NOM POUPA

Prénom Adrien

Promo M1 IL/SE 2018

Date 15/12/2016

POUPA Adrien
M1 - 2016

MATIÈRE C#

13

QCM/MCQ :

1. D ✓
2. B ✓
3. D ✗
4. E. ✗ I do not know if this is a typo or if it is done on purpose, but we should declare the RollNo property with a get accessor only, not rollNo.
5. C ✓
6. D ✓
7. C ✓
8. E ✗
9. B ✓
10. A ✓
11. B ✓
12. E ✗
13. C ✓
14. D ✗
15. A ✓
16. B ✓
17. B ✗
18. C ✓

⇒

19. Let's create the following two classes:

```
namespace test {
  class Animal {

    public string virtual whoAmI() {
        return "I am an animal";
    }
  }

  class Zebra : Animal {

    public string override whoAmI() {
        return "I am a Zebra";
    }
  }

class test {

    public static void Main(string[] args) {

        Animal a = new Animal();
        Console.WriteLine(a.whoAmI()); // I am an animal

        Zebra z = new Zebra();
        Console.WriteLine(z.whoAmI()); // I am a zebra

        Animal h = new Zebra();
        Console.WriteLine(h.whoAmI()); // I am a zebra
```

```
/* As we can see, this is late binding. It is done
at execution and allows true, natural polymorphism
like in Java ← We can do useful stuff like
this: /* annotation @Override

List<Animal> l = new List<Animal>();
l.Add(a);
l.Add(h); // Adding a Zebra in an Animal list
}

}
```

It is opposed to early binding, which is not about true polymorphism but rather "masking" using the "new" keyword:

```
namespace tst{
    class Animal{
        public string whoAmI(){
            return "I am an animal";
        }
    }
    class Zebra : Animal{
        public string new whoAmI(){
            return "I am a zebra";
        }
    }
    class tst{
        public static void Main(string[] args){
            // The first 2 examples output are the same.
            Animal h = new Zebra();
            Console.WriteLine(h.whoAmI()); // I am an animal
}}}
```

Early binding is the default behavior in C# (ie when omitting the "new" keyword, compiler emits a warning that early binding will be used).

Unlike dynamic/late binding, it does not allow using parent types as a generic type for all childs.

It is done at compilation and will not be aware of programmer's intentions unlike dynamic binding.

Programmers should use dynamic binding as much as possible.

Adoie

Fadi

C# EXAM

Documents forbidden

- **Encircle the correct answer (2 pts/multiple choice question)**

1. How will you complete the foreach loop in the C#.NET code snippet given below such that it correctly prints all elements of the array a?

```
int[][]a = new int[2][];
a[0] = new int[4]{6, 1 ,4, 3};
a[1] = new int[3]{9, 2, 7};
foreach (int[ ] i in a)
{
    /* Add loop here */
    Console.Write(j + " ");
    Console.WriteLine();
}
```

6 1 4 3

9 2 7

A. *foreach (int j = 1; j < a(0).GetUpperBound; j++)*
B. *foreach (int j = 1; j < a.GetUpperBound (0); j++)*
C. *foreach (int j in a.Length)* ✓
D. *foreach (int j in i)* ✗
E. *foreach (int j in a.Length -1)*

2. If a *Student* class has an indexed property which is used to store or retrieve values to/from an array of 5 integers, then which of the following are the correct ways to use this indexed property?

1. `Student[3] = 34;` ✗
2. `Student s = new Student();`
   `s[3] = 34;` ✓
3. `Student s = new Student();`
   `Console.WriteLine(s[3]);` ✗
4. `Console.WriteLine(Student[3]);` ✗
5. `Student.this s = new Student.this();`
   `s[3] = 34;`

A. 1, 2
B. 2, 3 ✓
C. 3, 4
D. 3, 5

3. Which of the following statements are correct about the C#.NET code snippet given below?

```
int[] a = {11, 3, 5, 9, 4};
```

1. The array elements are created on the stack.
2. Refernce a is created on the stack.
3. The array elements are created on the heap.
4. On declaring the array a new array class is created which is derived from *System.Array* Class.
5. Whether the array elements are stored in the stack or heap depends upon the size of the array.

A. 1, 2
B. 2, 3, 4
C. 2, 3, 5
D. 4, 5
E. None of these

4. A *Student* class has a property called *rollNo* and *stu* is a reference to a *Student* object and we want the statement *stu.RollNo = 28* to fail. Which of the following options will ensure this functionality?

A. Declare *rollNo* property with both get and set accessors.
B. Declare *rollNo* property with only set accessor.
C. Declare *rollNo* property with get, set and normal accessors.
D. Declare *rollNo* property with only get accessor.
E. None of the above

5. If a class *Student* has an indexer, then which of the following is the correct way to declare this indexer to make the C#.NET code snippet given below work successfully?

```
Student s = new Student();
s[1, 2] = 35;
```

A.

```
class Student
{
    int[ ] a = new int[5, 5];
    public property WriteOnly int this[int i, int j]
    {
        set
        {
            a[i, j] = value;
        }
```

```
    }
}
```

B.

```
class Student
{
    int[ , ] a = new int[5, 5];
    public int property WriteOnly
    {
        set
        {
            a[i, j] = value;
        }
    }
}
```

C.

```
class Student
{
    int[ , ] a = new int[5, 5];
    public int this[int i, int j]
    {
        set
        {
            a[i, j] = value;
        }
    }
}
```

D.

```
class Student
{
    int[ , ] a = new int[5, 5];
    int i, j;
    public int this
    {
        set
        {
            a[i, j] = value;
        }
    }
}
```

