**-----ENUM-----**

MANEIRA DE REPRESENTAR UM CONJUNTO DE CONSTANTES/PALAVRAS, SENDO CADA CONSTANTE/PALAVRA UM VALOR NUMÉRICO.

EX: UMA CLASSE PEDIDO, CUJO ATRIBUTO Status É UM ENUM DA SEGUINTE FORMA

class Order

{

public int Id { get; set; }

public DateTime Moment { get; set; }

public OrderStatus Status { get; set; }

}

ATRIBUTO Status É DO TIPO enum OrderStatus, SENDO ESSE enum

enum OrderStatus : int

{

PendingPayment = 0,

Processing = 1,

Shipped = 2,

Delivered = 3

}

ASSIM, OS ESTADOS DO PEDIDO PODEM SER RELACIONADOS COM ESSES VALORES NO MOMENTO DE INSTANCIAR UM OBJETO Order

NO PROGRAMA PRINCIPAL

Order order1 = new Order

{

Id = 01,

Moment = DateTime.Now,

Status = OrderStatus.PendingPayment //A PARTIR DO . O COMPILADOR SUGERE

};

PARA ORGANIZAR ESSA ESTRUTURA NUM PROJETO, O IDEAL É CRIAR UMA PASTA ENTIDADES, ONDE FICARÃO TODAS AS CLASSES QUE REPRESENTAM ENTIDADES DA APLICAÇÃO, E DENTRO DESSA PASTA ENTIDADES, CRIAR UMA SUBPASTA ENUMS, ONDE FICARÃO OS ENUMS RELACIONADOS.

ADICIONAR->CLASSE->NOMEAR->TROCAR class POR enum

LEMBRAR DE: PARA QUE UMA ENUM/CLASSE DE UM NAMESPACE DIFERENTE SEJA VISÍVEL, É NECESSÁRIO IMPORTÁ-LA NO CÓDIGO

using ProjetoOO\_3.Entities;

using ProjetoOO\_3.Entities.Enums;

**---CONVERTENDO ENUM PARA STRING---**

string txt = OrderStatus.PendingPayment.ToString();

**---CONVERTENDO STRING PARA ENUM---**

OrderStatus order\_status = Enum.Parse<OrderStatus>("Delivered");

//CONSIDERANDO QUE NA ESTRUTURA DO ENUM OrderStatus JÁ TENHA O VALOR Delivered

//ASSIM A STRING PASSADA NO PARÂMETRO SE TORNA O VALOR DO OBJETO order\_status

//NÃO FUNCIONARIA SE A STRING NÃO EXISTISSE DENTRO DA ESTRUTURA DO ENUM (Delivere POR EXEMPLO)

**REPRESENTAÇÃO UML ENUM**



**---COMPOSIÇÃO---**

UM OBJETO CONTÉM OUTRO - TEM UM / TEM VÁRIOS

DIAMANTE PRETO - TODO PARTE

**ENTRE ENTIDADES**



UM PEDIDO CONTÉM VÁRIOS ITENS

UM PEDIDO ESTÁ RELACIONADO A UM CLIENTE (COMPOSIÇÃO TAMBÉM)

