PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Informática e Ciências Exatas - Curso de Sistemas de Informação
Laboratório de Programação Orientada por Objeto - Professor: Paulo Amaral

RELATÓRIO 4 (2/2015) - TRABALHO EM GRUPO

FORMULÁRIOS, ASSOCIAÇÕES e TRATAMENTO DE EXCEÇÕES

1. ESPECIFICAÇÕES

Deve ser **entregue** um texto com o **relatório completo** do trabalho que conste na capa (1ª. página) os **nomes completos de todos os componentes do grupo**. O número de componentes do grupo será especificado pelo professor durante a aula, não sendo aceitos trabalhos anônimos.

No **relatório** deve constar tudo que foi especificado no exercício e os c**ódigos-fonte de todos os programas** envolvidos nos exercícios**.**

Todos programas devem **imprimir o nome completo de todos os componentes do grupo na tela** e os **arquivos fontes dos programas, além de serem incluídos no texto do relatório, devem ser entregues em arquivos separados no SGA**, devendo conter o **cabeçalho completo** com entrada, saída, descrição e nome completo de todos os componentes do grupo (não sendo aceitos trabalhos sem os nomes. **Cópias grosseiras** serão desconsideradas, ou seja, a **nota será igual a 0 (zero).**

A lista de exercícios tem partes que devem ser lidas (**Leitura**), outras (**Prática)** que devem ser exercitadas em casa e/ou nos laboratórios de informática e finalmente alguns **exercícios** (**Entregar) que devem ser entregues com o relatório.**

Haverá apresentação oral dos trabalhos

2. Interface gráfica de usuário (GUI) com Windows Forms do Visual Studio

Formulários são usados para o desenvolvimento de aplicativos usando a interface gráfica do Microsoft Windows.

Um **Formulário** é uma superfície visual na qual pode-se exibir informações para o usuário. Aplicativos de Formulários do Windows podem colocar controles em formulários e desenvolvendo respostas para ações do usuário, tais como cliques do mouse ou pressionamentos de teclas.

Um formulário é criado incialmente como um espaço em branco onde podem ser inseridos controles para criar uma interface do usuário com código para manipular dados. Para esse fim, o Visual Studio fornece um ambiente de desenvolvimento integrado (Integrated Development Environment - IDE) para ajudar na escrita de códigos, bem como um rico conjunto de controles escrito com o

.NET Framework. Complementando a funcionalidade desses controles com o código adequado, pode-se facilmente e rapidamente desenvolver as soluções.

Quando um usuário faz algo em seu formulário ou em um de seus controles, ele gera um evento. Seu aplicativo reage a esses eventos usando código e processa os eventos quando eles ocorrem.

2.1 Controles do Windows Forms

Um **controle** é um elemento discreto da interface do usuário (IU) que exibe dados ou aceita entrada de dados.

Windows Forms contém uma variedade de controles que podem ser adicionados nos formulários, entre os quais pode-se citar: Exibir caixas de texto, botões, caixas suspensas, botões de opção e mesmo páginas da Web. Se um controle existente não as necessidades do usuário, os Formulários do Windows também oferecem suporte à criação de seus próprios controles personalizados através da classe UserControl.

Windows Forms tem controles de UI que emulam recursos em aplicativos avançados como o Microsoft Office. Com os

Para exemplificar é exibida, a seguir, uma lista de alguns os controles disponíveis no Windows Forms.

Button: inicia, para ou interrompe um processo.

CheckBox: Exibe uma caixa de seleção e um rótulo de texto. Geralmente usado para definir opções

CheckedListBox: Exibe uma lista rolável de itens, acompanha cada por uma caixa de seleção.

ComboBox: Exibe uma lista drop-down de itens.

Label: exibe um texto, não sendo possível editar diretamente.

ListBox: exibe uma lista de texto e itens gráficos (ícones).

ListView: exibe itens em um dos quatro modos diferentes. Modos de exibição incluem somente texto, texto com pequenos ícones, texto com ícones grandes e um modo de exibição de detalhes.

RadioButton: exibe um botão que pode ser ativado ou desativado.

TextBox: exibe o texto inserido em tempo de design que pode ser editado por usuários em tempo de execução ou alterado programaticamente.

