Fatlum Maliqi / Kampusi Ferizaj

Chapter 14. Defining Classes

1. Define a class Student, which contains the following information about students: full name, course, subject, university, e-mail and phone number.

```
enum Specialty { KST, Telecommunication, Electronics, Unknown }
enum University { TechnicalUniversity, SofiaUniversity, Unknown }

class Student
{
   private string firstName;
   private string middleName;
   private string lastName;

   int course;
   Specialty specialty;
   University university;
   string email;
   string phoneNumber;
}
```

2. Declare several constructors for the class Student, which have different lists of parameters (for complete information about a student or part of it). Data, which has no initial value to be initialized with null. Use nullable types for all non-mandatory data.

```
public Student(string firstName, string lastName)
    : this(firstName, null, lastName, University.Unknown)
  {

  }

  public Student(string firstName, string middleName, string lastName, University university)
    : this(firstName, middleName, lastName, university, Specialty.Unknown)
  {

  }

  public Student(string firstName, string middleName, string lastName,
    University university, Specialty specialty)
    : this(firstName, middleName, lastName, 0, university, specialty, null, null)
  {

  }

  public Student(string firstName, string middleName, string lastName, int course,
    University university, Specialty specialty, string email, string phoneNumber)
  {
    this.firstName = firstName;
    this.middleName = middleName;
    this.lastName = lastName;
    this.course = course;
    this.specialty = specialty;
    this.university = university;
    this.email = email;
    this.phoneNumber = phoneNumber;
  }
```

3. Add a static field for the class Student, which holds the number of created objects of this class.

```
class Student
{

    private string firstName;
    private string middleName;
    private string lastName;
    private int course;
    Specialty specialty;
    University university;
    private string email;
    private string phoneNumber;

    static int numberOfSudents = 0;
…
}
```

4. Add a method in the class Student, which displays complete information about the student.

```
public override string ToString()
  {
     StringBuilder result = new StringBuilder();

     result.Append(String.Format("First Name: {0}{1}", this.firstName, Environment.NewLine));
     result.Append(String.Format("Middle Name: {0}{1}", this.middleName, Environment.NewLine));

     if (this.course != 0)
     {
       result.Append(String.Format("Couser: {0}{1}", this.course, Environment.NewLine));
     }

     result.Append(String.Format("Specialty: {0}{1}", this.specialty, Environment.NewLine));
     result.Append(String.Format("University: {0}{1}", this.university, Environment.NewLine));

     if (this.email != null)
     {
       result.Append(String.Format("Email: {0}{1}", this.email, Environment.NewLine));
     }

     if (this.phoneNumber != null)
     {
       result.Append(String.Format("Phone: {0}{1}", this.phoneNumber, Environment.NewLine));
     }

     return result.ToString();
  }
```

5. Modify the current source code of Student class so as to encapsulate the data in the class using properties.

```
enum Specialty { KST, Telecommunication, Electronics, Unknown }
enum University { TechnicalUniversity, SofiaUniversity, Unknown }

class Student
{
   private string firstName;
   private string middleName;
   private string lastName;
   private int course;
   Specialty specialty;
   University university;
   private string email;
   private string phoneNumber;

   public string FirstName
   {
      get
      {
         return this.firstName;
      }
      set
      {
         this.firstName = value;
      }
   }

   public string MiddleName
   {
      get
      {
         return this.middleName;
      }
      set
      {
         this.middleName = value;
      }
   }

   public string LastName
   {
      get
      {
         return this.lastName;
      }
      set
```

```
    {
        this.lastName = value;
    }
}

public int Course
{
    get
    {
        return this.course;
    }
    set
    {
        this.course = value;
    }
}

public Specialty Specialty
{
    get
    {
        return this.specialty;
    }
    set
    {
        this.specialty = value;
    }
}

public University University
{
    get
    {
        return this.university;
    }
    set
    {
        this.university = value;
    }
}

public string Email
{
    get
    {
        return this.email;
    }
    set
```

```
        {
            this.email = value;
        }
    }

    public string PhoneNumber
    {
        get
        {
            return this.phoneNumber;
        }
        set
        {
            this.phoneNumber = value;
        }
    }
}
```

6. Write a class StudentTest, which has to test the functionality of the class Student.

```
static class StudentTest
{
    private static Student studentWithTwoArguments;
    private static Student studentWithFourArguments;
    private static Student studentWithFiveArguments;
    private static Student studentWithAllArguments;

    static StudentTest()
    {
        studentWithTwoArguments = new Student("Joro", "Pelovski");
        studentWithFourArguments = new Student("Niki", "Petrov", "Kirov",
University.TechnicalUniversity);
        studentWithFiveArguments = new Student("Niki", "Mihailov", "Aleksiev",
University.TechnicalUniversity, Specialty.KST);
        studentWithAllArguments = new Student("Ivan", "Penchev", "Nikolov", 4,
University.TechnicalUniversity,
            Specialty.KST, "joro@abv.bg", "0883 30 24 12" );
    }

    public static void TestPrint()
    {
        Console.WriteLine(studentWithTwoArguments);
        Console.WriteLine(studentWithFourArguments);
        Console.WriteLine(studentWithFiveArguments);
        Console.WriteLine(studentWithAllArguments);
    }
}
```

7. Add a static method in class StudentTest, which creates several objects of type Student and store them in static fields. Create a static property of the class to access them. Write a test program, which displays the information about them in the console.

```
static class StudentTest
{
    private static Student studentWithTwoArguments;
    private static Student studentWithFourArguments;
    private static Student studentWithFiveArguments;
    private static Student studentWithAllArguments;

    static StudentTest()
    {
        studentWithTwoArguments = new Student("Joro", "Pelovski");
        studentWithFourArguments = new Student("Niki", "Petrov", "Kirov",
University.TechnicalUniversity);
        studentWithFiveArguments = new Student("Niki", "Mihailov", "Aleksiev",
University.TechnicalUniversity, Specialty.KST);
        studentWithAllArguments = new Student("Ivan", "Penchev", "Nikolov", 4,
University.TechnicalUniversity,
            Specialty.KST, "joro@abv.bg", "0883 30 24 12" );
    }

    public static void TestPrint()
    {
        Console.WriteLine(studentWithTwoArguments);
        Console.WriteLine(studentWithFourArguments);
        Console.WriteLine(studentWithFiveArguments);
        Console.WriteLine(studentWithAllArguments);
    }
}
```