UM ITEM TEM 1 PRODUTO

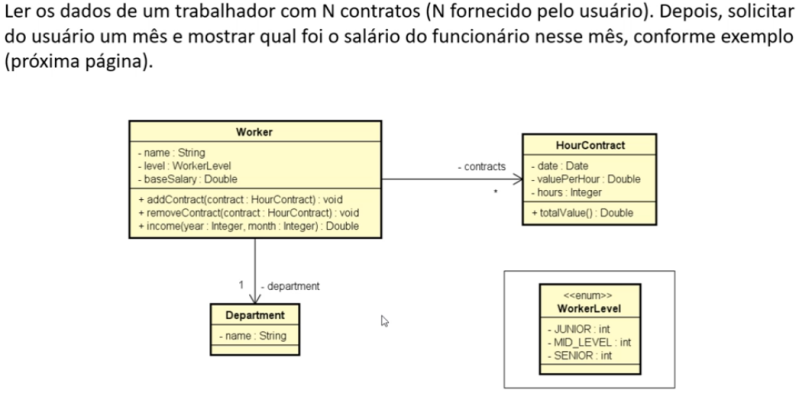
**ENTRE SERVIÇOS**



TANTO UM SERVIÇO DE PEDIDO QUANTO UM SERVIÇO DE AUTENTICAÇÃO ESTÃO RELACIONADOS COM SERVIÇO DE E-MAIL

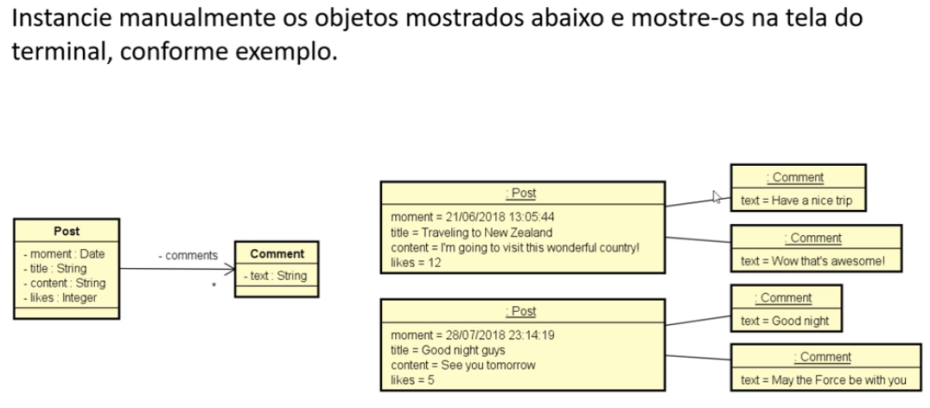
UM SERVIÇO DE PEDIDO ESTÁ RELACIONADO COM UM REPOSITÓRIO DO PEDIDO (QUE ACESSA O BD)

