PRÁTICA PROFISSIONAL I - PROJETO I - 2o Informática Matutino - 2014

Este projeto aplica os conhecimentos de Orientação a Objetos (modelagem, encapsulamento, sobrecarga e herança) aprendidos em Técnicas de Programação, bem como os de Programação Visual para desenvolvimento de uma aplicação completa de uma empresa de Transporte Ferroviário de Passageiros.

Usaremos a unit uMantemVetor, que descreve as classes TEntidade e TMantemVetor, com algumas pequenas alterações que deverão ser feitas para este projeto.

Descrição Geral da situação-problema

Uma empresa de transporte ferroviário efetua reservas de passagens para seus trens, por meio de um programa executado no balcão de atendimento.

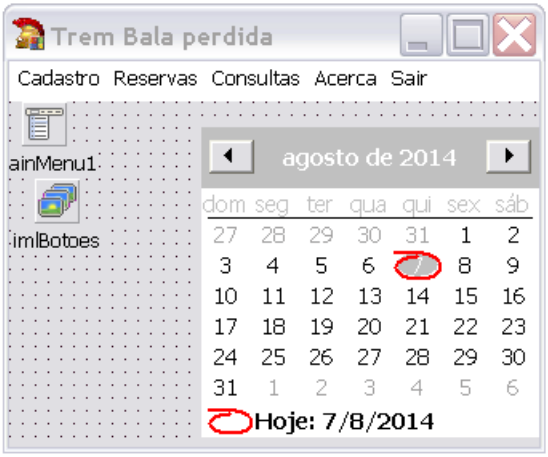
Para tanto, deverá haver um cadastro de cidades, um cadastro de viagens, um cadastro de trens e um cadastro de passageiros. Todos esses cadastros são mantidos em arquivos de registros compatíveis com os dados que armazenam.

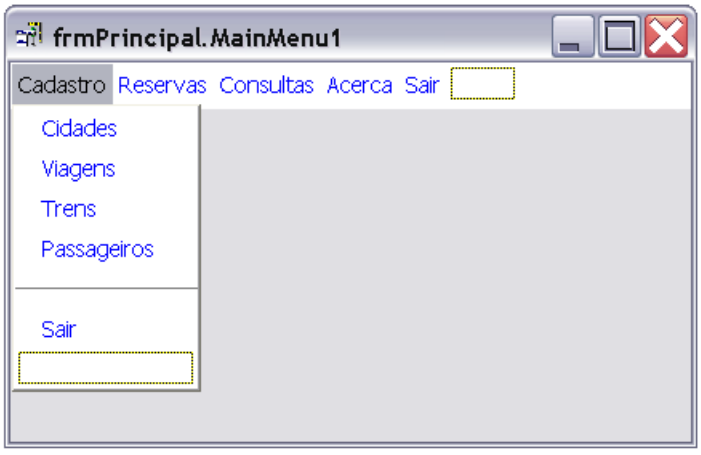
Será necessário criar formulários Windows para a manutenção desses cadastros e, para isso, seria interessante usar um mesmo padrão visual nos diversos formulários, um mesmo padrão de usabilidade e um mesmo padrão de armazenamento, recuperação e tratamento dos dados do sistema.

A seguir descrevemos o que se deseja para os formulários, sua identidade visual e seu funcionamento.

1. Programa Principal

O programa principal tem a seguinte tela e menu principal:





O componente imlBotoes possui as imagens que serão usadas para configurar os ToolButtons usados nos formulários descritos a seguir. Todos os formulários, menos o principal, deverão ser configurados como Available (não Auto-Create) no menu Options e serem instanciados (criados) sob demanda, ou seja, quando chamados pelo formulário principal, bem como liberados da memória quando forem fechados.

Quando ocorrer um clique em cada uma das opções do menu na figura acima, o sistema deverá instanciar o formulário associado à opção e colocá-lo em execução.

Para manter os registros de cada tipo de cadastro em vetores usaremos a unit uMantemVetor com as modificações que serão descritas oportunamente.

Cada tipo de cadastro deverá ser representado em uma unit separada, onde serão descritos o tipo de registro associado ao arquivo de cadastro e uma TEntidade específica, herança de TEntidade (declarada em uMantemVetor) capaz de encapsular um registro do tipo do cadastro.

Os formulários de cadastro deverão usar uma Toolbar com Toolbuttons para disponibilizar as operações de navegação, inclusão, exclusão, alteração, gravação dos registros do cadastro mantido em cada formulário, bem como as operações de pesquisa, cancelamento e impressão do cadastro.

Portanto, será necessário criar, na classe TMantemVetor, um método alterar(NovaEntidade:TEntidade; posicao:integer), que recebe um objeto encapsulado em TEntidade e o armazena no índice indicado pelo parâmetro posição, após testar a validade desse índice em relação ao tamanho do vetor dinâmico.

Destruir os formulários quando fechados

Para garantir que os formulários foram destruídos e os objetos que mantém os dados lidos e alterados tenham sido gravados e destruídos, de forma a serem lidos novamente quando os formulários forem novamente chamados pelo formulário principal, no evento onClose de cada um dos formulários de manutenção você deve executar o comando freeAndNil(<nome do formulário>) após gravar os dados dos vetores lidos. Exemplo:

```
procedure TfrmManPassageiro.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  uPassageiro.gravarDados('passageiros.dat');
  freeAndNil(frmManPassageiro);
end;
```

Passaremos agora a descrever os formulários de manutenção de cadastros.

