratas são definidos na classes, mas não podem ser implementados.**
    - **Os métodos devem ser redefinidos**




Braxis

Data: 26/09/06

- As classes abstratas não podem ser instanciadas, mas não significa que não tem uso. No entanto, classes abstratas podem ter grande utilização em subclasses.
- Classes Abstratas geralmente são usadas para declarações incompletas de subclasses, ou seja, non-abstract subclasses.
- Os métodos de classes abstratas não podem ser implementados na própria classe, somente podem ser redefinidos nas subclasses.
- Classes com pelo menos um método abstrato são abstratas
- Métodos Estáticos e Construtores não podem ser abstratos e também não podem ser redefinidos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Classe Final

- **Classe Final não podem ter subclasses**

```
CLASS lcl_truck   DEFINITION FINAL INHERITING FROM lcl_vehicle.
    ...

ENDCLASS.
```

- **Métodos finais não podem ser redefinidos**

```
CLASS lcl_bus   DEFINITION FINAL INHERITING FROM lcl_vehicle.
    METHODS estiamte_fuel FINAL.

ENDCLASS.
```

Data: 26/09/06

- Uma classe final não pode ter subclasses, e dessa forma ela pode proteger ela mesma,(o descontrole) especialização.
- Algumas características:
  - Uma classe final implicitamente somente contém métodos finais. Neste caso, você não pode definir um método, como um método final.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Friends

- **Classes Friends**
  - **Classes Friends é um conceito onde outras classes podem ver componentes, incluindo componentes (Private e Protected)**
    - **Acesso direto para os dados da classe provendo, relacionamento entre estas classes**
    - **Métodos pode ser visíveis em diversas classes.**

Data: 26/09/06

- O conceito de classes Friends permite acesso de componentes PRIVATE e PROTECTED, permitindo que todas as classes que sejam Friends possam visualizar estes componentes.
- Este conceito é somente de um lado. A classe provedora não é automaticamente uma Friend das Friends de si mesma. Se uma classe prove relacionamento Friend e é necessário acessar componentes que não são públicos de uma Friend, esta Friend tem que explicitamente prover relacionamento para ela.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Serviços Persistentes

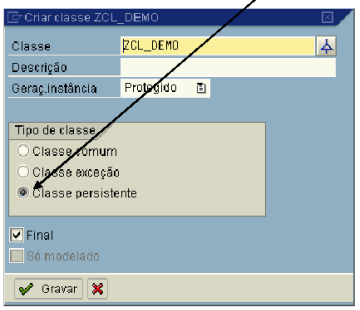
- **Serviços Persistentes**
  - Os serviços persistentes geram as informações de Orientação a Objeto em banco de dados.
  - Os objetos em runtime são transferidos quando o programa é parado, então os dados são gravados.

Data: 26/09/06


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Características de Serviços Persistentes

- **Serviços Persistentes**
  - **Objetos em runtime programa são transferidos**
  - **Os serviços persistentes permite você trabalhar com objetos persistentes**
  - **Classes persistentes são criadas no Class Builder.**




Data: 26/09/06

- Classes Persistentes somente podem ser criadas no Class Builder.
- No momento da criação de uma classe persistente, o Classe Builder cria uma classe correspondente, chamada de class actor ou agent, os métodos de quais serão usados para gerenciar os objetos de uma classe persistente.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Classe Agent

- **Classe Agent**
  - **Com os serviços persistentes, a classe actor ou agent gerencia os objetos persistentes.**
  - **Este agente atribui um range de serviços com métodos, para gerenciar os objetos e dados encapsulados.**

```
DATA: o_carrier   TYPE REF TO cl_carrier,
      o_agent     TYPE REF TO ca_carrier,
      carrname    TYPE        s_carrname.

r_agent = ca_carrier=>agent.
```

Data: 26/09/06

- Para cara classes persistente criada no Class Builder, ele gera duas adicionais classes ca_persistent e cb_persistent.
- A classe ca_persistent é a classe actor ou agent, no qual gerencia os objetos de uma classe cl_persistent, no qual todos acessos de database tomam seus lugares. A classe actor herda os métodos da classe abstrata cb_persistent. Ainda é possível redefinir estes métodos para modificar acessos as bases de dados. A superclasse cb_persistent não pode ser modificada. A classe actor é uma Friend da classe gerenciada. Ela tem um atributo CREATE PRIVATE e é exatamente uma instancia de uma classe actor é criada quando acessada.
- O atributo AGENT é uma variável de referencia com o tipo de classe ca_persistent. Quando o atributo é acessado pela primeira vez no programa, o construtor estático de uma ca_persistent cria exatamente uma instancia de uma classe, com um ponteiro para o atributo AGENT. Este objeto é parte de um serviço persistente e este método é usado para gerenciar os objetos da classe persistente. Para cada programa existe somente um objeto da classe ca_persistent, por que você não pode criar o objetos sem usar o CREATE OBJECT.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Manipulado Exceções

- Conteúdo
  - Definição de Exceções (Exception Handling)
  - Raising e Handling com classes predefinidas.

Data: 26/09/06

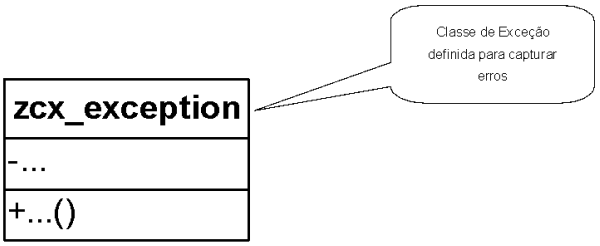

*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Exceptions (Execções)

```
REPORT ZDEMO.

....

RAISE EXCEPTION TYPE cx_exception
                EXPORTING erro1 = ...
                          erro2 = ...
```



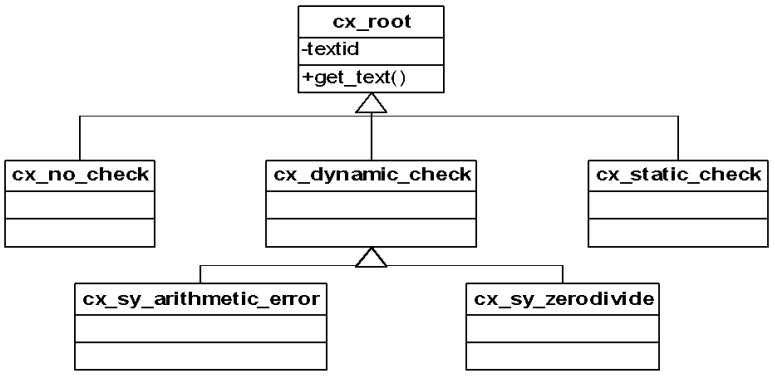

Braxis

Data: 26/09/06

- Manipular exceções é um procedimento no código de linguagens para capturar erros não previstos, ou possíveis erros que possam ser pré-definidos, exemplos divisão por zero, overflow. O conceito de classes de exceções está disponível desde a versão Sap Basis 6.10. Sendo possível visualizar estas classes com extensão CX, exemplo CX_ROOT que é a superclasse de exceção.
- O comando RAISE EXCEPTION conforme o slide acima, captura um erro através da classe de exemplo ZCX_EXCEPTION. Conforme o exemplo, é possível desenvolver classes Z de exceptions, ou utilizar as classes Standard de exceções, CX_ROOT, CX_SY_ZERODIVIDE.
- Com a Orientação a Objetos manipular exceções é de extrema importância para tratamento de erros. Pois se ocorre um erro no programa em tempo de execução, este consegue ser capturado o erro e pode ser tratado.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Exceptions (Exceções)

