ArrayList

|  |
| --- |
| ractical Learning: Introducing the ArrayList Class |
|  |

1. Access the ListCreator.cs file. On the main menu, click File -> New
2. Type the following:

|  |
| --- |
| using System;  namespace Business  {  interface IStockItem  {  string PartNumber { get; set; }  string PartName { get; set; }  decimal UnitPrice { get; set; }  }  }  namespace AutoParts  {  public class Part : Business.IStockItem  {  private string ID;  protected string name;  protected decimal price;  private int qty;  public string PartNumber  {  get { return ID;}  set { ID = value; }  }  public string PartName  {  get { return name; }  set { name = value; }  }  public decimal UnitPrice  {  get { return (price < 0) ? 0.00M : price; }  set { price = value; }  }  public int Quantity  {  get { return (qty < 0) ? 0 : qty; }  set { qty = value; }  }  public Part()  {  this.ID = null;  this.name = "Unknown";  this.qty = 0;  this.price = 0.00M;  }  public Part(string Nbr, string nm, int q, decimal pr)  {  this.ID = Nbr;  this.name = nm;  this.qty = q;  this.price = pr;  }  }  } |

1. On the main menu, click File -> New
2. When asked whether you want to save the changes, click Yes
3. Inside of the CSharp Lessons folder, locate the **Libraries1** folder and display it in the Save In combo box
4. Change the Save As Type to All Files
5. Set the file name to **Parts2.cs** and click Save
6. Open the Command Prompt and change to the above Libraries1 folder
7. To create the DLL, type **csc /target:library /out:PartCreatorR2.dll Parts2.cs**and press Enter
8. In the new empty file of Notepad, type the following:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class OrderProcessing  {  ArrayList ListOfParts;  public OrderProcessing()  {  ListOfParts = new ArrayList();  }  public void CreateInventory()  {  }  public void AddNewItem()  {  }  public void ShowInventory()  {  }  public void ProcessOrder()  {  }  public void DisplayReceipt()  {  }  }  } |

1. To save the file, on the main menu, click File -> Save
2. Locate the CSharp Lessons folder and display it in the Save In combo box
3. Create a new folder named **AutoParts2** and display it in the Save In combo box
4. Change the Save As Type to All Files. Set the File Name to **WorkOrder.cs**  and press Enter
5. Access the Exercise.cs file in the other instance of Notepad. On the main menu, click File -> New and type the following in it:

|  |
| --- |
| using System;  namespace AutoParts  {  class Exercise  {  static void Main()  {  OrderProcessing Order = new OrderProcessing();  }  }  } |

1. Save the new file as **Exercise.cs** in the same AutoParts2 folder
2. Open Windows Explorer or My Computer
3. From the CSharp Lessons\Libraries1 folder, copy ItemManagerR2.dll and paste it in the AutoParts2 folder

|  |
| --- |
| The Capacity of a List |
|  |

After declaring an **ArrayList** variable, it is empty. As objects are added to it, the list grows. The list can grow tremendously as you wish. The number of items of the list is managed through the memory it occupies and this memory grows as needed. The number of items that the memory allocated is currently using is represented by the **ArrayList.Capacity** property. This will usually be the least of your concerns.

If for some reason, you want to intervene and control the number of items that your **ArrayList** list can contain, you can manipulate the **Capacity** property. For example, you can assign it a constant to set the maximum value that the list can contain. Once, you will hardly have any reason to use the **Capacity** property: the compiler knows what to do with it.

If you set a fixed size on an **ArrayList** list, you may not be able to add a new item beyond the limit. In fact, if you attempt to do this, you may receive an error. A safe way is to check whether the list is fixed before performing a related operation. To find out whether a list is fixed, you can check the **ArrayList** variable's **IsFixedSize** property.

|  |
| --- |
| The Number of Items in the List |
|  |

When using a list, at any time, you should be able to know the number of items that the list contains. This information is provided by the **ArrayList.Count** property. The Capacity and the Count have this in common: the value of each increases as the list grows and the same value decreases if the list shrinks. It is important to know that, although they look alike, there are various differences between the capacity of a list and the number of items it contains. **Capacity** is a read/write property. This means that, as we saw above, you can assign a value to the capacity to fix the number of items that the list can contain. You can also retrieve the value of the Capacity. The **Count** is read-only because it is used by the compiler to count the current number of items of the items and this counting is performed without your intervention.