2. Manutenção de Cidades

Começamos pelo formulário de manutenção de Cidades, que efetua a manutenção do arquivo de registros de Cidade.

Você deverá descrever uma unit descrevendo o registro TCidade e a classe TEntidadeCidade, que herda de TEntidade e armazena registros de TCidade num vetor de TEntidade mantido pela classe TMantemVetor. Essa unit também deverá ter a variável asCidades acessível a todas as outras units do programa (ou seja, será declarada na seção Interface) e os procedimentos lerDados e gravarDados que, respectivamente, lerão os dados do arquivo de cidades e os armazenarão em asCidades e percorrerão os dados armazenados nesse objeto para gravá-los no arquivo de cidades.

A descrição do registro TCidade vem abaixo:

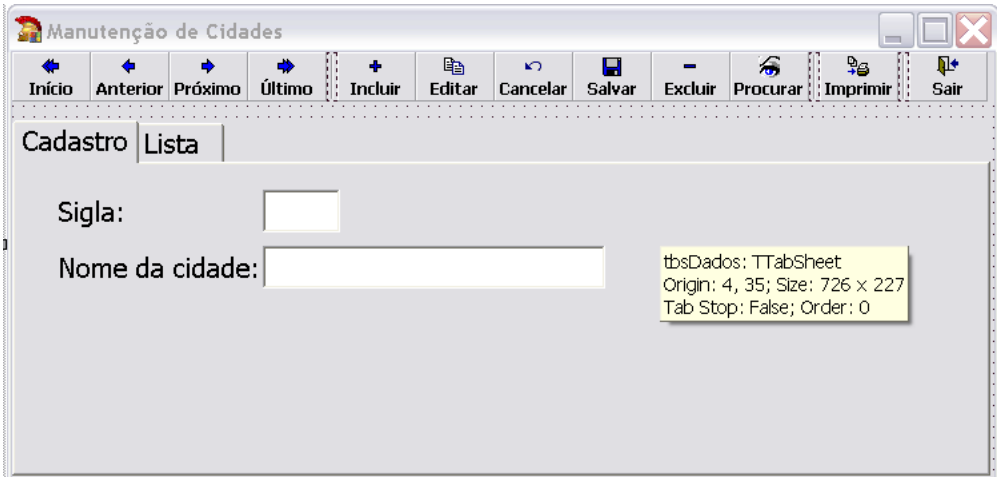
```
  TCidade = record
    siglaCidade : string[4];     // exemplos: CPS, GRU, BHZ, SPO, SBO, EFA
    nomeCidade  : string[30];
  end;
```

O nome do arquivo de cidades é “cidades.dat”.

A classe TEntidadeCidade deve ser herança de TEntidade, ter como atributo um campo **registro** do tipo TCidade, um construtor que recebe como parâmetro um novo TCidade que será encapsulado no campo registro durante a construção.

```
  TEntidadeCidade = class(TEntidade)  // herança!
    registro : TCidade;
    constructor create(novoRegistro:TCidade;
                       novaChave:string);
    function paraString : string;
  end;
```

Para criar este e os demais formulários de cadastro, devemos seguir as orientações da apostila de Técnicas de Programação, conforme desenvolvemos nosso último exercício no semestre passado. Quando o formulário for exibido, os arquivos que ele processa deverão ser lidos e armazenados em instâncias de TMantemVetor para que possam ser tratados. A seguir temos a aparência esperada desse formulário:



Quando se pressionar o botão [Incluir] deve-se dar início à digitação de um novo registro, conforme explicado na apostila.

Depois que se digitar a sigla da nova cidade, deve-se verificar, usando o método haDados do objeto TMantemVetor, se essa sigla ainda não está cadastrada, antes de permitir a digitação do nome da cidade. Se a sigla digitada já estiver armazenada, a inclusão deve ser cancelada e voltar a mostrar o registro que anteriormente aparecia na tela antes do pressionamento do botão [Incluir]. Se a sigla não for repetida, deve-se colocar o cursor no campo Nome da cidade e permitir sua digitação.

Para gravar o registro digitado, deve-se pressionar o botão [Salvar]. O evento OnClick desse botão fará as verificações necessárias e descritas na apostila e chamará o método de TMantemVetor que incluirá o novo registro na posição retornada como parâmetro da pesquisa binária.

O botão [Editar] permitirá que o registro atualmente exibido na tela seja alterado. Ao final de sua alteração, o usuário deverá pressiona o botão [Salvar], que fará as verificações necessárias e chamará o método alterar() de TMantemVetor.

O botão [Excluir] remove do objeto TMantemVetor de cidades o registro atualmente exibido na tela, através da chamada ao método de TMantemVetor que efetua essa remoção.

O botão [Cancelar] cancela a operação atual e recoloca o formulário em modo de navegação.

O botão [Procurar] fará com que o campo de sigla seja limpo e o cursor nele colocado. Em seguida, após a digitação da sigla procurada, será feita a chamada à pesquisa binária e, caso a sigla exista, o seu registro deverá ser exibido na tela e passar a ser o registro atual do objeto de manutenção de cidades.