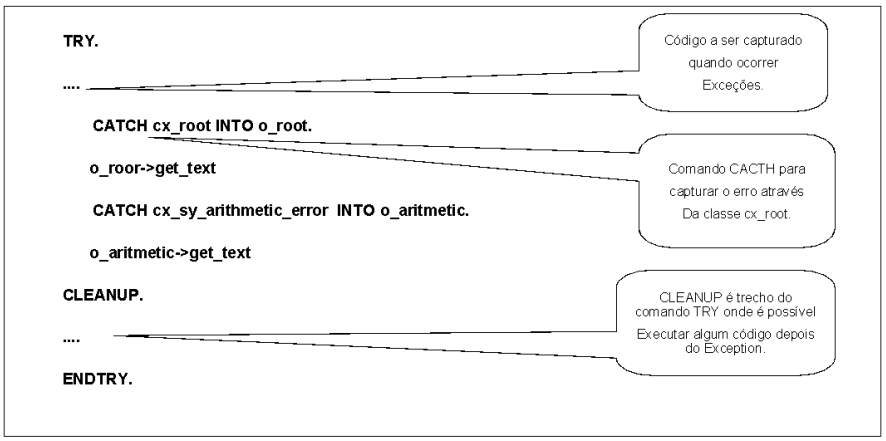



Data: 26/09/06

- As classes de exceções derivam da superclasse CX_ROOT. O slide acima mostra a hierarquia das classes de exceções e suas principais subclasses.
- As classes de exceções sempre começam com CX_ para classes globais, mas também é possível definir classes locais.
- No exemplo do slide acima a classe CX_ROOT tem dois métodos que são herdados para as subclasses. O método GET_SOURCE_POSITION retorna o nome do programa e o método GET_TEXT retorna um texto de exceção de uma classe em forma de string.
- Todas as classes de exceções possuem atributo KERNEL_ERRID que contém o nome do erro em runtime. Isto somente ocorre se houve uma captura de alguns para a classe instanciada.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## TRY – CATCH - Exceptions




Data: 26/09/06

- O comando TRY é importante comando para capturar exceções em linguagens de Orientações a Objetos, como Delphi, JAVA e .NET. Ele somente está disponível a partir da versão SAP BASIS 6.10 e a versão SAP APP 4.7.
- Ele é executado em blocos conforme o slide acima. E se uma exception ocorre o bloco TRY procura por uma declaração CATCH, onde a poderá manipular uma exceção. Se ele não encontrar um exceção de manipulação no TRY-ENDTRY, ele então passa a exceção para programa.
- O bloco CATCH contém exceções que são executadas se uma específica exceção ocorreu no bloco TRY. Depois do bloco é possível especificar outras classes de exceções. Depois que a exceção ocorre, o sistema procura todas as exceções declaradas no CATCH e a executa.
- Alguns casos, a exceção não é encontrada e então a exceção é passada para o programa e block CLEANUP é executado. O CLEANUP é opcional, mas ele tem uma importante função no controle de fluxo de manipulações, pois neste trecho de bloco é possível, voltar um estado de algum objeto, no caso de conexões com Banco de Dados ele pode fechar a conexão. O bloco CLEANUP tem a mesma função do FINALY no JAVA.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## TRY – CATCH - Exemplo

```
REPORT ZBX00.

DATA:           resultado   TYPE I.
                texto       TYPE STRING,
                o_root      TYPE CX_ROOT.

TRY.
    resultado = var1 * var2.
    CATCH cx_sy_arihtmetic_overflow  INTO o_root.
   texto = o_roor->get_text
    WRITE texto.

CLEANUP.

    ....

ENDTRY.
```

Se ocorrer um overflow na variável resultado, será capturada a exceção através da classe CX_SY_ARITMETIC_ERROR e não ocorrerá o DUMP.

Data: 26/09/06

- No slide acima, se ocorrer um overflow na variável resultado do tipo “I” , o runtime captura uma exceção CX_SY_ARITHMETIC_OVERFLOW no bloco CATCH.
- O objeto o_root recebe a instancia da classe de exceção, e então é possível exibir o conteúdo do erro através do método GET_TEXT que é herdado da classe CX_ROOT.
- A classe CX_SY_ARITHMETIC_OVERFLOW, CX_SY_ARITHMETIC_ERROR e CX_DYNAMIC_CHECK são subclasses da superclasse CX_ROOT.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Criando suas próprias Exceptions

```
CLASS lcl_airplane DEFINITION.
...
    METHODS get_techical_attrinutes
                IMPORTING          im_type      TYPE saplane-planetype
                EXEPORTING         ex_weight    TYPE s_plan_wei
                                   ex_tankcap   TYPE s_capacity.
...
ENDCLASS.

CLASS lcl_airplane IMPLEMENTION.
...
    METHODS get_techical_attrinutes
    SELECT SINGLE weight tankcap FROM saplane
            INTO ( ex_weight, ex_tankcap)
          WHERE planetype = im_type.
    IF sy_subrc <> 0.
       ex_weigth = 100000.
       ex_tankcap = 10000.
    ENDIF.
...
ENDCLASS.
```

Data: 26/09/06

- O código acima mostra o método get_technical_attributes de uma classe lcl_airplane, com a sua implementação. Ele importa o parâmetro im_planetype e retorna weigth e tank.
- Analisando a implementação do método acima, retorna os parâmetros weight = 1000000 e tankcap = 10000, caso não seja encontrado na tabela. Isto pode ser melhorado, implementando uma exceção para tratar, caso o SY_SUBRC seja diferente de 0.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Criando suas próprias Exceptions

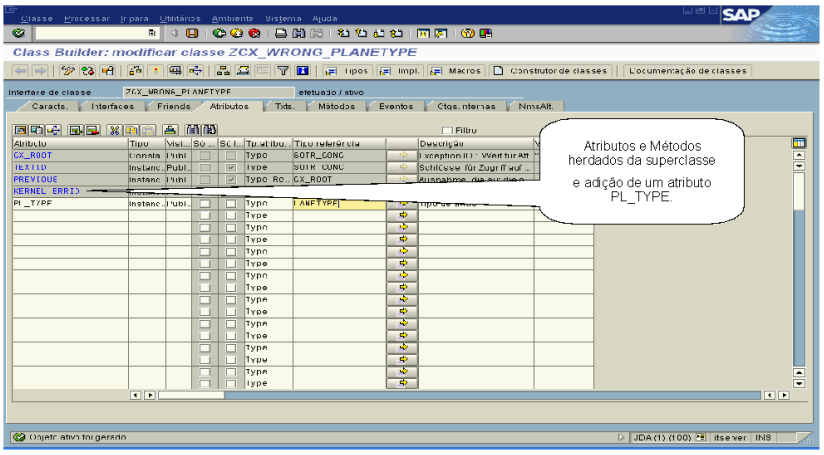



Data: 26/09/06

- È possível criar classes globais de exceções na SE24, conforme o slide acima. Geralmente as classes de exceções são geralmente classes globais e existe uma convenção de nomenclatura para este tipo de classes, onde para as classes iniciam com CX_ e para classes não standards, devem começar com ZCX_.
- Quando você cria uma classe de exceção ela, herda da classe CX_STATIC_CHECK como default.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Criando suas próprias Exceptions




