

Delphi Avançado



Ementa

- ▶ Bibliotecas e Componentes
- ▶ Objetos, Componentes e Controladores
- ▶ Manipulação de Arquivos
- ▶ Desenvolvendo DLL's
- ▶ Multi-Threading
- ▶ Desenvolvendo Aplicações com FireMonkey
- ▶ Relatórios
- ▶ Testes Unitários - Dunit
- ▶ Generics
- ▶ Interfaces

Ementa



Manipulação de Arquivos

Manipulação de Arquivos

- ▶ Arquivo Texto (*.txt)
- ▶ O processamento de arquivo representa um subconjunto das capacidades de processamento que permitem a um programa armazenar e processar volumes maciços de dados persistentes.
- ▶ Utilizamos uma variável do tipo **TextFile** e posteriormente os procedimentos **AssignFile**, **Reset** ou **Rewrite/Append**.

Manipulação de Arquivos

- ▶ Principais procedimentos utilizados:

- ▶ **TextFile** (Variável do tipo arquivo texto)
- ▶ **AssignFile** (Associa o nome lógico do arquivo ao arquivo físico)
- ▶ **Rewrite** (Cria novo arquivo em disco)
- ▶ **Reset** (Prepara um arquivo existente para leitura)
- ▶ **Append** (Abre um arquivo existente para gravação)
- ▶ **CloseFile** (Fecha o arquivo texto aberto)
- ▶ **Readln** (Lê uma linha do arquivo texto)
- ▶ **Read** (Lê um caractere do arquivo texto)
- ▶ **Writeln** (Grava no arquivo texto, deixando o marcador no final da linha)
- ▶ **Write** (Grava no arquivo texto)
- ▶ **Eoln** (Retorna se a leitura está no final da linha do arquivo)
- ▶ **Eof** (Retorna se a leitura do arquivo chegou ao fim)

Exercícios

01 - Criando Arquivo Texto

02 - Lendo Arquivo Texto

03 - Criando um Diário de Notas

The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the image, creating a modern, layered effect. The rest of the background is a solid, very light blue.

DLL's

Desenvolvendo DLL's

▶ Biblioteca de Vínculo Dinâmico



▶ Dynamic Link Library



▶ Biblioteca de enlaces dinámicos



▶ مكتبة الارتباط الحيوي



Desenvolvendo DLL's

- ▶ Uma DLL é uma biblioteca que contém código e dados que podem ser usados por mais de um programa ao mesmo tempo. Por exemplo, em sistemas operacionais Windows, a DLL **Comdlg32** executa funções comuns relacionadas à caixa de diálogo. Cada programa pode usar a funcionalidade contida nessa DLL para implementar uma caixa de diálogo **Abrir**. Isso ajuda a promover a reutilização de código e o uso eficiente de memória.

Desenvolvendo DLL's

- ▶ Alguns exemplos de DLL's em sistemas operacionais Windows:

Arquivos de Controles ActiveX (.ocx)*

Um exemplo de controle ActiveX é um controle de calendário que permita selecionar uma data de calendário.

Arquivos do Painel de Controle (.cpl)*

Um exemplo de arquivo .cpl é um item localizado em Painel de Controle. Cada item é uma DLL especializada.

Arquivos de Driver de Dispositivo (.drv)

Um exemplo de driver de dispositivo é um driver de impressora que controla a impressão em uma impressora.

Desenvolvendo DLL's

▶ *Vantagens da DLL*

- ▶ Usa menos recursos
- ▶ Promove a arquitetura modular
- ▶ Facilita a implantação e a instalação

Desenvolvendo DLL's

- ▶ A Embarcadero disponibiliza no Delphi o acesso a essas API's através da unit *Windows*, que realiza uma ponte entre o código Delphi e as várias DLLs disponibilizadas pelo sistema. As principais DLLs são:

User32.dll

Kernel32.dll

Comdlg32.dll

Gdi32.dll

Shell32.dll

Advapi32.dll

Winmm.dll

Desenvolvendo DLL's

- ▶ Carregar DLL explicitamente não consome a memória de toda a aplicação, somente no ponto do código.
- ▶ Carregar a DLL implicitamente consome a memória da aplicação e a mesma somente é liberado ao fechar o programa principal.

Exercícios

01 - Capturando Informações do Sistema Operacional

02 - Consumindo uma DLL feita em C#

03 - Criando uma DLL no Delphi

The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the left and right sides of the frame, creating a modern, architectural feel. The central area is a plain, light grayish-white.

Firemonkey

Desenvolvendo Aplicações com o Firemonkey

- ▶ Introduzido no Delphi desde a versão XE2.
- ▶ Framework voltado para Aplicações Modernas, Multiplataforma (rodando no Windows, MAC (OS X), IOS, Android e Linux).
- ▶ Faz uso dos recursos gráficos da GPU do Hardware.

