```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static Activity4.Program;

namespace Activity4
{
    internal class Program
    {
        #region Constants
        public const int MinParcels = 1;    // Minimum number of parcels in a collection
        public const int MaxParcels = 6;    // Maximum number of parcels in a collection
        public const int MaxWeight = 30;    // Maximum weight of a parcel
        public const int MaxSize = 450;     // Maximum size of a parcel
        #endregion

        #region Structs
        /// <summary>
        /// Represents a customer
        /// </summary>
        public struct Customer
        {
            public string Name;            // Name of customer
            public string Address;         // Address of customer
            public string PhoneNumber;     // Phone number of customer

            /// <summary>
            /// A Customer
            /// </summary>
```

```
    /// <param name="name">Name of customer</param>
    /// <param name="address">Address of customer</param>
    /// <param name="phoneNumber">Phone number of customer</param>
    public Customer(string name, string address, string phoneNumber)
    {
        Name = name;
        Address = address;
        PhoneNumber = phoneNumber;
    }
}

/// <summary>
/// Represents a parcel
/// </summary>
public struct Parcel
{
    public int HeightCm;        // Height of parcel in cm
    public int LengthCm;        // Length of parcel in cm
    public int WidthCm;         // Width of parcel in cm
    public int WeightKg;        // Weight of parcel in kg

    public bool IsTracked;         // Is parcel being tracked?
    public bool NeedsSignature;    // Does the parcel need a signature on delivery?

    /// <summary>
    /// A Parcel
    /// </summary>
    /// <param name="heightCm">Height of parcel in cm</param>
    /// <param name="lengthCm">Length of parcel in cm</param>
    /// <param name="widthCm">Width of parcel in cm</param>
    /// <param name="weightKg">Weight of parcel in kg</param>
```

```
        /// <param name="isTracked">Is the parcel being tracked?</param>
        /// <param name="needsSignature">Does the parcel need a signature on delivery?</param>
        public Parcel(int heightCm, int lengthCm, int widthCm, int weightKg, bool isTracked, bool needsSignature)
        {
            HeightCm = heightCm;
            LengthCm = lengthCm;
            WidthCm = widthCm;
            WeightKg = weightKg;
            IsTracked = isTracked;
            NeedsSignature = needsSignature;
        }
    }
    #endregion

    #region Primitive datatype getters

    /// <summary>
    /// Get a integer from the user
    /// </summary>
    /// <param name="prompt">Prompt to user</param>
    /// <param name="min">Minimum value possible (Inclusive)</param>
    /// <param name="max">Maximum value possible (Inclusive)</param>
    /// <returns>User inputed integer within the specified boundaries</returns>
    public static int GetInt(string prompt, int min, int max)
    {
        bool valid;     // Is input valid
        int retv;    // Double to return

        // Loop until a valid input
        do
```

```
    {
        Console.Write(prompt);
        string input = Console.ReadLine();
        valid = int.TryParse(input, out retv);

        // Skip remaining validity checks if already invalid due to non integer input
        if (!valid)
        {
            Console.WriteLine("Invalid input. Not an integer");
        }

        // Ensure retv is between given boundaries
        else if (retv < min || retv > max)
        {
            Console.WriteLine("Invalid input. Not between {0} and {1}", min, max);
            valid = false;
        }

    }
    while (!valid);

    return retv;
}

/// <summary>
/// Get a string from the user
/// </summary>
/// <param name="prompt">Prompt to user</param>
/// <returns>User inputted string</returns>
public static string GetString(string prompt)
{
```

```
    string input;
    bool valid;
    do
    {
        Console.Write(prompt);
        input = Console.ReadLine().Trim();

        // Ensure a string is present
        valid = !string.IsNullOrEmpty(input);

        if (!valid)
        {
            Console.WriteLine("Invalid input.");
        }
    }
    while (!valid);

    return input;
}

/// <summary>
/// Get a boolean from the user
/// </summary>
/// <param name="prompt">Promptr to user</param>
/// <returns>User inputted boolean</returns>
public static bool GetBoolean(string prompt)
{
    ConsoleKey keyPress;

    Console.Write(prompt);
    do
```

```
    {
        keyPress = Console.ReadKey(true).Key;
    }
    while (keyPress != ConsoleKey.Y && keyPress != ConsoleKey.N);
    Console.WriteLine(keyPress.ToString());
    return keyPress == ConsoleKey.Y;
}
#endregion

/// <summary>
/// Create a new Parcel from user inputs
/// </summary>
/// <returns>Parcel created from user's inputs</returns>
