Nome: _____ Nr: _____

Cada pergunta certa contabiliza 2 valores, **errada 1 valor negativo**, sem resposta 0 valores.

1. Dado: `class A { public string Foo() { return "Foo";} }`, considere: `A a = new A(); a.Foo();`
   O resultado da compilação de `a.Foo()` é: `ldloc.0; callvirt instance string A::Foo();`
   Se substituirmos `callvirt` por `call`, qual será o resultado da execução destas instruções:
   - ☐ Excepção `System.NullReferenceException`
   - ☐ Excepção `System.AccessViolationException`
   - ☐ Excepção `System.InvalidProgramException: Common Language Runtime detected an invalid program`
   - ✓ Execução sem excepções.

2. Dado: `class A { public string Foo() { return "Foo";} }`, considere: `A a = new A(); a.Foo();`
   O resultado da compilação de `a.Foo()` é: `ldloc.0; callvirt instance string A::Foo();`
   Qual é o resultado da execução de: **`ldnull; call`** `instance string Ficha03.A::Foo();`
   - ☐ Excepção `System.NullReferenceException`
   - ☐ Excepção `System.AccessViolationException`
   - ☐ Excepção `System.InvalidProgramException: Common Language Runtime detected an invalid program`
   - ✓ Execução sem excepções.

3. Dado: `class X { } class Y : X {}`, o construtor de Y resultante da compilação tem as instruções:
   `ldarg.0; call instance void X::.ctor(); ret;`
   - ☐ Não. Em vez de `call` tem `callvirt` porque o construtor é um método de instância.
   - ☐ Não. Em vez de `call` tem `callspecial` porque é esta a instrução IL usada na chamada a construtores.
   - ☐ Sim, porque o construtor é um método virtual.
   - ✓ Sim, porque `arg.0` não é `null`.

4. ```
   class X { public virtual string Foo() { return "X";  } }
   class Y : X { public override string Foo() { return base.Foo() + "Y"; } }
   ```
   O resultado da compilação de base.Foo() é:
   - ✓ `ldarg.0; call instance string Ficha03.X::Foo();`
   - ☐ `ldarg.0; callvirt instance string Ficha03.X::Foo();`
   - ☐ `ldarg.0; invoke instance string Ficha03.X::Foo();`
   - ☐ `ldarg.0; base.call instance string Ficha03.X::Foo();`

5. Qual o resultado da compilação de: `delegate void SomeHandler();`
   - ✓ Uma classe `SomeHandler`
   - ☐ Uma interface `SomeHandler`
   - ☐ Uma classe abstracta `SomeHandler`
   - ☐ Nenhuma das opções.
   ```

6.

```
delegate void SomeHandler();
```

```
class Program {
  static void println() { }
  static void Test(SomeHandler h) { }
  static void Main(string[] args) {
      Test(Program.println);
  }
}
```

Escreva TODAS as instruções IL resultantes da compilação de **`Test(Program.println)`**

⇔ **`Test(new SomeHander(Program.println))`**

```
ldnull
ldftn    Program::println
newobj   SomeHandler::ctor(object, int)
call     Program::Test
```

7.

```
interface I { void Foo();  }
class A: I { public void Foo() { Console.WriteLine("A");  } }
class B : A { public virtual void Foo() { Console.WriteLine("B"); } }
class C : B { public override void Foo() { Console.WriteLine("C"); } }
class D : C { public void Foo() { Console.WriteLine("D"); } }
```

```
static void Test() {
   I i = new C();
   i.Foo();
}
```

O output da execução da função `Test` é:

- ✓ A
- ☐ B
- ☐ C
- ☐ D

8. Se acrescentarmos à declaração da classe B: `class B : A, I {}` mantendo a mesma implementação, então o output da execução da função `Test` passa a ser:

- ☐ A
- ☐ B
- ✓ C
- ☐ D

9. Dada: `class A<T> { public void Foo<S, U>(T val, S x)  { } }` e uma variável `a`:
`A<string> a = new A<string>()` Qual a **única** utilização que NÃO dá erro de compilação:

- ☐ `a.Foo("ola", 91)`
- ☐ `a.Foo<string>("ola", 91)`
- ✓ `a.Foo<int, bool>("ola", 91)`
- ☐ `a.Foo<string, int>("ola", 91);`

10. Seja `EnumeratorConverter<T, R>` uma classe que implementa a interface `IEnumerator<R>` representando uma sequência de elementos de uma fonte (`src`) transformados por uma função (`transf`).
Sendo `src` e `transf` campos de `EnumeratorConverter<T, R>` do tipo `IEnumerator<T>` e `Func<T, R>`, qual a única implementação INCORRECTA da sua propriedade: `public R Current { get { ... } }`

- ✓ `src.MoveNext();  return transf(src.Current);`
- ☐ `return transf(src.Current);`
- ☐ `return transf.Invoke(src.Current);`
- ☐ `T v = src.Current;  return transf(v);`