Desenvolvendo Aplicações com o Firemonkey

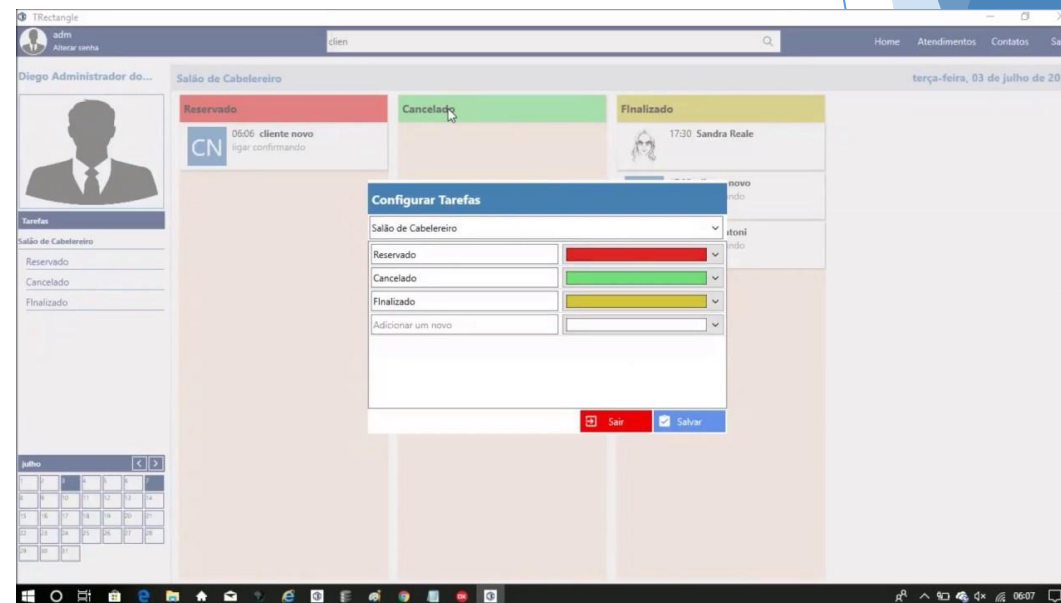
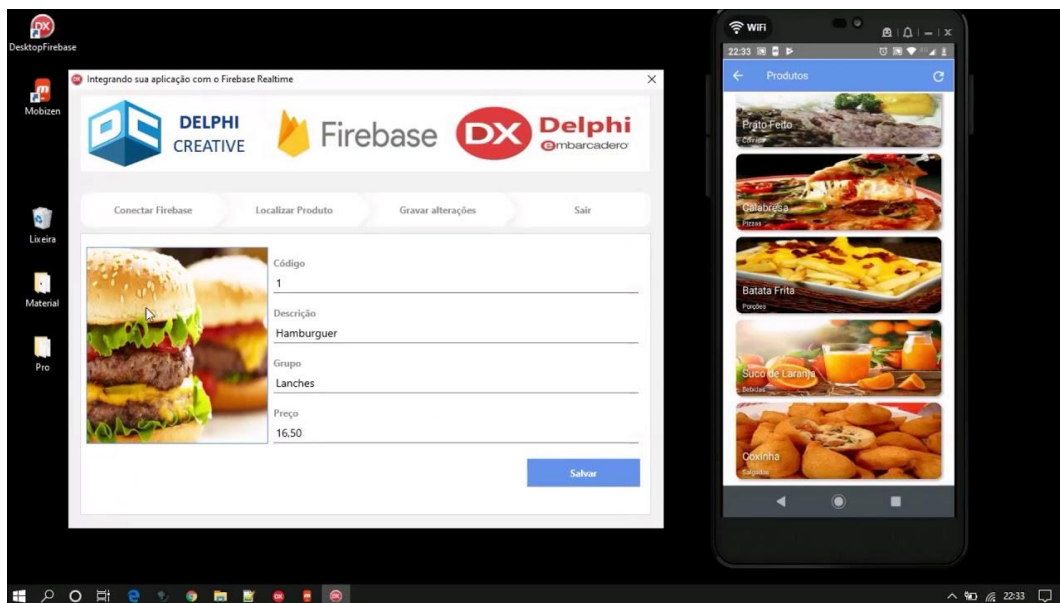
- ▶ Quando usar?
- ▶ Sua aplicação precisa rodar em outras plataformas?
- ▶ Sua aplicação precisa ter um layout mais rico?
- ▶ Sua aplicação precisa usar recursos gráficos da GPU que a VCL não utiliza?

Desenvolvendo Aplicações com o Firemonkey

- ▶ Onde gravar os dados?
- ▶ Algo simples usar o Shared Preferences
- ▶ Algo complexo usar o banco SQLite

Desenvolvendo Aplicações com o Firemonkey

► Exemplos de Aplicações?



Exercícios

01 - ToDo List

02 - Réplica NetFlix

03 - Réplica Mercado Livre

04 - IMC

05 - Jogo da Velha

Multi-Threading

Multi-Threading

- ▶ Threads são processamentos simultâneos. Muitas vezes utilizado para melhorar o tempo de processamento da rotina.
- ▶ Um bom exemplo desse tipo é a ação de verificação de grafia enquanto está digitando um texto.

