Lista de Exercícios

1. Escreva um programa que solicita que sejam digitadas três notas de um aluno e um peso para cada nota. Calcule e imprima a média do aluno.

Dica : Utiliza o recurso Console.ReadLine() para leitura dos parâmetros.

2. Escreva um programa semelhante ao do exercício 1, mas agora o número de notas e pesos pode variar. O usuário deve digitar quantas notas ele desejar e, para parar, a nota -1 deve ser digitada. Neste momento a média das notas e pesos digitados anteriormente deve ser calculada e o resultado impresso na tela.

Dica : Utiliza o recurso Console.ReadLine() para leitura dos parâmetros.

3. Complete as 4 tarefas:
a. Imprima todos os números de 10 a 25.
b. Imprima a soma dos números de 1 a 100, pulando de dois em dois (1, 3, 5, 7, etc.).
c. Começando em 0, imprima os números seguintes, enquanto a soma dos números já impressos for menor que 100.
d. Imprima a tabuada do 9 (até o décimo valor).

4. Escreva um programa que calcule o fatorial de 10. A regra do fatorial (!) é a seguinte:
0! = 1
1! = 0! x 1
2! = 1! x 2
...
n! = (n-1)! x n

Dica : http://pt.wikipedia.org/wiki/Factorial.

5. Imprima os 15 primeiros números da série de Fibonacci. A série de Fibonacci possui a seguinte sequência numérica: 0, 1, 1, 2, 3, 5, 8, 13, 21, etc.

Para calculá-la, o primeiro e segundo elementos valem 1, daí por diante, o n-ésimo elemento vale o (n-1)-ésimo elemento somado ao (n-2)-ésimo elemento (ex: 8 = 5 + 3).

Dica 1: http://pt.wikipedia.org/wiki/Número_de_Fibonacci.
Dica 2: Utiliza o recurso Console.ReadLine() para leitura dos parâmetros.

6. Escreva um programa que imprime na saída os valores assumidos por x. Esta variável x deve iniciar com algum valor inteiro, fornecido pelo usuário. Se x for par, x deve receber o valor dele mesmo somado com 5. Já se x for ímpar, x deve receber o valor dele multiplicado por 2. O programa termina assim que x for maior que 1000. Por exemplo, para x = 10, a saída deve ser: 15, 30, 35, 70, 75, 150, 155, 310, 315, 630, 635, 1270. Faça este exercício usando blocos if e depois usando blocos switch.

Dica: Utiliza o recurso Console.ReadLine() para leitura dos parâmetros.

7. Verifique a validade de uma data e mostre uma mensagem na tela dizendo se a data é válida ou inválida. Devem existir três variáveis para armazenar o dia, o mês e o ano, e o usuário deve fornecer os valores para estas variáveis via console. Considerar que fevereiro pode ter somente 28 dias e que anos válidos estão compreendidos entre 1900 e 2999.

Dica: Utiliza o recurso Console.ReadLine() para leitura dos parâmetros.

8. Crie a classe Calculadora com métodos que encapsulem os exercícios de 1 a 7, e a classe Program com o método Main() implementando as chamadas aos métodos da classe Calculadora.

9. Crie as classes Relogio e Ponteiro e escreva um método Main() para treinar a chamada aos métodos e atributos.

Atributos da classe Relogio:
- ponteiroHora (tipo Ponteiro)
- ponteiroMinuto (tipo Ponteiro)
- ponteiroSegundo (tipo Ponteiro)

Métodos da classe Relogio:
- AcertarRelogio(int, int, int): Acerta o relógio, posicionando adequadamente cada ponteiro do relógio. Os parâmetros passados são hora, minuto e segundo.
- LerHora(): retorna a hora atual do relógio.
- LerMinuto(): retorna o minuto atual do relógio.
- LerSegundo(): retorna o segundo atual do relógio.

Atributos da classe Ponteiro:
- posicao(int): indica em qual posição está o ponteiro (1, 2, 3, 4, etc.).