8. Define a class, which contains information about a mobile phone: model, manufacturer, price, owner, features of the battery (model, idle time and hours talk) and features of the screen (size and colors).

```
class GSM
{
   private Battery battery;
   private Display display;
   private string model;
   private string manufacture;
   private decimal? price;
   private string owner;
}


class Battery
{
   private string model;
   private TimeSpan? hoursIdle, hoursTalk;
}


class Display
{
   private decimal? size;
   private decimal? colors;
}
```

9. Declare several constructors for each of the classes created by the previous task, which have different lists of parameters (for complete information about a student or part of it). Data fields that are unknown have to be initialized respectively with null or 0.

```
class GSM
{
    private Battery battery;
    private Display display;

    private string model;
    private string manufacturer;
    private decimal? price;
    private string owner;

    public GSM(string model, string manufacturer)
        : this(model, manufacturer, null)
    {
    }

    public GSM(string model, string manufacturer, decimal? price)
        : this(model, manufacturer, price, null)
    {
    }

    public GSM(string model, string manufacturer, decimal? price,
        string owner)
        : this(model, manufacturer, price, owner, new Battery())
    {
    }

    public GSM(string model, string manufacturer, decimal? price,
        string owner, Battery battery)
        : this(model, manufacturer, price, owner, battery, new Display())
    {
    }

    public GSM(string model, string manufacturer, decimal? price,
        string owner, Battery battery, Display display)
    {
        this.model = model;
        this.manufacturer = manufacturer;
        this.price = price;
        this.owner = owner;
        this.battery = battery;
        this.display = display;
    }
}
class Battery
```

```
{
    private string model;
    private TimeSpan? hoursIdle, hoursTalk;

    public Battery()
        : this(null)
    {
    }

    public Battery(string model)
        : this(model, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle)
        : this(model, hoursIdle, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle, TimeSpan? hoursTalk)
    {
        this.model = model;
        this.hoursIdle = hoursIdle;
        this.hoursTalk = hoursTalk;
    }


class Display
{
    private decimal? size;
    private decimal? colors;

    public Display()
        : this(null)
    {
    }

    public Display(decimal? size)
        : this(size, null)
    {
    }

    public Display(decimal? size, decimal? numberOfColors)
    {
        this.size = size;
        this.colors = numberOfColors;
    }
}
```

10. To the class of mobile phone in the previous two tasks, add a static field nokiaN95, which stores information about mobile phone model Nokia N95. Add a method to the same class, which displays information about this static field.

```
class GSM
{
   private Battery battery;
   private Display display;

   static private GSM nokiaN95;

   private string model;
   private string manufacturer;
   private decimal? price;
   private string owner;

   public GSM(string model, string manufacturer)
      : this(model, manufacturer, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price)
      : this(model, manufacturer, price, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner)
      : this(model, manufacturer, price, owner, new Battery())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery)
      : this(model, manufacturer, price, owner, battery, new Display())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery, Display display)
   {
      this.model = model;
      this.manufacturer = manufacturer;
      this.price = price;
      this.owner = owner;
      this.battery = battery;
      this.display = display;
   }
   static GSM()
   {
      nokiaN95= new GSM("N95", "Nokia", 1000M, "Me", new Battery("H223", new TimeSpan(20, 30, 0),
new TimeSpan(5, 45, 0), new Display(5M));
   }

   static public GSM NokiaN95
   {
      get
      {
         return nokiaN95;
      }
   }


   public static string NokiaN95ToString()
   {

      StringBuilder printInfo = new StringBuilder();
      printInfo.Append(String.Format("Manufacture: {0}\n", nokiaN95.manufacturer));
      printInfo.Append(String.Format("Model: {0}\n", nokiaN95.model));
      printInfo.Append(nokiaN95.price == null ? String.Format("Price: {0}\n") : String.Format("Price:
{0}\n", nokiaN95.price));
      printInfo.Append(nokiaN95.owner == null ? String.Format("Owner: {0}\n") : String.Format("Owner:
{0}\n", nokiaN95.owner));

      if (nokiaN95.battery != null)
      {
         printInfo.Append("\nBattery\n");
         printInfo.Append(String.Format("{0,2}{1,-10} - {2}\n", "-", "Model", nokiaN95.battery.Model));
         printInfo.Append(nokiaN95.battery.HoursIdle == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursIdle", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursIdle",
nokiaN95.battery.HoursIdle));
         printInfo.Append(nokiaN95.battery.HoursTalk == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursTalk", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursTalk",
nokiaN95.battery.HoursTalk));
      }
      else
      {
         printInfo.Append(String.Format("{0, -13} - {1}\n", "Battery", "unknown"));
      }

      if (nokiaN95.display != null)
      {
         printInfo.Append("\nDisplay\n");
         printInfo.Append(String.Format("{0,2}{1,-10} - {2}\"\n", "-", "Size", nokiaN95.display.Size));
```

```
            printInfo.Append(nokiaN95.display.Colors == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"Colors", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors", nokiaN95.display.Colors));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Display", "unknown"));
        }
        return printInfo.ToString();
    }
}
```

11. Add an enumeration BatteryType, which contains the values for type of the battery (Li-Ion, NiMH, NiCd, …) and use it as a new field for the class Battery.

```
enum BatteryType { LiIon, NiMH, NiCd }

class Battery
{
    private string model;
    private TimeSpan? hoursIdle, hoursTalk;
    private BatteryType? type;

    public Battery()
        : this(null)
    {
    }

    public Battery(string model)
        : this(model, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle)
        : this(model, hoursIdle, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle, TimeSpan? hoursTalk)
        : this(model, hoursIdle, hoursTalk, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle, TimeSpan? hoursTalk, BatteryType? type)
    {
        this.model = model;
        this.hoursIdle = hoursIdle;
        this.hoursTalk = hoursTalk;
        this.type = type;
    }
}
```