Multi-Threading

- ▶ Quando Utilizar?
- ▶ São úteis quando um aplicativo necessita realizar uma operação longa.
- ▶ Evitar o famoso “Sistema parou de responder”

Multi-Threading

- ▶ Métodos
- ▶ Execute: Faz o trabalho da Thread em si.
- ▶ Terminated: Verifica se a execução da Thread foi finalizada
- ▶ Sincronize: Interage com a Thread principal da aplicação

Multi-Threading

- ▶ Thread x Thread Anônima
- ▶ Threads é uma classe que precisa ser instanciada para sua utilização. Recomendada para casos mais complexos.
- ▶ Threads Anônimas quando queremos executar algo simples dentro da Thread principal da aplicação.

Bibliotecas e Componentes

Bibliotecas e Componentes

- ▶ Existem dois tipos de Bibliotecas e Componentes
- ▶ Nativos X Terceiros

Bibliotecas e Componentes

- ▶ Nativos
- ▶ São os que vem com a instalação do Delphi. São mais seguros e menor risco de descontinuação. Nele temos o selo da Embarcadero.
- ▶ Terceiros
- ▶ São os que instalamos como se fosse “pacotes” ou “plugins” adicionais. Há um certo risco na utilização. E há sempre uma dependência do autor evoluir a biblioteca caso a mesma não seja open-source.

Bibliotecas e Componentes

- ▶ <https://github.com/skia4delphi/skia4delphi>
- ▶ <https://github.com/rzaripov1990/ZMaterialComponents>

Relatórios no Delphi

Relatórios no Delphi

- ▶ Incorporado
- ▶ FastReport - <https://www.fast-report.com/pt/>
- ▶ Terceiros
- ▶ ReportBuilder, FortesReport, QuickReport e entre outros

Relatórios no Delphi

- ▶ Como instalar
- ▶ Tools -> GetIt Package Manager -> Pesquisar por FastReport
- ▶ Realizar a instalação

Relatórios no Delphi

- ▶ Principais Componentes (Tela):
 - ▶ TfrxReport, TfrxDBDataset, TfrxPDFExport e TFDQuery
- ▶ Principais Recursos (Relatório):
 - ▶ Bandas: ReportTitle, Header, MasterData e DetailData
 - ▶ Textos: TfrxMemoView

Exercícios

01 - PedidoCompra

02 - Banco Sakila

Testes Unitários - DUnit

Testes Unitários - DUnit

- ▶ VANTAGENS
- ▶ Facilidade em encontrar falhas e eliminá-las de forma rápida.
- ▶ Cobrir todas as possibilidades do seu código para garantir que tudo esteja funcionando adequadamente
- ▶ Economia de tempo

Testes Unitários - DUnit

- ▶ COMO ESCREVER BONS TESTES?
- ▶ Não utilize lógica nos testes
- ▶ Precisam ser fáceis de serem executados e lidos
- ▶ Faça testes com os valores esperados primeiro
- ▶ Independentes
- ▶ Rode seus testes constantemente!!!

Testes Unitários - DUnit

- ▶ AAA - Arrange, Act e Assert
- ▶ Arrange: Preparar todas as condições necessárias para a execução do teste
- ▶ Act: Executar o que será testado
- ▶ Assert: Validar as informações depois do teste realizado

Exercícios

01 - Calculadora

02 - PedidoCompra

Generics

Generics

- ▶ Recurso novo no Delphi que permite que criemos tipos parametrizados.
- ▶ Pode ser usado como tipo de um Field, Parâmetro e Classe.
- ▶ `TMyClass<T> = class`
- ▶ `FCampo: T;`
- ▶ `function MeuMetodo(const aTipo: T): T;`

Exercícios

- 01 - Conhecendo o Generics
- 02 - Validação de Tipos
- 03 - Enum To List
- 04 - Método Genérico - “IIF” - Ternário
- 05 - Constructor e Constraint
- 06 - TList
- 07 - TQueue (Fila)
- 08 - TStack (Pilha)
- 09 - TDictionary (Dicionário)
- 10 - Desafio (Criar uma tela de Senha de Hospital - TQueue)

Interfaces

Interfaces

- ▶ É um tipo que define um conjunto de operações que uma classe deve implementar.
- ▶ A interface estabelece um **contrato** que a classe deve cumprir.

```
IShape = interface  
['{EEC6993B-CB8D-4740-9C71-EF19785E11E0}']  
function Area: Double;  
function Perimetro: Double;  
end;
```

```
TRectangle = class(TInterfacedObject, IShape)  
private  
public  
function Area: Double;  
function Perimetro: Double;  
end;
```

Interfaces

- ▶ Pra quê interfaces?
- ▶ R: Para criar sistemas com **baixo acoplamento e mais flexíveis**

Interfaces (Exemplo na Prática)

- ▶ Uma locadora brasileira de carros cobra um valor por hora para locações de até 12 horas. Porém, se a duração da locação ultrapassar 12 horas, a locação será cobrada com base em um valor diário. Além do valor da locação, é acrescido no preço o valor do imposto conforme regras do país que, no caso do Brasil, é 20% para valores até R\$ 100,00, ou 15% para valores acima de R\$ 100,00.
- ▶ Fazer um programa que lê os dados da locação (modelo do carro, instante inicial e final da locação), bem como o valor por hora e o valor diário de locação. O programa deve então gerar uma nota de pagamento (contendo valor da locação, valor do imposto e valor total do pagamento) e informar os dados na tela. Veja os exemplos:

Interfaces (Exemplo na Prática)

- ▶ Exemplo 1 (Por Hora):

Entre com os dados do aluguel

Modelo do carro: Civic

Retirada (dd/MM/yyyy hh:mm): 25/06/2018 10:30

Retorno (dd/MM/yyyy hh:mm): 25/06/2018 14:40

Entre com o preço por hora: 10.00

Entre com o preço por dia: 130.00

FATURA:

Pagamento basico: 50.00

Imposto: 10.00 Pagamento

total: 60.00

- ▶ Cálculos: Duração = (25/06/2018 14:40) - (25/06/2018 10:30) = 4:10 = 5 horas
- ▶ Pagamento básico = $5 * 10 = 50$
- ▶ Imposto = $50 * 20\% = 50 * 0.2 = 10$

Interfaces (Exemplo na Prática)

- ▶ Exemplo 2 (Por Dia):

Entre com os dados do aluguel

Modelo do carro: Civic

Retirada (dd/MM/yyyy hh:mm): 25/06/2018 10:30

Retorno (dd/MM/yyyy hh:mm): 27/06/2018 11:40

Entre com o preço por hora: 10.00

Entre com o preço por dia: 130.00

FATURA:

Pagamento basico: 390.00

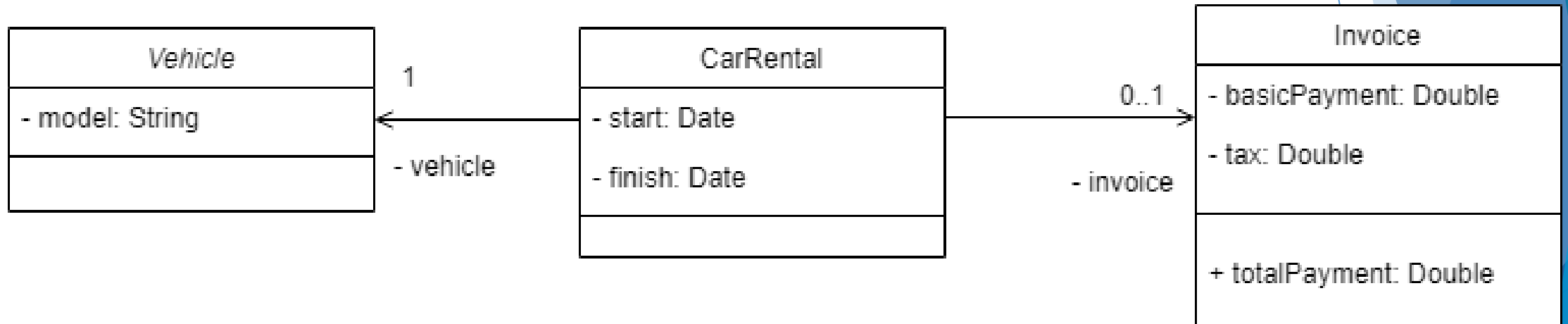
Imposto: 58.50 Pagamento

total: 448.50

- ▶ Cálculos: Duração = (27/06/2018 11:40) - (25/06/2018 10:30) = 2 dias + 1:10 = 3 dias
- ▶ Pagamento básico = $3 * 130 = 390$
- ▶ Imposto = $390 * 15\% = 390 * 0.15 = 58.50$

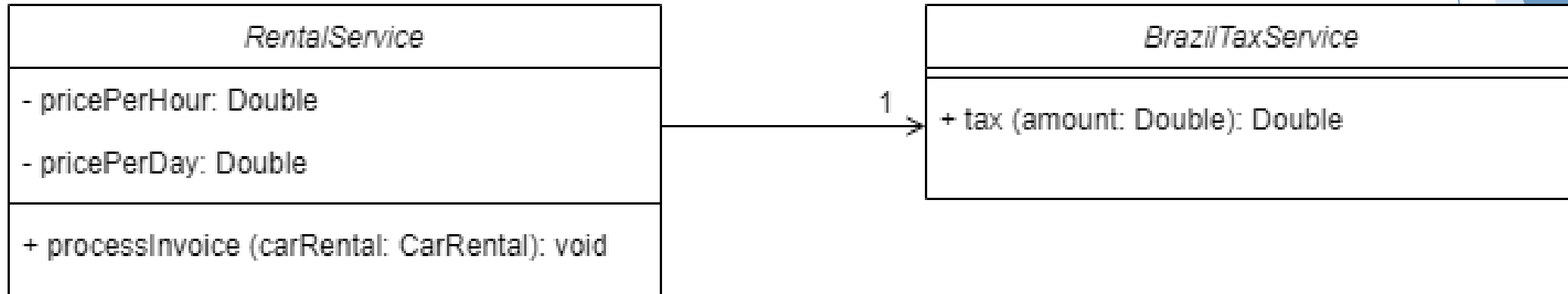
Interfaces (Solução sem Interface)