Data: 26/09/06

- Todos os componentes (Atributos e Métodos) da classe CX_ROOT são herdados, então você pode adicionar novos componentes. Um construtor de instancia é criado automaticamente.
- Também podem ser definidos textos para as exceções na TAB TEXT como um ID de exceção. O método GET_TEXT exporta este TEXT ID.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Capturando próprias Exceptions

```
CLASS lcl_airplane DEFINITION.
    METHODS get_techical_attrinutes
                IMPORTING          im_type      TYPE saplane-planetype
                EXEPORTING         ex_weight    TYPE s_plan_wei
                                   ex_tankcap   TYPE s_capacity. .
ENDCLASS.

CLASS lcl_airplane IMPLEMENTION. ...
    METHODS get_techical_attrinutes
    SELECT SINGLE weight tankcap FROM saplane
            INTO ( ex_weight, ex_tankcap)
          WHERE planetype = im_type.
    IF sy_subrc <> 0.
      TRY.

      RAISE EXCEPTION TYPE zcx_wrong_planetype EXPORTING
                                          pl_type = im_type.
      CATCH zcx_wrong_planetype.
    texto = o_root->get_text( ).
        WRITE texto.
      ENDTRY.
    ENDIF.
...
ENDCLASS.
```

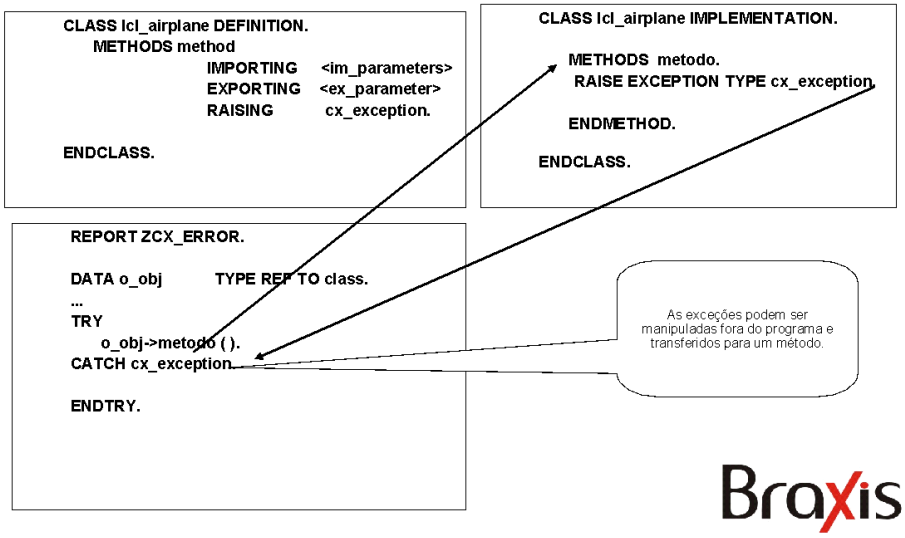
Braxis

Data: 26/09/06

- Com a clausula TRY e RAISE no slide acima, se tipo de airplane for passado de forma incorreta, através da classe de  exceção desenvolvida, caso o SY-SUBRC seja diferente de 0.
- Dentro do bloco TRY existe a clausula RAISE importando o parâmetro pl_type do constructor e assim automaticamente gera a exceção. Então possível capturar a exceção e capturar TEXTID.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Passando Exceptions

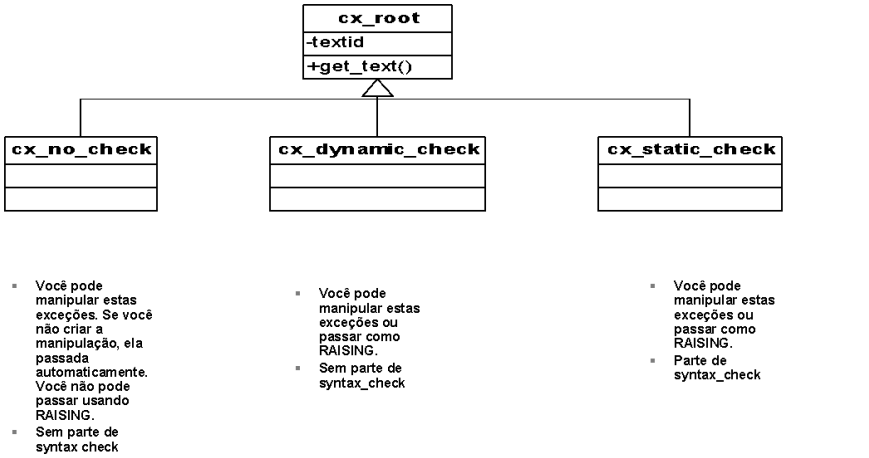



Data: 26/09/06

- As exceções que ocorrem em procedures( métodos, funções, reports e subrotinas) podem ser transferidas para outro programa, ou include, conforme exemplo acima.
- Para passar exceções de uma procedure, você geralmente usa o RAISING.
- Se for uma classe local, o RAISING deve ser especificado no método com a sintaxe na definição da classe acima.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## Passando Exceptions




Data: 26/09/06

- CX_STATIC_CHECK. As exceções dessas subclasses podem ser manipuladas, ou passadas explicitamente com RAISING. Então, somente as exceções que você mesmo define no código de sua aplicação são Subclasses de CX_STATIC_CHECK.
- CX_DYNAMIC_CHECK. As relevantes exceções não têm que ser declaradas. Se uma exceção ocorre em runtime, somente como subclasses de CX_STATIC_CHECK, ela deve ser manipulada ou passada explicitamente usando RAISING. Entretanto, isto não é verificado no syntax check.
- CX_NO_CHECK. Estas exceções não podem ser declaradas. Estas exceções podem ser manipuladas. Se não, elas podem ser passadas para o outro programa. A checagem de sintaxe nunca encontra um erro na manipulação. Todas as exceções da categoria CX_NO_CHECK que não são manipuladas na chamada da hierarquia são automaticamente passadas para o nível mais auto, se eles não capturados lá, eles causam runtime error. Algumas predefinidas exceções com o prefixo CX_SY para situações de erros em ambiente  runtime são subclasses de CX_NO_CHECK.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**EXERCÍCIOS:**

**1. Exceções**

1.1 Utilizando o include ZBXOO_EVE_XX crie um novo chamado ZBXOO_EXCS_XX .

1.2 Após ser criado implemente o método get_technical_attributes utilizando o comando TRY, com os seguintes componentes:

Na classe lcl_airplane
- Crie um atributo privado weight type saplane-weight
- Crie um atributo privado tankcap type saplane-tankcap.
- Crie um método privado chamado get_technical_atributes com a definição e os seguintes parâmetros.:

```
            IMPORTING
   im_planetype type saplane-planetype
EXPORTING
   ex_weight   type saplane-weight
   ex_tankcap  type saplane-tankcap
RAISING
   ZCX_WRONG_PLANETYPE.
```

- Implemente o método com seguinte código abaixo:

```
 METHOD get_technical_atributes.

            SELECT SINGLE weight tankcap
 FROM saplane
INTO (ex_weight, ex_tankcap)
WHERE planetype = im_planetype.
IF SY-SUBRC NE 0.
RAISE EXCEPTION TYPE ZCX_WRONG_PLANETYPE
      EXPORTING planetype = im_planetype.

ENDIF.
  ENDMETHOD.
```

No método display_atributes.
- Declare um objeto o_planetype TYPE REF TO ZCX_WRONG_PLANETYPE
- Declare uma variável text TYPE STRING.