12. Add a method to the class GSM, which returns information about the object as a string.

```
class GSM
{
   private Battery battery;
   private Display display;

   static private GSM nokiaN95;

   private string model;
   private string manufacturer;
   private decimal? price;
   private string owner;


   public GSM(string model, string manufacturer)
      : this(model, manufacturer, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price)
      : this(model, manufacturer, price, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner)
      : this(model, manufacturer, price, owner, new Battery())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery)
      : this(model, manufacturer, price, owner, battery, new Display())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery, Display display)
   {
      this.model = model;
      this.manufacturer = manufacturer;
      this.price = price;
      this.owner = owner;
      this.battery = battery;
      this.display = display;
   }
```

```
    static GSM()
    {
        nokiaN95 = new GSM("IPhone4S", "Apple", 1000M, "Me", new Battery("H223", new TimeSpan(20,
30, 0), new TimeSpan(5, 45, 0),
                BatteryType.NiCd), new Display(5M));
    }

    static public GSM IPhone4S
    {
        get
        {
            return nokiaN95;
        }
    }

    public static string NokiaN95ToString()
    {

        StringBuilder printInfo = new StringBuilder();
        printInfo.Append(String.Format("Manufacture: {0}\n", nokiaN95.manufacturer));
        printInfo.Append(String.Format("Model: {0}\n", nokiaN95.model));
        printInfo.Append(nokiaN95.price == null ? String.Format("Price: {0}\n") : String.Format("Price:
{0}\n", nokiaN95.price));
        printInfo.Append(nokiaN95.owner == null ? String.Format("Owner: {0}\n") : String.Format("Owner:
{0}\n", nokiaN95.owner));

        if (nokiaN95.battery != null)
        {
            printInfo.Append("\nBattery\n");
            printInfo.Append(String.Format("{0,2}{1,-10} - {2}\n", "-", "Model", nokiaN95.battery.Model));
            printInfo.Append(nokiaN95.battery.HoursIdle == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursIdle", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursIdle",
nokiaN95.battery.HoursIdle));
            printInfo.Append(nokiaN95.battery.HoursTalk == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursTalk", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursTalk",
nokiaN95.battery.HoursTalk));
            printInfo.Append(nokiaN95.battery.Type == null ? String.Format("{0,2}{1,-10} - {2}\n", "-", "Type",
"unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Type", nokiaN95.battery.Type));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Battery", "unknown"));
        }

        if (nokiaN95.display != null)
        {
            printInfo.Append("\nDisplay\n");
            printInfo.Append(String.Format("{0,2}{1,-10} - {2}\"\n", "-", "Size", nokiaN95.display.Size));
            printInfo.Append(nokiaN95.display.Colors == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"Colors", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors", nokiaN95.display.Colors));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Display", "unknown"));
        }
        return printInfo.ToString();
    }

    public override string ToString()
    {

        StringBuilder printInfo = new StringBuilder();
        printInfo.Append("--------------------------\n");
        printInfo.Append(String.Format("{0, -13} - {1}\n", "Manufacture", this.manufacturer));
        printInfo.Append(String.Format("{0, -13} - {1}\n", "Model", this.model));
        printInfo.Append(this.price == null ? String.Format("{0, -13} - unknown\n", "Price") :
String.Format("{0, -13} - {1}\n", "Price", this.price));
        printInfo.Append(this.owner == null ? String.Format("{0, -13} - unknown\n", "Owner") :
String.Format("{0, -13} - {1}\n", "Owner", this.owner));

        if (this.battery != null)
        {
            printInfo.Append("\nBattery\n");
            printInfo.Append(String.Format("{0,2}{1,-10} - {2}\n", "-", "Model", this.battery.Model));
            printInfo.Append(this.battery.HoursIdle == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursIdle", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursIdle", this.battery.HoursIdle));
            printInfo.Append(this.battery.HoursTalk == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursTalk", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursTalk", this.battery.HoursTalk));
            printInfo.Append(this.battery.Type == null ? String.Format("{0,2}{1,-10} - {2}\n", "-", "Type",
"unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Type", this.battery.Type));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Battery", "unknown"));
        }

        if (this.display != null)
        {
            printInfo.Append("\nDisplay\n");
            printInfo.Append(String.Format("{0,2}{1,-10} - {2}\"\n", "-", "Size", this.display.Size));
            printInfo.Append(this.display.Colors == null ? String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors",
"unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors", this.display.Colors));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Display", "unknown"));
        }
        printInfo.Append("--------------------------\n");
        return printInfo.ToString();
    }
}
```

