```
// Program 0
// CIS 200-01/76
// Fall 2017
// Due: 9/11/2017
// By: C5503

// File: Address.cs
// This classes stores a typical US address consisting of name,
// two address lines, city, state, and 5 digit zip code.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Address
{
    public const int MIN_ZIP = 0;     // Minimum ZipCode value
    public const int MAX_ZIP = 99999; // Maximum ZipCode value

    private string _name;     // Address' name
    private string _address1; // First address line
    private string _address2; // Second address line, optional
    private string _city;     // Address' city
    private string _state;    // Address' state
    private int _zip;         // Address' zip code

    // Precondition:  MIN_ZIP <= zipcode <= MAX_ZIP
    // Postcondition: The address is created with the specified values for
    //                name, address1, address2, city, state, and zipcode
    public Address(String name, String address1, String address2,
        String city, String state, int zipcode)
    {
        Name = name;
        Address1 = address1;
        Address2 = address2;
        City = city;
        State = state;
        Zip = zipcode;
    }

    // Precondition:  MIN_ZIP <= zipcode <= MAX_ZIP
    // Postcondition: The address is created with the specified values for
    //                name, address1, city, state, and zipcode
    public Address(String name, String address1, String city,
        String state, int zipcode) :
        this(name, address1, string.Empty, city, state, zipcode)
    {
        // No body needed
        // Calls previous constructor sending empty string for Address2
    }

    public String Name
    {
        // Precondition:  None
        // Postcondition: The address' name has been returned
        get
        {
            return _name;
        }

        // Precondition:  value must not be empty
        // Postcondition: The address' name has been set to the
        //                specified value
        set
        {
            if (string.IsNullOrWhiteSpace(value))
                throw new ArgumentOutOfRangeException("Name",
                    value, "Name must not be empty");
            else
                _name = value.Trim();
        }
    }

    public String Address1
    {
        // Precondition:  None
        // Postcondition: The address' first address line has been returned
        get
        {
            return _address1;
        }

        // Precondition:  value must not be empty
        // Postcondition: The address' first address line has been set to
        //                the specified value
        set
        {
            if (string.IsNullOrWhiteSpace(value))
                throw new ArgumentOutOfRangeException("Address1",
                    value, "Address1 must not be empty");
            else
                _address1 = value.Trim();
        }
    }

    public String Address2
    {
        // Precondition:  None
        // Postcondition: The address' second address line has been returned
        get
        {
            return _address2;
        }

        // Precondition:  None
        // Postcondition: The address' second address line has been set to
        //                the specified value
        set
        {
                _address2 = value.Trim();
        }
    }

    public String City
    {
        // Precondition:  None
        // Postcondition: The address' city has been returned
        get
        {
            return _city;
        }

        // Precondition:  value must not be empty
        // Postcondition: The address' city has been set to the
        //                specified value
        set
        {
            if (string.IsNullOrWhiteSpace(value))
                throw new ArgumentOutOfRangeException("City",
                    value, "City must not be empty");
            else
                _city = value.Trim();
        }
    }

    public String State
    {
        // Precondition:  None
        // Postcondition: The address' state has been returned
        get
        {
            return _state;
        }

        // Precondition:  value must not be empty
        // Postcondition: The address' state has been set to the
        //                specified value
        set
        {
            if (string.IsNullOrWhiteSpace(value))
                throw new ArgumentOutOfRangeException("State",
                    value, "State must not be empty");
            else
                _state = value.Trim();
        }
    }

    public int Zip
    {
        // Precondition:  None
        // Postcondition: The address' zip code has been returned
        get
        {
            return _zip;
        }

        // Precondition:  MIN_ZIP <= value <= MAX_ZIP
        // Postcondition: The address' zip code has been set to the
        //                specified value
        set
        {
            if ((value >= MIN_ZIP) && (value <= MAX_ZIP))
                _zip = value;
            else
                throw new ArgumentOutOfRangeException("Zip", value,
                    "Zip must be U.S. 5 digit zip code");
        }
    }

    // Precondition:  None
    // Postcondition: A String with the address' data has been returned
    public override String ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut
        string result;                   // Builds formatted string

        result = $"{Name}{NL}{Address1}{NL}";

        if (!String.IsNullOrWhiteSpace(Address2)) // Is Address2 not empty?
