Exercícios de Revisão - Parte 2
Unidade 1 - Linguagem C#

Resolva todos os exercícios de revisão dentro de uma mesma solução (*solution*) no Visual Studio e crie um projeto para cada um deles.

1. Levando em consideração que os dados a seguir são dados de um cliente no cadastro de uma empresa, reestruture o exercício anterior sobre validação de dados para atender às seguintes regras:

Campo	Regras	Tipo
Nome	Pelo menos 5 caracteres	string
CPF	Validado de acordo com anexo A	long
Data de nascimento	Lida no formato DD/MM/AAAA O cliente deve ter pelo menos 18 anos na data atual	DateTime
Renda mensal	Valor ≥ 0 Lida com duas casas decimais e vírgula decimal	float
Estado civil	C, S, V ou D (maiúsculo ou minúsculo)	char
Dependentes	0 a 10	int

- Todos os dados devem ser lidos em formato **string**, em sequência, e a validação de todos os dados deverá ocorrer somente **após** a leitura do **último** dado (Dependentes).
- Se todos os dados estiverem corretos, o programa deve imprimi-los e terminar.
- Se um ou mais campos estiverem com erro, o programa deve mostrar a lista de erros com nome do campo seguido do dado incorreto e da mensagem de erro correspondente, e ler novamente **somente** os campos incorretos.

A solução não deve ser implementada em uma única classe ou toda no método Main. O desafio desse exercício é estruturar a solução usando a **separação de responsabilidades** (*separations of concerns*), de forma que cada classe tenha uma responsabilidade única e bem definida.

2. Crie uma classe **Turma** que possui uma lista de **Alunos**. Cada aluno tem matrícula e nome (obrigatórios). Durante o semestre os alunos fazem duas provas (P1 e P2). Crie métodos para:
   - Inserir e remover aluno da turma.
   - Lançar a nota (seja ela P1 ou P2) de um aluno.
   - Imprimir os alunos da turma em ordem alfabética e suas notas finais: NF = (P1 + P2) / 2
   - Imprimir as estatísticas da turma: média da P1, média da P2, média da turma (média das NFs), maior NF da turma com os dados do respectivo aluno.

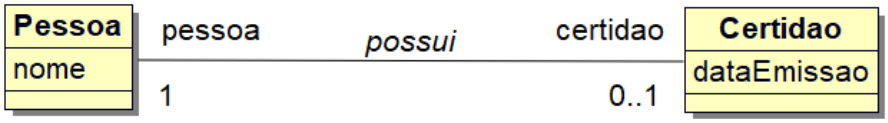
3. Crie as classes **Curso**, **Aluno** e **Turma** com os atributos, propriedades, construtores e métodos adequados para representar o seguinte cenário:
   - Um curso tem nome (obrigatório).
   - Um aluno tem matrícula e nome (obrigatórios, onde a matrícula é gerada pelo próprio programa).
   - Uma turma tem código (obrigatório).
   - Um curso tem zero ou mais alunos e zero ou mais turmas.
   - Não pode haver dois alunos com a mesma matrícula nem duas turmas com o mesmo código.
   - Um aluno pode estar associado a uma turma ou não, mas somente a uma turma.
   - Uma turma pode ter zero ou mais alunos.

   Crie métodos para:
   - Matricular um aluno no curso.
   - Remover um aluno do curso (somente se aluno não está associado a uma turma).
   - Criar uma nova turma no curso.
   - Remover uma turma do curso (somente se a turma não tiver nenhum aluno associado a ela).
   - Inserir e remover aluno de uma turma (o mesmo aluno não pode ser inserido duas vezes na mesma turma, mas pode estar em mais de uma turma).
   - Listar alunos de uma turma específica em ordem alfabética.
   - Listar todas as turmas do curso que possuem alunos (turmas em ordem de código e alunos da turma em ordem alfabética).

4. Crie as classes **Pessoa** e **CertidaoNascimento** com os atributos, propriedades e construtores adequados para representar o seguinte cenário:

   - Uma pessoa tem sempre um nome.
   - Uma certidão de nascimento tem sempre uma data de emissão.
   - Uma pessoa pode ter ou não uma única certidão, mas uma certidão sempre pertence a uma única pessoa, ou seja, é possível criar uma pessoa sem certidão, mas não é possível criar uma certidão sem ter uma pessoa associada a ela.
   - Não é possível alterar a certidão de uma pessoa, nem é possível alterar a pessoa associada a uma certidão.

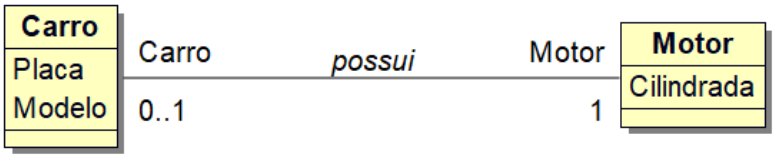
   Esse cenário pode ser representado pelo seguinte diagrama de classes da UML:



5. Crie as classes **Carro** e **Motor** com os atributos, propriedades, construtores e métodos adequados para representar o seguinte cenário:

   - Um carro possui placa (string), modelo (string) e motor, todos obrigatórios.
   - Um motor possui cilindrada (1.0, 1.6, 1.8, 2.0, etc.), que é obrigatória.
   - É possível trocar o motor de um carro, mas o carro nunca deve ficar sem motor.
   - Um motor pode estar instalado em um carro ou não.
   - Não é possível colocar o mesmo motor em dois carros distintos (deve ser gerada uma exceção).
   - Não é possível alterar a placa ou modelo de um carro nem a cilindrada de um motor.
   - A velocidade máxima de um carro é calculada como: 140 km/h para carros com motor até 1.0, 160 km/h para carros com motor até 1.6, 180 km/h para carros com motor até 2.0 e 220 km/h para carros com motor > 2.0.