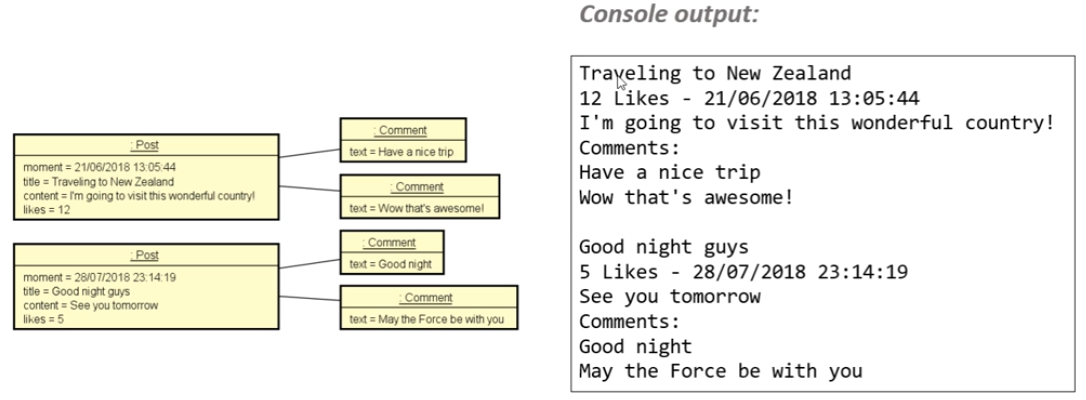
**EXERCÍCIO RESOLVIDO 1**

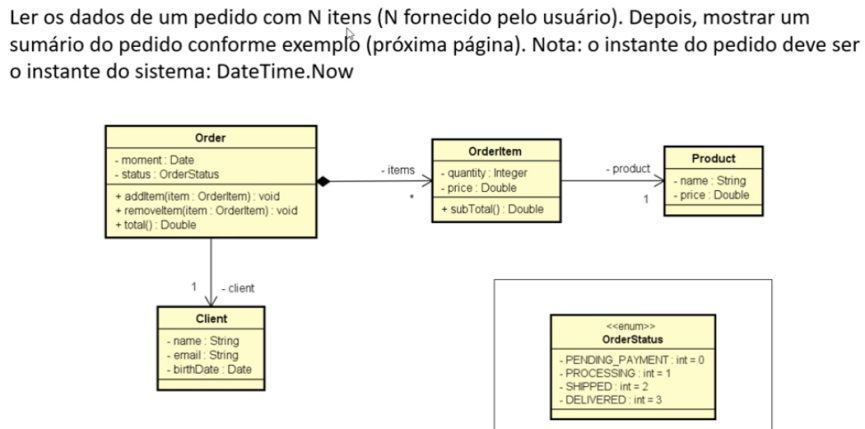
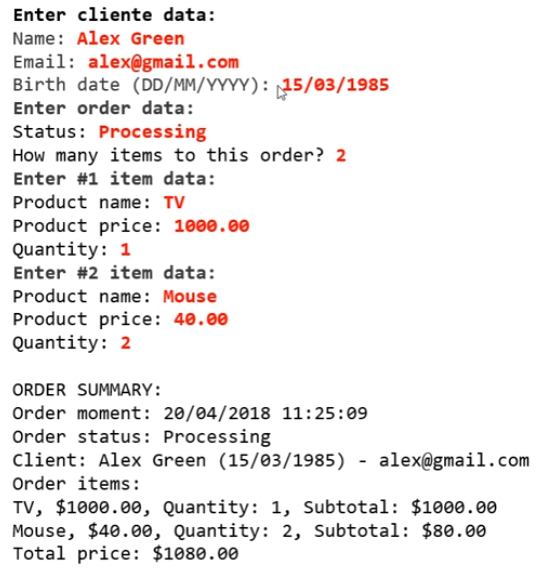
\*NO CONSTRUTOR DE WORKER, NÃO USAR COMO ARGUMENTO A LISTA. SEMPRE NUMA RELAÇÃO DE COMPOSIÇÃO DE UM PARA MUITOS, O CONSTRUTOR DE “UM” NÃO INICIA JÁ COM A LISTA DE “MUITOS”, O MUITOS É ADICIONADO AOS POUCOS.

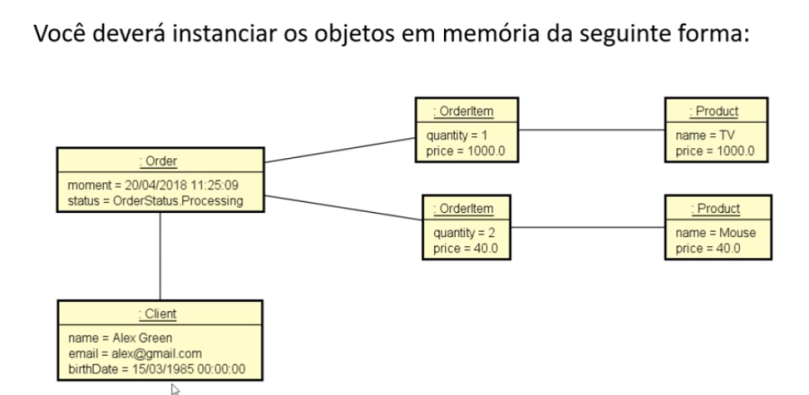
**EXERCÍCIO RESOLVIDO 2**





**EXERCÍCIO RESOLVIDO 3**



**StringBuilder**

UMA CLASSE QUE PODE SER UTILIZADA PARA OTIMIZAR O MÉTODO ToString DE UMA CLASSE. QUANDO O MÉTODO ToString TEM UM CONTEÚDO MUITO GRANDE, É RECOMENDÁVEL UTILIZAR O StringBuilder PARA CRIAR A SAÍDA.

public override string ToString()

{

StringBuilder sb = new StringBuilder();//MONTAR STRING COM MAIS EFICIÊNCIA

sb.AppendLine(Title);

sb.AppendLine($"{Likes} Likes - {Moment.ToString("dd/MM/yyyy HH:mm:ss")}");

sb.AppendLine(Content);

sb.AppendLine("Comments");

foreach(Comment c in Comments)