- Insira o comando TRY e dentro do bloco TRY e CATCH inclua a chamada do método get_technical_atributes EXPORTING im_planetype = planetype
IMPORTING ex_weight = weight
ex_tankcap = tankcap ).

- Na clausula CATCH declare a classe ZCX_WRONG_PLANETYPE e inseria a exceção no objeto o_planetype.

- Atribua o método get_text do objeto a variável text e liste com um WRITE: / text COLOR COL_NEGATIVE.

1.3 No programa ZBXOO_CLSS_MAIN_00 utilizado anteriormente comente o include *INCLUDE ZBXOO_EVE_00 e atribua o novo include ZBXOO_EXCS_XX com as alterações acima.

1.4 Execute o programa principal ZBXOO_CLSS_MAIN_XX a exceção será capturada para o ariplane 747-300 que existe na tabela.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**RESPOSTAS EXERCÍCIOS:**

## 1. Exceções

1.1 Resposta do include ZBX00_EXCS_00 com as alterações de exceções:

```
*&---------------------------------------------------------------------*
*&  Include          ZBXOO_EVE_00                                      *
*&---------------------------------------------------------------------*

*----------------------------------------------------------------------*
*   CLASS lcl_airplane DEFINITION.                                     *
*----------------------------------------------------------------------*

CLASS lcl_airplane DEFINITION.

  PUBLIC SECTION.

    CONSTANTS: pos_1 TYPE i VALUE 30.

    METHODS: constructor IMPORTING
               im_name       TYPE string
               im_planetype  TYPE saplane-planetype,

             display_atributes.

     CLASS-METHODS: display_n_o_airplanes.

     EVENTS airplane_created.

  PRIVATE SECTION.

  DATA:  name       type     string,
         planetype   type     saplane-planetype,
         weight     type     saplane-weight,
         tankcap    type     saplane-tankcap.


  CLASS-DATA: n_o_airplanes  type i.

   METHODS: get_technical_atributes
         IMPORTING
              im_planetype type saplane-planetype
         EXPORTING
              ex_weight   type saplane-weight
              ex_tankcap  type saplane-tankcap
         RAISING
              ZCX_WRONG_PLANETYPE.


ENDCLASS.

*----------------------------------------------------------------------*
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*   CLASS lcl_airplane IMPLEMENTATION.                          *
*---------------------------------------------------------------------*

CLASS lcl_airplane  IMPLEMENTATION.


  METHOD constructor.
    name       = im_name.
    planetype  = im_planetype.
    n_o_airplanes = n_o_airplanes + 1.
    raise event airplane_created.
  ENDMETHOD.


  METHOD display_atributes.

    WRITE: / icon_ws_plane as icon,
           / 'Name of airplane: '(001), AT pos_1 name,
           / 'Airplane type '(002), AT pos_1 planetype.


    DATA: o_planetype   TYPE REF TO ZCX_WRONG_PLANETYPE,
          text          TYPE   STRING.

    TRY.

      get_technical_atributes( EXPORTING im_planetype = planetype
                               IMPORTING ex_weight = weight
                                         ex_tankcap = tankcap ).

    CATCH ZCX_WRONG_PLANETYPE INTO o_planetype.
      text = o_planetype->get_text( ).
      WRITE: / text COLOR COL_NEGATIVE.

    ENDTRY.

  ENDMETHOD.


  METHOD display_n_o_airplanes.
    WRITE: /,/ 'Total nember of planes'(ca1),
           AT pos_1 n_o_airplanes LEFT-JUSTIFIED,/.


  ENDMETHOD.


  METHOD get_technical_atributes.

    SELECT SINGLE weight tankcap
      FROM saplane
      INTO (ex_weight, ex_tankcap)
     WHERE planetype = im_planetype.
     IF SY-SUBRC NE 0.
     RAISE EXCEPTION TYPE ZCX_WRONG_PLANETYPE
                EXPORTING planetype = im_planetype.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
    ENDIF.
  ENDMETHOD.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane DEFINITION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane DEFINITION INHERITING FROM lcl_airplane.

  PUBLIC SECTION.

    METHODS: constructor IMPORTING
          im_name       TYPE string
          im_planetype  TYPE saplane-planetype
          im_maxcargo   TYPE scplane-cargomax,

          display_atributes REDEFINITION.

  PRIVATE SECTION.

    DATA: max_cargo TYPE scplane-cargomax.

ENDCLASS.



*---------------------------------------------------------------------*
*   CLASS lcl_cargo_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_cargo_plane IMPLEMENTATION.

    METHOD: constructor.

     CALL METHOD super->constructor( im_name = im_name
              im_planetype  = im_planetype ).
              max_cargo = im_maxcargo.

    ENDMETHOD.

    METHOD: display_atributes.

      CALL METHOD super->display_atributes( ).

      WRITE: / 'Max Cargo = ', max_cargo.

    ENDMETHOD.



ENDCLASS.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane DEFINITION.
*---------------------------------------------------------------------*
CLASS lcl_passenger_plane DEFINITION INHERITING FROM lcl_airplane.

    PUBLIC SECTION.

      METHODS: constructor IMPORTING
                im_name       TYPE string
                im_planetype  TYPE saplane-planetype
                im_seats      TYPE sflight-seatsmax,

               display_atributes REDEFINITION.


    PRIVATE SECTION.

      DATA: max_seats TYPE sflight-seatsmax.


ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_passenger_plane IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_passenger_plane IMPLEMENTATION.

  METHOD constructor.

     CALL METHOD super->constructor( im_name = im_name
                im_planetype  = im_planetype ).
                max_seats = im_seats.

  ENDMETHOD.

  METHOD display_atributes.

    CALL METHOD super->display_atributes( ).

     WRITE: / 'Max Seats = ', max_seats.

  ENDMETHOD.


ENDCLASS.


*---------------------------------------------------------------------*
*   CLASS lcl_carrier DEFINITION.
*---------------------------------------------------------------------*
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
CLASS lcl_carrier DEFINITION .

   PUBLIC SECTION.

     METHODS: constructor IMPORTING
              im_name       TYPE string,
              get_name      RETURNING value(ex_name) TYPE string,
              add_airplane  FOR EVENT airplane_created of lcl_airplane
                        IMPORTING sender,
              display_airplanes,
              display_atributes.

   PRIVATE SECTION.

     DATA: name TYPE string,
          airplane_list TYPE TABLE OF REF TO lcl_airplane.

ENDCLASS.

*---------------------------------------------------------------------*
*   CLASS lcl_carrier IMPLEMENTATION.
*---------------------------------------------------------------------*

CLASS lcl_carrier IMPLEMENTATION.

  METHOD constructor.
    name = im_name.
    SET HANDLER add_airplane for all instances.
  ENDMETHOD.

  METHOD get_name.
    ex_name = name.
  ENDMETHOD.

  METHOD add_airplane.
   APPEND sender TO airplane_list.
  ENDMETHOD.


  METHOD display_atributes.
   display_airplanes( ).
  ENDMETHOD.


  METHOD display_airplanes.
   DATA: r_plane TYPE REF TO lcl_airplane.
     LOOP AT airplane_list INTO r_plane.
       r_plane->display_atributes( ).
     ENDLOOP.
  ENDMETHOD.

ENDCLASS.
```