|  |
| --- |
| A Read-Only List |
|  |

One of the reason for creating a list is to be able to add items to it, edit its items, retrieve an items, or delete items from it. These are the default operations. You can still limit these operations as you judge them unnecessary. For example, you may create a list and then initialize it with the items that you want the list to only have. If you don't intend to have the user adding items to it, you can create the list as read-only. To do this, you can call the **ArrayList.ReadOnly()** method. It is overloaded with two versions as follows:

public static ArrayList ReadOnly(ArrayList);

public static IList ReadOnly(IList);

This method is static. This means that you don't need to declare an instance of **ArrayList** to call them. Instead, to make the list read-only, call the **ArrayList.ReadOnly()** method and pass your **ArrayList** variable to it.

As we will see in the next sections, some operations cannot be performed on a read-only list. To perform such operations, you can first find out whether an **ArrayList** list is read-only. This is done by checking its **IsReadOnly** property.

|  |
| --- |
| Item Addition |
|  |

The primary operation performed on a list is to create one. One of the biggest advantages of using a linked list is that you don't have to specify in advance the number of items of the list as done for an array. You can just start adding items. The **ArrayList** class makes this possible with the **Add()** method. Its syntax is:

public virtual int Add(object value);

The argument of this method is the value to add to the list. If the method succeeds with the addition, it returns the position where the value was added in the list. This is usually the last position in the list. If the method fails, the compiler would throw an error. One of the errors that could result from failure of this operation would be based on the fact that either a new item cannot be added to the list because the list is read-only, or the list was already full prior to adding the new item. Normally, a list can be full only if you had specified the maximum number of items it can contain using the **ArrayList.Capacity** property. As mentioned above, the list can be made read-only by passing its variable to the **ArrayList.ReadOnly()** method.

|  |
| --- |
| Practical Learning Practical Learning: Adding Items to an ArrayList List |
|  |