{

sb.AppendLine(c.Text);

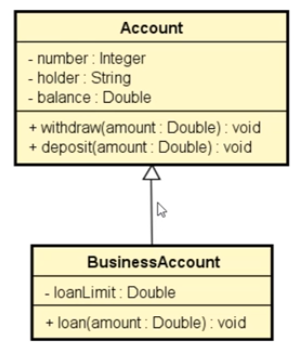
}

return sb.ToString();

}

**HERANÇA**

ASSOCIAÇÃO ENTRE CLASSES QUE PERMITE REUTILIZAR ATRIBUTOS E MÉTODOS DA CLASSE BASE PARA A CLASSE DERIVADA, ADICIONANDO SOMENTE O QUE FOR NECESSÁRIO, AO INVÉS DE REPETIR TUDO.



PARA GARANTIR QUE UM ATRIBUTO/MÉTODO SÓ É ACESSÍVEL NA CLASSE OU EM UMA DE SUAS SUBCLASSES, USAR protected

**---USANDO CONSTRUTOR DA BASE NA DERIVADA---**

CONSTRUTOR DA BASE

public Account(int number, string holder, double balance)

{

Number = number;

Holder = holder;

Balance = balance;

}

CONSTRUTOR DA DERIVADA - USA AS INFORMAÇÕES E ADICIONA UM PARÂMETRO

public BusinnesAccount(int number, string holder, double balance, double loanLimit)

: base(number, holder, balance)

{

LoanLimit = loanLimit;

}

**---UPCASTING E DOWNCASTING---**

SÃO AÇÕES DE:

UPCASTING - INSTANCIAR UM OBJETO DA CLASSE BASE RECEBENDO UM CONSTRUTOR DA CLASSE DERIVADA

DOWNCASTING-INSTANCIAR UM OBJETO DA CLASSE DERIVADA RECEBENDO UM CONSTRUTOR DA CLASSE BASE.

AMBOS SÃO ACEITOS

CONSIDERANDO AS CLASSES Account(BASE) , BusinessAccount(DERIVADA) E SavingAccount(DERIVADA)

Account a = new Account(1, "Luciano", 10500);

BusinnesAccount b = new BusinnesAccount(2, "Comercial Luciano", 35000, 12000);

Account c = new SavingsAccount(3, "Lara", 12000, 0.05);//UPCASTING

//CASTING USANDO as

//Account d = (BusinnesAccount)b;

//Account d = b as BusinnesAccount;

SavingsAccount e = (SavingsAccount)c; //DOWN CASTING

//c É SavingAccount - COMPATÍVEL COM e

//BusinnesAccount e = (BusinnesAccount)a;//DOWNCASTING DEFEITUOSO - NÃO COMPATÍVEL

//BusinnesAccount f = (BusinnesAccount)c;//DOWNCASTING DEFEITUOSO - NÃO COMPATÍVEL

//PARA TESTAR ANTES DE FAZER O CASTING

if (c is BusinnesAccount)

{

BusinnesAccount f = (BusinnesAccount)c;//DOWNCASTING DEFEITUOSO - NÃO COMPATÍVEL

}

else

{

Console.WriteLine("NÃO COMPATÍVEL");

}

DOWNCASTING - OPERAÇÃO INSERGURA - DEVE SER USADA SOMENTE QUANDO NECESSÁRIO

**SOBREPOSIÇÃO COM VIRTUAL E OVERRIDE**

SOBREPOR UM MÉTODO SIGNIFICA QUE, NA CLASSE DERIVADA, O MÉTODO HERDADO DA CLASSE BASE SERÁ REESCRITO, MANTENDO SEU NOME MAS MUDANDO SUA LÓGICA. PRA QUE ISSO SEJA POSSÍVEL, NA CLASSE BASE ELE PRECISA USAR A PALAVRA virtual E NA CLASSE DERIVADA override.