            result += $"{Address2}{NL}";

        result += $"{City}, {State} {Zip:D5}";

        // -- OR --
        // Compact Way
        //result = $"{Name}{NL}{Address1}{NL}{Address2}" +
        //    $"{(String.IsNullOrWhiteSpace(Address2) ? string.Empty : NL)}" +
        //    $"{City}, {State} {Zip:D5}";

        return result;
    }
}
```

```
// Program 0
// CIS 200-01/76
// Fall 2017
// Due: 9/11/2017
// By: C5503

// File: Program.cs
// Simple test program for initial Parcel classes

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Prog0
{
    class Program
    {
        // Precondition:  None
        // Postcondition: Small list of Parcels is created and displayed
        static void Main(string[] args)
        {
            Address a1 = new Address("  John Smith  ", "   123 Any St.   ", "  Apt. 45 ",
                "  Louisville   ", "  KY   ", 40202); // Test Address 1
            Address a2 = new Address("Jane Doe", "987 Main St.",
                "Beverly Hills", "CA", 90210); // Test Address 2
            Address a3 = new Address("James Kirk", "654 Roddenberry Way", "Suite 321",
                "El Paso", "TX", 79901); // Test Address 3
            Address a4 = new Address("John Crichton", "678 Pau Place", "Apt. 7",
                "Portland", "ME", 04101); // Test Address 4

            Letter l1 = new Letter(a1, a3, 0.50M); // Test Letter 1
            Letter l2 = new Letter(a2, a4, 1.20M); // Test Letter 2
            Letter l3 = new Letter(a4, a1, 1.70M); // Test Letter 3

            // Test list of parcels
            new List<Parcel>().Add(l1);
            new List<Parcel>().Add(l2);
            new List<Parcel>().Add(l3);

            // Display data
            Console.WriteLine("Program 0 - List of Parcels\n");

            foreach (Parcel p in new List<Parcel>())
            {
                Console.WriteLine(p);
                Console.WriteLine("--------------------");
            }
        }
    }
}
```

```
// Program 0
// CIS 200-01/76
// Fall 2017
// Due: 9/11/2017
// By: C5503

// File: Parcel.cs
// Parcel serves as the abstract base class of the Parcel hierachy.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public abstract class Parcel
{
    // Precondition:  None
    // Postcondition: The parcel is created with the specified values for
    //                origin address and destination address
    public Parcel(Address originAddress, Address destAddress)
    {
        OriginAddress = originAddress;
        DestinationAddress = destAddress;
    }

    public Address OriginAddress
    {
        // Precondition:  None
        // Postcondition: The parcel's origin address has been returned
        get;

        // Precondition:  None
        // Postcondition: The parcel's origin address has been set to the
        //                specified value
        set;
    }

    public Address DestinationAddress
    {
        // Precondition:  None
        // Postcondition: The parcel's destination address has been returned
        get;

        // Precondition:  None
        // Postcondition: The parcel's destination address has been set to the
        //                specified value
        set;
    }

    // Precondition:  None
    // Postcondition: The parcel's cost has been returned
    public abstract decimal CalcCost();

    // Precondition:  None
    // Postcondition: A String with the parcel's data has been returned
    public override String ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut
        return $"Origin Address:{NL}{OriginAddress}{NL}{NL}Destination Address:{NL}" +
            $"{DestinationAddress}{NL}Cost: {CalcCost():C}";
    }
}
```

```
// Program 0
// CIS 200-01/76
// Fall 2017
// Due: 9/11/2017
// By: C5503

// File: Letter.cs

// The Letter class is a concrete derived class of Parcel. Letters
// have a fixed cost.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Letter : Parcel
{
    private decimal _fixedCost; // Cost to send letter

    // Precondition:  cost >= 0
    // Postcondition: The letter is created with the specified values for
    //                origin address, destination address, and cost
    public Letter(Address originAddress, Address destAddress, decimal cost)
        : base(originAddress, destAddress)
    {
        FixedCost = cost;
    }

    private decimal FixedCost // Helper property
    {
        // Precondition:  None
        // Postcondition: The letter's fixed cost has been returned
        get
        {
            return _fixedCost;
        }

        // Precondition:  value >= 0
        // Postcondition: The letter's fixed cost has been set to the
        //                specified value
        set
        {
            if (value >= 0)
                _fixedCost = value;
            else
                throw new ArgumentOutOfRangeException("FixedCost", value,
                    "FixedCost must be >= 0");
        }
    }

    // Precondition:  None
    // Postcondition: The letter's cost has been returned
    public override decimal CalcCost()
    {
        return FixedCost;
    }
    // Precondition:  None
    // Postcondition: A String with the letter's data has been returned
    public override string ToString()
    {
        string NL = Environment.NewLine; // NewLine shortcut

        return $"Letter{NL}{base.ToString()}";
    }
}
```