13. Define properties to encapsulate the data in classes GSM, Battery and Display.

```
class GSM
{
    private Battery battery;
    private Display display;

    static private GSM nokiaN95;

    private string model;
    private string manufacturer;
    private decimal? price;
    private string owner;

    List<Call> calls;

    public GSM(string model, string manufacturer) : this(model, manufacturer, null)
    {
    }

    public GSM(string model, string manufacturer, decimal? price) : this(model, manufacturer, price, null)
    {
    }

    public GSM(string model, string manufacturer, decimal? price,
        string owner) : this(model, manufacturer, price, owner, new Battery())
    {
    }

    public GSM(string model, string manufacturer, decimal? price,
        string owner, Battery battery) : this(model, manufacturer, price, owner, battery, new Display())
    {
}

public GSM(string model, string manufacturer, decimal? price,
   string owner, Battery battery, Display display)
{
   this.Model = model;
   this.Manufacturer = manufacturer;
   this.Price = price;
   this.Owner = owner;
   this.battery = battery;
   this.display = display;
}

public string Model
{
   get
   {
      return this.model;
   }
   set
   {
      this.model = value;
   }
}

public string Manufacturer
{
   get
   {
      return this.manufacturer;
   }
   set
   {
      if (String.IsNullOrEmpty(value))
         throw new ArgumentException("Invalid manufacturer: " + value);
      this.manufacturer = value;
   }
}

public decimal? Price
{
   get
   {
      return this.price;
   }
   set
   {
      if (value < 0)
```

```
            {
                throw new ArgumentException("Invalid price: " + value);
            }
            this.price = value;
        }
    }

    public string Owner
    {
        get
        {
            return this.owner;
        }
        set
        {
            if (String.IsNullOrEmpty(value))
                throw new ArgumentException("Invalid name: " + value);
            this.owner = value;
        }
    }

    static GSM()
    {
        nokiaN95 = new GSM("IPhone4S", "Apple", 1000M, "Me", new Battery("H223", new TimeSpan(20,
30, 0), new TimeSpan(5, 45, 0),
            BatteryType.NiCd), new Display(5M));
    }

    static public GSM IPhone4S
    {
        get
        {
            return nokiaN95;
        }
    }

    public static string NokiaN95ToString()
    {
        StringBuilder printInfo = new StringBuilder();
        printInfo.Append(String.Format("Manufacture: {0}\n", nokiaN95.manufacturer));
        printInfo.Append(String.Format("Model: {0}\n", nokiaN95.model));
        printInfo.Append(nokiaN95.price == null ? String.Format("Price: {0}\n") : String.Format("Price:
{0}\n", nokiaN95.price));
        printInfo.Append(nokiaN95.owner == null ? String.Format("Owner: {0}\n") : String.Format("Owner:
{0}\n", nokiaN95.owner));

        if (nokiaN95.battery != null)
        {
          printInfo.Append("\nBattery\n");
          printInfo.Append(String.Format("{0,2}{1,-10} - {2}\n", "-", "Model", nokiaN95.battery.Model));
          printInfo.Append(nokiaN95.battery.HoursIdle == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursIdle", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursIdle",
nokiaN95.battery.HoursIdle));
          printInfo.Append(nokiaN95.battery.HoursTalk == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursTalk", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursTalk",
nokiaN95.battery.HoursTalk));
          printInfo.Append(nokiaN95.battery.Type == null ? String.Format("{0,2}{1,-10} - {2}\n", "-", "Type",
"unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Type", nokiaN95.battery.Type));
       }
       else
       {
          printInfo.Append(String.Format("{0, -13} - {1}\n", "Battery", "unknown"));
       }

       if (nokiaN95.display != null)
       {
          printInfo.Append("\nDisplay\n");
          printInfo.Append(String.Format("{0,2}{1,-10} - {2}\"\n", "-", "Size", nokiaN95.display.Size));
          printInfo.Append(nokiaN95.display.Colors == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"Colors", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors", nokiaN95.display.Colors));
       }
       else
       {
          printInfo.Append(String.Format("{0, -13} - {1}\n", "Display", "unknown"));
       }
       return printInfo.ToString();
    }

    public override string ToString()
    {
       StringBuilder printInfo = new StringBuilder();
       printInfo.Append("--------------------------\n");
       printInfo.Append(String.Format("{0, -13} - {1}\n", "Manufacture", this.manufacturer));
       printInfo.Append(String.Format("{0, -13} - {1}\n", "Model", this.model));
       printInfo.Append(this.price == null ? String.Format("{0, -13} - unknown\n", "Price") :
String.Format("{0, -13} - {1}\n", "Price", this.price));
       printInfo.Append(this.owner == null ? String.Format("{0, -13} - unknown\n", "Owner") :
String.Format("{0, -13} - {1}\n", "Owner", this.owner));

       if (this.battery != null)
       {
          printInfo.Append("\nBattery\n");
          printInfo.Append(String.Format("{0,2}{1,-10} - {2}\n", "-", "Model", this.battery.Model));
          printInfo.Append(this.battery.HoursIdle == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursIdle", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursIdle", this.battery.HoursIdle));
```

```
            printInfo.Append(this.battery.HoursTalk == null ? String.Format("{0,2}{1,-10} - {2}\n", "-",
"HoursTalk", "unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "HoursTalk", this.battery.HoursTalk));
            printInfo.Append(this.battery.Type == null ? String.Format("{0,2}{1,-10} - {2}\n", "-", "Type",
"unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Type", this.battery.Type));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Battery", "unknown"));
        }

        if (this.display != null)
        {
            printInfo.Append("\nDisplay\n");
            printInfo.Append(String.Format("{0,2}{1,-10} - {2}\"\n", "-", "Size", this.display.Size));
            printInfo.Append(this.display.Colors == null ? String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors",
"unknown") : String.Format("{0,2}{1,-10} - {2}\n", "-", "Colors", this.display.Colors));
        }
        else
        {
            printInfo.Append(String.Format("{0, -13} - {1}\n", "Display", "unknown"));
        }
        printInfo.Append("--------------------------\n");
        return printInfo.ToString();
    }
}


class Battery
{
    private string model;
    private TimeSpan? hoursIdle, hoursTalk;
    private BatteryType? type;

    public Battery()
        : this(null)
    {
    }

    public Battery(string model)
        : this(model, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle)
        : this(model, hoursIdle, null)
    {
    }
    public Battery(string model, TimeSpan? hoursIdle, TimeSpan? hoursTalk)
        : this(model, hoursIdle, hoursTalk, null)
    {
    }

    public Battery(string model, TimeSpan? hoursIdle, TimeSpan? hoursTalk, BatteryType? type)
    {
        this.Model = model;
        this.HoursIdle = hoursIdle;
        this.HoursTalk = hoursTalk;
        this.Type = type;
    }

    public string Model
    {
        get
        {
            return this.model;
        }
        set
        {
            this.model = value;
        }
    }

    public TimeSpan? HoursIdle { get; set; }

    public TimeSpan? HoursTalk { get; set; }

    public BatteryType? Type { get; set; }
}


class Display
{
    private decimal? size;
    private decimal? colors;

    public Display() : this(null)
    {
    }

    public Display(decimal? size) : this(size, null)
    {
    }

    public Display(decimal? size, decimal? numberOfColors)
    {
        this.size = size;
        this.colors = numberOfColors;
    }

    public decimal? Size
    {
        get
        {
            return this.size;
        }
        set
        {
            if (value <= 0)
            {
                throw new ArgumentException("Invalid size: " + value);
            }
            this.size = value;
        }
    }

    public decimal? Colors
    {
        get
        {
            return this.size;
        }
        set
        {
            if (value <= 0)
            {
                throw new ArgumentException("Invalid number of colors: " + value);
            }
            this.colors = value;
        }
    }
}
```

14. Write a class GSMTest, which has to test the functionality of class GSM. Create few objects of the class and store them into an array. Display information about the created objects. Display information about the static field nokiaN95.

```
static class GSMTest
{

    private static GSM[] testGSMs = new GSM[2];

    static void Main(string[] args)
    {
        GSMTest.TestPrint();
    }

    public static void TestPrint()
    {
        testGSMs[0] = new GSM("Desire S", "HTC", 1000M, "Pesho", new Battery("H223", new
TimeSpan(20, 30, 0), new TimeSpan(10, 30, 0),
            BatteryType.NiCd), new Display(5M));
        testGSMs[1] = new GSM("3310", "Nokia", 150M, "Ivan", new Battery("H223", new TimeSpan(10, 15,
0), new TimeSpan(5, 20, 0),
            BatteryType.LiIon), new Display(3M));

        foreach (GSM gsm in testGSMs)
        {
            Console.WriteLine(gsm);
        }

        Console.WriteLine(GSM.NokiaN95ToString());
}
```

15. Create a class Call, which contains information about a call made via mobile phone. It should contain information about date, time of start and duration of the call.

```
class Call
{
   DateTime timeOfCall;
   string dialedNumber;
   int duration;

   public Call(string dialedNumber, int duration)
   {
      this.timeOfCall = DateTime.Now;
      this.dialedNumber = dialedNumber;
      this.duration = duration;
   }

   public DateTime TimeOfCall
   {
      get
      {
         return this.timeOfCall;
      }
   }

   public string DialedNumber
   {
      get
      {
         return this.dialedNumber;
      }
   }

   public int Duration
   {
      get
      {
         return this.duration;
      }
   }
}
```