O botão [Imprimir] deverá gerar um relatório, usando Rave Reports, sobre as cidades cadastradas. Esse relatório é uma lista simples, com a sigla e o nome de cada cidade, em ordem crescente de sigla.

Ou seja, você deverá pesquisar como se usa o Rave Reports e gerar um relatório das cidades.

Quando o usuário fechar este formulário ou clicar no botão [Sair], os dados deverão ser gravados no arquivo de cidades e retornar-se ao formulário principal.

3.Manutenção de viagens

O formulário de manutenção de Viagens efetua a manutenção do arquivo de registros de Viagem.

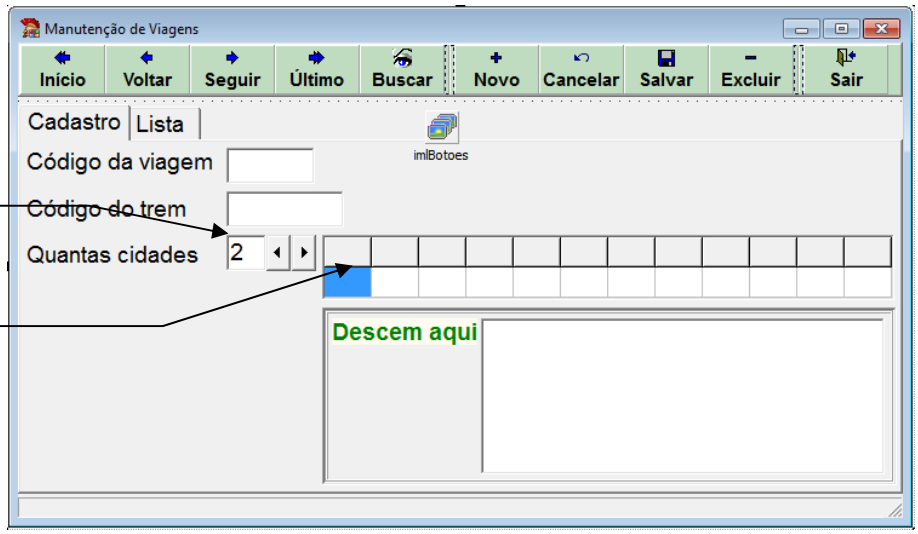
Você deverá descrever uma unit descrevendo o registro TViagem e a classe TEntidadeViagem, que herda de TEntidade e armazena registros de TViagem num vetor de TEntidade mantido pela classe TMantemVetor. Essa unit também deverá ter a variável asViagens acessível a todas as outras units do programa (ou seja, será declarada na seção Interface) e os procedimentos lerDados e gravarDados que, respectivamente, lerão os dados do arquivo de viagens e os armazenarão em asViagens e percorrerão os dados armazenados nesse objeto para gravá-los no arquivo de viagens.

A descrição do registro TViagem vem abaixo:

```
TViagem = record
  codigoViagem : String[4]; // chave primária
  codigoTrem   : string[6];
  qtasCidades  : byte;      // 1 a 12
  cidadesPorOndePassa : array[1..12] of string[4];
end;
```

Observe que TViagem guardará as **siglas** de **até** 12 cidades por onde a viagem poderá passar. O nome do arquivo de viagens é “viagens.dat”.

Ao lado temos a aparência básica do formulário de manutenção de viagens. Inclua botões para Editar, Pesquisar e Imprimir, conforme descrito para as cidades anteriormente.




```
edQtasCidades.maxLength=2
udQtasCidades.max = 12
```

```
stgCidade.DefaultColWidth=50
ColCount inicial = 12
```

No **momento da gravação de um novo registro** de viagem incluído, verifica-se se cada um dos códigos de cidades digitados no stringGrid stgCidade existe no arquivo de cidades.

Portanto, o arquivo de cidades também é lido e armazenado em um objeto TMantemVetor, que encapsula entidades TEntidadeCidade, baseado em TCidade, que também já foram criados.

A leitura desse arquivo no formulário de manutenção de viagens é feita no evento OnShow do formulário, e a variável asCidades (usada para armazenar os dados lidos das cidades) é uma variável declarada na unit de Manutenção de Cidades. Como usamos o mesmo modelo para os objetos de manutenção de vetores de registro, o objeto asCidades tem, também, o método haDados, que faz pesquisa binária do código da cidade no vetor de cidades lidas.

Quando for selecionado uma célula no stgCidade, são procurados, no vetor de passageiros, todos os passageiros que realizam a viagem apresentada na tela acima e que descem na cidade cujo código foi selecionado no stringGrid. O código e o nome desses passageiros são apresentados no lsbPassageiros.

4. Manutenção de Trens

Um trem é composto por até 10 vagões. Cada vagão possui 14 fileiras, com 4 poltronas em cada fileira. As fileiras são numeradas de 1 a 14, e as poltronas, de 'A' a 'D'.

Quando uma poltrona é reservada, ela é associada com o passageiro através do código do passageiro e do código da viagem que esse passageiro está fazendo.