ToolStrip e MenuStrip: pode-se criar barras de ferramentas e menus que contêm texto e imagens, exibir submenus e hospedar outros controles como caixas de texto e caixas suspensas.

O namespace **System.Drawing** permite criar elementos da IU personalizados e contém uma grande seleção de classes para processar linhas, círculos e outras formas diretamente em um formulário.

O Visual Studio tem o Windows Forms Designer, onde através de ações tipo arrastar-e-soltar, pode-se criar facilmente aplicativos de Formulários do Windows. Basta selecionar os controles com o cursor e adicioná-los no local desejado no formulário. O designer fornece ferramentas, como linhas de grade e snap lines para facilitar o trabalho de alinhar controles.

2.2 Exemplos

Exemplo1 (prática):

Criar um aplicativo do Windows Forms a partir da linha de comando (código fonte no SGA)

O procedimento a seguir descreve os passos básicos para criar e executar um aplicativo de Formulários do Windows a partir da linha de comando.

1-Inclua no código as diretivas ***using*** para especificar as bibliotecas que o programa pode utilizar.

```
using System;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
```

A diretiva *using System* especifica que o programa pode utilizar a biblioteca no namespace *System*, ou seja, evitam a necessidade de especificar o nome para cada classe

2-Declare a classe **Form1** que herda da classe Form.

public class **Form1 : Form**

3-Crie um construtor padrão para Form1, incluindo o nome do formulário: “Formulario 1".

```
public Form1( ) {
        this.Text = "Formulario 1"; // nome do formulário
}
```

4- Adicione o método Main à classe.

```
[STAThread] // especifica que o aplicativo utiliza um único thread.

public static void Main(){
  Application.EnableVisualStyles(); // habilita estilos visuais: cores, fontes, etc
  Application.Run(new Form1());  // Cria uma instância do formulário e executa.

}
```

5- Compilar e executar o programa:
csc.exe Form1.cs
Form1.exe

Exemplo2 (prática):
Criar um aplicativo do Windows Forms na IDE do Visual Studio

Para criar uma aplicação usando IDE do Visual Studio C#, deve-se selecionar **File -> New->Project** e será aberta a janela de novo projeto. Nessa janela escolhe-se tipo **Windows Form Application**. O novo projeto pode ser denominado de **Form2.**

Selecione o menu **View**, escolha a opção **Toolbox** e será apresentada, ao lado esquerdo da tela, a janela **Toolbox.**

Neste aplicativo serão inseridos três controles: um para mecanismo de entrada de um nome (**TextBox)**, um de ação (**Button)** e outro para apresentar o nome informado (**Label).**

Na lista de ferramentas **Toolbox**, mantenha selecionada a guia **Common Controls** para que seja apresentado o conjunto completo dos controles visuais. Selecione o controle **TextBox**, o qual será usado para a entrada de um determinado dado. Para selecioná-lo basta um clique no controle desejado e levá-lo para dentro do formulário, o qual será fixado por meio de um clique dentro da área pretendida. Insira o controle, arrastando-o para o canto superior esquerdo. Agora insira um controle tipo **Button** abaixo do controle anterior e finalmente insira um controle tipo **Label.**

Para terminar a construção do programa, dê um duplo clique no controle **Button1** do formulário ativo. Na janela do código apresentado, escreva como linha código a seguinte instrução:

label1.Text = textBox1.Text no método button1_Click(object sender, EventArgs e).

Observe que o conteúdo do controle tipo **Label1** vai mostrar o conteúdo digitado e armazenado no controle **TextBox1.**

Para salvar o programa: **FILE->Save All**.

Para compilar e executar o programa: **DEBUG->Start Debugging**

OBS:
O código fonte de uma versão do aplicativo gerado a partir da linha de comando está disponível no SGA.

3. EXERCÍCIOS PARA ENTREGAR (Exercícios para entregar)

Utilizando as especificações dos exercícios do LAB1 escrever programas em C#, utilizando formulários, para:

3.1 Converter temperaturas de Celsius para Fahrenheit e de Fahrenheit para Celsius.
3.2 Implementar uma calculadora que efetue as **operações aritméticas básicas** (+, -, x e /) com números reais.
3.3 Implementar a classe Conta com as mesmas especificações do exercício da seção 3.9 do LAB1.

4. Associação de classes

Objetos de classes diferentes podem se relacionar através de troca de mensagens, mas em algumas situações existem relacionamentos mais fortes.