10. Crie a classe Fracao, que representa uma fração matemática. Esta classe deve ser capaz de armazenar o numerador e o denominador da fração. Ela ainda deve ter um método que recebe uma fração como parâmetro, multiplica ambas as frações, e retorna uma nova fração como resultado. Crie um programa simples que instancia duas frações, define seus valores, calcula o valor da multiplicação entre elas e mostra o resultado.

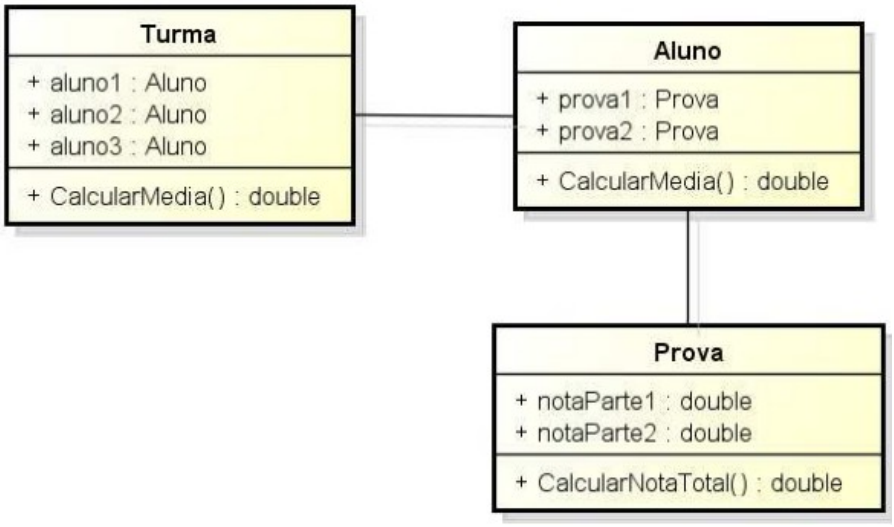
11. Crie classes que representam as figuras geométricas: Triangulo, Quadrado, Circunferencia e Trapezio. Cada uma destas classes deve ter um método para calcular a sua área, com a seguinte assinatura: double CalcularArea().

Perceba que o método CalcularArea() não recebe parâmetros. Portanto todos os dados necessários devem ser armazenados no objeto da classe em atributos, para depois serem utilizados pelo método. As fórmulas para o cálculo da área são as seguintes:

Figura	Fórmula	Elementos da Fórmula
Triângulo	$A = \frac{b \times h}{2}$	**b** = base **h** = altura
Quadrado	$A = l^2$	**l** = lado
Circunferência	$A = \pi \times r^2$	**r** = raio
Trapézio	$A = \frac{(B + b)}{2} \times h$	**B** = base maior **b** = base menor **h** = altura

Dica: O valor de π pode ser obtido através da chamada Math.PI no C#

12. Desenvolva um sistema escolar para cálculos de médias. Ele é composto pelas seguintes classes:



Uma turma é composta por três alunos. Cada um dos alunos realizou duas provas, onde cada prova possuía duas partes. Observe uma descrição sobre o que cada método faz:

Classe	Método	Descrição
Turma	***CalcularMedia()***	Calcula a média da turma. A média é calculada utilizando a média de cada aluno da turma.
Aluno	***CalcularMedia()***	Calcula a média do aluno. A média é calculada utilizando a nota total das duas provas realizadas por ele.
Prova	***CalcularNotaTotal()***	Calcula a nota total da prova. Esta nota é data pela soma das notas das partes 1 e 2. A nota total não pode ultrapassar 10.0.

Crie uma aplicação que instancia a turma, os três alunos e as duas provas para cada aluno. Defina também notas para as provas. A aplicação deve mostrar mensagens informando a média de cada aluno e a média geral da turma.

