1)

```
public class MyClass   i
{
public MyClass ()
{
}
// остаток определения класса
```

2)

```
public MyClass () // конструктор без параметров
{
   //код конструктора
}
public MyClass(int number) // другая перегрузка
{
   //код конструктора
}
```

3)

```
public class MyNumber
{
   private int number;
   public MyNumber(int number)
   {
      this.number = number;
   }
}
```

4)

```
MyNumber numb = new MyNumber () ; //вызовет ошибку при компиляции
```

5)

```
public class MyNumber
{
   private int number;
   private MyNumber(int number) // другая перегрузка
   {
      this.number = number;
   }
}
```

6)

```
class MyClass
{
   static MyClass ()
   {
      // код инициализации
   }
   // остаток определения класса
}
```

7)

```
namespace Wrox.ProCSharp.StaticConstructorSample
{
 public class UserPreferences
 {
   public static readonly Color BackColor;
   static UserPreferences ()
   {
```

```
 {
      DateTime now = DateTime.Now;
      if (now.DayOfWeek == DayOfWeek.Saturday ||
        now.DayOfWeek == DayOfWeek.Sunday)
          BackColor = Color.Green;
      else
          BackColor = Color.Red;
   }
   private UserPreferences ()
   {
   }
 }
}
```

8)

```
using System;
using System.Drawing;
```

9)

```
class MainEntryPoint
{
  static void Main(string[] args)
  {
     Console.WriteLine("Предпочтение пользователя: BackColor равно: "
          + UserPreferences.BackColor.ToString());
  }
}
```

10)

Предпочтение пользователя: BackColor равно: Color [Red]

11)

```
class Car
{
  private string description;
  private uint nWheels;
```

```
public Car(string description, uint nWheels)
 {
    this.description = description;
    this.nWheels = nWheels;
 }
 public Car(string description)
 {
    this.description = description;
    this.nWheels = 4;
 }
 // и т.д.
12)
class Car
{
 private string description;
 private uint nWheels;
 public Car(string description, uint nWheels)
 {
    this.description = description;
    this.nWheels = nWheels;
 }
```

```
  public Car(string description)  : this(description, 4)
  {
  }
  // и т.д.
13)
Car myCar = new Car("Proton Persona");
14)
public class DocumentEditor
{
  public static readonly uint MaxDocuments;
  static DocumentEditor ()
  {
     MaxDocuments = DoSomethingToFindOutMaxNumber();
  }
15)
public class Document
{
  public readonly DateTime CreationDate;
  public Document()
  {
```

```
   // Читаем дату создания файла. Предположим, что в результате
   // получаем 1 января 2010 г., но вообще даты могут быть
разными
   // для различных экземпляров класса.
   CreationDate = new DateTime(2008, 1, 1);
  }
}
16)
void SomeMethod()
{
   MaxDocuments = 10;
   // Здесь - ошибка компиляции. MaxDocuments объявлен как
readonly.
}
17) var captain = new {FirstName = "James", MiddleName = "T",
                LastName = "Kirk"};
18) var doctor = new {FirstName = "Leonard", MiddleName = "",
               LastName = "McCoy"};
19)
var captain = new (person.FirstName, person.MidleName,
                person.LastName};
```

20) Листинг 2.8

```
// Использовать модификатор static.

using System;

class StaticDemo
{
  // Переменная типа static.
  public static int Val = 100;

  // Метод типа static.
  public static int ValDiv2()
  {
    return Val/2;
  }
}
```

```
class SDemo
{
  static void Main()
  {

    Console.WriteLine("Исходное значение переменной "
                      + StaticDemo.Val);
    StaticDemo.Val = 8;
    Console.WriteLine("Текущее значение переменной" +
                      "StaticDemo.Val равно " + StaticDemo.Val);
    Console.WriteLine("StaticDemo.ValDiv2(): " +
                                  StaticDemo.ValDiv2());
  }
}
```

21)

Исходное значение переменной StaticDemo.Val равно 100
Текущее значение переменной StaticDemo.Val равно 8
StaticDemo.ValDiv2(): 4

22) Листинг 2.9

```
class StaticError
{
  public int Denom = 3; // обычная переменная экземпляра
  public static int Val = 1024; // статическая переменная

  /* Ошибка! Непосредственный доступ к нестатической
     переменной из статического метода недопустим. */
  static int ValDivDenom()
  {
    return Val/Denom; // не подлежит компиляции!
  }
}
```

23) Листинг 2.10

```
using System;

class AnotherStaticError
{
  // Нестатический метод.
```

```
  void NonStaticMeth()
  {
    Console.WriteLine("В методе NonStaticMeth().");
  }

  /* Ошибка! Непосредственный вызов нестатического
     метода из статического метода недопустим. */
  static void staticMeth()
  {
    NonStaticMeth(); // не подлежит компиляции!
  }
}
```

24) Листинг 2.11

```
class MyClass
{
  // Нестатический метод.
  void NonStaticMeth()
  {
    Console.WriteLine("В методе NonStaticMeth().");
  }
```

```
/* Нестатический метод может быть вызван из
    статического метода по ссылке на объект. */
  public static void staticMeth(MyClass ob)
  {
    ob.NonStaticMeth(); // все верно!
  }
}
```