16. Add a property for keeping a call history – CallHistory, which holds a list of call records.

```
class GSM
{
   private Battery battery;
   private Display display;

   static private GSM nokiaN95;
```

```
    private string model;
    private string manufacturer;
    private decimal? price;
    private string owner;

    List<Call> calls = new List<Call>();
}
```

16. Add a property for keeping a call history – CallHistory, which holds a list of call records.

```
class GSM
{
    private Battery battery;
    private Display display;

    static private GSM nokiaN95;

    private string model;
    private string manufacturer;
    private decimal? price;
    private string owner;

    List<Call> calls = new List<Call>();
}
```

17. In GSM class add methods for adding and deleting calls (Call) in the archive of mobile phone calls. Add method, which deletes all calls from the archive.

```
class GSM
{
   private Battery battery;
   private Display display;

   static private GSM nokiaN95;

   private string model;
   private string manufacturer;
   private decimal? price;
   private string owner;

   List<Call> calls = new List<Call>();

   public GSM(string model, string manufacturer) : this(model, manufacturer, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price) : this(model, manufacturer, price, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner) : this(model, manufacturer, price, owner, new Battery())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery) : this(model, manufacturer, price, owner, battery, new Display())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery, Display display)
   {
      this.Model = model;
      this.Manufacturer = manufacturer;
      this.Price = price;
      this.Owner = owner;
      this.battery = battery;
      this.display = display;
   }

   public string Model
   {
    get
    {
        return this.model;
    }
    set
    {
        this.model = value;
    }
}

public string Manufacturer
{
    get
    {
        return this.manufacturer;
    }
    set
    {
        if (String.IsNullOrEmpty(value))
            throw new ArgumentException("Invalid manufacturer: " + value);
        this.manufacturer = value;
    }
}

public decimal? Price
{
    get
    {
        return this.price;
    }
    set
    {
        if (value < 0)
        {
            throw new ArgumentException("Invalid price: " + value);
        }
        this.price = value;
    }
}

public string Owner
{
    get
    {
        return this.owner;
    }
    set
    {
            if (String.IsNullOrEmpty(value))
                throw new ArgumentException("Invalid name: " + value);
            this.owner = value;
        }
    }

    static GSM()
    {
        nokiaN95 = new GSM("IPhone4S", "Apple", 1000M, "Me", new Battery("H223", new TimeSpan(20,
30, 0), new TimeSpan(5, 45, 0),
            BatteryType.NiCd), new Display(5M));
    }

    static public GSM IPhone4S
    {
        get
        {
            return nokiaN95;
        }
    }

    public void ViewCalls()
    {
        if (calls.Count > 0)
        {
            Console.WriteLine("---------------- Calls List ------------------");
            Console.WriteLine("{0,-10}{1,-13} | {2,-8} | {3}", "Date", "Time", "Number", "Duration");
            foreach (var call in this.calls)
            {
                Console.WriteLine("{0,-23} | {1,-8} | {2}", call.TimeOfCall, call.DialedNumber, call.Duration);
            }
            Console.WriteLine("---------------------------------------------");
        }
        else
        {
            Console.WriteLine("\nNo calls\n");
        }
    }

    public void AddCall(string dailedNumber, int duration)
    {
        Call newCall = new Call(dailedNumber, duration);
        this.calls.Add(newCall);
    }

    public void RemoveCall(string numberToDel)
    {
        int removedCallsCount;
        if (this.calls.Count > 0)
        {
            Console.WriteLine();
        }
        else
        {
            Console.WriteLine("No calls in history");
            return;
        }

        removedCallsCount = this.calls.RemoveAll(
            delegate(Call call)
            {
                return call.DialedNumber == numberToDel;
            });

        if (removedCallsCount > 0)
        {
            Console.WriteLine("{0} calls removed\n", removedCallsCount);
        }
        else
        {
            Console.WriteLine("Call not found");
        }
    }

    public void ClearHistory()
    {
        this.calls.Clear();

        Console.WriteLine("\nAll calls cleared\n");
    }
}
```

18. In GSM class, add a method that calculates the total amount of calls (Call) from the archive of phone calls (CallHistory), as the price of a phone call is passed as a parameter to the method.

```
class GSM
{
   private Battery battery;
   private Display display;

   static private GSM nokiaN95;

   private string model;
   private string manufacturer;
   private decimal? price;
   private string owner;

   List<Call> calls = new List<Call>();

   public GSM(string model, string manufacturer) : this(model, manufacturer, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price) : this(model, manufacturer, price, null)
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner) : this(model, manufacturer, price, owner, new Battery())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery) : this(model, manufacturer, price, owner, battery, new Display())
   {
   }

   public GSM(string model, string manufacturer, decimal? price,
      string owner, Battery battery, Display display)
   {
      this.Model = model;
      this.Manufacturer = manufacturer;
      this.Price = price;
      this.Owner = owner;
      this.battery = battery;
      this.display = display;
   }

   public string Model
```

```
{
   get
   {
      return this.model;
   }
   set
   {
      this.model = value;
   }
}

public string Manufacturer
{
   get
   {
      return this.manufacturer;
   }
   set
   {
      if (String.IsNullOrEmpty(value))
         throw new ArgumentException("Invalid manufacturer: " + value);
      this.manufacturer = value;
   }
}

public decimal? Price
{
   get
   {
      return this.price;
   }
   set
   {
      if (value < 0)
      {
         throw new ArgumentException("Invalid price: " + value);
      }
      this.price = value;
   }
}

public string Owner
{
   get
   {
      return this.owner;
   }
   set
```

```
        {
            if (String.IsNullOrEmpty(value))
                throw new ArgumentException("Invalid name: " + value);
            this.owner = value;
        }
    }

    static GSM()
    {
        nokiaN95 = new GSM("IPhone4S", "Apple", 1000M, "Me", new Battery("H223", new TimeSpan(20,
30, 0), new TimeSpan(5, 45, 0),
            BatteryType.NiCd), new Display(5M));
    }

    static public GSM IPhone4S
    {
        get
        {
            return nokiaN95;
        }
    }

    public void AddCall(string dailedNumber, int duration)
    {
        Call newCall = new Call(dailedNumber, duration);
        this.calls.Add(newCall);
    }

    public void RemoveCall(string numberToDel)
    {
        int removedCallsCount;

        if (this.calls.Count > 0)
        {
            Console.WriteLine();
        }
        else
        {
            Console.WriteLine("No calls in history");
            return;
        }

        removedCallsCount = this.calls.RemoveAll(
            delegate(Call call)
            {
                return call.DialedNumber == numberToDel;
            });
        if (removedCallsCount > 0)
        {
            Console.WriteLine("{0} calls removed\n", removedCallsCount);
        }
        else
        {
            Console.WriteLine("Call not found");
        }
    }

    public void ClearHistory()
    {
        this.calls.Clear();

        Console.WriteLine("\nAll calls cleared\n");
    }

    public decimal CalculatePrice(decimal costPerMinute)
    {
        int totalSecondCount = 0;

        foreach (var call in calls)
        {
            totalSecondCount += call.Duration;
        }

        return (totalSecondCount / 60) * costPerMinute;
    }
}
```