Os principais tipos de associação entre classes são:

- **Associação simples**: relação *semântica estrutural* entre classes. Ex: uma Pessoa trabalha para uma Companhia, uma Companhia tem vários Escritórios, etc.
- **Agregação**: relação *todo-parte* entre classes, sendo que a parte pode existir sem o todo. Ex: Carro e Roda. Uma Roda é parte de um Carro, porém pode a Roda existir por si só fora do Carro, como por exemplo, remover a roda de um carro para colocar em outro.
- **Composição**: relação *todo-parte* entre classes, sendo que a parte NÃO existe sem o todo. Ex: Pedido e Itens de Pedido. Se destruir o Pedido, os Itens são também destruídos junto, eles não tem sentido se não houver um Pedido.

4.1 Associações simples

Nas **associações simples** os objetos são associados, mas não há relação de *pertinência*.

Exemplos: uma pessoa jantou em um restaurante, um aluno cursou diversas disciplinas, vários alunos podem estar associados à um único professor e um único aluno pode estar associado à vários professores.

Nestes casos, não existe um relacionamento de posse entre esses objetos. Todos os objetos são independentes. Um aluno pode existir sem a necessidade de um professor, da mesma forma que é possível existir um professor sem a necessidade da existência de um aluno.

4.2 Agregação

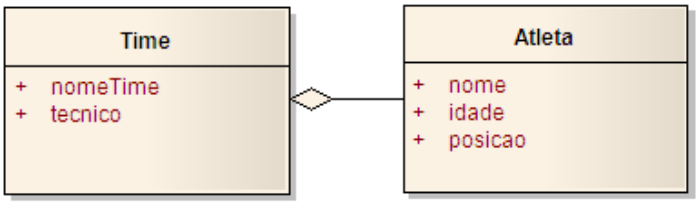
Na **agregação** representa-se um todo que é composto de várias partes. Exemplo: um conselho é um agregado de membros, da mesma forma que uma reunião é um agregado de uma pauta, uma sala e participantes.

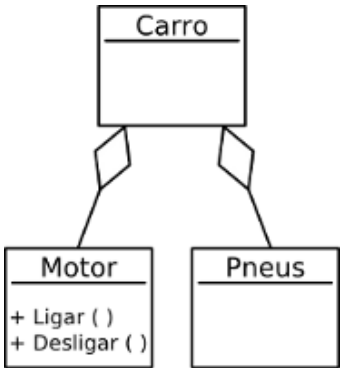
A implementação deste relacionamento não é de posse, pois uma reunião não CONTÉM uma sala.

As partes da agregação podem fazer outras coisas em outras partes da aplicação, podem ser referenciados por outros objetos e não somente por um objeto.

Em UML, a agregação é representada por uma linha com um losango vazio do lado da classe que manda no relacionamento.

Exemplos de agregação:





4.3 Composição

A composição, diferentemente da agregação, é um relacionamento de "posse".

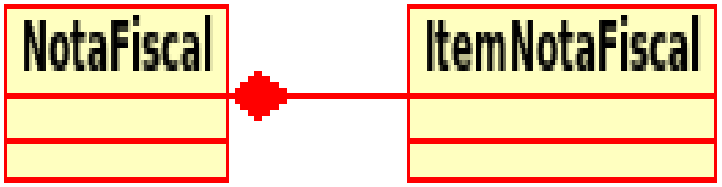
Um objeto (container) CONTÉM outros objetos (elementos) e estes elementos que estão contidos dentro de outro objeto e dependem dele para existir. Eles são criados e destruídos de acordo com o seu container.

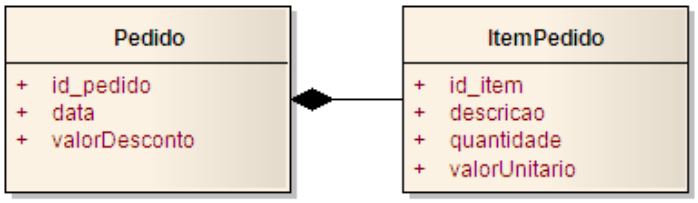
A composição, na UML, é representada por uma linha com um losango preenchido do lado da classe dona do relacionamento