Você deverá descrever uma unit descrevendo o registro TTrem e a classe TEntidadeTrem, que herda de TEntidade e armazena registros de TTrem num vetor de TEntidade mantido pela classe TMantemVetor

Essa unit também deverá ter a variável osTrens acessível a todas as outras units do programa (ou seja, será declarada na seção Interface) e os procedimentos lerDados e gravarDados que, respectivamente, lerão os dados do arquivo de viagens e os armazenarão em asViagens e percorrerão os dados armazenados nesse objeto para gravá-los no arquivo de viagens. O registro TTrem está descrito abaixo:

```
TTrem = record
  CodigoTrem : string[6]
  QtosVagoes : byte
  Vagoes     : array[1..10[ of TVagao;
end;
```

TVagao é um registro com os seguintes campos :

```
TVagao = record
  qtasPessoas : byte
  poltrona    : array[1..14, 'A'..'D' ] of
                  record
                    CodViagem   : string[5];
                    RG          : string[6]
                  end;
end;
```

Um trem terá até 10 vagões, e cada vagão poderá ter até 56 pessoas viajando em viagens diferentes. O registro de um trem, portanto, tem a aparência abaixo:

CodTrem	QtosVagoes	Vagoes																													
		1								2								.	10												
			1	2	3	4	5	...	13	14		1	2	3	4	5	...	13	14	.		1	2	3	4	...	13	14			
		A						...			A						...			.	A					...					
		B						...			B						...			.	B					...					
		C						...			C						...				C					...					
		D						...			D						...				D					...					

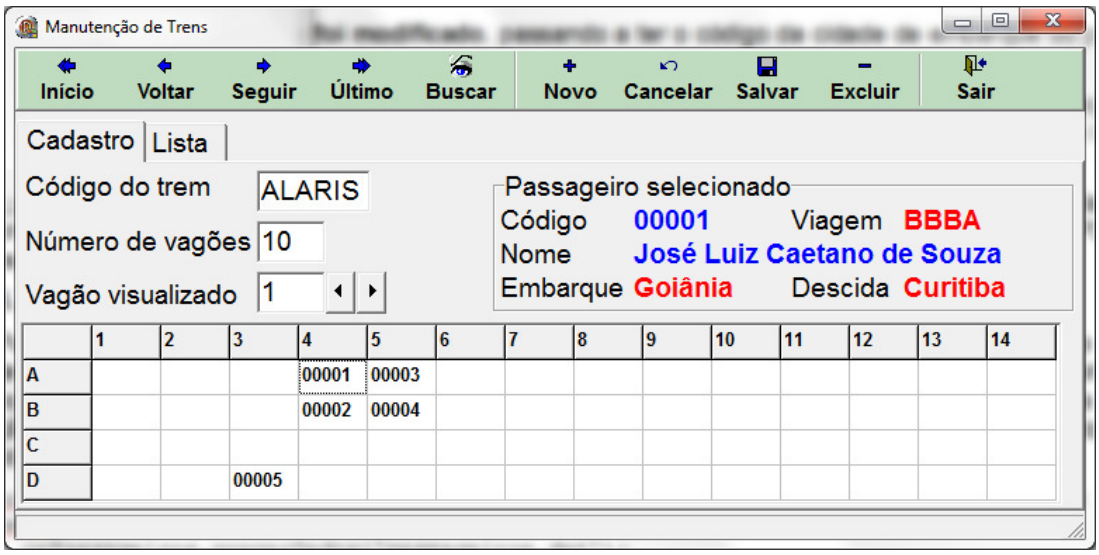
O arquivo de passageiros possui a sigla da cidade de embarque do passageiro (cidadeSobe) e a sigla da cidade de descida (cidadeDesce).

Crie a unit que descreve o registro de trens, sua TEntidade e os procedimentos de leitura e gravação dos dados; declare também a variável osTrens, que armazenará os dados de trens lidos e incluídos e será acessada pelas demais units do sistema.

O formulário tem sua aparência básica apresentada na figura abaixo:

Observe que o passageiro de código 00001 reservou a poltrona 4A do vagão 1, e que os nomes das cidades onde ele sobe e desce são apresentados (e não os códigos dessas cidades, que seriam GOI e CBA, respectivamente).

Inclua os botões [Editar], [Pesquisar] e [Imprimir] e codifique o evento onClick de cada um.



5. Manutenção de Passageiros

Siga os modelos descritos anteriormente para realizar o cadastro de passageiros. O registro de passageiros tem a descrição abaixo:

```
TPassageiro = Record
  codPassageiro : String[5]
  Nome          : String[30]
  codViagem     : string[5]
  CidadeSobe    : string[4]
  CidadeDesce   : string[4]
  Vagao         : byte (1 a 10)
  fileira       : byte (1 a 20)
  poltrona      : char ('A' a 'D')
end;
```