► Entidades



Interfaces (Solução sem Interface)

► Services



Interfaces (Solução sem Interface)

- O que acontece se a empresa usar a taxa do USA?

```
Welcome Page | Aluguel | URentalService X
URentalService
├─ unit URentalService;
├─ interface
├─ uses
│   UBrazilTaxService, UCarRental;
├─ type
├─ TRentalService = class
├─   private
├─     FPricePerDay: Double;
├─     FPricePerHour: Double;
├─     FTaxService: TBrazilTaxService;
├─   public
├─     constructor Create(const aPricePerDay, aPricePerHour: Double; aTaxService: TBrazilTaxService);
├─     procedure ProcessInvoice(aCarRental: TCarRental);
├─   end;
├─ implementation
├─   uses
├─     Math, DateUtils, UInvoice;
├─   { TRentalService }
├─   constructor TRentalService.Create(const aPricePerDay, aPricePerHour: Double; aTaxService: TBrazilTaxService);
├─   begin
```

```
Welcome Page | Aluguel | URentalService
Writeln('Retirada (dd/mm/yyyy hh:mm):');
Readln(xStrStart);
xStart := StrToDateTime(xStrStart);

Writeln('Retorno (dd/mm/yyyy hh:mm):');
Readln(xStrFinish);
xFinish := StrToDateTime(xStrFinish);

xCarRental := TCarRental.Create(xStart, xFinish, TVehicle.Create(xCarModel));

Writeln('Entre com o preço por hora:');
Readln(xStrPricePerHour);
xPricePerHour := StrToFloatDef(xStrPricePerHour, 0);

Writeln('Entre com o preço por dia:');
Readln(xStrPricePerDay);
xPricePerDay := StrToFloatDef(xStrPricePerDay, 0);

xRentalService := TRentalService.Create(xPricePerDay, xPricePerHour, TBrazilTaxService.Create);
xRentalService.ProcessInvoice(xCarRental);

Writeln('FATURA:');
Writeln('Pagamento básico... ', xCarRental.Invoice.BasicPayment.ToString);
Writeln('Imposto..... ', xCarRental.Invoice.Tax.ToString);
Writeln('Total..... ', xCarRental.Invoice.TotalPayment.ToString);

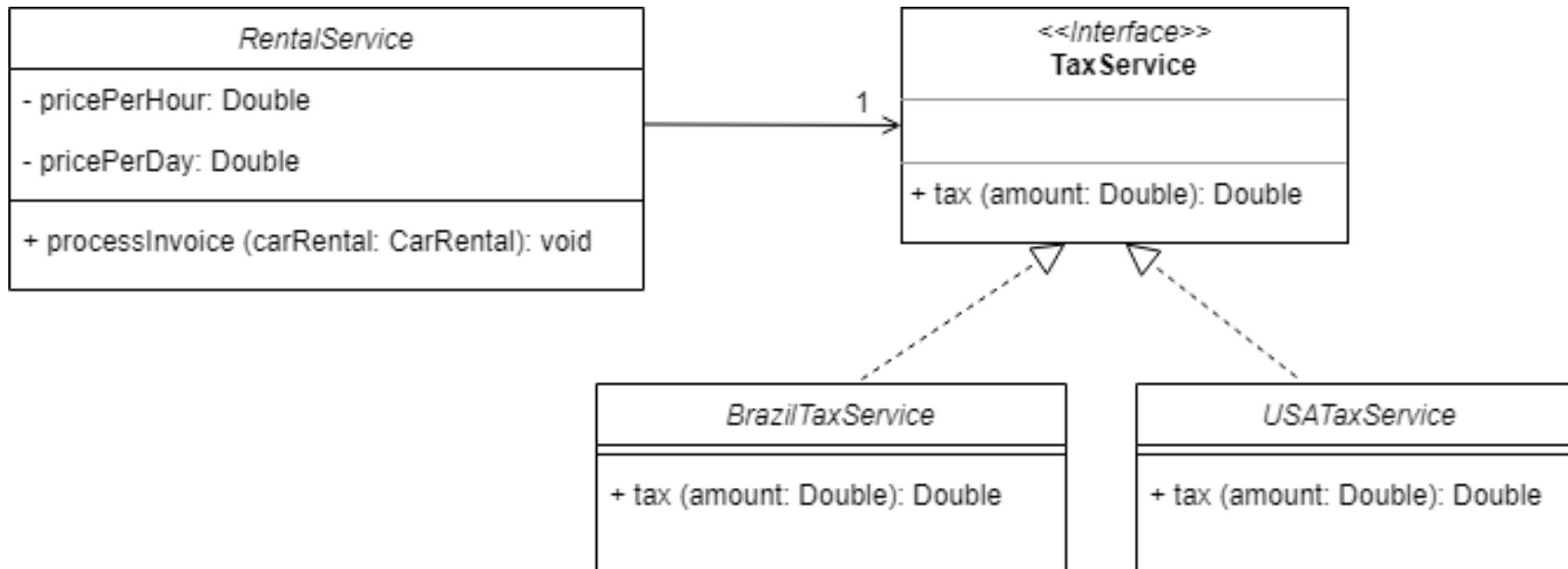
Readln;
except
```

Interfaces (Solução sem Interface)

- ▶ Violamos o princípio “O” do SOLID
- ▶ “Sua classe deve ser fechada para modificações e aberta para extensões”
- ▶ Então como corrigir? Usando interfaces!!!