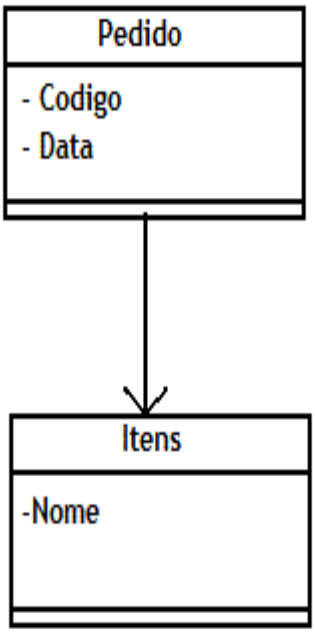
Exemplos de composição:

Um exemplo de container poderia ser uma nota fiscal, e seus elementos seriam seus itens.
Não faz sentido existirem itens de nota fiscal sem existir uma nota fiscal onde tais itens estariam contidos.

Eles só existem se existir uma nota fiscal da qual eles fazem parte. Se a nota fiscal é destruída, todos os seus itens também são, o que não acontece com a agregação, onde, se uma reunião é destruída, seus participantes continuam existindo, pois podem participar de outras reuniões.







Pedido - Classe que contém uma instância da classe **Itens;**

Um pedido **TEM UM** Item;

```
public class Pedido {
    private Itens i;
    public Pedido( ) {
            i = new itens();
       }
}
```

```
public class Itens {
    public Itens( ){   }
}
```

5. EXERCÍCIOS PARA ENTREGAR (Exercícios para entregar)

Ex5.1 (exercício para entregar): Implemente uma classe Veículo. O veículo é composto por várias partes: um motor, um tanque de combustível e 4 pneus. O funcionamento do veículo depende das suas partes, da seguinte forma:

Motor

- Possui uma potência (em hp), uma taxa fixa de consumo (em km/litro) e um tanque de combustível (ver abaixo).
- Possui um comando para avançar uma determinada quantidade de km. Se a pressão de mais de um pneu (ver abaixo) estiver abaixo de 20 lb, o consumo do veículo aumenta em 30%. Se não houver combustível suficiente, o veículo avança até o tanque esvaziar.

Tanque de Combustível

- Possui uma determinada capacidade e quantidade atual de combustível (ambos em litros).
- Pode ser abastecido com uma certa quantidade de combustível, limitado à sua capacidade máxima.

Pneu

- Cada pneu possui uma determina pressão (em lb).
- Pode ser calibrado com determinada pressão informada (positiva ou negativa, sendo somada à atual).

Implemente as classes acima, usando composição quando necessário. Modularize o código. Lembre-se de implementar gets e sets necessários, bem como construtores adequados.

No programa principal, faça as seguintes operações:

- Instancie um veículo cujo motor tem 71 hp, consumo de 12 km/litro, tanque com capacidade para 50 litros, pneus dianteiros com 27 lb e traseiros com 23 lb.
- Abasteça o tanque com 30 litros.
- Exiba na tela as informações sobre cada componente do veículo.
- Avance 300 km.
- Reduza a pressão do pneu traseiro esquerdo para 17 lb.
- Avance 100 km.
- Reduza a pressão do pneu dianteiro direito para 18 lb.
- Abasteça mais 10 litros.
- Avance 200 km.

A cada operação de movimento, mostre na tela a distância percorrida e a quantidade de combustível restante.

Ex5.2 (exercício para entregar): Crie a classe **ContaDePoupanca**. Use a variável **static taxaDeJurosAnual** para armazenar a taxa de juros de todos os correntistas. Cada objeto da classe contém uma variável de instância **private saldoPoupanca**, indicando a quantidade que o poupador possui atualmente depositada. Forneça o método **CalcularJuroMensal** para calcular os juros mensais, multiplicando **saldoPoupanca** por **taxaDeJurosAnual**, dividindo por 12; esses juros

devem ser acrescidos a **saldoPoupanca**. Forneça um método **static AlterarTaxaDeJuros** que configure **taxaDeJurosAnual** para um novo valor. Escreva um programa driver para testar a classe **ContaDePoupanca**. Instancie dois objetos **ContaDePoupanca**, **poupador1** e **poupador2**, com saldos de R$2000,00 e R$3000,00, respectivamente. Configure **taxaDeJurosAnual** como 4% e, em seguida, calcule os juros mensais e imprima os novos saldos para cada um dos poupadores. Depois, configure **taxaDeJurosAnual** como 5%, calcule os juros do próximo mês e imprima os novos saldos para cada um dos poupadores.