6. Reservas de passagens

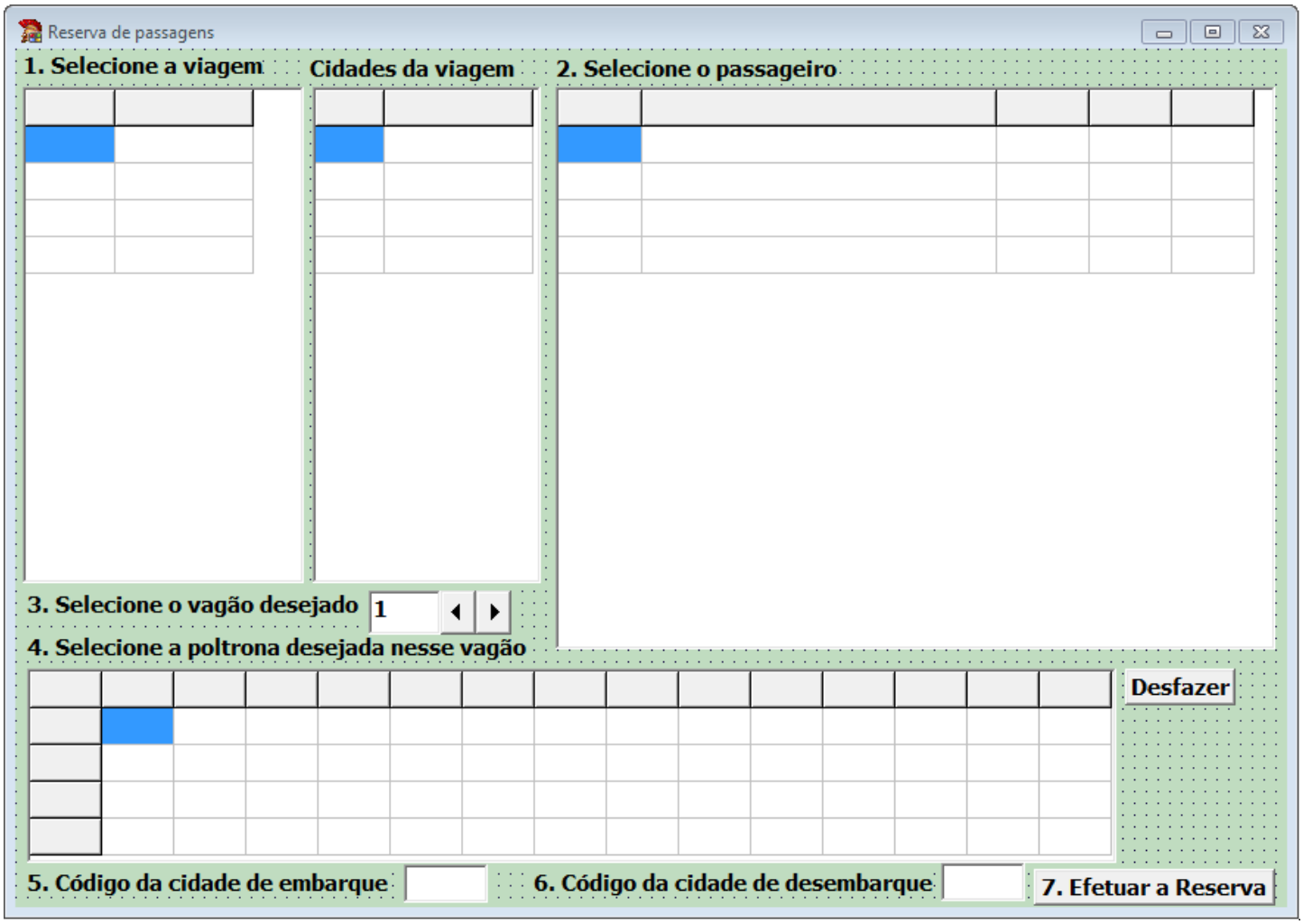
Resta fazermos a reserva de passagens. Para isso, serão necessários todos os arquivos que criamos anteriormente, bem como as classes de manutenção dos mesmos.

Uma maneira de facilitar a obtenção dos dados para efetuar a reserva é através de stringGrids que mostrem os registros de passageiros, de viagens, de cidades da viagem selecionada e os vagões do trem que faz essa viagem, como vemos na próxima figura.

Note que os labels do formulário estão numerados, indicando a ordem em que as informações devem ser fornecidas. Observe também que a última operação é, justamente, o botão que efetua a reserva, que será acionado pelo usuário quando este já tiver fornecido os dados necessários para concretizar uma reserva.

Você receberá este formulário pronto, com a declaração das variáveis principais, a leitura dos arquivos necessários, armazenando-os em objetos adequados para sua manutenção e a apresentação dos dados desses arquivos nos stringGrids.

Você deverá programar os eventos necessários para a efetivação de reservas e que são descritos abaixo.



Quando o formulário for criado, são instanciados os objetos de manutenção dos registros e lidos os arquivos correspondentes.

Após isso, os dados de viagens e de passageiros são apresentados nos stringGrids stgViagem e stgPassageiro. Note o uso dos métodos iniciarPercursoSequencial e podePercorrer, que permitem o percurso de todos os registros armazenados no objeto de manutenção, de forma que eles podem ser acessados e apresentados nos controles do formulário.

Itens a serem programados

1. Quando uma viagem for selecionada no stgViagem, as cidades por onde a viagem passa devem ser apresentadas no stringGrid stgCidade. Também devem ser exibidas as poltronas do primeiro vagão do trem que faz a viagem.

2. Quando um passageiro for selecionado, deve-se verificar se ele não tem uma viagem já reservada, através do campo codViagem desse passageiro. Caso esse campo seja diferente de '0000', deve-se emitir uma mensagem para o usuário, através de um showMessage, informando que o passageiro não pode fazer nova reserva.

3. Quando o upDown udVagao for clicado, deve-se exibir as poltronas do vagão indicado pela propriedade Position do udVagao.