1. To create an inventory, access the WorkOrder.cs file and change it as follows:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class OrderProcessing  {  ArrayList ListOfParts;  public OrderProcessing()  {  ListOfParts = new ArrayList();  }  // This method is used to create an inventory  public void CreateInventory()  {  Part One;    // Create a Part object using its properties  One = new Part();  One.PartNumber = "GD646";  One.PartName = "Bearing Clutch Pilot ";  One.UnitPrice = 9.75M;  One.Quantity = 4;  // Add the new part to the list  ListOfParts.Add(One);  // Create a Part object using a constructor  One = new Part("EU473", "Belt Accessory Drive ", 10, 6.75M);  // Add the new part to the list  ListOfParts.Add(One);  // Do the same to complete the list  One = new Part("AH325", "Break Drum ", 5, 20.55M);  ListOfParts.Add(One);  One = new Part("KS745", "Right Mirror ", 2, 9.35M);  ListOfParts.Add(One);  One = new Part("KE374", "Break Shoe ", 6, 20.25M);  ListOfParts.Add(One);  One = new Part("GD943", "Signal Lamp Assembly ", 4, 74.55M);  ListOfParts.Add(One);  One = new Part("GH386", "Bearing Input Shaft ", 3, 45.25M);  ListOfParts.Add(One);  One = new Part("WD394", "Brake Disc ", 14, 85.50M);  ListOfParts.Add(One);  One = new Part("TR944", "Front Wheel Lug Nut ", 7, 10.75M);  ListOfParts.Add(One);  One = new Part("GD844", "Front Pump Gasket ", 6, 10.72M);  ListOfParts.Add(One);  One = new Part("GD933", "Filter Steering ", 4, 12.55M);  ListOfParts.Add(One);  One = new Part("GW478", "Air Control Valve ", 8, 35.25M);  ListOfParts.Add(One);  One = new Part("LA943", "Clutch Master Clndr ", 5, 124.55M);  ListOfParts.Add(One);  One = new Part("RU688", "Tie Rod ", 12, 32.55M);  ListOfParts.Add(One);  One = new Part("PP797", "Ball Joint ", 14, 25.75M);  ListOfParts.Add(One);  One = new Part("RA292", "Drive Belt ", 10, 10.65M);  ListOfParts.Add(One);  One = new Part("AG778", "Oil Filter ", 8, 6.25M);  ListOfParts.Add(One);  One = new Part("KQ820", "Timing Belt ", 1, 45.95M);  ListOfParts.Add(One);  One = new Part("GT722", "Intake Manifold Gask ", 4, 18.55M);  ListOfParts.Add(One);  One = new Part("WA502", "Spark Plug Seal ", 24, 4.15M);  ListOfParts.Add(One);  One = new Part("AL848", "Air Filter ", 32, 15.65M);  ListOfParts.Add(One);  One = new Part("RU382", "Fuel Injector Clip ", 12, 17.05M);  ListOfParts.Add(One);  One = new Part("HJ624", "Brk Caliper w/o Pads ", 3, 190.50M);  ListOfParts.Add(One);  One = new Part("RL555", "Crankshaft Seal ", 7, 10.55M);  ListOfParts.Add(One);  One = new Part("PQ273", "Oil Pump ", 16, 218.75M);  ListOfParts.Add(One);  One = new Part("ER162", "Timing Belt Tensioner ", 12, 264.55M);  ListOfParts.Add(One);  One = new Part("EY275", "Camshaft Seal ", 8, 8.95M);  ListOfParts.Add(One);  One = new Part("LM357", "Valve Cover Gasket ", 1, 22.75M);  ListOfParts.Add(One);  One = new Part("RU473", "Valve Stem Seal ", 1, 3.95M);  ListOfParts.Add(One);  One = new Part("QW374", "Starter ", 1, 320.65M);  ListOfParts.Add(One);  One = new Part("QR374", "Radiator Cap ", 14, 12.75M);  ListOfParts.Add(One);  One = new Part("PQ902", "Thermostat Gasket ", 9, 4.20M);  ListOfParts.Add(One);  One = new Part("QT847", "Water Pump ", 5, 12.95M);  ListOfParts.Add(One);  One = new Part("PY784", "Spark Plug Platinum ", 14, 145.85M);  ListOfParts.Add(One);  One = new Part("TQ483", "Tie Rod Assembly ", 12, 3.95M);  ListOfParts.Add(One);  One = new Part("EQ173", "Oil Pump ", 20, 155.75M);  ListOfParts.Add(One);  One = new Part("UG376", "Piston Ring Set ", 13, 218.75M);  ListOfParts.Add(One);  One = new Part("PI489", "Distributor Cap ", 1, 275.55M);  ListOfParts.Add(One);  One = new Part("BT389", "Oil Seal Front Pump ", 18, 7.05M);  ListOfParts.Add(One);  One = new Part("CQ274", "Transmitter Filter Kit", 22, 9.25M);  ListOfParts.Add(One);  One = new Part("QX202", "Tail Lamp Assembly ", 7, 5.05M);  ListOfParts.Add(One);  One = new Part("GN780", "Bearing Wheel ", 5, 40.15M);  ListOfParts.Add(One);  One = new Part("XZ485", "Left Mirror ", 8, 7.25M);  ListOfParts.Add(One);  One = new Part("BD199", "Caliper Bolt/Pin ", 8, 3.55M);  ListOfParts.Add(One);  }  // This method is used to add a new part to the list  public void AddNewItem()  {  string ID;  string Name;  decimal Price;  int qty;  // Ask the user to type a number for the new part  Console.Write("Enter Item Number (Example: PD764): ");  ID = Console.ReadLine();  // Then ask the user to provide additional information about the part  Console.WriteLine("Enter the name or a short description: ");  Name = Console.ReadLine();  Console.Write("Enter Unit Price: ");  Price = decimal.Parse(Console.ReadLine());  Console.Write("How Many? ");  qty = int.Parse(Console.ReadLine());  // Using the new information that the user provided  // Create a new Part object using the second constructor  Part NewPart = new Part(ID, Name, qty, Price);    // Once the part is ready, add it to the database  ListOfParts.Add(NewPart);  }  public void ShowInventory()  {  }  public void ProcessOrder()  {  }  public void DisplayReceipt()  {  }  }  } |