Ex5.3 (exercício para entregar): Sistema Bancário

a) No sistema de um banco algumas informações dos clientes como nome e código precisam ser armazenadas. Crie uma classe para modelar os objetos que representarão os clientes.

```
class Cliente {
        String nome;
        int codigo;
}
```

b) Os bancos oferecem aos clientes a possibilidade de obter um cartão de crédito que pode ser utilizados para fazer compras. Um cartão de crédito possui um número e uma data de validade. Crie uma classe para modelar os objetos que representarão os cartões de crédito.

```
class CartaoDeCredito {
        int numero;
        String dataDeValidade;
}
```

c) Defina um vínculo entre a classe Cliente e a classe CartaoDeCredito. Para isso deve-se alterar a classe CartaoDeCredito.

```
class CartaoDeCredito {
        int numero;
        String dataDeValidade;
        // Adicione a linha abaixo
        Cliente cliente;
}
```

d) Teste o relacionamento entre clientes e cartões de crédito. Escreva o código a seguir no método *main()*.

```
// Criar alguns objetos
Cliente c = new Cliente();
CartaoDeCredito cdc = new CartaoDeCredito();

// Carregar alguns dados
c.nome = "José da Silva";
cdc.numero = 111111;

// Ligar os objetos (liga o objeto CartãodeCredito cdc ao objeto Cliente c)
cdc.cliente = c;
```

e) As agências bancárias possuem um número. Crie uma classe para modelar as agências.

```
class Agencia {
        int numero;
}
```

f) As contas do banco possuem saldo e estão vinculadas a uma agência. Crie uma classe para modelar as contas e estabeleça um vínculo com a classe AGENCIA.

```
class Conta {
        double saldo;
        Agencia agencia;
}
```

g) Teste o relacionamento entre contas e agências. Escreva o código abaixo no método *main()*.

```
// Criar alguns objetos

Agencia a = new Agencia();

Conta c = new Conta();

// Carregar alguns dados

a.numero = 178;

c.saldo = 1000.0;

// Ligar os objetos (liga o objeto Conta c ao objeto Agencia a)

c.agencia = a;
```

h) Acrescente alguns métodos na classe CONTA para definir as lógicas de depositar, sacar e imprimir extrato.

```
class Conta {
        double saldo;
        Agencia agencia;
        void deposita(double valor) { }
        void saca(double valor) { }
        void imprimeExtrato( ) { }
}
```

i) Testar os métodos da classe CONTA. Escrever o código abaixo no método *main()*.

```
c.deposita(1000);
c.imprimeExtrato();
c.saca(100);
c.imprimeExtrato();
```

j) Criar uma classe para modelar as faturas dos cartões de crédito.

```
class Fatura {
        double total;
        // Adiciona o valor ao total da fatura
        void adiciona(double valor) { }
        // Calcula multa de 2% sobre o total da fatura
        double calculaMulta() { }
        // Imprime o total da fatura
        void imprimeDados() { }
}
```

k) Teste a classe FATURA. Escrever o código abaixo no método *main()*.

```
Fatura f = new Fatura();
f.adiciona(100);
f.adiciona(200);
f.imprimeDados();
double multa = f.calculaMulta();
```

l) Alterar a classe CartaodeCredito para incluir as faturas de um cartão.

```
class CartaoDeCredito {
        int numero;
        String dataDeValidade;
        Cliente cliente;
        // Adicione a linha abaixo
        Fatura[] faturas;
}
```

l)Acrescentar um método à classe CONTA para transferir um valor para uma conta de destino.

// Deve-se verifica se a conta possui saldo suficiente para a transferência

```
void transfere(Conta destino, double valor) { }
```

Ex5.4 (exercício para entregar) Fazer um **programa em C#,** orientado por objetos que utilize obrigatoriamente os conceitos de **herança e composição,** para implementar uma ***agenda*** para armazenar os dados de ***contatos*** de pessoas, sejam elas **físicas ou jurídica**s.