Interfaces (Solução com Interface)

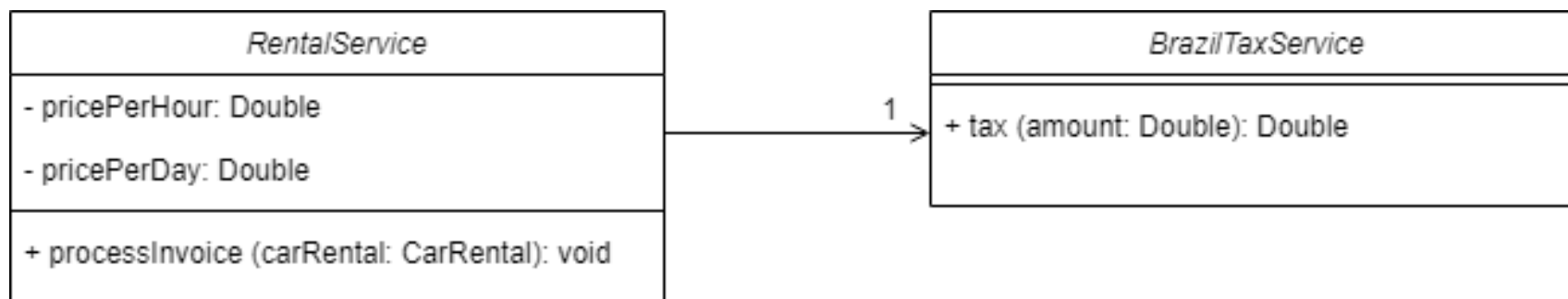
► Services




Inversão de Controle e Injeção de Dependência

Interfaces (Inversão e Injeção)

- ▶ Acoplamento forte
- ▶ A classe RentalService conhece a dependência concreta
- ▶ Se a classe concreta mudar, é preciso mudar a classe RentalService

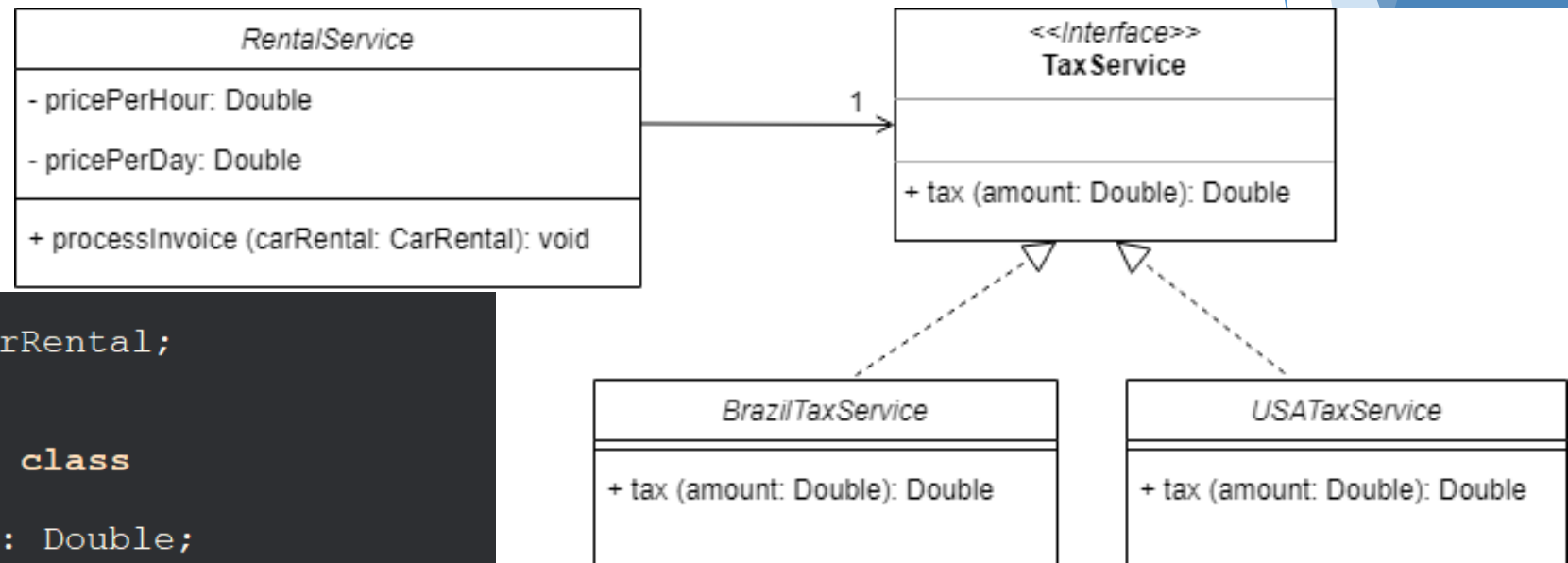


```
· type
· □ TRentalService = class
0   private
·     FPricePerDay: Double;
·     FPricePerHour: Double;
·
·
4   FTaxService: TBrazilTaxService;
```



