1)

```
public  class  HelloCollection
{
  public  IEnumerator  GetEnumerator()
  {
    return  new  Enumerator (0);
  }
  public  class  Enumerator: IEnumerator<string>, IEnumerator, IDisposable
  {
    private  int  state;
    private  object  current;
    public  Enumerator(int state)
    {
      this.state = state;
    }
 bool System.Collections.IEnumerator.MoveNext()
    {
      switch (state)
      {
        case 0:
          current = "Hello";
          state = 1;
```

```
        state = 2;
      return  true;
      case 2:
        break;
    }
    return  false;
}
string  System.Collections.Generic.IEnumerator<string>.Current
{
    get
    {
      return current;
    }
}
object  System.Collections.IEnumerator.Current
{
    get
    {
      return  current;
    }
}
```

```
    void  IDisposable.Dispose()
    {
    }
  }
}
```

2)

```
public  class  MusicTitles
{
  string[]  names = {
                "Tubular Bells",  "Hergest  Ridge",
                "Ommadawn",  "Platinum" };
  public  IEnumerator<string> GetEnumerator()
  {
    for (int  i = 0; i < 4; i++)
    {
      yield  return  names[i];
    }
  }
```

```
  public  IEnumerable<string> Reverse()
  {
    for (int  i = 3; i >= 0; i--)
    {
      yield  return  names[i];
    }
  }
  public  IEnumerable<string> Subset(int  index, int  length)
  {
    for (int  i = index; i < index + length; i++)
    {
      yield  return  names[i];
    }
  }
}
```

3)

```
var  titles = new  MusicTitles();
foreach (var title in titles)
{
   Console.WriteLine(title);
}
Console.WriteLine();
Console.WriteLine("обратная");
foreach (var  title  in  titles.Reverse()) ,
{
   Console.WriteLine(title);
}
Console.WriteLine();
Console.WriteLine("подмножество");
foreach (var  title  in  titles.Subset(2,2))
{
   Console.WriteLine(title);
}
```

4)

```
public  class  GameMoves
{
  private  IEnumerator  cross;
  private  IEnumerator  circle;
  public  GameMoves()
  {
    cross = Cross();
    circle = Circle();
  }
  private  int  move = 0;
  const  int  MaxMoves = 9;
  public  IEnumerator Cross()
  {
    while (true)
    {
      Console.WriteLine("Крестик, ход {0}", move);
      if (++move >= MaxMoves)  yield  break;
      yield  return  circle;
    }
  }
```

```
public  IEnumerator  Circle()
  {
    while (true)
    {
      Console.WriteLine("Нолик, ход {0}”, move);
      if(++move >= MaxMoves)  yield  break;
      yield return cross;
    }
  }
}
```

5)

```
GameMoves  game = new GameMoves();
IEnumerator  enumerator = game.Cross();
while (enumerator.MoveNext())
{
  enumerator = enumerator.Current  as  IEnumerator;
}
```

6)

Крестик, ход 0
Нолик, ход 1
Крестик, ход 2
Нолик, ход 3
Крестик, ход 4
Нолик, ход 5
Крестик, ход 6
Нолик, ход 7
Крестик, ход 8

7)

```
public  static  Tuple<int,int> Divide (int dividend, int divisor)
{
   int  result = dividend/divisor;
   int  reminder = dividend%divisor;
   return  Tuple.Create<int, int>(result, reminder);
}
```

8)

```
Var  result = Divide(5,2);
Console.WriteLine("результат деления: (0), остаток: {1}",
                  result.Item1,  result.Item2);
```

9)

```
public  class  Tuple<T1, T2, T3, T4, T5, T6, T7, TRest>
```

10)

```
var  tuple = Tuple.Create<string,string,string,int,int,int,double,
  Tuple<int,int> ("Stephanie", "Alina", "Nagel", 2009, 6, 2, 1.37,
    Tuple.Create<int,int>(52, 3490));
```

11)

```
Public  class  Person: IEquatable<Person>
{
  public  int Id {get; private set; }
  public  string FirstName {get; set;}
  public  string LastName {get; set;}
```

```
public  class  Person: IEquatable<Person>
{
  public  int  Id {get; private set; }
  public  string  FirstName {get; set;}
  public  string  LastName {get; set;}
  public  override  string  ToString()
  {
    return  String.Format("{0}, {1} {2}", Id, FirstName, LastName);
  }
  public  override  bool  Equals(object obj)
  {
    if(obj == null) throw  new  ArgumentNullException("obj");
    return  Equals(obj  as  Person);
  }
  public  override  int GetHashCode()
  {
    return  Id.GetHashCode();
  }
```

```
  public  bool  Equals(Person other)
  {
    if (other == null) throw new ArgumentNullException("other");
    return this.Id==other.Id && this.FirstName==other.FirstName &&
       this.LastName == other.LastName;
  }
}
```

12)

```
var  janet = new  Person {FirstName = "Janet", LastName = "Jackson"};
Person []  persons1 = { new  Person
                  {
                    FirstName = "Michael",
                    LastName = "Jackson"
                  },
                  janet
                 };
```

```
Person[]  persons2 = { new Person
                {
                  FirstName = "Michael",
                  LastName = "Jackson"
                },
                janet
               };
if (persons1 != persons2)
  Console.WriteLine("разные ссылки");
```

13)

```
if ((persons1 as IStructuralEquatable).Equals(persons2,
   EqualityComparer<Person>.Default))
{
  Console.WriteLine("одинаковое содержимое");
}
```

14)

```
var  t1 = Tuple.Create<int, string>(1, "Stephanie");
var  t2 = Tuple.Create<int, string>(1, "Stephanie");
if (t1 != t2) Console.WriteLine("не одинаковое содержимое");
```

15)

```
if (t1.Equals(t2)) Console.WriteLine("одинаковое содержимое");
```

16)

```
class TupleComparer: IEqualityComparer
{
  public new bool Equals(object x, object y)
  {
    return x.EquaLs(y);
  }
  public int GetHashCode(object obj)
  {
    return obj.GetHashCode();
  }
```

17)

TupleComparer используется при передаче нового экземпляра методу Equals() класса Tuple<T1,T2>. Метод Equals() класса Tuple вызывает метод Equals() класса TupleComparer для каждого сравниваемого элемента. Поэтому с классом Tuple<T1, T2> класс TupleComparer вызывается два раза для проверки эквивалентности всех элементов:

```
if (t1.Equals(t2, new  TupleComparer()))
 Console.WriteLine("равны после проверки с помощью TupleComparer");
```