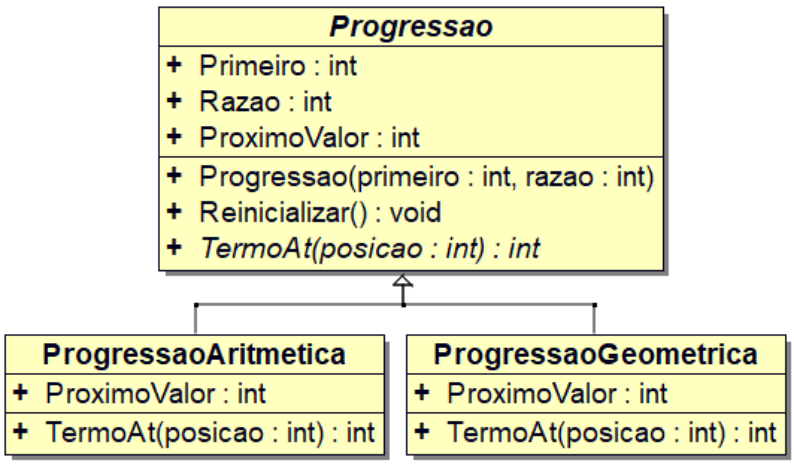
   A partir dessas regras, crie métodos para construir objetos do tipo Carro e Motor, alterar seus atributos (quando possível) e calcular a velocidade máxima de um carro.



6. Dada a classe abstrata **Progressao** implemente as classes concretas **ProgressaoAritmetica** e **ProgressaoGeometrica** definidas a seguir:

   - **Primeiro** e **Razao** são duas propriedades concretas alteráveis.
   - **ProximoValor** é uma propriedade abstrata *readonly* que retorna o próximo valor da progressão a cada vez que é chamada, iniciando em Primeiro.
   - **TermoAt** é um método abstrato que retorna o termo da progressão que está na posição indicada, iniciando na posição 1.
   - **Reinicializar** é um método que reinicializa a propriedade **ProximoValor**, fazendo com que ela volte a retornar o **Primeiro** termo da progressão.

   Você pode adicionar outros atributos, propriedades ou métodos que achar necessários. Ao final, usando as classes **ProgressaoAritmetica** e **ProgressaoGeometrica**, imprima 10 termos de uma PA e PG que iniciam em 3 e tem razão 4.



7. Um arquivo de **propriedades** (*properties*) é um arquivo **texto** onde cada linha tem o formato **chave=valor**. Exemplo:

   url=http://empresa.com.br/app
   porta=8080
   endereco=192.161.35.101
   email=admin@gmai.com

   Normalmente, esse tipo de arquivo é usado para armazenar configurações ou parâmetros de uma aplicação. Crie a classe **Propriedades** com os atributos, propriedades, construtores e métodos adequados para manipular um conjunto de propriedades. Essa classe deve conter dois construtores:

   - **Propriedades()**: cria o objeto sem nenhuma propriedade.
   - **Propriedades(string path)**: cria o objeto com as propriedades armazenadas no arquivo *path*. Gere uma exceção caso haja algum erro na manipulação do arquivo.

   Além disso, também devem ser implementados métodos para:

   - Recuperar o valor de uma chave
   - Alterar o valor de uma chave (se a chave não existir gere uma exceção)
   - Verificar se uma chave existe ou não
   - Incluir uma nova chave (se a chave já existir gere uma exceção)
   - Salvar as propriedades em um arquivo texto (pode ser no mesmo arquivo de onde as propriedades foram carregadas ou pode ser diferente).

   Tanto a chave quanto o valor devem ser do tipo **string** e não pode haver duas propriedades com a mesma chave.

   Observação: no C# já existe uma classe denominada Properties para manipular arquivos de propriedades, mas não use essa classe. Implemente a sua!

Anexo A - Validação de CPF

Um CPF é válido se obedece às seguintes regras:

- Possui exatamente 11 dígitos
- Os 11 dígitos não podem ser todos iguais (tudo 1 ou tudo 2, por exemplo)
- Os 9 primeiros dígitos compõem o número do CPF e os 2 últimos dígitos são os dígitos verificadores (DV). Os DVs são calculados da seguinte forma:

  J é o **1° dígito verificador** do CPF.

  K é o **2° dígito verificador** do CPF.

  Primeiro Dígito

  Para obter **J** multiplicamos os 9 primeiros dígitos do CPF (A a I) pelas constantes da tabela a seguir:

A	B	C	D	E	F	G	H	I
x 10	x 9	x 8	x 7	x 6	x 5	x 4	x 3	x 2

  O resultado da soma 10A + 9B + 8C + 7D + 6E + 5F + 4G + 3H + 2I é **dividido por 11.** Analisamos então o **resto** dessa divisão: se for 0 ou 1, o dígito **J** é **0** (zero). Se for de 2 a 10, o dígito **J** é **11 – RESTO**.

  Segundo Dígito

  Já temos **J**. Para obter **K** multiplicamos os 10 primeiros dígitos do CPF (A a J) pelas constantes da tabela a seguir:

A	B	C	D	E	F	G	H	I	J
x 11	x 10	x 9	x 8	x 7	x 6	x 5	x 4	x 3	x 2

O resultado da soma 11A + 10B + 9C + 8D + 7E + 6F + 5G + 4H + 3I + 2J é **dividido por 11.** Analisamos então o **resto** dessa divisão: se for 0 ou 1, o dígito **K** é **0** (zero). Se for 2 a 10, o dígito **K** é **11 – RESTO**.