4. Quando uma célula do stringGrid stgVagao for selecionada, deve-se verificar se essa célula contém a cadeia '00000' ou cadeia vazia, indicando que a poltrona correspondente à célula está disponível para reserva. Se não estiver, deve-se apresentar mensagem de advertência ao usuário, através de um showMessage.

5. Quando o cursor sair do edCidadeSobe, deve-se verificar se o texto digitado nesse componente é um dos códigos de cidade por onde a viagem passa. Se não for, deve-se limpar esse componente e nele colocar o cursor para nova digitação.

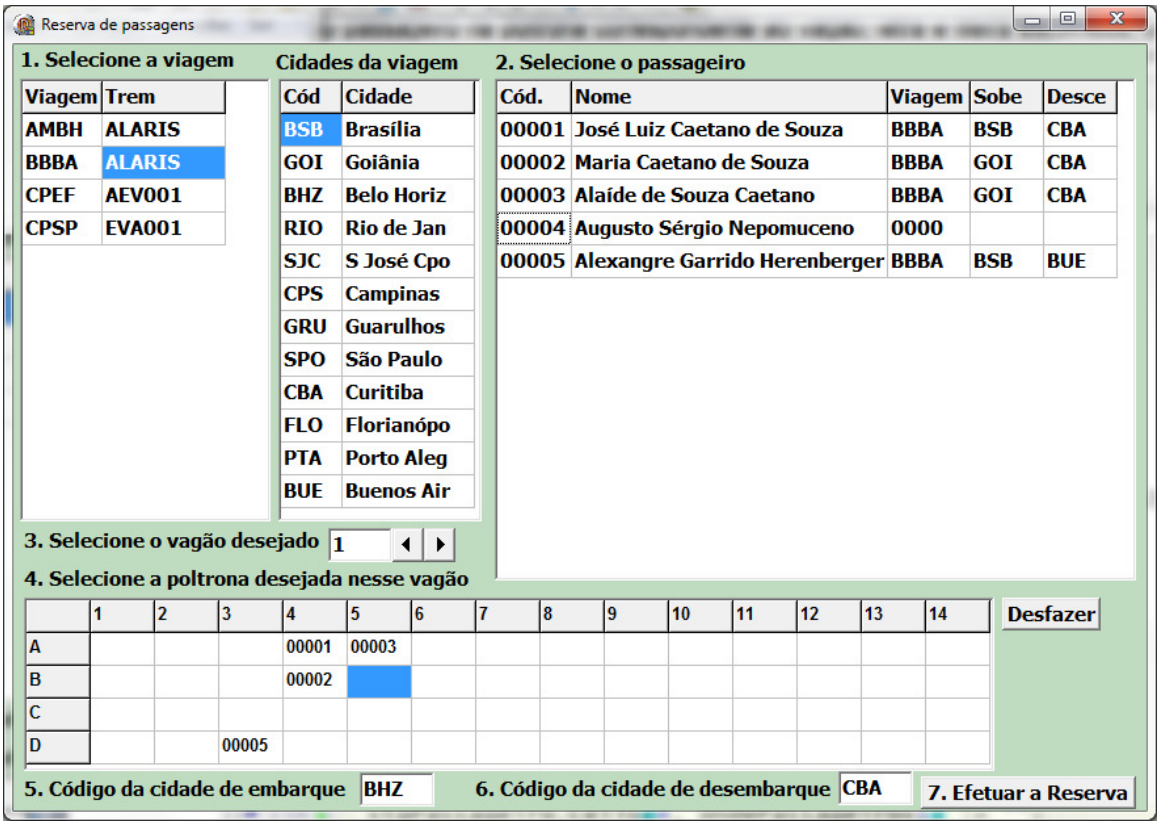
6. Quando o cursor sair do edCidadeDesce, deve-se verificar se o texto digitado nesse componente é um dos códigos de cidade por onde a viagem passa. Se não for, deve-se limpar esse componente e nele colocar o cursor para nova digitação.

7. Quando o botão btnReserva for clicado, deve-se efetuar a reserva. Para isso, deve-se garantir que a célula selecionada no stgVagao é igual a '00000' ou cadeia vazia, que os textos dos TEdit edCidadeSobe e edCidadeDesce são diferentes de cadeia vazia. No registro de passageiro, deve-se copiar o código da viagem selecionada, o número do vagão selecionado, a fileira e a letra da poltrona escolhida, bem como os códigos das cidades de embarque e de desembarque. Em seguida, atualiza-se o registro de passageiro no objeto de manutenção de passageiros. Após isso, no registro do trem que faz a viagem escolhida, deve-se colocar o código do passageiro na poltrona correspondente ao vagão, letra e fileira escolhidos, bem como atualizar esse registro no objeto de manutenção de trens. Após isso, execute o método que atualiza a tela para que as modificações sejam visíveis.

8. Quando o botão btnDesfazer for clicado, deve-se desfazer a reserva do passageiro está selecionado no stgVagao, colocando '0000' no codViagem desse passageiro, limpando os campos cidadeSobe, cidadeDesce, vagao, fileira e poltrona, atualizando esse registro. Deve-se também limpar a célula do stringGrid e a poltrona correspondente a esse vagão e poltrona do trem onde a viagem é feita, atualizando o registro do trem.

9. Quando o formulário for fechado, deve-se gravar os dados dos objetos de passageiros e de trens, para que as reservas efetuadas sejam gravadas em disco.