1.2 Reposta do programa principal com as alterações de exceções:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*&---------------------------------------------------------------------*
*& Report  ZBXOO_EXSS_00                                               *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

REPORT  ZBXOO_EXCS_MAIN_00              .


TYPE-POOLS icon.

INCLUDE ZBXOO_INT_00.
*INCLUDE ZBXOO_EVE_00.
INCLUDE ZBXOO_EXCS_00.

DATA: r_plane TYPE REF TO lcl_airplane,
      plane_list TYPE TABLE OF REF TO lcl_airplane,
      r_cargo type ref to lcl_cargo_plane,
      r_passenger type ref to lcl_passenger_plane,
      r_carrier  type ref to lcl_carrier,
      r_agency TYPE REF TO lcl_travel_agency,
      r_rental TYPE REF TO lcl_rental,
      r_truck TYPE REF TO lcl_truck,
      r_bus TYPE REF TO lcl_bus,
      r_hotel TYPE REF TO ZCL_HOTEL_00.



DATA: count TYPE i.

START-OF-SELECTION.

CREATE OBJECT r_carrier EXPORTING im_name = 'Smile$Fly-Travel'.

CREATE OBJECT r_agency EXPORTING im_name = 'CVC - Turismo'.

*r_agency->add_partner( r_carrier ).


CREATE OBJECT r_passenger EXPORTING
        im_name = 'LH Berlin'
        im_planetype = '747-400'
        im_seats = 345.

CREATE OBJECT r_cargo EXPORTING
        im_name = 'AA New York'
          im_planetype = '747-300'
          im_maxcargo = 533.

*r_carrier->add_airplane( r_passenger ).
*
*r_carrier->add_airplane( r_cargo ).

CALL METHOD lcl_airplane=>display_n_o_airplanes( ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
CREATE OBJECT r_agency EXPORTING im_name = 'CVC'.

CREATE OBJECT r_rental EXPORTING im_name = 'RENT A CAR'.

CREATE OBJECT r_truck  EXPORTING im_name = 'MACK'
                         im_cargo = '458'.

r_rental->add_vehicle( r_truck ).

CREATE OBJECT r_bus  EXPORTING   im_name = 'MERCEDES'
                         im_passengers = '80'.

r_rental->add_vehicle( r_bus ).

CREATE OBJECT r_truck  EXPORTING im_name = 'VOLVO'
                         im_cargo = '48'.

CREATE OBJECT r_hotel EXPORTING
                       im_name = 'HILTOM HOTEL'
                       im_beds = 320.
       .

r_rental->add_vehicle( r_truck ).

r_agency->add_partner( r_rental ).

r_carrier->display_atributes( ).

r_agency->display_agency_partners( ).

r_hotel->display_attributes( ).
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 12. EXERCÍCIO EXTRA

**EXERCÍCIOS:**

**1. Criação de Arquivo XML**

1.1 Criar um programa ZBXXX_XML de acordo o grupo.
1.2 Incluir a declaração TYPE-POOLS: ixml para `biblioteca ixml.`
1.3 Declarar as seguintes estruturas para gerar a estrutura XML.

```
l_ixml            TYPE REF TO if_ixml,
l_streamfactory   TYPE REF TO if_ixml_stream_factory,
l_ostream         TYPE REF TO if_ixml_ostream,
l_renderer        TYPE REF TO if_ixml_renderer,
l_document        TYPE REF TO if_ixml_document.
```

1.4 Declarar as seguintes estruturas abaixo para gerar elementos do XML.

```
l_element_flights TYPE REF TO if_ixml_element,
l_element_airline TYPE REF TO if_ixml_element,
l_element_flight  TYPE REF TO if_ixml_element,
l_element_from    TYPE REF TO if_ixml_element,
l_element_to      TYPE REF TO if_ixml_element,
l_element_dummy   TYPE REF TO if_ixml_element,
l_value           TYPE string,
l_value2(50)      TYPE c.
```

1.5 Declarar as seguintes estruturas abaixo para seleção dos dados e armazenamento XML.

```
l_xml_table       TYPE TABLE OF xml_line,
l_xml_size        TYPE i,
l_rc              TYPE i.

lt_spfli          TYPE TABLE OF spfli.
l_spfli           TYPE spfli.
```

1.6 Declarar as variáveis abaixo para utilização de gravação do arquivo.

```
w_filename TYPE string,
w_path     TYPE string,
w_fullpath TYPE string,
w_rc     TYPE i,
w_title  TYPE string.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

1.7 Utilizar o método estático da classe `cl_gui_frontend_services=>file_save_dialog`, para exibir a caixa de diálogo para solicitar a local de gravação de arquivo. Informando os parâmetros:

```
 EXPORTING
       window_title          = 'Gravar arq XML'
       DEFAULT_FILE_NAME     = 'C:\Temp\flights.xml'
       initial_directory     = 'C:\Temp\'
CHANGING
       filename              = w_filename
       path                  = w_path
       fullpath              = w_fullpath
EXCEPTIONS
       cntl_error            = 1
       error_no_gui          = 2
       not_supported_by_gui  = 3
       OTHERS                = 4
```

1.8 Criar um evento START-OF-SELECTION.

1.9 Fazer um select na tabela spfli e gravá-lo em lt_spfli.

1.10 Ordenar a tabela interna lt_spfli por CARRID.

1.11 Criar um LOOP na tabela já preenchida e atribuir a work área a variável l_spfli.

1.12 Carregar a definição da classe CL_IXML com o comando CLASS <classe a ser carregada> DEFINITION.

1.13 No comando “AT FIRST” do LOOP implementar o seguinte código:

- Criar um ixml factory (Referencia de um Objeto) , atribuindo o método estático CREATE() da classe CL_IXML para o objeto l_ixml.
- Criar dom object model XML ( Instancia do Objeto ) , atribuindo o método de instancia ( CREATE_DOCUMENT) da referencia l_ixml para o objeto l_document.
- Com o objeto l_document criado. Cria o root node com os valores flights, utlizando o método create_simple_element, passado como parâmetros ( name = ‘flight_OO’ e parent = l_document ).
- Finalizar ENDAT.

1.14 Implementar o comando “AT NEW carrid” do LOOP o seguinte código:

- Criar elementos 'airline' como child de 'flights', utilizando o objeto l_document com o método create_simple_element, passado como parâmetros ( name = ‘airline’ e parent = l_element_flights ).
- Atribuir o campo l_spfli-carrid a variável l_value.
- Atribuir o método set_attribute do objeto l_element_airline a variável l_rc, passando os parâmetros (name = 'code' value =l_value ). Para criar o atributo name.
- Incluir um SELECT SINGLE da tabela scarr selecionado o campo carrname gravando em l_value2, onde carrid é igual a l_spfli-carrid.
- Atribuir l_value2 a l_value.
- Atribuir o método set_attribute do objeto l_element_airline a variável l_rc, passando os parâmetros (name = 'code' value =l_value ). Para criar o atributo name.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

- Finalizar ENDAT.

1.15 Implementar o comando “AT NEW connid” do LOOP o seguinte código:

- Criar elemento 'flight' como child de 'airline', utilizando o objeto l_document com o método create_simple_element, passado como parâmetros ( name = ‘flight’ e parent = l_element_airline ).
- Atribuir _spfli-connid a l_value.
- Atribuir o método set_attribute do objeto l_element_flight a variável l_rc, passando os parâmetros (name = 'number' value =l_value ). Para criar o atributo name.
- Finalizar ENDAT.

1.16 No corpo do LOOP definir os seguintes métodos:

- Concatenar l_spfli-cityfrom e l_spfli-countryfr e atribuir a l_value.
- Utilizando o objeto l_document com o método create_simple_element atribua ao `l`_element_from, passado como parâmetros ( name = ‘from’, value = l_value e parent = l_element_flight )
- Atribuir l_spfli-airpfrom para l_value.
- Atribuir o método set_attribute do objeto l_element_from a variável l_rc, passando os parâmetros (name = 'airport' value =l_value ). Para criar o atributo name.
- Concatenar l_spfli-cityto e l_spfli-countryto e gravar em l_value.
- Utilizando o objeto l_document com o método create_simple_element atribua ao `l`_element_from, passado como parâmetros ( name = ‘to’, value = l_value e parent = l_element_flight )
- Atribuir `l_spfli-airpto a l_value.`
- Atribuir o método set_attribute do objeto l_element_to a variável l_rc, passando os parâmetros (name = 'airport' value =l_value ). Para criar o atributo name.
- Atribuir `l_spfli-deptime ao l_value.`
- Utilizando o objeto l_document com o método create_simple_element atribua ao `l`_element_from, passado como parâmetros ( name = ‘departure’, value = l_value e parent = l_element_flight )
- Atribuir `l_spfli-arrtime ao l_value.`
- Utilizando o objeto l_document com o método create_simple_element atribua ao `l`_element_from, passado como parâmetros ( name = ‘arrival’, value = l_value e parent = l_element_flight ).
- Finalizar LOOP.

1.17 Implementar o seguinte código para carregar o XML Stream.

- Criar um stream factory, atribuindo o método create_stream_factory do objeto l_ixml para o objeto l_streamfactory.
- Conectar Tabela XML para o stream factory através da declaração abaixo:

```
l_ostream = l_streamfactory->create_ostream_itable( table =
l_xml_table ).
```

- Normalizar o documento XML através da declaração abaixo:

```
l_renderer = l_ixml->create_renderer( ostream  = l_ostream
                                      document = l_document ).
 l_rc = l_renderer->render( ).
```

- Utilizar o método get_num_written_raw do objeto l_ostream e atribuir em l_xml_size, para obter o tamanho do XML.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

1.18 Após todas as etapas acima executadas é possível agora gravar o arquivo XML no local selecionado. Para isto, vamos utilizar a classe `cl_gui_frontend_services` com o método `gui_download`. Implementando os seguintes parâmetros.

```
EXPORTING
      bin_filesize            = l_xml_size
      filename                = w_fullpath
      filetype                = 'BIN'
CHANGING
      data_tab                = l_xml_table
EXCEPTIONS
      file_write_error        = 1
      no_batch                = 2
      gui_refuse_filetransfer = 3
OTHERS                  = 4.
```

1.19 Ativar e executar o programa.

## RESPOSTA DO EXERCÍCIO

### 1. Criação de Arquivo XML

```
*---------------------------------------------------------------------*
*& Report  ZBX00_XML                                                  *
*&                                                                    *
*&---------------------------------------------------------------------*
*&  Exercício para geração de Arquivo XML - Curso ABAP Objects        *
*&  Autor: Fábio Ferri                                                *
*&---------------------------------------------------------------------*

REPORT  zbx00_xml                                                    .
TYPE-POOLS: ixml.

TYPES: BEGIN OF xml_line,
        data(256) TYPE x,
       END OF xml_line.

DATA: l_ixml            TYPE REF TO if_ixml,
      l_streamfactory   TYPE REF TO if_ixml_stream_factory,
      l_ostream         TYPE REF TO if_ixml_ostream,
      l_renderer        TYPE REF TO if_ixml_renderer,
      l_document        TYPE REF TO if_ixml_document.

DATA: l_element_flights TYPE REF TO if_ixml_element,
      l_element_airline TYPE REF TO if_ixml_element,
      l_element_flight  TYPE REF TO if_ixml_element,
      l_element_from    TYPE REF TO if_ixml_element,
      l_element_to      TYPE REF TO if_ixml_element,
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
      l_element_dummy   TYPE REF TO if_ixml_element,
      l_value           TYPE string,
      l_value2(50)      TYPE c.

DATA: l_xml_table       TYPE TABLE OF xml_line,
      l_xml_size        TYPE i,
      l_rc              TYPE i.

DATA: lt_spfli          TYPE TABLE OF spfli.
DATA: l_spfli           TYPE spfli.

DATA: w_filename TYPE string,
      w_path     TYPE string,
      w_fullpath TYPE string,
      w_rc     TYPE i,
      w_title  TYPE string.

* Utilizar o método file_save_dialog para escolher onde salvar
* o arquivo.

CALL METHOD cl_gui_frontend_services=>file_save_dialog
  EXPORTING
    window_title         = 'Gravar arq XML'
*    DEFAULT_EXTENSION    =
*    DEFAULT_FILE_NAME    = 'C:\Temp\flights.xml'
*    FILE_FILTER          =
    initial_directory    = 'C:\Temp\'
*    WITH_ENCODING        =
*    PROMPT_ON_OVERWRITE  = 'X'
  CHANGING
    filename             = w_filename
    path                 = w_path
    fullpath             = w_fullpath
*    USER_ACTION          =
*    FILE_ENCODING        =
  EXCEPTIONS
    cntl_error           = 1
    error_no_gui         = 2
    not_supported_by_gui = 3
    OTHERS               = 4
        .
IF sy-subrc <> 0.
  MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
             WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
ENDIF.



START-OF-SELECTION.
*   Selecionar dados em SPFLI
  SELECT * FROM spfli INTO TABLE lt_spfli.