Os dados de pessoas físicas e jurídicas, são armazenados com os seguintes itens especificados a seguir:

Pessoas Físicas

- Nome
- Endereço
- Data de nascimento
- Sexo
- CPF

Pessoas Jurídicas

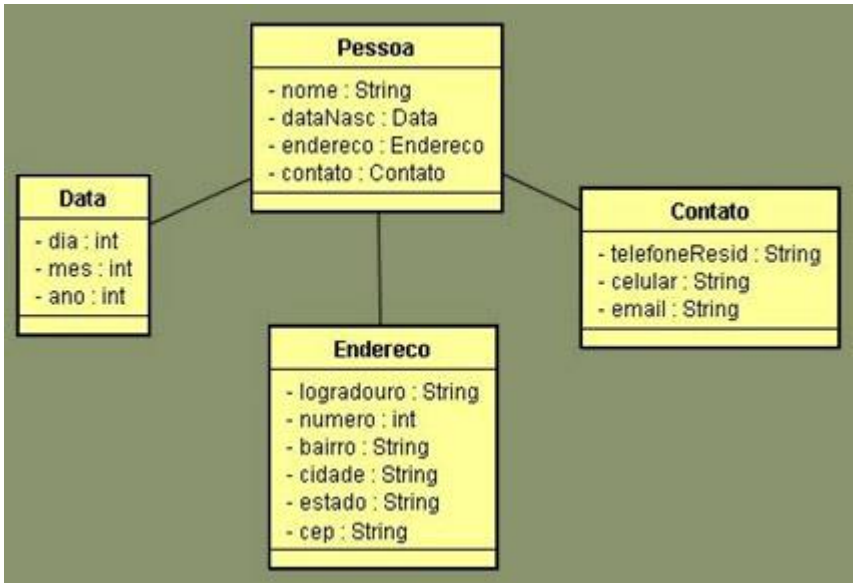
- Nome
- Endereço
- Data de estabelecimento
- CNPJ

A agenda *tem* vários contatos: AGREGAÇÃO / COMPOSIÇÃO

Um contato ***TEM*** uma pessoa*:* associação contato → pessoa

Ex5.5 (exercício para entregar):

Crie as classes apresentadas no diagrama abaixo e aplique a Composição para a classe Pessoa, que além de possuir um atributo Nome será composta pelas classes Data, Endereço e Contato para os atributosdataNasc, endereço e contato, respectivamente
.



OBS: Para melhorar os aspectos de encapsulamento, crie construtores para cada uma das classes e faça a inicialização dos objetos via construtores.

6. Tratamento de exceções, eventos e delegados (introdução) e técnicas avançadas de interface gráfica de usuário (GUI) com Windows no C#.

Ex6.1 (exercício para entregar):
Usar técnicas de tratamento de exceções **(C# e Java**), interface gráfica de usuário (GUI) **(C#**) e eventos e delegados **(C#**) para aperfeiçoar os programas do exercício 3.3 dos Labs 1 e 4 (Formulários):
3.3.1 Converter C-F/F-C
3.3.2 Calculadora
3.3.3 Conta

Ex6.2 (exercíciospara entregar):
Usar técnicas de tratamento de exceções, interface gráfica de usuário (GUI)**,** eventos e delegados para aperfeiçoar os programas codificados em **C#** do exercício 7 do lab2 (Herança e arquivos):
7.1 C**lasse Pessoa**
7.2 Classe Conta e por herança as **classes Poupança e Investimento.**

Ex6.3 (exercício para entregar): Compile os **programas da pasta Eventos_delegados,** disponível no SGA, **traduza** os comentários, **explique** o que **os programas fazem**, **identifique** em que partes dos códigos os conceitos de **tratamento de exceções, eventos, delegados** e **interfaces gráficas** foram utilizados.

Ex6.4 (exercício para entregar)
Compile os **programas da pasta Excecoes,** disponível no SGA, **traduza** os comentários, **explique** o que **os programas fazem**, **identifique** em que partes dos códigos os conceitos de **tratamento de exceções, eventos, delegados** e **interfaces gráficas** foram utilizados.

Ex6.5 (exercício para entregar):
Compile os **programas da pasta GUI,** disponível no SGA, **traduza** os comentários, **explique** o que **os programas fazem**, **identifique** em que partes dos códigos os conceitos de **tratamento de exceções, eventos, delegados** e **interfaces gráficas** foram utilizados.