1.

В результате выполнения фрагмента программы:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Xml.Serialization;

class A {
    public int y;

    public A() { }
}

class Program {
    static void Main() {
        XmlSerializer xml = new
            XmlSerializer(typeof(List<A>));
        FileStream fs = new FileStream("out.xml",
FileMode.Create);
        using (new StreamWriter(fs)) {
            List<A> list = new List<A>();
            for (int i = 0; i < 5; i++) {
                list.Add(new A());
                list[i].y = i + 1;
            }
            xml.Serialize(fs, list);
        }
        fs = new FileStream("out.xml", FileMode.Open);
        List<A> list2 = (List<A>)xml.Deserialize(fs);
        for (int i = 0; i < list2.Count; i++) {
            Console.Write(list2[i].y);
        }
    }
}
```

на экран будет выведено:

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

2.

В результате выполнения фрагмента программы:

```
using System;
using System.IO;
using System.Xml.Serialization;

public class A {
    public int x = 5;
    [NonSerialized]
    public int y = 7;

    public A() { }
}

class Program {
    static void Main() {
        XmlSerializer xml = new XmlSerializer(typeof(A));
        FileStream fs = new FileStream("out.xml",
FileMode.Create);
        using (new StreamWriter(fs)) {
            A a = new A();
            a.x = a.y + 10;
            xml.Serialize(fs, a);
        }
        fs = new FileStream("out.xml", FileMode.Open);
        A a2 = (A)xml.Deserialize(fs);
            Console.Write(a2.y + a2.x);
    }
}
```

на экран будет выведено:

3.

Выберите верные утверждения (*укажите все верные ответы***):**

1) XML-сериализация требует атрибут [Serializable].
2) XML-сериализация сериализует открытые и внутрисборочные поля класса.
3) XML-сериализация игнорирует атрибут [NonSerialized].
4) В JSON-сериализации нужно классы или структуры помечать атрибутом [DataMember], а их члены – [DataContract].
5) Бинарная сериализация требует наличия конструктора без параметров.

4.

В результате выполнения фрагмента программы:

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization.Json;
using System.Runtime.Serialization;
using System.IO;

[DataContract]
public class A {
    [DataMember]
    internal int x = 5;
}
[DataContract]
class B : A {
    public int x;
}

class Program {
    static void Main() {
        DataContractJsonSerializer bf = new
DataContractJsonSerializer(typeof(List<A>));
        FileStream fs = new FileStream("out.xml",
FileMode.Create);
        using (new StreamWriter(fs)) {
            List<A> list = new List<A>();
            for (int i = 0; i < 5; i++) {
                list.Add(new A());
                list[i].x = i + 1;
            }
            bf.WriteObject(fs, list);
        }
        fs = new FileStream("out.xml", FileMode.Open);
        bf = new DataContractJsonSerializer(typeof(List<B>));
        List<B> list2 = (List<B>)bf.ReadObject(fs);
        for (int i = 0; i < list2.Count; i++) {
            Console.Write(list2[i].x);
        }
    }
}
```

на экран будет выведено:

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

5.

Существуют следующие виды сериализации (*укажите все верные ответы***):**
1) XML. 2) Бинарная. 3) XAML. 4) SOAP. 5) JSON.

6.

Про XML-сериализацию верно (*укажите все верные ответы***):**
1) Сериализует поля и методы. 2) При десериализации НЕ требует приведения типа к объекту, тип которого был сериализован. 3) Делает атрибут [Serializable] наследуемым. 4) Требует наличие конструктора без параметров. 5) Все сериализуемые поля должны быть public. В противном случае выбрасывается исключение.

7.

Расставьте в правильной последовательности шаги двоичной сериализации:
1) Создать объект сериализации, называемый форматером. 2) Закрыть байтовый поток. 3) Создать объект класса. 4) Создать байтовый поток (FileStream) и связать его с файлом для записи. 5) Используя метод Serialize() объекта-форматера сохранить в файле представление объекта.

8.

Укажите номера строк кода, которые не содержат в себе ошибок компиляции:

```
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

class Program {
    static void Main() {
        Person person = new Person("Tom", 29);//1
        Person person2 =
new Person(new string(new char[] { }), new int());//2
        BinaryFormatter formatter = new
BinaryFormatter(typeof(Person));//3
        using (FileStream fs = new FileStream("people.dat"))
{//4
            formatter.Serialize(fs, person);//5
        }
    }
}
[Serializable]
class Person {
    public string Name { get; set; }
    public int Age { get; set; }
    public Person(string name, int age) {
        Name = name;
        Age = age;
    }
}
```

1) 1
2) 2
3) 3
4) 4
5) 5

9.

В результате выполнения фрагмента программы:

```
using System;
using System.IO;
using System.Xml.Serialization;

public class A {
    public int x = 7;
    public int y = 10;
    public A() { }
}

class Program {
    static void Main() {
        XmlSerializer xml = new XmlSerializer(typeof(A[]));
        FileStream fs = new FileStream("out.xml",
FileMode.Create);
        using (new StreamWriter(fs)) {
            A a = new A();
            xml.Serialize(fs, a);
        }
        fs = new FileStream("out.xml", FileMode.Open);
        A a2 = (A)xml.Deserialize(fs);
        Console.Write(a2.y + a2.x);
    }
}
```

на экран будет выведено:

Примечание:
Если возникнет ошибка компиляции, введите: ***
Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---
Если возникнет ошибка исполнения или исключение, введите: +++

10.

Выберите все верные утверждения:

1) Атрибут [**NotSerialized**], как и атрибут [**Serializable**] может использоваться для классов.
2) Атрибуты [**NotSerialized**] и [**Serializable**] могут быть унаследованы.
3) В JSON-сериализации для сериализации используется метод WriteObject();
4) JSON-сериализация, в отличие от XML-сериализации, работает только с открытыми типами данных.
5) В бинарной сериализации, все классы, поля и свойства должны быть помечены атрибутом [**Serializable**].

1	+++
2	24
3	3
4	00000
5	1245
6	4
7	34152
8	125
9	+++
10	3