*   Ordenar dados em SPFLI
  SORT lt_spfli BY carrid.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
*   LOOP para gerar o XML
  LOOP AT lt_spfli INTO l_spfli.

    AT FIRST.

* Carrega classe CL_IXML
      CLASS cl_ixml DEFINITION LOAD.
*       Criar um ixml factory (Referencia de um Objeto)
      l_ixml = cl_ixml=>create( ).
*       Criar dom object model XML ( Instancia do Objeto )
      l_document = l_ixml->create_document( ).
*       Criar o root node com value flights no create_simple_element
      l_element_flights  = l_document->create_simple_element(
                  name = 'flights_OO'
                  parent = l_document ).
    ENDAT.

    AT NEW carrid.
*       Criar elementos 'airline' como child de 'flights'
      l_element_airline  = l_document->create_simple_element(
                  name = 'airline'
                  parent = l_element_flights  ).

*       Criar  atributo 'code' do node 'airline'
      l_value = l_spfli-carrid.
      l_rc = l_element_airline->set_attribute( name = 'code' value =
l_value ).

*       Criar atributo 'name' do node 'airline'
      SELECT SINGLE carrname FROM scarr INTO l_value2 WHERE carrid EQ
l_spfli-carrid.
      l_value = l_value2.
      l_rc = l_element_airline->set_attribute( name = 'name' value =
l_value ).

    ENDAT.

    AT NEW connid.
*       Criar elemento 'flight' como child of 'airline'
      l_element_flight  = l_document->create_simple_element(
                  name = 'flight'
                  parent = l_element_airline  ).

*       Criar atributo 'number' de node 'flight'
      l_value = l_spfli-connid.
      l_rc = l_element_flight->set_attribute( name = 'number' value =
l_value ).

    ENDAT.

*     Criar elemento 'from' como child de 'flight'
    CONCATENATE l_spfli-cityfrom ',' l_spfli-countryfr INTO l_value.
    l_element_from  = l_document->create_simple_element(
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
                name = 'from'
                value = l_value
                parent = l_element_flight  ).

*     Criar atribute 'airport' do node 'from'
    l_value = l_spfli-airpfrom.
    l_rc = l_element_from->set_attribute( name = 'airport' value =
l_value ).

*     Criar elemento 'to' como child de 'flight'
    CONCATENATE l_spfli-cityto ',' l_spfli-countryto INTO l_value.
    l_element_to  = l_document->create_simple_element(
                name = 'to'
                value = l_value
                parent = l_element_flight  ).

*     Criar atributo 'airport' do node 'from'
    l_value = l_spfli-airpto.
    l_rc = l_element_to->set_attribute( name = 'airport' value =
l_value ).

*     Criar elemento 'departure' como child de 'flight'
    l_value = l_spfli-deptime.
    l_element_dummy  = l_document->create_simple_element(
                name = 'departure'
                value = l_value
                parent = l_element_flight ).

*     Criar elemento 'arrival' como child de 'flight'
    l_value = l_spfli-arrtime.
    l_element_dummy  = l_document->create_simple_element(
                name = 'arrival'
                value = l_value
                parent = l_element_flight ).

  ENDLOOP.

*   Criando um stream factory
  l_streamfactory = l_ixml->create_stream_factory( ).
*   Conectar Tabela XML para o stream factory
*   ele criar a saida de stream do XML
  l_ostream = l_streamfactory->create_ostream_itable( table =
l_xml_table ).

*   Rendering ( Normaliza o documento XML) o documento
  l_renderer = l_ixml->create_renderer( ostream  = l_ostream
                                        document = l_document ).
  l_rc = l_renderer->render( ).

*   Salvar o arquivo XML document
  l_xml_size = l_ostream->get_num_written_raw( ).

  CALL METHOD cl_gui_frontend_services=>gui_download
    EXPORTING
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

```
      bin_filesize            = l_xml_size
      filename                = w_fullpath
      filetype                = 'BIN'
    CHANGING
      data_tab                = l_xml_table
    EXCEPTIONS
      file_write_error        = 1
      no_batch                = 2
      gui_refuse_filetransfer = 3
      OTHERS                  = 4.
  IF sy-subrc <> 0.
    MESSAGE ID sy-msgid TYPE sy-msgty NUMBER sy-msgno
               WITH sy-msgv1 sy-msgv2 sy-msgv3 sy-msgv4.
  ENDIF.
```


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

## 13. TESTES

Com o objetivo de avaliar o que você aprendeu no curso, segue abaixo alguns testes para você se auto-avaliar sobre seu conhecimento em ABAP Objects. As perguntas têm múltiplas respostas, onde podem conter apenas uma reposta correta, ou todas as respostas corretas, ou todas as respostas incorretas.

1) **Pergunta 1**

O que uma instancia de uma classe?

**Resposta 1**
É um atributo público que pode ser referenciado.

**Resposta 2**
É uma referencia de memória de um objeto.

**Resposta 3**
É uma referencia de memória de classe com atributos estáticos.

**Resposta 4**
É uma chamada de uma classe sem necessidade do CREATE OBJETC.

2) **Pergunta 2**

O que é verdade sobre Orientação a Objeto?

**Resposta 1**
As linguagens orientadas a objeto representam melhor o mundo real, do que as linguagens procedurais.

**Resposta 2**
É possível utilização de objetos, ou classes, mas outros aplicativos podem acessar somente um único objeto por vez.

**Resposta 3**
Através da UML e a forma de documentação dos objetos, existe uma melhoria na estrutura do software desenvolvido.

**Resposta 4**
Reduz custos de manutenção e desenvolvimento.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 1**

O que uma instancia de uma classe?

**Resposta 1**
Errada:

**Resposta 2**
Correta:

**Resposta 3**
Errada:

**Resposta 4**
Errada:

**Pergunta 2**

O que é verdade sobre Orientação a Objeto?

**Resposta 1**
Correta:

**Resposta 2**
Errada:

**Resposta 3**
Correta:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

3) **Pergunta 3**

O que encapsulamento?

**Resposta 1**
É criação de uma classe com atributos (Privados ou Públicos) com referencia de outra classe.

**Resposta 2**
É a implementação de uma classe com métodos e atributos privados para somente utilização do próprio objeto.

**Resposta 3**
É implementação de atributos públicos em uma classe, para que todos os objetos possam visualizar.

**Resposta 4**
É a implementação de um objeto com determinadas funções(métodos) disponíveis para outros objetos utilizarem.

4) **Pergunta 4**

O que é verdade em UML?

**Resposta 1**
É um padrão aberto de modelagem de linguagens Orientadas o Objeto, permitindo especificação, construção, visualização e documentação de modelos de software.

**Resposta 2**
Em UML, uma classe possui: Nome da Classe, Atributos, Métodos, e Eventos.

**Resposta 3**
Atributos e Métodos descrevem as características de um objeto do mundo real.

**Resposta 4**
A UML promove melhor maior qualidade, documentação e principalmente garante os fortes conceitos de Orientação a Objetos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 3**

O que encapsulamento?

**Resposta 1**
Errada:

**Resposta 2**
Errada:

**Resposta 3**
Errada:

**Resposta 4**
Correta:

**Pergunta 4**

O que é verdade em UML?