public static Parcel CreateParcel()
{
    // Get parcel information from user
    int height = GetInt("What is the height of the parcel? (cm): ", 0, int.MaxValue);
    int length = GetInt("What is the length of the parcel? (cm): ", 0, int.MaxValue);
    int width = GetInt("What is the width of the parcel? (cm): ", 0, int.MaxValue);
    int weight = GetInt("What is the weight of the parcel? (kg): ", 0, int.MaxValue);
    bool isTracked = GetBoolean("Is the parcel tracked?: ");
    bool needsSignature = GetBoolean("Does the parcel need a signature?: ");


    return new Parcel(height, length, width, weight, isTracked, needsSignature);
}

/// <summary>
/// Create a new Customer from user inputs
/// </summary>
/// <returns>Customer created from user's inputs</returns>
```

```
public static Customer CreateCustomer()
{
   string name = GetString("What is the customer's name?: ");
   string address = GetString("What is the customer's address?: ");
   string phoneNumber = GetString("What is the customer's phone number?: ");

   return new Customer(name, address, phoneNumber);
}

/// <summary>
/// Find the price of collecting a Parcel
/// </summary>
/// <param name="parcel">Parcel to calculate the cost of</param>
/// <returns>Price of prarcel collection or null if not collectable</returns>
public static decimal? ParcelCost(Parcel parcel)
{
   int size = parcel.HeightCm + parcel.LengthCm + parcel.WidthCm;
   decimal price;

   // Check if parcel is collectable
   if (size > MaxSize || parcel.WeightKg > MaxWeight)
      return null;

   // If collectable set price
   else if (size > 150 || parcel.WeightKg > 15)
      price = 30M;
   else if (size > 95 || parcel.WeightKg > 2)
      price = 20M;
   else
      price = 5M;
```

```
    return price;
}

/// <summary>
/// Display customer details
/// </summary>
/// <param name="customer">Customer to display details of</param>
public static void CustomerDetails(Customer customer)
{
    Console.WriteLine("Customer:");
    Console.WriteLine("\tName: {0}",customer.Name);
    Console.WriteLine("\tAddress: {0}", customer.Address);
    Console.WriteLine("\tPhone Number: {0}", customer.PhoneNumber);
}

/// <summary>
/// Display Parcel's total cost
/// </summary>
/// <param name="parcel">Parcel to calculate and display details of</param>
/// <returns>Total cost of parcel or null if uncollectable</returns>
public static decimal? TotalParcelCost(Parcel parcel)
{
    decimal totalCost = 0;
    decimal? cost = ParcelCost(parcel);

    // Cannot be delivered
    if (cost == null)
    {
        Console.WriteLine("\t\tCannot be delivered!");
        return null;
    }
```

```
    else
    {
        // Default parcel cost
        decimal parcelCost = cost.Value;
        Console.WriteLine("\t\tCost: £{0:#.00}", parcelCost);

        // Add tracking feature
        if (parcel.IsTracked)
        {
            Console.WriteLine("\t\t\tTracking: + £5.00");
            parcelCost += 5M;
        }

        // Add signature feature
        if (parcel.NeedsSignature)
        {
            Console.WriteLine("\t\t\tSignature: + £2.00");
            parcelCost += 2M;
        }

        // Total cost only if additional features
        if (parcel.IsTracked || parcel.NeedsSignature)
            Console.WriteLine("\t\tTotal Cost: £{0:#.00}", parcelCost);

        totalCost += parcelCost;
    }
    return totalCost;
}

/// <summary>
/// Display the cost of the collection
```

```
/// </summary>
/// <param name="parcels"></param>
public static void CollectionCost(Parcel[] parcels)
{
   decimal totalCost = 0;
   Console.WriteLine("Collection:");

   // Display each parcel in collection
   for(int i = 0; i < parcels.Length; i++)
   {
      Console.WriteLine("\tParcel {0}", i + 1);
      decimal? parcelCost = TotalParcelCost(parcels[i]);

      // Increase collections cost if collectable
      if(parcelCost != null)
      {
         totalCost += parcelCost.Value;
      }

   }

   // Total cost of collection
   Console.WriteLine("Collection Cost: £{0:#.00}", totalCost);
}

/// <summary>
/// Entrypoint of program
/// </summary>
static void Main(string[] args)
{
   // Get Customer Info
```

```
        Customer customer = CreateCustomer();

        // Get parcels in collection
        int parcelsToCollect = GetInt("How many parcels need to be collected?: ", MinParcels,
MaxParcels);

        // Create and populate an array of parcels of the users choice
        Parcel[] parcels = new Parcel[parcelsToCollect];
        for(int i = 0; i < parcelsToCollect; i++)
        {
           parcels[i] = CreateParcel();
        }

        // Display Customer Details
        CustomerDetails(customer);

        // Display cost of collection
        CollectionCost(parcels);

        // Prevent closing until user input
        Console.WriteLine("Press any key to close!");
        Console.ReadKey(true);
      }
   }
}
```