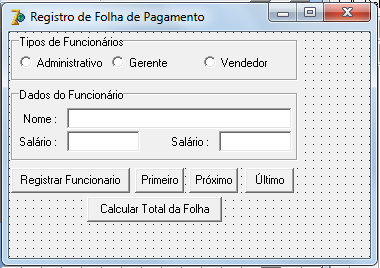
unit untFolhaDePagamento;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ExtCtrls, uFuncionario, uAdministrativo, uGerente,

 uVendedor;

type

TfrmFolhaDePagamento = class(TForm)

rgTipos: TRadioGroup;

gbxDados: TGroupBox;

Label1: TLabel;

edtNome: TEdit;

lblComplemento: TLabel;

edtComplemento: TEdit;

edtSalario: TEdit;

Label3: TLabel;

btnRegistrarFuncionario: TButton;

btnPrimeiro: TButton;

btnProximo: TButton;

btnUltimo: TButton;

btnCalculoFolhoa: TButton;

procedure rgTiposClick(Sender: TObject);

procedure btnRegistrarFuncionarioClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure btnPrimeiroClick(Sender: TObject);

procedure btnProximoClick(Sender: TObject);

procedure btnUltimoClick(Sender: TObject);

procedure btnCalculoFolhoaClick(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

function movimentarRegistro(direcao : Integer) : Boolean;

function exibeDados(registro : Integer) : Boolean;

var

frmFolhaDePagamento: TfrmFolhaDePagamento;

funcionarios : Array[1..10] of Funcionario;

registroAtual, qtdeFuncionarios : Integer;

implementation

{$R \*.dfm}

procedure TfrmFolhaDePagamento.rgTiposClick(Sender: TObject);

begin

lblComplemento.Visible := False;

edtComplemento.Visible := False;

if (rgTipos.ItemIndex = 1) then

lblComplemento.Caption := 'Gratificação :'

else if (rgTipos.ItemIndex = 2) then

lblComplemento.Caption := 'Comissões :';

if (rgTipos.ItemIndex = 1) or (rgTipos.ItemIndex = 2) then begin

lblComplemento.Visible := True;

edtComplemento.Visible := True;

end;

gbxDados.Enabled := True;

end;

procedure TfrmFolhaDePagamento.btnRegistrarFuncionarioClick(Sender: TObject);

var

ger : Gerente;

ven : Vendedor;

begin

if (edtNome.Text = '') or (edtSalario.Text = '') then begin

MessageDlg('É necessário informar ao menos o nome e salário do funcionário', mtInformation, [mbOk], 0);

Abort;

end;

Inc(qtdeFuncionarios);

if (rgTipos.ItemIndex = 0) then

funcionarios[qtdeFuncionarios] := Administrativo.Create

else if (rgTipos.ItemIndex = 1) then begin

ger := Gerente.Create;

ger.setGratificacao(StrToFloat(edtComplemento.Text));

funcionarios[qtdeFuncionarios] := ger;

end

else if (rgTipos.ItemIndex = 2) then begin

ven := Vendedor.Create;

ven.setComissoes(StrToFloat(edtComplemento.Text));

funcionarios[qtdeFuncionarios] := ven;

end;

funcionarios[qtdeFuncionarios].setNome(edtNome.Text);

funcionarios[qtdeFuncionarios].setSalario(StrToFloat(edtSalario.Text));

{ Após o registro de cada funcionário, os objetos respectivos à entrada são

limpos, esperando os dados para registro de um novo funcionário }

rgTipos.ItemIndex := -1;

gbxDados.Enabled := False;

edtNome.Text := '';

edtSalario.Text := '';

edtComplemento.Text := '';

if (qtdeFuncionarios = 10) then begin

MessageDlg('Limite máximo atingido', mtInformation, [mbOk], 0);

btnRegistrarFuncionario.Enabled := False;

end;

end;

procedure TfrmFolhaDePagamento.FormCreate(Sender: TObject);

begin

qtdeFuncionarios := 0;

end;

procedure TfrmFolhaDePagamento.btnPrimeiroClick(Sender: TObject);

begin

if (not movimentarRegistro(0)) then

MessageDlg('Registro fora dos limites. Movendo-se para o último registro', mtWarning, [mbOk], 0);

end;

function movimentarRegistro(direcao : Integer) : Boolean;

begin

if (qtdeFuncionarios = 0) then

Result := False

else if (direcao = 0) then

Result := exibeDados(1)

else if (direcao = 1) then

Result := exibeDados(registroAtual+1)

else if (direcao = 10) then

Result := exibeDados(qtdeFuncionarios);

end;

function exibeDados(registro : Integer) : Boolean;

begin

registroAtual := registro;

if (registroAtual < 1) or (registroAtual > qtdeFuncionarios) then

Result := False

else begin

frmFolhaDePagamento.edtNome.Text := funcionarios[registroAtual].getNome();

frmFolhaDePagamento.edtSalario.Text := FloatToStr(funcionarios[registroAtual].getSalario());