Interfaces (Inversão e Injeção)

- ▶ Acoplamento fraco
- ▶ A classe RentalService não conhece a dependência concreta
- ▶ Se a classe concreta mudar, a classe RentalService não muda nada.



```
- uses
.   UTaxService, UCarRental;
.
. type
.   TRentalService = class
10     private
.       FPricePerDay: Double;
.       FPricePerHour: Double;
.
.       FTaxService: ITaxService;
-     public
.       constructor Create(const aPricePerHour: Double, const aPricePerDay: Double)
```

Injeção de dependência por meio do construtor

```
Aluguel x URentalService

Writeln('Entre com o preço por hora:');
Readln(xStrPricePerHour);
xPricePerHour := StrToFloatDef(xStrPricePerHour, 0);

Writeln('Entre com o preço por dia:');
Readln(xStrPricePerDay);
xPricePerDay := StrToFloatDef(xStrPricePerDay, 0);

//xRentalService := TRentalService.Create(xPricePerDay, xPricePerHour, TBrazilTaxService.Create);
xRentalService := TRentalService.Create(xPricePerDay, xPricePerHour, TUSATaxService.Create);
xRentalService.ProcessInvoice(xCarRental);

Writeln('FATURA:');
Writeln('Pagamento básico....: ', xCarRental.Invoice.BasicPayment.ToString);
Writeln('Imposto.....: ', xCarRental.Invoice.Tax.ToString);
Writeln('Total.....: ', xCarRental.Invoice.TotalPayment.ToString);
```

upcasting

Inversão de Controle

- ▶ **Inversão de Controle**
- ▶ Padrão de desenvolvimento que consiste em retirar da classe a responsabilidade de instanciar suas dependências.
- ▶ **Injeção de Dependência**
- ▶ É uma forma de realizar a inversão de controle: um componente externo instancia a dependência, que é então injetada no objeto “pai”.

Exercícios

01 - Conhecendo Interface

02 - Calculadora

Ementa (Final)

- ▶ Bibliotecas e Componentes
- ▶ Objetos, Componentes e Controladores
- ▶ Manipulação de Arquivos
- ▶ Trabalhando com Listas
- ▶ Trabalhando com Formulários
- ▶ Manipulação de Erros
- ▶ Desenvolvendo DLL's
- ▶ Multi-Threading
- ▶ Injeção de Dependência
- ▶ Conexão com Banco de Dados Utilizando o FireDAC
- ▶ Desenvolvendo Aplicações com FireMonkey (Mobile)
- ▶ Relatórios (Fast Report)
- ▶ Testes Unitários - Dunit
- ▶ Generics
- ▶ Interfaces

Questões da Certificação

Question 20 of 60

Classes and Objects

A class in Delphi can have only one base class.

A. True

B. False

Question 20 of 60

Classes and Objects

A class in Delphi can have only one base class.

Correct answer: A.

You chose: A.

✓ A. True

B. False

Question 22 of 60

Classes and Objects

In Delphi, methods can have...

- A. Default values for parameters
 - B. Overloading
 - C. Support for inline code
 - D. a virtual definition
 - E. All of the above
-

Question 22 of 60

Classes and Objects

In Delphi, methods can have...

Correct answer: E.

You chose: E.

- A. Default values for parameters
 - B. Overloading
 - C. Support for inline code
 - D. a virtual definition
 - ✓ E. All of the above
-

Question 23 of 60

Classes and Objects

Which is the keyword used to mark a virtual method that has no implementation in the given class?

This question displayed answers in random order during the test.

- A. sealed
 - B. unique
 - C. abstract
 - D. interface
-

Question 23 of 60

Classes and Objects

Which is the keyword used to mark a virtual method that has no implementation in the given class?

Correct answer: C.

You chose: C.

This question displayed answers in random order during the test.

- A. sealed
 - B. unique
 - ✓ C. abstract
 - D. interface
-

Question 24 of 60

Classes and Objects

If a class has a field referring to another class, which of the following is true?

This question displayed answers in random order during the test.

- A. Delphi will use reference counting for the internal object
 - B. Delphi will allocate the memory for the internal objects if you mark the field as auto
 - C. Delphi will allocate memory for the internal object automatically
 - D. Delphi won't allocate the memory for the internal object, so you must do it before using the object (and generally Free it in the destructor)
 - E. Delphi won't allocate the memory for the internal object, so you must do it in the constructor and desctructor
 - F. Delphi won't allocate the memory for the internal object, so you must override special memory allocation functions
-

Question 24 of 60

Classes and Objects

If a class has a field referring to another class, which of the following is true?

Correct answer: D.

You chose: D.

This question displayed answers in random order during the test.