virtual - INDICA QUE O MÉTODO PODE SER SOBREPOSTO NAS SUAS SUBCLASSES

override - INDICA QUE O MÉTODO É UMA SOBREPOSIÇÃO DO MESMO MÉTODO NA SUA CLASSE BASE

NA CLASSE BASE

//SAQUE - CADA SAQUE TEM TAXA DE 5$

public virtual void WithDraw(double ammount)

{

Balance -= ammount+5;

}

NA CLASSE DERIVADA

//MÉTODO SOBRESCRITO - CONTA POUPANÇA NÃO TEM DESCONTO PARA SAQUE

public override void WithDraw(double ammount)

{

Balance -= ammount;

}

CASO SEJA NECESSÁRIO, POR ALGUM MOTIVO, USAR NA CLASSE DERIVADA ALGUM MÉTODO DA CLASSE BASE, SÓ É PRECISO USAR A PALAVRA base

NA CLASSE DERIVADA

base.WithDraw(ammount);

**---CONTROLADOR sealed---**

NAS CLASSES

USADO NA DEFINIÇÃO DE UMA CLASSE PARA IMPEDIR QUE ELA SEJA HERDADA

sealed class SavingsAccount

NOS MÉTODOS

USADO EM MÉTODOS QUE JÁ FORAM SOBREPOSTOS PARA EVITAR QUE SEJAM SOBREPOSTOS NOVAMENTE EM SUAS SUBCLASSES

public sealed override void WithDraw(double ammount)

\*BOA PRÁTICA: SEMPRE SELAR MÉTODOS SOBREPOSTOS - MAIS SEGURO E MAIS EFICIENTE

**POLIMORFISMO**

RECURSO PRA POO QUE PERMITE QUE VARIÁVEIS DE UM MESMO TIPO GENÉRICO POSSAM APONTAR PARA OBJETOS DE TIPOS ESPECÍFICOS DIFERENTES (EM TEMPO DE EXECUÇÃO).

TIPO GENÉRICO Account APONTANDO PARA DOIS TIPOS ESPECÍFICOS (BusinessAccount E SavingsAccount)

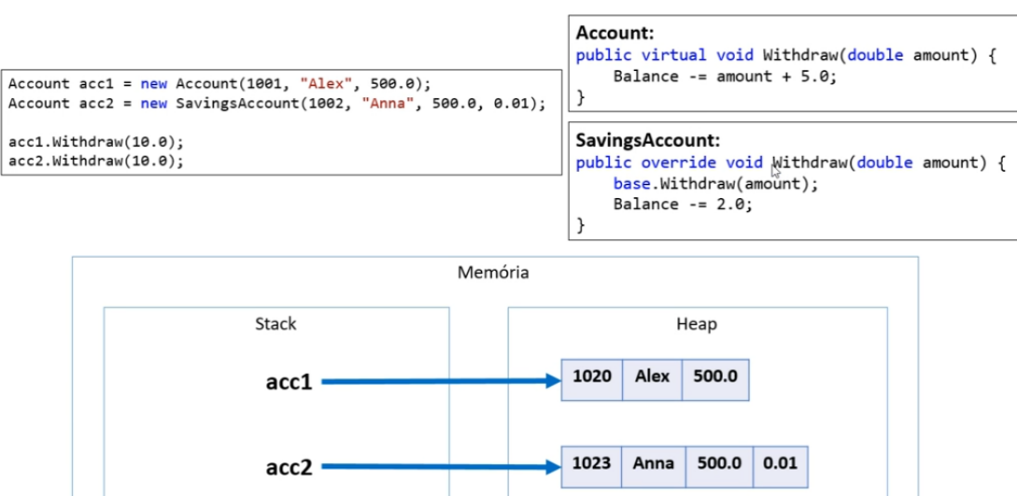
Account a = new BusinnesAccount(1, "Luciano", 400,1200);

Account b = new SavingsAccount(2, "Ana", 400, 0.05);

a.WithDraw(10);

b.WithDraw(10);

Console.WriteLine($"{a.Balance}\n{b.Balance}");