if (funcionarios[registroAtual].ClassName = 'Administrativo') then

frmFolhaDePagamento.rgTipos.ItemIndex := 0

else if (funcionarios[registroAtual].ClassName = 'Gerente') then begin

frmFolhaDePagamento.rgTipos.ItemIndex := 1;

frmFolhaDePagamento.edtComplemento.Text := FloatToStr((funcionarios[registroAtual] as Gerente).getGratificacao());

end

else if (funcionarios[registroAtual].ClassName = 'Vendedor') then begin

frmFolhaDePagamento.rgTipos.ItemIndex := 2;

frmFolhaDePagamento.edtComplemento.Text := FloatToStr((funcionarios[registroAtual] as Vendedor).getComissoes());

end;

Result := True;

end;

end;

procedure TfrmFolhaDePagamento.btnProximoClick(Sender: TObject);

begin

if (not movimentarRegistro(1)) then

MessageDlg('Registro fora dos limites. Movendo-se para o último registro', mtWarning, [mbOk], 0);

end;

procedure TfrmFolhaDePagamento.btnUltimoClick(Sender: TObject);

begin

if (not movimentarRegistro(10)) then

MessageDlg('Registro fora dos limites. Movendo-se para o último registro', mtWarning, [mbOk], 0);

end;

procedure TfrmFolhaDePagamento.btnCalculoFolhoaClick(Sender: TObject);

var

i, contador : Integer;

totalFolha : Real;

begin

i := 0;

contador := 1;

totalFolha := 0;

while (movimentarRegistro(i)) do begin

i := 1;

{ Uso do Polimorfismo }

totalFolha := totalFolha + funcionarios[contador].calcularSalario();

inc(contador);

end;

MessageDlg('O total a ser pago para todos os funcionários registrados é de : ' +

FormatFloat('###,##0.00', totalFolha), mtInformation, [mbOk], 0);

end;

end.

unit uAdministrativo;

interface

uses uFuncionario;

type

Administrativo = class (Funcionario)

public

function calcularSalario: Real; override;

constructor Create; overload;

destructor Destroy; override;

end;

implementation

{implementation of Administrativo}

constructor Administrativo.Create;

begin

inherited Create;

end;

destructor Administrativo.Destroy;

begin

inherited Destroy;

end;

function Administrativo.calcularSalario: Real;

begin

Result := getSalario();

end;

end.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

unit uGerente;

interface

uses uFuncionario;

type

Gerente = class (Funcionario)

private

gratificacao: Real;

public

function calcularSalario: Real; override;

procedure setGratificacao(NewVal: Real);

function getGratificacao: Real;

constructor Create; overload;

destructor Destroy; override;

end;

implementation

{implementation of Gerente}

constructor Gerente.Create;

begin

inherited Create;

end;

destructor Gerente.Destroy;

begin

inherited Destroy;

end;

function Gerente.calcularSalario: Real;

begin

Result := (gratificacao + getSalario());

end;

procedure Gerente.setGratificacao(NewVal: Real);

begin

gratificacao := NewVal;

end;

function Gerente.getGratificacao: Real;

begin

Result := gratificacao;

end;

end.

----------------------------------------------------------------------------

unit uFuncionario;

interface

type

Funcionario = class

private

nome: String;

salario: Real;

public

function calcularSalario: Real; virtual;

function getSalario: Real;

procedure setNome(NewVal: String);

procedure setSalario(NewVal: Real);

function getNome: String;

constructor Create; overload;

destructor Destroy; override;

end;

implementation

{implementation of Funcionario}

constructor Funcionario.Create;

begin

inherited Create;

end;

destructor Funcionario.Destroy;

begin

inherited Destroy;

end;

function Funcionario.getSalario(): Real;

begin

Result := salario;

end;

procedure Funcionario.setNome(NewVal: String);

begin

nome := NewVal;

end;

procedure Funcionario.setSalario(NewVal: Real);

begin

salario := NewVal;

end;

function Funcionario.getNome: String;

begin

Result := nome;

end;

function Funcionario.calcularSalario: Real;

begin

Result := getSalario();

end;

end.

---------------------------------------------------------------------------------------

unit uVendedor;

interface

uses uFuncionario;

type

Vendedor = class (Funcionario)

private

comissoes: Real;

public

function calcularSalario: Real; override;

procedure setComissoes(NewVal: Real);

function getComissoes: Real;

constructor Create; overload;

destructor Destroy; override;

end;

implementation

{implementation of Vendedor}

constructor Vendedor.Create;

begin

inherited Create;

end;

destructor Vendedor.Destroy;

begin

inherited Destroy;

end;

function Vendedor.calcularSalario: Real;

begin

Result := (comissoes + getSalario());

end;

procedure Vendedor.setComissoes(NewVal: Real);

begin

comissoes := NewVal;

end;

function Vendedor.getComissoes: Real;

begin

Result := comissoes;

end;

end.