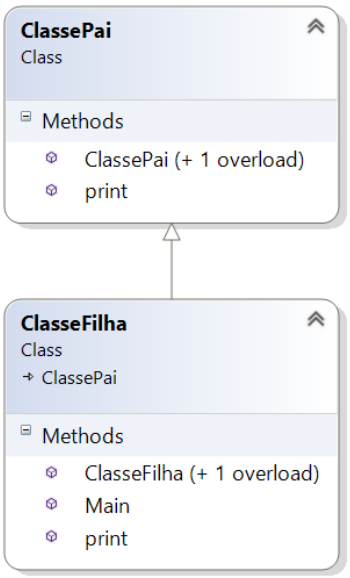
Ficha de Exercícios Nº2 (POO – C#):

Os exercícios desta ficha destinam-se a efetuar uma revisão de programação orientada a objetos com C#. Pretende-se rever alguns tópicos como ***herança***, ***polimorfismo*** *da linguagem C#.*

1. Analise o código abaixo e verifique o output que o mesmo apresenta na página seguinte. Existem duas classes, a *ClasseFilha* e *ClassePai*, com a estrutura e métodos apresentados na seguinte figura.



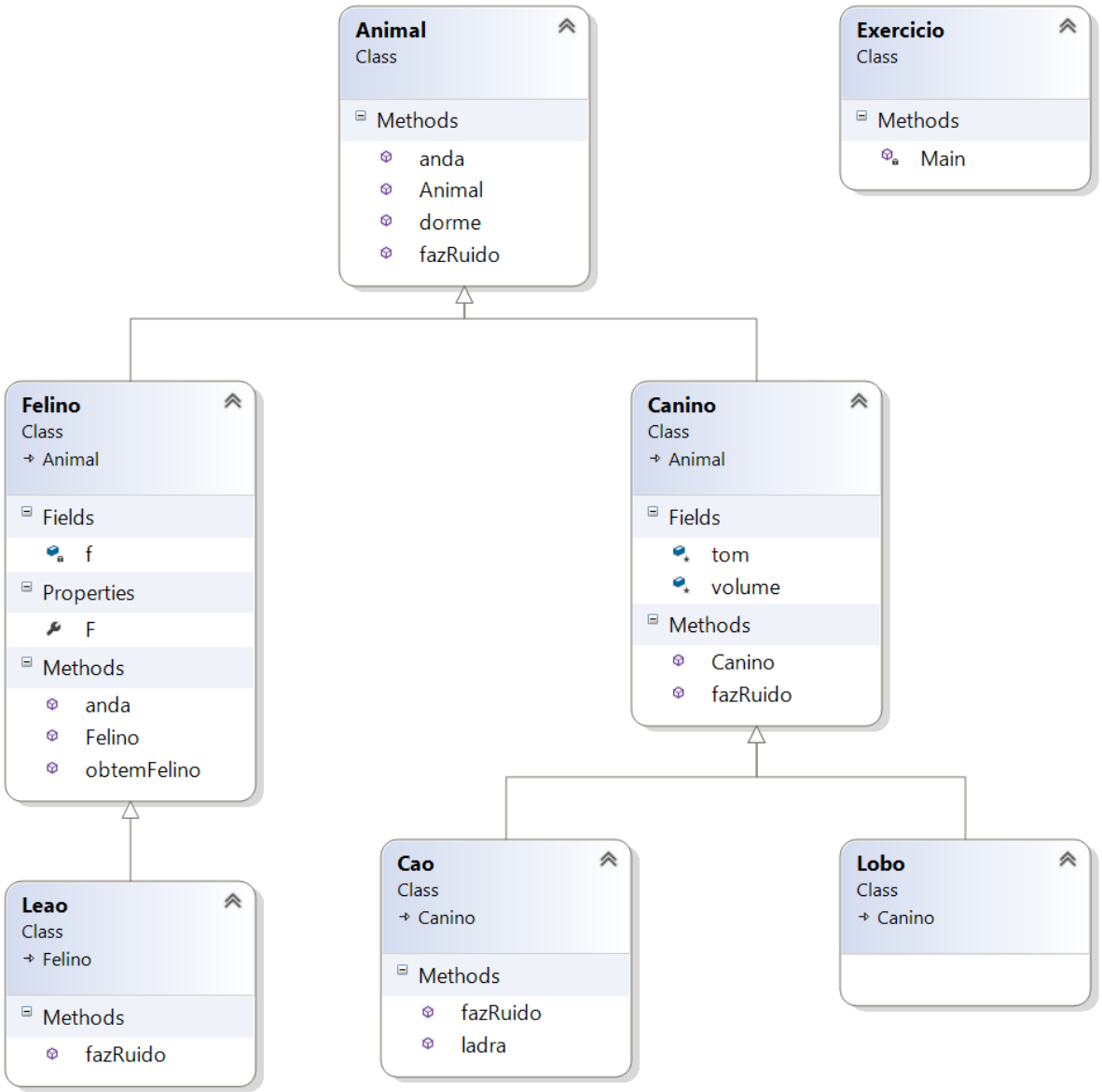

```
using System;

namespace Heranca
{
    public class ClassePai
    {
        public ClassePai()
        {
            Console.WriteLine("Construtor Pai");
        }
        public ClassePai(string texto)
        {
            Console.WriteLine("Construtor Pai " + texto);
        }
        public void print()
        {
            Console.WriteLine("Print() da classe Pai.");
        }
    }

    public class ClasseFilha : ClassePai
    {
        public ClasseFilha() : base("Mensagem Ola...")
        {
            Console.WriteLine("Construtor Filha");
        }
        public ClasseFilha(string x) : base(x)
        {
            Console.WriteLine("Construtor Filha com string");
        }
        public new void print()
        {
            base.print();
            Console.WriteLine("Print() da classe filha");

        }
        public static void Main()
        {
            ClasseFilha filha = new ClasseFilha();

            ClasseFilha filha2 = new ClasseFilha("String Parametro");
            filha.print();
            ((ClassePai)filha).print();

            Console.ReadKey();
        }
    }

}
```