19. Create a class GSMCallHistoryTest, with which to test the functionality of the class GSM, from task 12, as an object of type GSM. Then add to it a few phone calls (Call). Display information about each phone call. Assuming that the price per minute is 0.37, calculate and display the total cost of all calls. Remove the longest conversation from archive with phone calls and calculate the total price for all calls again. Finally, clear the archive.

```
static class GSMTest
{

  private static GSM[] testGSMs = new GSM[2];

  static void Main(string[] args)
  {
    GSMTest.TestPrint();
  }

  public static void TestPrint()
  {
    testGSMs[0] = new GSM("Desire S", "HTC", 1000M, "Pesho", new Battery("H223", new
TimeSpan(20, 30, 0), new TimeSpan(10, 30, 0),
        BatteryType.NiCd), new Display(5M));
    testGSMs[1] = new GSM("3310", "Nokia", 150M, "Ivan", new Battery("H223", new TimeSpan(10, 15,
0), new TimeSpan(5, 20, 0),
        BatteryType.LiIon), new Display(3M));

    foreach (GSM gsm in testGSMs)
    {
      Console.WriteLine(gsm);
    }

    Console.WriteLine(GSM.NokiaN95ToString());

    testGSMs[0].AddCall("8427153", 60);
    testGSMs[0].AddCall("8435153", 60);
    testGSMs[0].AddCall("8437153", 90);

    testGSMs[0].ViewCalls();
    Console.WriteLine("Total price = {0:C}", testGSMs[0].CalculatePrice(0.37M));

    testGSMs[0].RemoveCall("8437153");
    testGSMs[0].ViewCalls();
    Console.WriteLine("Total price = {0:C}", testGSMs[0].CalculatePrice(0.37M));

    testGSMs[0].ClearHistory();
    testGSMs[0].ViewCalls();
    Console.WriteLine("Total price = {0:C}", testGSMs[0].CalculatePrice(0.37M));
  }
}
```

20. There is a book library. Define classes respectively for a book and a library. The library must contain a name and a list of books. The books must contain the title, author, publisher, release date and ISBN-number. In the class, which describes the library, create methods to add a book to the library, to search for a book by a predefined author, to display information about a book and to delete a book from the library.

```
enum SearchOption { NameOfBook, Author }
enum DisplayOption { NameOfBook, Author }

class Library
{
   private string name;
   private List<Book> bookList;

   public Library(string name, List<Book> bookList = null)
   {
      this.name = name;

      if (bookList == null)
      {
         this.bookList = new List<Book>();
      }
      else
      {
         this.bookList = bookList;
      }
   }

   public string Name
   {
      get
      {
         return name;
      }
      set
      {
         this.name = value;
      }
   }

   public void Add(Book newBook)
   {
      this.bookList.Add(newBook);
   }

   public int IndexOf(string name, int startIndex, SearchOption option)
   {
      int foundIndex = -1;
```

```
    name = name.Trim();

    for (int i = startIndex; i < this.bookList.Count; i++)
    {

        if (option == SearchOption.Author)
        {
            if (this.bookList[i].Author == name)
            {
                foundIndex = i;
            }
        }
        else
        {
            if (this.bookList[i].Title == name)
            {
                foundIndex = i;
            }
        }
    }

    return foundIndex;
}

public void Display(string name, DisplayOption option)
{
    Console.WriteLine("{0,-3}|{1,-15}|{2,-15}", "N", "Title", "Author");
    for (int i = 0; i < this.bookList.Count; i++)
    {

        if (option == DisplayOption.Author)
        {
            if (this.bookList[i].Author == name)
            {
                Console.WriteLine("{0,-3}|{1,-15}|{2,-15}", i + 1, bookList[i].Title, bookList[i].Author);
            }
        }
        else
        {
            if (this.bookList[i].Title == name)
            {
                Console.WriteLine("{0,-3}|{1,-15}|{2,-15}", i + 1, bookList[i].Title, bookList[i].Author);
            }
        }
    }
}
```

```
    public void DisplayAll()
    {
        int count = 1;

        Console.WriteLine("{0,-3}|{1,-15}|{2,-15}","N","Title","Author");
        foreach (Book elem in this.bookList)
        {
            Console.WriteLine("{0,-3}|{1,-15}|{2,-15}",count, elem.Title, elem.Author);
            count++;
        }
    }

    public void DeleteAt(int index)
    {
        this.bookList.RemoveAt(index);
    }
}


class Book
{
    public Book(string title, string author, string publisher = "unknow", DateTime? date = null,
string isbn = "unknow")
    {
        this.Title = title;
        this.Author = author;
        this.Publisher = publisher;

        if (date == null)
        {
            this.PublishedDate = DateTime.Now;
        }
        else
        {
            this.PublishedDate = (DateTime) date;
        }

        this.ISBN = isbn;
    }

    public string Title { get; set; }
    public string Author { get; set; }
    public string Publisher { get; set; }
    public DateTime PublishedDate { get; set; }
    public string ISBN { get; set; }
}
```

21. Write a test class, which creates an object of type library, adds several books to it and displays information about each of them. Implement a test functionality, which finds all books authored by Stephen King and deletes them. Finally, display information for each of the remaining books.

```
class TestLibrary
{
  static void Main(string[] args)
  {
    Book firstBook = new Book("C#", "Svetlin Nakov");
    Book secondBook = new Book("Java", "Svetlin Nakov");
    Book thirdBook = new Book(".NET", "Svetlin Nakov");
    Book fourthBook = new Book("Ice and fire", "George Martin");

    Library telerikLib = new Library("Telerik Library");

    telerikLib.Add(firstBook);
    telerikLib.Add(secondBook);
    telerikLib.Add(thirdBook);
    telerikLib.Add(fourthBook);
    Console.WriteLine("Display library :");
    telerikLib.DisplayAll();

    int startIndex = 0;
    int indexFound;

    while (telerikLib.IndexOf("Svetlin Nakov", startIndex, SearchOption.Author) != -1)
    {
      indexFound = telerikLib.IndexOf("Svetlin Nakov", startIndex, SearchOption.Author);
      telerikLib.DeleteAt(indexFound);
    }
    Console.WriteLine("\nAfter deleting :");
    telerikLib.DisplayAll();
  }
}
```