Para a definição das notas, utilize as seguintes informações:

Aluno 1	Prova 1	Nota Parte 1	**4.0**
		Nota Parte 2	**2.5**
	Prova 2	Nota Parte 1	**1.0**
		Nota Parte 2	**7.0**
Aluno 2	Prova 1	Nota Parte 1	**6.5**
		Nota Parte 2	**3.5**
	Prova 2	Nota Parte 1	**0.0**
		Nota Parte 2	**3.0**
Aluno 3	Prova 1	Nota Parte 1	**5.0**
		Nota Parte 2	**4.0**
	Prova 2	Nota Parte 1	**6.0**
		Nota Parte 2	**1.5**

13. Crie uma classe Lampada que possui um atributo ligada, que indica se a lâmpada está ligada ou desligada.

Ao construir uma lâmpada, o estado dela (ligada ou desligada) deve ser fornecido. Para ligar e desligar a lâmpada, os métodos Ligar() e Desligar() devem ser chamados, respectivamente. Aliás, esta é a única forma de alterar o estado da lâmpada, já que o atributo ligada não pode ser acessado de fora da classe.

A lâmpada também possui um método Imprimir(). Quando chamado, ele mostra as mensagens “Lâmpada ligada” ou “Lâmpada desligada”, dependendo do estado atual.

Construa uma aplicação que cria uma lâmpada ligada, muda o estado dela e também imprime o estado atual após cada chamada a Ligar() e Desligar().

14. Crie uma classe Data que possui dois construtores. O primeiro recebe um dia, mês e ano e o segundo, além destas informações, recebe também uma hora, minuto e segundo (a hora fornecida deve estar entre 0 e 23). É importante que este segundo construtor invoque o primeiro para evitar a duplicação de código.

Os construtores devem armazenar os dados fornecidos como parâmetros em atributos privados. Estes atributos devem ter seus valores expostos para fora da classe usando read-only properties.

A classe Data deve ter também um método Imprimir(), utilizado para imprimir a data e hora representados pelo objeto. Este método recebe como parâmetro o formato de hora que deve ser utilizado para imprimir as horas (12 ou 24h). Se o objeto for construído sem informação de horário, este parâmetro não afeta a impressão.

Os formatos da hora são do tipo int, mas devem ser representados por duas constantes na classe Data: FORMATO_12H e FORMATO_24H.
Para entender melhor o funcionamento do método Imprimir(), observe como ele deve se comportar em diversas situações:

Código	Resultado

`Data d1 = new Data(10, 03, 2000, 10, 30, 10);`
`d1.imprimir(Data.FORMATO_12H);`
`d1.imprimir(Data.FORMATO_24H);`
	`10/3/2000 10:30:10 AM`
	`10/3/2000 10:30:10`

`Data d2 = new Data(15, 06, 2000, 23, 15, 20);`
`d2.imprimir(Data.FORMATO_12H);`
`d2.imprimir(Data.FORMATO_24H);`
	`15/6/2000 11:15:20 PM`
	`15/6/2000 23:15:20`

`Data d3 = new Data(5, 10, 2005);`
`d3.imprimir(Data.FORMATO_12H);`
`d3.imprimir(Data.FORMATO_24H);`
	`5/10/2005`
	`5/10/2005`

15. Crie duas classes: Ponto2D e Ponto3D. Ponto2D possui como atributos as coordenadas x e y, enquanto Ponto3D, além delas, também possui a coordenada z. Utilize a relação de herança para representar estas classes.

A respeito dos construtores, Ponto2D deve ter apenas um construtor, que recebe os valores de x e y como parâmetros (tipo double). Já Ponto3D também deve ter apenas um construtor, que deve receber x, y e z como parâmetros (também do tipo double). Dica: Se a relação de herança e a declaração dos construtores foram feitas corretamente, você deverá, obrigatoriamente, chamar o construtor da superclasse explicitamente.

Ambas as classes devem implementar o método Imprimir(), que exibe no console os valores das coordenadas do objeto.

16. O C# possui uma interface chamada ICloneable, que pode ser implementada por classes que são capazes de gerar cópias de objetos. Classes que implementam esta interface devem implementar o método Clone(). Dentro deste método é implementada a lógica para criar um novo objeto com base no objeto original.

Com base nisto, crie uma classe Porta que suporta a criação de novos objetos (cópia). Ela deve ter os atributos altura (double), largura (double) e aberta (boolean). Também deve possuir os métodos Abrir(), Fechar() e os valores dos atributos devem ser expostos para fora da classe através de read-only properties.