Output:

```
Construtor Pai Mensagem Ola...
Construtor Filha
Construtor Pai String Parametro
Construtor Filha com string
Print() da classe Pai.
Print() da classe filha
Print() da classe Pai.
```

Nota: C# suporta a herança de apenas uma classe.

2. Considere o código abaixo e a seguinte estrutura de classes e métodos:



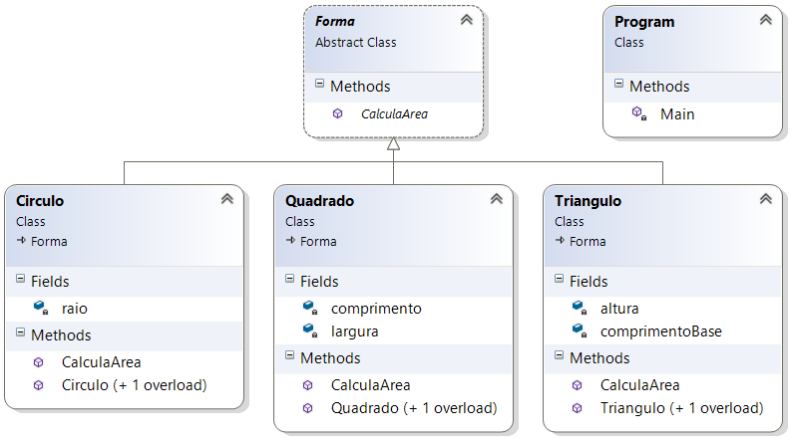
a. Analise e especifique **no papel** qual o output do programa apresentado na página seguinte.