22. We have a school. In school we have classes and students. Each class has a number of teachers. Each teacher has a variety of disciplines taught. Students have a name and a unique number in the class. Classes have a unique text identifier. Disciplines have a name, number of lessons and number of exercises. The task is to shape a school with C# classes. You have to define classes with their fields, properties, methods and constructors. Also define a test class, which demonstrates, that the other classes work correctly.

```
class SchoolProjectExample
{
  static void Main(string[] args)
  {
    SchoolExample();
  }

  private static void SchoolExample()
  {
    Teacher teacher = new Teacher("Gosho", "Stoqnov", 33);

    teacher.AddDiscipline(new Discipline("Bio", 5, 5));
    teacher.AddDiscipline(new Discipline("Chem", 1, 7));

    SchoolClass classB = new SchoolClass("123");
    classB.Add(new Student("Mitko", "Ivanov", 18, 13));
    classB.Add(new Student("Milko", "Ivanov", 19, 23));
    classB.Add(teacher);

    School johnAtanasov = new School();
    johnAtanasov.Add(classB);
  }
}


class School
{
  List<SchoolClass> classes;

  public School()
  {
    this.classes = new List<SchoolClass>();
  }

  public void Add(SchoolClass classToAdd)
  {
    this.classes.Add(classToAdd);
  }
}
```

```
class SchoolClass
{
   string UniqueId { get; set; }

   List<Student> students;
   List<Teacher> teachers;

   public SchoolClass(string uniqueId)
   {
      this.UniqueId = uniqueId;
      this.students = new List<Student>();
      this.teachers = new List<Teacher>();
   }

   public void Add(Student student)
   {
      this.students.Add(student);
   }

   public void Add(Teacher teacher)
   {
      this.teachers.Add(teacher);
   }

   public void Remove(Student student)
   {
      this.students.Remove(student);
   }

   public void Remove(Teacher teacher)
   {
      this.teachers.Remove(teacher);
   }
}


class Person
{
   string FirstName { get; set; }
   string LastName { get; set; }
   int Age { get; set; }

   public Person(string firstName, string lastName, int age)
   {
      this.FirstName = firstName;
      this.LastName = lastName;
      this.Age = age;
   }
}


class Teacher : Person
{
    List<Discipline> disciplinesTeached;

    public Teacher(string firstName, string lastName, int age)
        : base(firstName, lastName, age)
    {
        this.disciplinesTeached = new List<Discipline>();
    }

    public void AddDiscipline(Discipline discipline)
    {
        this.disciplinesTeached.Add(discipline);
    }
}


class Student : Person
{
    int UniqueClassNumber { get; set; }

    public Student(string firstName, string lastName, int age, int uniqueClassNumber)
        : base(firstName, lastName, age)
    {
        this.UniqueClassNumber = uniqueClassNumber;
    }
}


class Discipline
{
    string Name { get; set; }
    int LecturesCount { get; set; }
    int ExercisesCount { get; set; }

    public Discipline(string name, int lecturesCount, int exercisesCount)
    {
        this.Name = name;
        this.LecturesCount = lecturesCount;
        this.ExercisesCount = exercisesCount;
    }
}
```

23. Writea generic class GenericList<T>, which holds a list of elements of type T. Store the list of elements into an array with a limited capacity that is passed as a parameter of the constructor of the class. Add methods to add an item, to access an item by index, to remove an item by index, to insert an item at given position, to clear the list, to search for an item by value and to override the method ToString().

```
class GenericList<T>
{
   private int capacity;
   private int indexForNextAdd;

   T[] genericList;

   public GenericList(int capacity = 2)
   {
      this.capacity = capacity;
      this.genericList = new T[capacity];
      this.indexForNextAdd = 0;
   }

   public void Add(T elemToAdd)
   {
      if (indexForNextAdd == genericList.Length)
      {
         T[] newList = new T[this.capacity * 2];
         genericList.CopyTo(newList, 0);
         genericList = newList;
         this.capacity = this.capacity * 2;
      }

      genericList[indexForNextAdd] = elemToAdd;

      indexForNextAdd++;
   }


   public T this[int indx]
   {
      get
      {
         return genericList[indx];
      }
   }

   public void RemoveAt(int position)
   {
      if (position >= indexForNextAdd)
      {
         throw new ArgumentOutOfRangeException();
      }

      T[] resultList = new T[this.capacity];

      int elemLeft = this.indexForNextAdd - position - 1;

      Array.Copy(genericList, 0, resultList, 0, position);
      Array.Copy(genericList, position + 1, resultList, position, elemLeft);

      indexForNextAdd--;

      genericList = resultList;
   }

   public void AddAt(int position, T value)
   {
      if (position >= indexForNextAdd)
      {
         throw new ArgumentOutOfRangeException();
      }

      T[] newList;

      int elemLeft = this.indexForNextAdd - position;

      if (indexForNextAdd == genericList.Length)
      {
         newList = new T[this.capacity * 2];

         this.capacity = this.capacity * 2;
      }
      else
      {
         newList = new T[this.capacity];
      }

      Array.Copy(genericList, 0, newList, 0, position);
      newList[position] = value;
      Array.Copy(genericList, position, newList, position + 1, elemLeft);

      indexForNextAdd++;

      genericList = newList;
   }

   public void Clear()
   {
      genericList = null;
      this.indexForNextAdd = 0;
   }

   public int Find(T searcedValue, int start = 0)
   {
      int indexFound = -1;

      for (int i = start; i < this.genericList.Length; i++)
         {
         if (this.genericList[i].Equals(searcedValue))
         {
             indexFound = i;
             break;
         }
      }

      return indexFound;
   }

   public override string ToString()
   {
      StringBuilder result = new StringBuilder();

      for (int i = 0; i < indexForNextAdd; i++)
             {
         result.Append(String.Format("{0} ", genericList[i]));
      }

      if (indexForNextAdd == 0)
      {
         result.Append("No elements");
      }

      return result.ToString();
   }
}
```

24. Implement auto-resizing functionality of the array from the previous task, when by adding an element, it reaches the capacity of the array.