25) Листинг 2.12

```
// Использовать поле типа static для подсчета
// количества экземпляров существующих объектов.

using System;

class CountInst
{
  static int count = 0;
```

```
  // Инкременировать подсчет при создании объекта.
  public CountInst()
  {
    count++;
  }

  // Декременировать подсчет при уничтожении объекта.
  ~CountInst()
  {
    count--;
  }

  public static int GetCount() {
    return count;
  }
}
```

```
class CountDemo
{
  static void Main()
  {
    CountInst ob;


    for(int i=0; i < 10; i++) {
      ob = new CountInst();
      Console.WriteLine("Текущий подсчет: " + CountInst.GetCount());
    }
  }
}
```

26)

```
Текущий подсчет: 1
Текущий подсчет: 2
Текущий подсчет: 3
Текущий подсчет: 4
Текущий подсчет: 5
```

Текущий подсчет: 6
Текущий подсчет: 7
Текущий подсчет: 8
Текущий подсчет: 9
Текущий подсчет: 10

27) Листинг 2.13

```
// Использовать статическую фабрику класса.

using System;

class MyClass
{
  int a, b;

  // создать фабрику для класса MyClass.
  static public MyClass Factory(int i, int j)
  {
    MyClass t = new MyClass();
```

```
    t.a = i;
    t.b = j;

    return t; // возвратить объект
  }

  public void Show()
  {
    Console.WriteLine("a и b: " + a + " " + b);
  }
}

class MakeObjects
{
  static void Main()
  {
    int i, j;
```

```
   // Создать объекты, используя фабрику.
   for(i=0, j=10; i < 10; i++, j--) {
     MyClass ob = MyClass.Factory(i, j); // создать объект
     ob.Show();
   }

   Console.WriteLine();
  }
}
```

28)

```
MyClass ob = MyClass.Factory(i, j); // создать объект
```

29)

```
class Dimensions
{
  public double Length;
  public double Width;
}
```

30)

```
struct Dimensions
{
    public double Length;
    public double Width;
}
```

31) Листинг 2.14

```
struct Dimensions
{
  public double Length;
  public double Width;
```

```
Dimensions(double length, double width)
 {
    Length=length;
    Width=width;
 }
 public int Diagonal
 {
    get
    {
       return Math.Sqrt(Length*Length + Width*Width);
    }
 }
}

32)
Dimensions point = new Dimensions();
point.Length = 3;
point.Width = 6;
```

33)

```
Dimensions point;
point.Length = 3;
point.Width = 6;
```

34)

```
Dimensions point;
Double D = point.Length;
```

35)

```
struct Dimensions
{
  public double Length = 1;
  // Ошибка. Начальные значения не разрешены,
  public double Width = 2;
  // Ошибка. Начальные значения не разрешены.
}
```

36)

```
partial class TheBigClass
{
  public void MethodOne()
  {
  }
}
// BigClassPart2.cs
partial class TheBigClass
{
  public void MethodTwo()
  {
  }
}
```

37)

public, private, protected, internal, abstract, sealed, new,
общие ограничения

38) Листинг 2.15

```
[CustomAttribute]
partial class TheBigClass : TheBigBaseClass, IBigClass
{
  public void MethodOne()
  {
  }
}
// BigClassPart2.cs
[AnotherAttribute]
partial class TheBigClass : IotherBigClass
{
  public void MethodTwoO
  {
  }
}
```

39)

```
[CustomAttribute]
[AnotherAttribute]
partial class TheBigClass:TheBigBaseClass,IBigClass,OtherBigClass
{
  public void MethodOne()
  {
  }
  public void MethodTwo()
  {
  }
}
```

40)

```
static class StaticUtilities
{
  public static void HelperMethod()
  {
  }
}
```

41) StaticUtilities.HelperMethod();

42)

```
int i = -50;
string str = i.ToString(); // возвращает "-50"
```

43)

```
enum Colors {Red, Orange, Yellow};
// далее в коде. . .
Colors favoriteColor = Colors.Orange;
string str = favoriteColor.ToString(); // возвращает "Orange"
```

44) Листинг 2.16

```
using System;
namespace Wrox
{
  class MainEntryPoint
  {
```

```
static void Main(string[] args)
   {
      Money cashl = new Money();
      cashl.Amount = 40M;
      Console.WriteLine("cashl.ToString() возвращает: " +
                        cashl.ToString());
      Console.ReadLine();
   }
}
 class Money
 {
   private decimal amount;
   public decimal Amount
   {
      get
      {
         return amount;
      }
```

```
set
        {
            amount = value;
        }
    }
    public override string ToString()
    {
        return "$'* + Amount.ToString();
    }
  }
}
```

45) cash1.ToString() возвращает: $40

47) cash1.AddToAmount (10M);

46)

```
namespace Wrox
{
  public static class MoneyExtension
  {
     public static void AddToAmount(this Money money,
                                    decimal amountToAdd)
     {
        money.Amount += amountToAdd;
     }
  }
}
```