b. Execute o *Visual Studio*, crie um projeto *Visual C# > Windows > Console Application* com nome `Exercicio`, copie o código e compare o resultado obtido com a sua solução da alínea a). Caso existam diferenças, tente compreender o que correu mal…

```
using System;
namespace Exercicio {

    public class Animal
    {
        public Animal()  {    Console.WriteLine("\nCria animal..."); }
        public void dorme() { Console.WriteLine("Dorme.. zzzzzzzz"); }
        public virtual void fazRuido() { Console.WriteLine("Ruido...");    }
        public void anda(int metros)   { Console.WriteLine("Anda por aí..."); }
    }

    public class Felino : Animal {

        static int f = 1;
        public int F {  get { return f; } }

        public Felino() {   Console.WriteLine("Criei Felino...");  f++;
        }
        public void anda() {  Console.WriteLine("Andando por aí sozinho!...");
        }
        public int obtemFelino() {  return f; }
    }

    public class Canino : Animal {
        protected int volume;
        protected int tom;
        public Canino() {  Console.WriteLine("Criei Canino...");
                           volume = 0; tom = 0;
        }
        public int fazRuido(int volume, int tom) {
             return (volume * tom); }
    }
    public class Leao : Felino {
        public override void fazRuido() {
                 Console.WriteLine("Ruge..... Rrrrrrr!"); }
    }

    public class Cao : Canino {
        public int fazRuido(int tom) {  return (volume * tom);
        }
        public void ladra(int v, int t) {   volume = v;  tom = t;
        }
    }

    public class Lobo : Canino { }
```

```
class Exercicio {

        static void Main(string[] args)  {
            int x;

            Cao kiko= new Cao();
            kiko.fazRuido();

            Leao leonce = new Leao();
            leonce.fazRuido();
            x = Kiko.fazRuido(5, 5);
            kiko.ladra(10, 10);
            Console.WriteLine("Valor 1 = {0}, Valor 2 = {1}", x, kiko.fazRuido(3));

            leonce.dorme();
            leonce.fazRuido();
            leonce.anda();

            Felino f = new Felino();
            Console.WriteLine("Valor1 = {0}, Valor 2 = {1}", f.F, f.obtemFelino());

            Lobo lobinho = new Lobo();
        }
    }
}
```

3. Implemente um programa, que usando os conceitos de POO em C#, implemente a área de um círculo, de um triângulo e de um quadrado.



Implemente os seguintes elementos:

a. Deve criar uma classe de nome ***Forma***, e esta deve ter um método abstrato de nome *CalculaArea()*

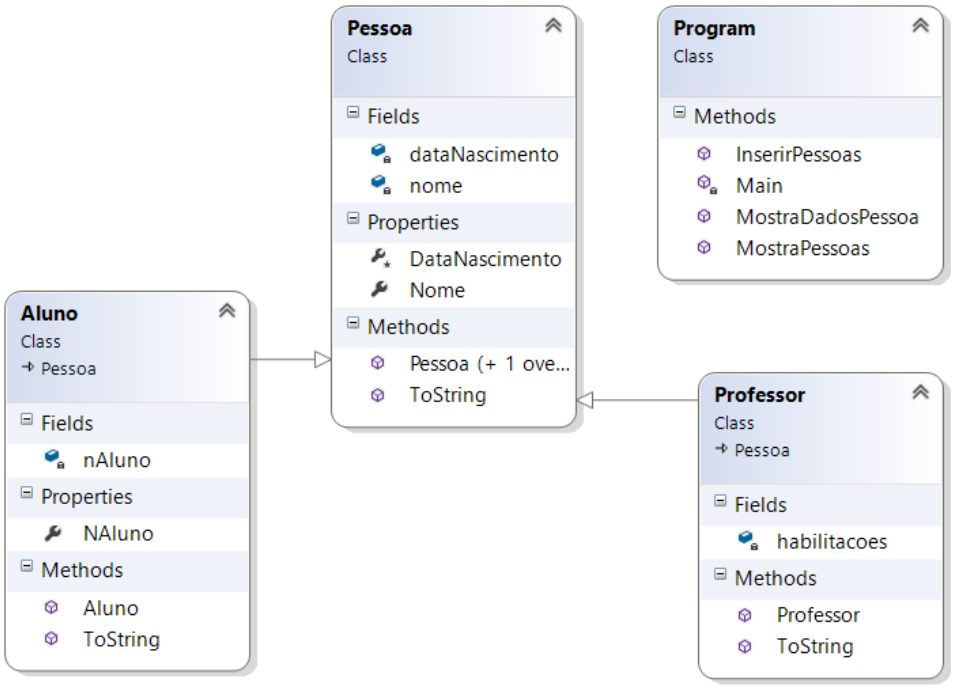
b. Sabendo que as formas Quadrado, Triangulo e Circulo, terão atributos diferentes para calculo da área, crie as seguintes classes que herdam da classe *Forma*
   i. ***Quadrado***: Deverão ser especificados dois atributos *comprimento* e *largura*
   ii. ***Triangulo***: Deverão ser especificados dois atributos *comprimentoBase* e *altura*
   iii. ***Circulo***: Deverá ser especificado o atributo *raio*

c. Para cara uma destas classes especifique os construtores e o que considere necessário para que as mesmas fiquem funcionais.