Como uma porta pode criar outras cópias dela mesma, você deve implementar o método Clone() na classe, o qual deve criar um novo objeto com os valores dos atributos copiados e retorná-lo.

17. Crie a classe Figura que representa figuras geométricas, representadas pelas classes Quadrado e Retangulo. Uma figura pode ter sua área calculada a partir do método CalcularArea(), que retorna a área calculada da figura em forma de um double.

Crie também a classe FiguraComplexa. Uma figura complexa é também uma figura, mas a diferença é que ela é composta por várias figuras (quadrados, retângulos ou até outras figuras complexas). Para calcular a área de uma figura complexa, basta somar a área de todas as figuras que a compõem.

Para executar a aplicação, crie a classe Program, que é responsável por criar uma figura complexa e calcular a sua área. Esta figura deve ser composta por:

- 1 quadrado com 3 de lado
- 1 quadrado com 10 de lado
- 1 retângulo com lados 2 e 7
- 1 retângulo com lados 5 e 3

Dica: Perceba a diferença entre uma classe ser uma figura e ter uma ou mais figuras. A primeira relação é de herança, enquanto a segunda implica em uma composição.

18. Implemente a classe Colecao e duas subclasses: Pilha e Fila. Uma coleção tem um array de dados que fazem parte da coleção.

Tanto a pilha como a fila são coleções. A diferença entre elas está na disciplina de acesso. Na pilha, o último elemento inserido é o primeiro a ser removido (como numa pilha de pratos). Na fila, o primeiro elemento inserido é o primeiro a ser removido (como numa fila de banco).

Os métodos da classe Colecao responsáveis por estas operações são:

- void InserirItem(object item)
- object RemoverItem()

Crie um método que recebe uma coleção, adiciona alguns elementos e remove estes mesmo elementos. Imprima os elementos removidos e veja a diferença no resultado.

19. Crie uma classe Veiculo com um field ligado (privado), que indica se o carro está ligado ou não. Esta classe deve ter também os métodos Ligar() e Desligar(), que definem o valor para este field, e uma read-only property para retornar o valor do field.

Depois crie três subclasses de Veiculo: Automovel, Motocicleta e Onibus. Cada classe destas deve sobrescrever os métodos Ligar() e Desligar() e deve imprimir mensagens como “Automóvel ligado”, “Motocicleta desligada”, etc. Para manter a consistência do modelo, descubra como fazer para que o atributo ligado de Veiculo tenha o valor correto quando os métodos são chamados.

Crie uma aplicação que instancia três veículos, um de cada tipo, e chama os métodos Ligar(), Desligar() e a property Ligado. O resultado obtido deve ser consistente com o que o modelo representa. Por exemplo, ao chamar o método Ligar() de um Automovel, é esperado que a property Ligado retorne true.

20. Crie uma classe ContaBancaria, que representa uma conta bancária genérica que não pode ser instanciada. Esta classe deve ter uma property Saldo (visível apenas para ela e para as suas subclasses) e os métodos Depositar(double), Sacar(double) e Transferir(double, ContaBancaria). Estes métodos devem depositar um valor na conta, sacar um valor da conta e transferir um valor da conta de origem para uma conta de destino, respectivamente.

Além destes, ContaBancaria deve ter um método CalcularSaldo(). Este método possui a regra do cálculo do saldo final (que pode ser diferente do saldo armazenado em Saldo) e deve ser obrigatoriamente implementado pelas subclasses de ContaBancaria, pois cada classe possui suas próprias regras de cálculo.

Crie duas subclasses de ContaBancaria: ContaCorrente e ContaInvestimento. Cada uma deverá implementar suas regras para calcular o saldo (método CalcularSaldo()). No caso de ContaCorrente, o saldo final é o saldo atual subtraído de 10%, referente a impostos que devem ser pagos. Já para a ContaInvestimento, o saldo final é o saldo atual acrescido de 5%, referente aos rendimentos do dinheiro investido.

Crie uma aplicação que instancia uma conta corrente e uma conta investimento e executa as operações de depósito, saque, transferência e cálculo de saldo. Verifique se os resultados obtidos são consistentes com a proposta do modelo e com as regras de cálculo estabelecidas.