É possível visualizar e verificar se as reservas foram efetuadas corretamente, usando os formulários de manutenção de passageiros e de trens.



7. Ordenações para listagens em ordens alternativas

A classe TMantemVetor já possui um método **ordenar()**, que ordena crescentemente o vetor **dados** (array of TEntidade), sendo o atributo **Chave** usado para determinar a ordem entre os registros armazenados nesse vetor.

Mas imagine que fosse necessário ordenar os registros armazenados por outros campos, de acordo com as necessidades de cada tipo de entidade e aplicação que usam a classe TMantemVetor.

Por exemplo, na manutenção de passageiros poderá ser preciso listar os dados em ordem de nome ou departamento.

Para fazer isso, a classe TMantemVetor **deverá ter** um método chamado ordenacaoAlternativa, cujo objetivo é permitir que se ordene o vetor **dados** pelo campo desejado, não apenas pelo campo Chave.

No entanto, a classe TMantemVetor não tem como prever, para cada aplicação específica, quais campos serão usados para as ordenações alternativas. Portanto, cada **aplicação** deverá fornecer uma **função que compare os campos** que ela deseja, e essa **função será passada como parâmetro** para o método ordenacaoAlternativa. Essa função receberá como parâmetros os índices Lento e Rapido que serão usados para acessar as posições do vetor dados onde estão os registros a serem comparados na aplicação.

Para passar uma função como parâmetro para o método ordenacaoAlternativa, deve-se declarar, na unit **uClasseMantemVetor**, um **novo tipo** que descreva a função, como se vê abaixo:

```
TFuncaoDeOrdenacao = function (lento, rapido:integer):integer;
```

Essa classe usará esse tipo para declarar o **novo** método **ordenacaoAlternativa()**, como abaixo:

```
procedure ordenacaoAlternativa(comparar : TFuncaoDeOrdenacao);
```

Essa declaração acima informa que o método **ordenacaoAlternativa** recebe como parâmetro uma função **comparar()** que, por ser do tipo **TFuncaoDeOrdenacao**, terá 2 parâmetros inteiros e devolverá um valor inteiro, pois essa estrutura de parämetros é a que está definida no tipo TFuncaoDeOrdenacao.

A classe **TMantemVetor** implementará o método **ordenacaoAlternativa()**. Esse método deverá variar os índices Lento e Rapido como já é feito no método ordenar() existente (onde se ordena o vetor **dados** pelo campo **chave**), e trocar os elementos que estejam fora de ordem, como também já era feito anteriormente no método ordenar().

Para determinar se os elementos do vetor estão fora de ordem e devem ser trocados, o novo método **ordenacaoAlternativa()** deverá chamar a função **comparar()**, que foi recebida como parâmetro. Ao chamar essa função, deverá passar Lento e Rapido como parâmetro, como vemos abaixo:

```
Procedure TVetor.ordenacaoAlternativa(comparar : TFuncaoDeOrdenacao);
... variação de Lento e Rapido ...
        if comparar(lento, rapido) > 0 then
              faz a troca entre os elementos Lento e Rapido do vetor dados
```

A função **comparar(),** chamada por **ordenacaoAlternativa(),** não poderá ser implementada diretamente na classe TMantemVetor, pois essa classe também é genérica e não sabe quais os campos que serão usados em cada ordenação alternativa. Portanto, será cada **APLICAÇÃO** específica que **implementará as funções de comparação** entre os campos usados para as ordenações alternativas que cada aplicação precisar realizar.

Por exemplo, no caso da **classe de manutenção de passageiros**, caso ela necessite que os dados armazenados no vetor de passageiros sejam ordenados pelo campo nome, ela deverá implementar a função **compararNomes(Lento, Rapido)**, que devolve um valor inteiro. Se o nome da posição lento for menor que o nome da posição Rapido, essa função devolverá -1, se forem nomes iguais a função devolverá 0 e se o nome da posição lento for maior que o título da posição Rapido, essa função devolverá 1, de forma muito semelhante à função **compararCom() da classe TEntidade**, descrita na apostila de Técnicas de Programação.

Quando for necessário ordenar os dados por **nome de passageiro**, a aplicação de manutenção de passageiros chamará o método **ordenacaoAlternativa()** do objeto **osPassageiros**, passando como parâmetro o **nome da função** que faz a comparação entre nomes de professores. Essa função será

associada ao parâmetro **comparar** dentro da classe TVetor e será chamada pelo comando **if** do quadro acima , sempre que for necessário comparar dois departamentos. O código da chamada para ordenação por nomes de disciplinas seria como:

```
osPassageiros.ordenacaoAlternativa(compararNomes);
```

Quando a aplicação necessitar ordenar dados por cidade de subida, deverá implementar uma função **compararCidadeSobe()**, que recebe como parâmetros dois inteiros (Lento e Rapido) devolve -1 se a cidade de subida armazenada em Dados[Lento] for menor que a cidade de subida armazenada em Dados[Rapido], devolve 0 se forem iguais e devolve 1 se a cidade de subida armazenada em Dados[Lento] for maior que a cidade de subida armazenada em Dados[Rapido]. A chamada do método **ordenacaoAlternativa()**, nesse caso, seria como:

```
osPassageiros.ordenacaoAlternativa(compararCidadeSobe);
```