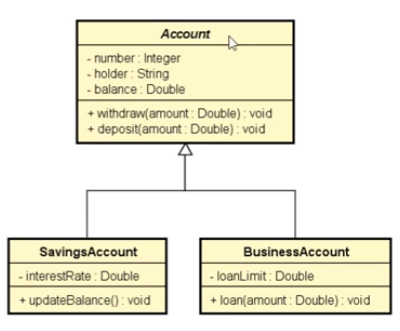
**CLASSES ABSTRATAS**

NÃO PODEM SER INSTANCIADAS, APENAS SUAS SUBCLASSES

NOTAÇÃO UML: ITÁLICO

MESMO SENDO ABASTRATA, AINDA É POSSÍVEL DECLARAR UMA LISTA DO TIPO DESSA CLASSE

abstract class Account



\*MÉTODOS ABSTRATOS SÃO AQUELES QUE NÃO POSSUEM IMPLEMENTAÇÃO, CONSIDERANDO QUE SÃO TÃO GENÉRICOS, QUE SUA LÓGICA SÓ PODE SER IMPLEMENTADA NA SUBCLASSE ESPECÍFICA.

SE UMA CLASSE POSSUI AO MENOS UM MÉTODO ABSTRATO, A CLASSE TAMBÉM SE TORNA ABSTRATA.

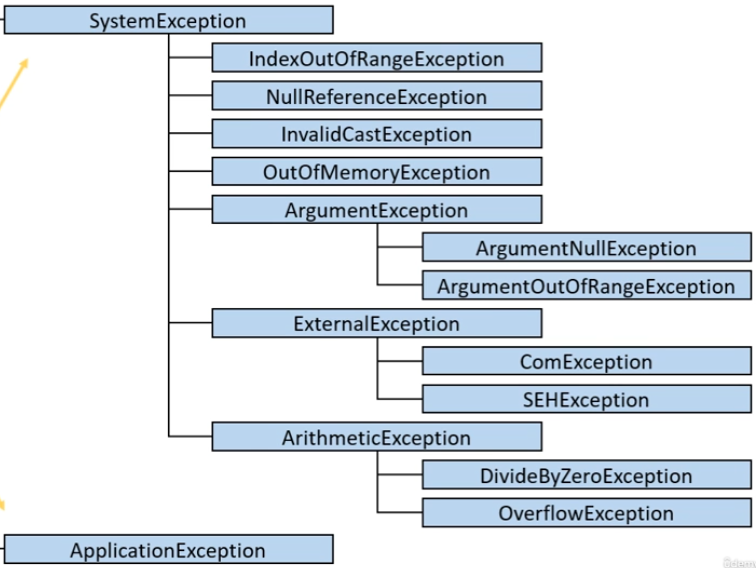
public abstract double Area();

//SEM IMPLEMENTAÇÃO - SEM CHAVES

**---EXCEÇÕES---**

SystemException - Exceções do próprio sistema.

ApplicationException - Para criação de exceções pelo programador na aplicação.



**--TRY-CATCH--**

try

{

CÓDIGO COM POSSIBILIDADE DE ERRO

}

catch (TIPO EXCEÇÃO x)

{

Console.WriteLine($"Erro! {x.Message}");

}

**--FINALLY--**

Finally é um bloco opcional que poder ser incluído ao try-catch. Ele será executado independente de um catch ter sido ativado ou não. Muito usado para fechar conexão com arquivo de texto e banco de dados

try

{

int a = int.Parse(Console.ReadLine());

int b = int.Parse(Console.ReadLine());

int c = a / b;

Console.WriteLine(c);

}

catch (FormatException e)//VARIÁVEL DO TIPO ERRADO

{

Console.WriteLine($"Erro! {e.Message}");

}

catch (DivideByZeroException e1)//DIVISÃO POR ZERO

{

Console.WriteLine($"Erro! {e1.Message}");

}

catch(Exception e)//GENERIC ERROR

{

Console.WriteLine($"Erro! {e.Message}");

}

finally

{

Console.WriteLine("Aparece de qualquer forma");