d. Crie no programa principal, um objeto de cada tipo, especificando valores para que seja apresentada a área dos mesmos, como por exemplo:

```
Area do quadrado: 50 unidades
Area do triangulo: 160 unidades
Area do circulo: 153 unidades
```

4. Considere a seguinte estrutura de classes, e implemente alíneas que se seguem:



4.1 Considere a seguinte estrutura de classes, e implemente alíneas que se seguem:

a. Classe ***Pessoa***

i. Especifique os atributos *nome* do tipo *String* e *dataNascimento* do tipo *DateTime*

ii. Especifique as propriedades apenas de leitura de cada um destes atributos.

iii. Crie um construtor, de forma que receba por parâmetro o nome e a data de nascimento. Por omissão, devem ser assumidos os valores nome=”Desconhecido” e para a Data de Nascimento, deve ser criada uma instância de DateTime sem parâmetros.

iv. Especifique o método ***ToString()*** de forma a retornar uma string composta pelo nome e data de nascimento.

b. Especifique uma classe **Aluno**, seguindo a hierarquia apresentada na figura anterior.

i. Especifique o atributo e a propriedade apenas de leitura para o número de aluno;

ii. Especifique um construtor para o aluno;

iii. Especifique o método ***ToString()*** de forma a retornar uma *string* não só o texto do método ToString existente na classe pessoa, mas também com o número do aluno.

c. Especifique uma classe **Professor**, seguindo a hierarquia apresentada na figura anterior

i. Especifique o atributo e a propriedade apenas de leitura para habilitações;

ii. Especifique um construtor para o professor;

iii. Especifique o método ***ToString()*** de forma a retornar uma *string* não só o texto do método *ToString* existente na classe pessoa, mas incluindo também as habilitações do professor.

d. Na classe principal

i. Especifique o método **InserirPessoas()** que solicita dados ao utilizador, e devolve uma lista das pessoas();

ii. Especifique o método **MostraPessoas** que recebe por parâmetro uma lista de *Pessoas* e percorre a lista, apresentando no ecrã os dados da pessoa, com recurso ao método `MostraDadosPessoa(Pessoa pessoa)`

4.2 Após implementar a alínea anterior, efetue as seguintes alterações ao projeto desenvolvido:

a. Classe ***Pessoa***

iii. A classe deve ter um novo atributo id, sendo que este deverá guardar o indentificador único para cada objecto da classe pessoa e deve ser gerado de forma automática, sequencialmente.

iv. Os atributos *nome, datade Nascimento e id*, devem ser acedidos publicamente através das respectivas propriedades.

1. O Id deverá ser só de leitura;
2. A propriedade DatadeNascimento, deve garantir que a data seja um formato correcto e além disso, a propriedade deve garantir que só se aceitem datas de nascimento entre 1/1/1900 e a data atual, caso contrário devem ser geradas excepções:

```
Dica: throw new FormatException("Formato de Data Incorrecto!");
      throw new ArgumentOutOfRangeException(null, "Data com valores
                                  fora dos parâmetros!");
```

v. O método ToString deverá também apresentar o id da pessoa;

vi. Efetue alteração de forma a que não seja possível criar instâncias da classe Pessoa. Altere o programa principal para que este funcione corretamente,

vii. Especifique uma propriedade pública de nome GetNextId, apenas de leitura, que permita obter o próximo Id (não se pretende que seja necessário criar um objecto Pessoa para usar esta propriedade).

e. Altere a classe ***Professor e Aluno*** de forma que os seus atributos sejam acessível através de propriedades publicas.

f. Crie uma classe ***Turma***

i. Devem ser especificados atributos como *nome*, *tipo* (que podem ser Teorica, Pratica ou Laboratorial), *numero*, um *professor* e uma *lista de alunos*.

ii. Especifique as propriedades de forma a aceder aos atributos;

iii. Especifique o construtor e os métodos que considere necessários;

iv. Implemente no programa principal o que considere necessário de forma a poder testar a classe Turma.