**Resposta 1**
Correta:

**Resposta 2**
Errada:

**Resposta 3**
Correta:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

5) **Pergunta 5**

O que é Especialização e Generalização?

**Resposta 1**
Especialização é um nível de informação de uma SuperClasse que será herdada para as SubClasses

**Resposta 2**
A SuperClasse é a Generalização de uma Subclasse.

**Resposta 3**
A SubClasse é a Especialização de uma SuperClasse.

**Resposta 4**
A Generalização é um nível de informação de uma SubClasse que possuem maiores detalhes do que a SuperClasse.

6) **Pergunta 6**

Sobre os Componentes de uma classe eles podem ser?

**Resposta 1**
Privados onde outros objetos podem acessar.

**Resposta 2**
Públicos onde outros objetos podem acessar.

**Resposta 3**
Privados e Protected componentes podem somente serem acessados por suas próprias classes ou  classes Friends.

**Resposta 4**
E UML o símbolo de Componentes são:
Privados: "-"
Públicos: "+"
Protected: "#"


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 5**

O que é Especialização e Generalização?

**Resposta 1**
Errada:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Errada:

**Pergunta 6**

Sobre os Componentes de uma classe eles podem ser?

**Resposta 1**
Errada:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

7) **Pergunta 7**

O que é verdade sobre Atributo de Instancia?

**Resposta 1**
Existem separadamente para cada objeto ou referencia criado.

**Resposta 2**
Ele é declarado com a declaração DATA na parte DEFINITION de uma classe.

**Resposta 3**
Para acessar um atributo de instancia não necessário utilizar a declaração TYPE REF TO e o CREATE OBJETC.

**Resposta 4**
Ele é declarado com a declaração CLASS-DATA na parte DEFINITION de uma classe.

8) **Pergunta 8**

O que é verdade sobre Atributo Estático?

**Resposta 1**
Ele é declarado com a declaração DATA na parte DEFINITION de uma classe.

**Resposta 2**
Ele é acessado com a seguinte declaração CLASSE=>ATRIBUTO.

**Resposta 3**
Este atributo usualmente, contém informações gerais de uma classe. Exemplos. Numero de objetos criados, contadores, etc...

**Resposta 4**
Ele é acessado com a seguinte declaração CLASSE->ATRIBUTO.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 7**

O que é verdade sobre Atributo de Instancia?

**Resposta 1**
Correta:

**Resposta 2**
Correta:

**Resposta 3**
Errada:

**Resposta 4**
Errada:

**Pergunta 8**

O que é verdade sobre Atributo Estático?

**Resposta 1**
Errada:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Errada:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

9) **Pergunta 9**

O que é verdade sobre Métodos?

**Resposta 1**
São funções/ações de classes que podem modificar o comportamento de um objeto.

**Resposta 2**
São declarados no DEFINITION de uma classe e implementados o código no IMPLEMENTATION.

**Resposta 3**
Podem ser Privados, Públicos e Protected.

**Resposta 4**
Eles possuem assinaturas que são chamados como parâmetros: os quais são: IMPORTING, EXPORTING, CHANGING, RETURNING, EXCEPTIONS E RAISING.

10) **Pergunta 10**

O que é um Método Constructor?

**Resposta 1**
È um método que pode ser executado a qualquer momento para iniciar os valores dos atributos.

**Resposta 2**
É um método que sempre deve ser declarado como PRIVADO para garantir a integridade da classe.

**Resposta 3**
Ele somente pode ser criado com CREATE OBJECT.

**Resposta 4**
É um método que somente é executado na criação do objeto, para iniciar os componentes necessários, exemplos: Atributos.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 9**

O que é verdade sobre Métodos?

**Resposta 1**
Correta:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Correta:

**Pergunta 10**

O que é um Método Constructor?

**Resposta 1**
Errada:

**Resposta 2**
Errada:

**Resposta 3**
Correta:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

11) **Pergunta 11**

O que é verdade sobre Herança?

**Resposta 1**
É o relacionamento de Subclasses entre Subclasses.

**Resposta 2**
È uma excelente forma de centralizar alguns componentes comuns que poderão ser utilizados por demais SubClasses.

**Resposta 3**
Possui conceitos fortes como:
- Centralização de Códigos
- Reuso de Código pelas SubClasses
- É possível fazer definições ( Alterações de Métodos )
- As alterações realizadas são automaticamente Herdadas pelas SubClasses.

**Resposta 4**
Em ABAP Objetcs é possível implementar herança múltipla, com relacionamento de SuperClasses e SubClasses.

12) **Pergunta 12**

O que é verdade sobre Redefinição?

**Resposta 1**
Atributos e Métodos (Privados e Públicos) podem ser redefinidos.

**Resposta 2**
Na alteração do código da redefinição, o método super-> sempre deve ser executado, para garantir a integridade da SuperClasse.

**Resposta 3**
Somente na redefinição é possível redefinir métodos Privados.

**Resposta 4**
É uma redefinição de código do método da subclasse herdado pela superclasse.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 11**

O que é verdade sobre Herança?

**Resposta 1**
Errada:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Errada:

**Pergunta 12**

O que é verdade sobre Redefinição?

**Resposta 1**
Errada:

**Resposta 2**
Correta:

**Resposta 3**
Errada:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

13) **Pergunta 13**

O que é verdade sobre Interfaces

**Resposta 1**
As declarações de Interfaces são iguais às classes e somente possuem código extra de implementações.

**Resposta 2**
Os atributos de interfaces devem ser declarados sempre como Privados.

**Resposta 3**
As interfaces são adições e implementações em classe. E tem a função de manter um padrão de implementação de classes.

**Resposta 4**
As interfaces somente podem ser implementadas uma única vez por classe. Assim não permitindo a Herança Múltipla em ABAP OBJECTS.

14) **Pergunta 14**

O que é verdade sobre  Eventos?

**Resposta 1**
Somente podem ser acionados por métodos.

**Resposta 2**
Somente podem exportar parâmetros.

**Resposta 3**
Existem dois tipos de Eventos ( Estáticos e de Instancia)

**Resposta 4**
O comando SET HANDLER permite registrar um EVENTO.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 13**

O que é verdade sobre Interfaces?

**Resposta 1**
Errada:

**Resposta 2**
Errada:

**Resposta 3**
Correta:

**Resposta 4**
Errada:

**Pergunta 14**

O que é verdade sobre  Eventos?

**Resposta 1**
Correta:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

15) **Pergunta 15**

O que é verdade sobre Classes Globais?

**Resposta 1**
Classes Globais podem ser acessadas de qualquer objeto R/3. Porque estão gravadas no Repositório SAP.

**Resposta 2**
Existe uma transação específica para criação de classes globais a SE24.

**Resposta 3**
Podem ser utilizadas pelo Objetc Navigator.

**Resposta 4**
È possível criar classes persistentes.


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*

**Respostas página anterior:**

**Pergunta 15**

O que é verdade sobre Classes Globais?

**Resposta 1**
Correta:

**Resposta 2**
Correta:

**Resposta 3**
Correta:

**Resposta 4**
Correta:


*Documento de propriedade da Braxis IT Services – Proibida a reprodução sem autorização*