}

---EXCEÇÕES PERSONALIZADAS---

PARA TRATAR EXCEÇÕES QUE SÃO REGRAS DE NEGÓCIOS, USA-SE CONDICIONAIS ALIADAS COM COMANDO THROW E UMA ESTRURA DE UMA CLASSE DomainException NA PASTA Execptions QUE HERDA DE ApplicationException

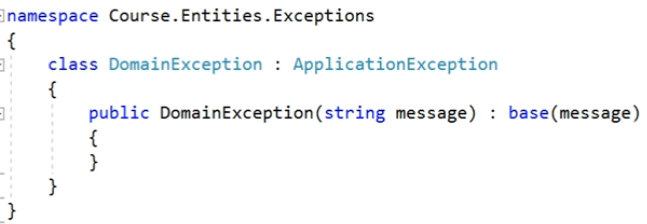
DomainException - NA ANÁLISE DE SISTEMA, Domain É O ESCOPO DO NEGÓCIO

Herdar de ApplicationException - SIGNIFICA QUE OS ERROS SÃO QUEBRAS DAS REGRAS DE NEGÓCIO

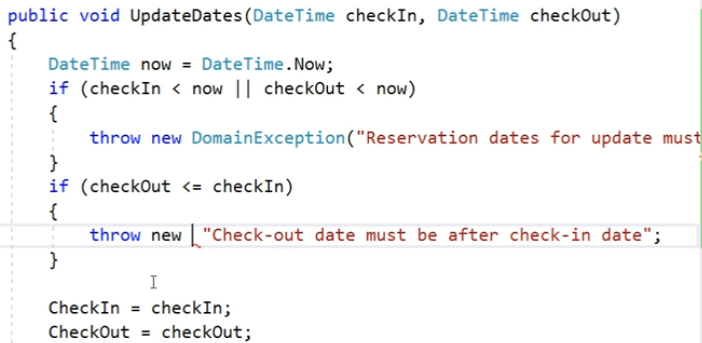
CONSIDERANDO O EXEMPLO DE UM SISTEMA DE RESERVERSAS ONDE

- A DATA DE RESERVA NÃO PODE SER NO PASSADO

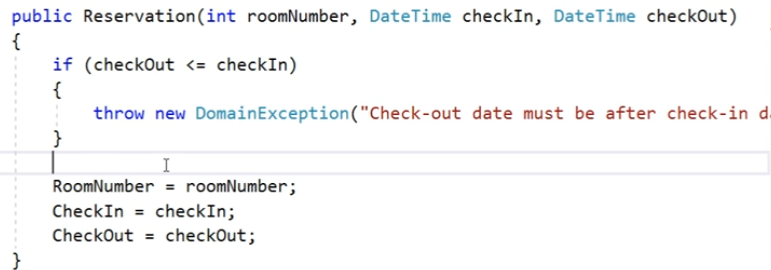
-A DATA DE CHECKIN NÃO PODE SER DEPOIS DE CHECKOUT



CLASSE DomainException com um construtor que usa o construtor da classe base (ApplicationException)



NA CLASSE RESERVA, NA FUNÇÃO DE ATUALIZAR A RESERVA, QUANDO FEITA A COMPARAÇÃO DE DATAS, SE QUEBRAR A REGRA DE NEGÓCIO, É LANÇADA A EXCEÇÃO, QUE É UMA INSTÂNCIA DE DomainException RECEBENDO COMO ARGUMENTO A MENSAGEM DE ERRO.



TAMBÉM É FEITA A VERIFICAÇÃO DE ERRO NO CONSTRUTOR DA CLASSE. CASO AS DATAS QUEBREM AS REGRAS DE NEGÓCIO, É LANÇADA A EXCEÇÃO.

E PARA QUE ESSA EXECÇÕES SEJAM LANÇADAS, NO PROGRAMA PRINCIPAL É NECESSÁRIO COLOCAR TODAS AS LINHAS QUE POSSAM GERAR ESSES ERROS DENTRO DE UM BLOCO TRY, COM O CATCH ABAIXO COM TIPO DE ERRO DomainException



.

.