25. Define a class Fraction, which contains information about the rational fraction (e.g. ¼ or ½). Define a static method Parse() to create a fraction from a sting (for example -3/4). Define the appropriate properties and constructors of the class. Also write property of type Decimal to return the decimal value of the fraction (e.g. 0.25).

```
class Fraction
{
   private int sign;
   private int denominator;
   private int numerator;

   decimal decFraction;

   public Fraction(int numerator, int denominator, int sign)
   {
      this.numerator = numerator;
      this.denominator = denominator;
      this.sign = sign;

      if (this.numerator == 0)
      {
         this.decFraction = 0.00M;
      }
      else
      {
         this.decFraction = Convert.ToDecimal(this.numerator) / Convert.ToDecimal(this.denominator);
      }
   }

   public int Numerator
   {
      get
      {
         return this.numerator;
      }
   }

   public int Denominator
   {
      get
      {
         return this.denominator;
```

```
    }
}

public int Sign
{
    get
    {
        return this.sign;
    }
}

public decimal DecimalValue
{
    get
    {
        if (this.Sign == 1)
        {
            return this.decFraction * (-1);
        }
        else
        {
            return this.decFraction;
        }
    }
}

public static Fraction Parse(string fullFraction)
{
    int den;
    int num;
    int sign = 0;

    Match match;

    string pattern = @"([\-]*)\s*([0-9]+)\s*/\s*([0-9]+)";

    if (!Regex.IsMatch(fullFraction, pattern))
    {
        throw new FormatException("Incorrect Format");
    }
    else
    {
        match = Regex.Match(fullFraction, pattern);

        num = int.Parse(match.Groups[2].Value);
        den = int.Parse(match.Groups[3].Value);
    }
```

```
        if (match.Groups[1].Value != "")
              {
                     sign = 1;
              }

        return new Fraction(num, den, sign);
    }

    public override string ToString()
    {
        if (this.Numerator == 0)
        {
            return "0";
        }

        StringBuilder result = new StringBuilder();

        if (this.Sign == 1)
        {
            result.Append("-");
        }

        result.Append(this.Numerator);
        result.Append("/");
        result.Append(this.Denominator);

        return result.ToString();
    }
}
```

26. Write a class FractionTest, which tests the functionality of the class Fraction from previous task. Pay close attention on testing the function Parse with different input data.

```
class FractionTest
{
    static void Main(string[] args)
    {
        TestPrint();
    }

    public static void TestPrint()
    {
        Fraction a = Fraction.Parse(" -1 / 8");
        Fraction b = new Fraction(7, 8, 0);

        Console.WriteLine("Parse : \"-1 / 8\" : {0}", a);
        Console.WriteLine("a.Numerator : {0}", a.Numerator);
        Console.WriteLine("a.Denominator : {0}", a.Denominator);
        Console.WriteLine("a.DecimalValue : {0}", a.DecimalValue);

        Console.WriteLine("\nConstructor ///0 for sign//");
        Console.WriteLine("Fraction b = new Fraction(7, 8, 0) : {0}", b);

        Console.WriteLine("b + a = {0}", b + a);
    }
}
```

27. Write a function to cancel a fraction (e.g. if numerator and denominator are respectively 10 and 15, fraction to be cancelled to 2/3).

```
class Fraction
{
   int sign;
   int denominator;
   int numerator;

   decimal decFraction;

   public Fraction(int numerator, int denominator, int sign)
   {
      this.numerator = numerator;
      this.denominator = denominator;
      this.sign = sign;

      if (this.denominator != 0)
      {
         if (this.numerator == 0)
         {
            this.decFraction = 0.00M;
         }
         else
         {
            this.decFraction = Convert.ToDecimal(this.numerator) / Convert.ToDecimal(this.denominator);
         }
      }
      else
      {
         throw new DivideByZeroException("Invalid operation: devision by zero!");
      }
   }

   public int Numerator
   {
      get
      {
         return this.numerator;
      }
   }

   public int Denominator
   {
      get
      {
         return this.denominator;
```

```
    }
}

public int Sign
{
    get
    {
        return this.sign;
    }
}

public decimal DecimalValue
{
    get
    {
        if (this.Sign == 1)
        {
            return this.decFraction * (-1);
        }
        else
        {
            return this.decFraction;
        }

    }
}

public static Fraction Parse(string fullFraction)
{
    int den;
    int num;
    int sign = 0;

    Match match;

    string pattern = @"([\-]*)\s*([0-9]+)\s*/\s*([0-9]+)";

    if (!Regex.IsMatch(fullFraction, pattern))
    {
        throw new FormatException("Incorrect Format");
    }
    else
    {
        match = Regex.Match(fullFraction, pattern);

        num = int.Parse(match.Groups[2].Value);
        den = int.Parse(match.Groups[3].Value);
    }
```

```
   if (match.Groups[1].Value != "")
          {
                 sign = 1;
          }

   return new Fraction(num, den, sign);
}


public static Fraction operator +(Fraction firstFrac, Fraction secondFrac)
{
   int firstDen = firstFrac.Denominator;
   int secDen = secondFrac.Denominator;
   int sign = 0;

   int finalDen;
   int finalNum;

   int CommonDivisor;

   int firstNum = firstFrac.Numerator;
   int secondNum = secondFrac.Numerator;

   int LCM = LeastCommonMultiple(firstDen, secDen);

   firstNum = firstNum * (LCM / firstDen);
   secondNum = secondNum * (LCM / secDen);

   if (firstFrac.Sign == 1 && secondFrac.Sign == 1)
   {
      finalNum = firstNum + secondNum;
      sign = 1;
   }
   else if (firstFrac.Sign == 1)
   {
      finalNum = secondNum - firstNum;

      if (finalNum < 0)
      {
         finalNum = finalNum * (-1);
         sign = 1;
      }
   }
   else if (secondFrac.Sign == 1)
   {
      finalNum = firstNum - secondNum;
```

```
        if (finalNum < 0)
        {
            finalNum = finalNum * (-1);
            sign = 1;
        }
    }
    else
    {
        finalNum = firstNum + secondNum;
        sign = 0;
    }



    finalDen = LCM;

    if(finalNum == 0)
    {
        finalDen = 0;
        sign = 0;
    }
    else
    {
        CommonDivisor = GCD(finalNum, firstDen);

        finalDen = finalDen / CommonDivisor;
        finalNum = finalNum / CommonDivisor;
    }

    return new Fraction(finalNum, finalDen, sign);

}

static int LeastCommonMultiple(int first, int second)
{
    return (first * second) / GCD(first, second);
}

static int GCD(int first,int second)
{
    int temp;

    while (second != 0)
    {
       temp = second;
       second = first % second;
       first = temp;
    }
```

```
            return first;
        }

        public override string ToString()
        {
            if (this.Numerator == 0)
            {
                return "0";
            }

            StringBuilder result = new StringBuilder();

            if (this.Sign == 1)
            {
                result.Append("-");
            }

            result.Append(this.Numerator);
            result.Append("/");
            result.Append(this.Denominator);

            return result.ToString();
        }
    }
```