- A. Delphi will use reference counting for the internal object
- B. Delphi will allocate the memory for the internal objects if you mark the field as auto
- C. Delphi will allocate memory for the internal object automatically
- ✓ D. Delphi won't allocate the memory for the internal object, so you must do it before using the object (and generally Free it in the destructor)
- E. Delphi won't allocate the memory for the internal object, so you must do it in the constructor and destructor
- F. Delphi won't allocate the memory for the internal object, so you must override special memory allocation functions

Question 26 of 60

Classes and Objects

In a derived class, can you override methods declared as virtual in a base class?

A. True

B. False

Question 26 of 60

Classes and Objects

In a derived class, can you override methods declared as virtual in a base class?

Correct answer: A.

You chose: A.

✓ A. True

B. False

Question 27 of 60

Standard Routines and I/O

What is the effect of the compiler directive `{I-}`?

- A. When an I/O error happens, an `EFileError` exception gets raised
 - B. When an I/O error happens, it will not generate an exception, and the program will have to check the I/O operation by using the `IOResult` routine
 - C. When an I/O error happens, an `EInOutError` exception gets raised
 - D. None of the above
-

Question 27 of 60

Standard Routines and I/O

What is the effect of the compiler directive `{ $I- }`?

Correct answer: B.

You chose: B.

- A. When an I/O error happens, an `EFileError` exception gets raised
 - ✓ B. When an I/O error happens, it will not generate an exception, and the program will have to check the I/O operation by using the `IOResult` routine
 - C. When an I/O error happens, an `EInOutError` exception gets raised
 - D. None of the above
-

Question 28 of 60

Standard Routines and I/O

In Delphi, what function allows you to create a new directory including the creation of parent directories as needed?

- A. CreateSubDirectory
 - B. ForceDirectories
 - C. CreateDir
 - D. None of the above
-

Question 28 of 60

Standard Routines and I/O

In Delphi, what function allows you to create a new directory including the creation of parent directories as needed?

Correct answer: B.

You chose: B.

- ☐ A. CreateSubDirectory
 - ☒ B. ForceDirectories
 - ☐ C. CreateDir
 - ☐ D. None of the above
-

Question 36 of 60

Object Interfaces

In Delphi all interfaces are implicitly derived from an interface called:

- A. TInterface
 - B. IInterface
 - C. Unknown
 - D. TObject
 - E. None of the above
-

Question 36 of 60

Object Interfaces

In Delphi all interfaces are implicitly derived from an interface called:

Correct answer: B.

You chose: B.

- ☐ A. TInterface
 - ☒ B. IInterface
 - ☐ C. Unknown
 - ☐ D. TObject
 - ☐ E. None of the above
-

Which statement best describes the error in the following code?

```
type
  IAncestor = interface
  end;

  IDescendant = interface(IAncestor)
    procedure Method1;
  end;
  TMyClass = class(TInterfacedObject, IDescendant)
    procedure Method1;
    procedure Method2;
  end;

implementation

var
  D: IDescendant;
  A: IAncestor;
begin
  D := TMyClass.Create;
  D.Method1;
  D.Method2;
end;
```

This question displayed answers in random order during the test.

- A. D is declared as a variable of type TMyClass and can not be used to access TMyClass.Method2
- B. The Method2 is not declared on the IAncestor interface, so you can not use it to access TMyClass.Method2
- C. D is declared as a variable of type IDescendant and references an instance of TMyClass, and you can not use it to access TMyClass.Method2

Which statement best describes the error in the following code?

```
type
  IAncestor = interface
  end;

  IDescendant = interface(IAncestor)
    procedure Method1;
  end;
  TMyClass = class(TInterfacedObject, IDescendant)
    procedure Method1;
    procedure Method2;
  end;
```

implementation

```
var
  D: IDescendant;
  A: IAncestor;
begin
  D := TMyClass.Create;
  D.Method1;
  D.Method2;
end;
```

Correct answer: C.

You chose: C.

This question displayed answers in random order during the test.

- A. D is declared as a variable of type TMyClass and can not be used to access TMyClass.Method2
- B. The Method2 is not declared on the IAncestor interface, so you can not use it to access TMyClass.Method2
- ✓ C. D is declared as a variable of type IDescendant and references an instance of TMyClass, and you can not use it to access TMyClass.Method2

Question 45 of 60

Generics Attrib & Anon Methods

Generics allow you to define classes that do not specifically define the type of certain data members.

A. True

B. False

Question 45 of 60

Generics Attrib & Anon Methods

Generics allow you to define classes that do not specifically define the type of certain data members.

Correct answer: **A.**

You chose: **A.**

✓ **A. True**

B. False

Question 50 of 60

Exceptions and Assertions

The exception handling block starts and ends with what keywords?

- A. finally ... end
 - B. try ... end
 - C. except ... end
 - D. None of the above
-

Question 50 of 60

Exceptions and Assertions

The exception handling block starts and ends with what keywords?

Correct answer: C.

You chose: C.

- A. finally ... end
 - B. try ... end
 - ✓ C. except ... end
 - D. None of the above
-