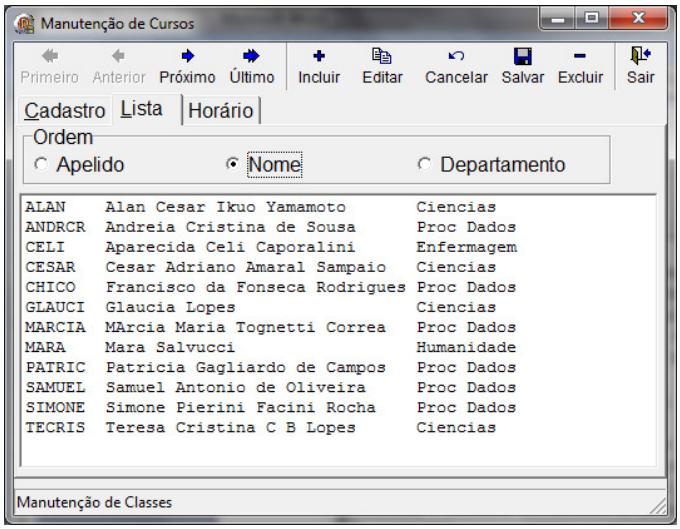
Assim, para ter essas ordenações, a aplicação deverá implementar as funções abaixo:

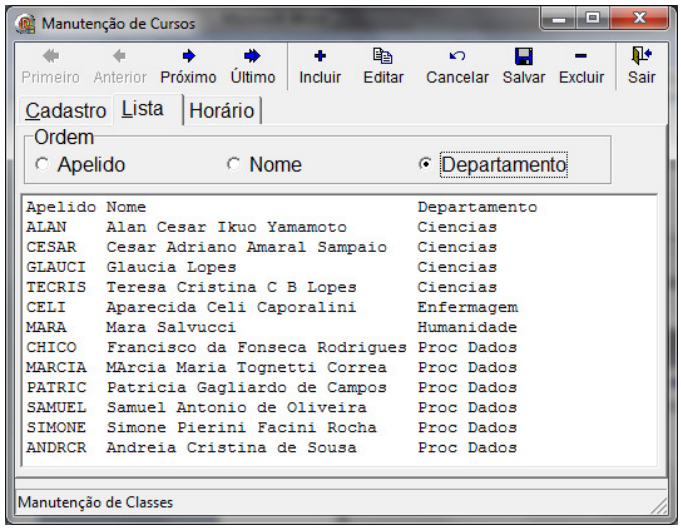
```
Function compararNomes(Lento, Rapido:integer):integer;      // devolve –1, 0 ou 1
```
e
```
Function compararCidadeSobe(Lento, Rapido:integer):integer;// devolve –1, 0 ou 1
```

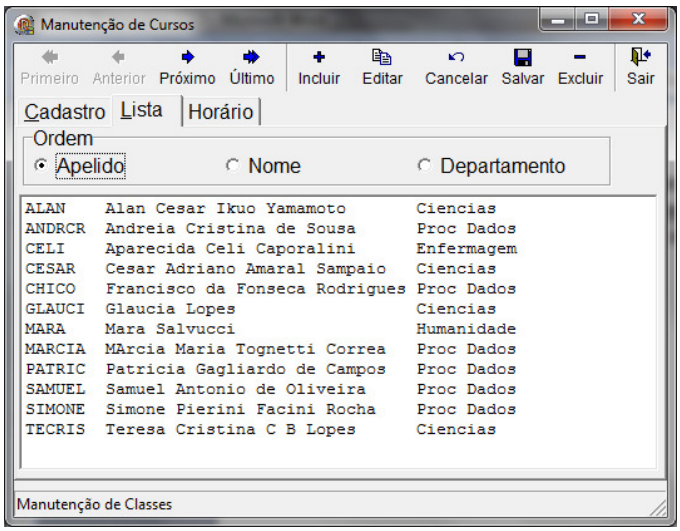
Essas funções devolverão -1, 0 e 1, conforme descrito acima, e serão passadas como parâmetro para o método **ordenacaoAlternativa()** quando for necessário ordenar os dados de passageiros por nomes ou por cidades.

Usando essa estratégia, podemos ter um único método de ordenação para qualquer tipo de registro que seja encapsulado na classe TMantemVetor. Poderíamos ter, na manutenção de cada tipo de registro, funções de comparação dos principais campos, que determinariam a ordem entre registros tendo como base o campo.

Levando em conta a descrição da técnica acima, implemente as alterações descritas na classe TMantemVetor. Na aba [Lista] dos formulários de manutenção, programe o evento onClick do radioGroup para ordenar e listar professores de acordo com o indicado no botão, apresentando os dados no memo, como vemos nas figuras de exemplo (que não tem relação com o projeto de reserva de passagens de trens).







Este projeto foi baseado no site de venda de passagens de trens da Espanha, indicado abaixo:

www.renfe.es

Outro site de venda de passagens, dessa vez na Itália, segue abaixo:

www.trenitalia.it

NoBrasil, infelizmente essa é uma realidade que terá de esperar pelo "Trem Bala" ou "Trem de Alta Velocidade"que sabe-se lá se realmente existirá. Enquanto isso, você pode ir fazendo seu projeto e viajar um pouco no site abaixo:

http://www.dicaseturismo.com.br/passeios-de-trem-pelo-brasil/#