6. Which of the following is the correct way to implement a write only property *Length* in a *Sample* class?

A.

```
class Sample
{
    public int Length
    {
        set
        {
            Length = value;
        }
    }
}
```

B.

```
class Sample
{
    int len;
    public int Length
    {
        get
        {
            return len;
        }
        set
        {
            len = value;
        }
    }
}
```

C.

```
class Sample
{
    int len;
    public int Length
    {
        WriteOnly set
        {
            len = value;
        }
    }
}
```

D.

```
class Sample
{
    int len;
    public int Length
    {
        set
        {
            len = value;
        }
    }
}
```

19. Explain dynamic/late binding, its benefits using an interesting example in C#. [12 pts]

7. Which of the following statements is correct about the C#.NET code snippet given below?

```
class Student s1, s2; // Here 'Student' is a user-defined class.
s1 = new Student();
s2 = new Student();
```

A. Contents of *s1* and *s2* will be exactly same.
B. The two objects will get created on the stack.
C. Contents of the two objects created will be exactly same.
D. The two objects will always be created in adjacent memory locations.
E. We should use *delete()* to delete the two objects from memory.

8. Which of the following statements are correct?

1. Instance members of a *class* can be accessed only through an object of that *class*.
2. A *class* can contain only instance data and instance member *function*.
3. All objects created from a *class* will occupy equal number of bytes in memory.
4. A *class* can contain Friend functions.
5. A *class* is a blueprint or a template according to which objects are created.

A. 1, 3, 5
B. 2, 4
C. 3, 5
D. 2, 4, 5
E. None of these

9. Which of the following statements are correct about the *this* reference?

1. *this* reference can be modified in the instance member function of a class.
2. Static functions of a class never receive the *this* reference.
3. Instance member functions of a class always receive a *this* reference.
4. *this* reference continues to exist even after control returns from an instance member function.
5. While calling an instance member function we are not required to pass the *this* reference explicitly.

A. 1, 4
B. 2, 3, 5
C. 3, 4
D. 2, 5
E. None of these

10. Which of the following statements are correct about static functions?

1. Static functions can access only static data.
2. Static functions cannot call instance functions.
3. It is necessary to initialize static data.
4. Instance functions can call static functions and access static data.
5. *this* reference is passed to static functions.

A. 1, 2, 4
B. 2, 3, 5
C. 3, 4
D. 4, 5
E. None of these

11. Which of the following statements are correct about constructors in C#.NET?

1. Constructors cannot be overloaded.
2. Constructors always have the name same as the name of the class.
3. Constructors are never called explicitly.
4. Constructors never return any value.

A. 1, 3
B. 2, 3, 4
C. 3
D. 4
E. None of these

12. Which of the following statements should be added to the subroutine fun( ) if the C#.NET code snippet given below is to output 9 13?

```
class BaseClass
{
    protected int i = 13;
}
class Derived: BaseClass
{
    int i = 9;
    public void fun()
    {
        // [*** Add statement here ***]
    }
}
```

A. *Console.WriteLine(base.i + " " + i);*
B. *Console.WriteLine(i + " " + base.i);*
C. *Console.WriteLine(mybase.i + " " + i);*
D. *Console.WriteLine(i + " " + mybase.i);*
E. *Console.WriteLine(i + " " + this.i);*

on vet 9 13

13. How can you prevent inheritance from a class in C#.NET ?

A. Declare the class as *shadows*.
B. Declare the class as *overloads*.
C. Declare the class as *sealed*.
D. Declare the class as *suppress*.
E. Declare the class as *override*.

14. Which of the following statements is incorrect about delegate?

A. Delegates are reference types.
B. Delegates are object oriented.
C. Delegates are type-safe.
D. Delegates serve the same purpose as function pointers in C and pointers to member function operators in C++.
E. Only one method can be called using a delegate.

15. Which of the following is the necessary condition for implementing delegates?

A. Class declaration
B. Inheritance
C. Run-time Polymorphism
D. Exceptions
E. Compile-time Polymorphism

16. Which of the following statements are correct about the delegate declaration given below?

```
delegate void del(int i);
```

1. On declaring the delegate a class called *del* will get created.
2. The signature of del need not be same as the signature of the method that we intend to call using it.
3. The *del* class will be derived from the *MulticastDelegate* class.
4. The method that can be called using del should not be a *static* method.
5. The *del* class will contain a one-argument constructor and an *Invoke()* method.

A. 1, 2 and 3 only
B. 1, 3 and 5 only
C. 2 and 4 only
D. 4 only
E. All of the above

17. Which of the following is the correct way to call *MyFun()* of the *Sample* class given below?

```
class Sample
{
    public void MyFun(int i, Single j)
    {
        Console.WriteLine("Welcome to IndiaBIX !");
    }
}
```

A.
```
delegate void del(int i);
Sample s = new Sample();
del d = new del(s.MyFun);
d(10, 1.1f);
```

B.
```
delegate void del(int i, Single j);
del d;
Sample s = new Sample();
d = new del(s.MyFun);
d(10, 1.1f);
```

C.
```
Sample s = new Sample();
delegate void d = new del(MyFun);
d(10, 1.1f);
```

D.
```
delegate void del(int i, Single]);
Sample s = new Sample();
del = new delegate(ref MyFun);
del(10, 1.1f);
```

18. Which of the following are the correct ways to declare a delegate for calling the function *func()* defined in the sample class given below?

```
class Sample
{
    public int func(int i, Single j)
    {
        /* Add code here. */
    }
}
```

A. *delegate d(int i, Single j);*
B. *delegate void d(int, Single);*
C. *delegate int d(int i, Single j);*
D. *delegate void (int i, Single j);*
E. *delegate int sample.func(int i, Single j);*