1. Save the file
2. To test the above code, access the Exercise.cs file and change it as follows:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class Exercise  {  static void Main()  {  int Choice = 0;  OrderProcessing Order = new OrderProcessing();  Order.CreateInventory();  // Display a short menu to the user before taking an action  try  {  Console.WriteLine(" =-= College Park Auto Parts =-=");  Console.WriteLine("How may I help you?");  Console.WriteLine("1. I want to process a customer's order");  Console.WriteLine("2. I want to see the current inventory");  Console.WriteLine("3. I want to add a new item to the inventory");  Console.Write("Your choice (1, 2, or 3)? ");  Choice = int.Parse(Console.ReadLine());  }  catch(FormatException)  {  Console.WriteLine("\nInvalid Choice - The program will terminate\n");  }  // Take an action based on the user's choice  switch(Choice)  {  case 1:  Order.ProcessOrder();  break;  case 2:  Order.ShowInventory();  break;  case 3:  Order.AddNewItem();  break;  }  }  }  } |

1. Save the file
2. Access the Command Prompt and change to the AutoParts2 folder
3. To compile the application, **csc /reference:PartCreatorR2.dll / out:"Four-Corner Auto-Parts R2".exe WorkOrder.cs Exercise.cs**and press Enter
4. To execute the application, type **"Four-Corner Auto-Parts R2"**and press Enter
5. After testing the program, return to Notepad

|  |
| --- |
| Item Retrieval |
|  |

Once a list is ready, you can perform different types of operations on it. Besides adding items, one of the most regular operations performed on a list consists of locating and retrieving its items. You have various options. To retrieve a single item based on its position, you can apply the square brackets of arrays to the variable. Like a normal array, an **ArrayList** list is zero-based. Another issue to keep in mind is that the **ArrayList**[] returns an Object value. Therefore, you may have to cast this value to your type of value to get it right.

|  |
| --- |
| Practical Learning Practical Learning: Retrieving Items From an ArrayList List |
|  |

1. To show an inventory, access the WorkOrder.cs file and change it as follows:

|  |
| --- |
| using System;  using System.Collections;  namespace AutoParts  {  class OrderProcessing  {  ArrayList ListOfParts;  public OrderProcessing()  {  ListOfParts = new ArrayList();  }  public void CreateInventory()  {  . . . *No Change*  }  public void AddNewItem()  {  string ID;  string Name;  decimal Price;  int qty;  Console.Write("Enter Item Number (Example: PD764): ");  ID = Console.ReadLine();  Console.WriteLine("Enter the name or a short description: ");  Name = Console.ReadLine();  Console.Write("Enter Unit Price: ");  Price = decimal.Parse(Console.ReadLine());  Console.Write("How Many? ");  qty = int.Parse(Console.ReadLine());  Part NewPart = new Part(ID, Name, qty, Price);  ListOfParts.Add(NewPart);  ShowInventory();  }  public void ShowInventory()  {  Console.WriteLine("\n===============================================");  Console.WriteLine("=-= College Park Auto Parts =-= Store Inventory");  Console.WriteLine("-----------------------------------------------");  Console.WriteLine(" Item # Description Price Qty");  for(int i = 0; i < ListOfParts.Count; i++)  {  Part One = (Part)ListOfParts[i];  Console.WriteLine(" {0} {1} {2,6}{3,5}",  One.PartNumber, One.PartName,  One.UnitPrice, One.Quantity);  }  Console.WriteLine("===============================================\n");  }  public void ProcessOrder()  {  ArrayList Choices = new ArrayList();  Part AnItem;  string PartID;  int Qty;    do  {  Console.Write("Enter the part number (q to stop): ");  PartID = Console.ReadLine();  for(int i = 0; i < ListOfParts.Count; i++)  {  AnItem = new Part();  if( PartID == ((Part)ListOfParts[i]).PartNumber)  {  AnItem.PartNumber = ((Part)ListOfParts[i]).PartNumber;  AnItem.PartName = ((Part)ListOfParts[i]).PartName;  AnItem.UnitPrice = ((Part)ListOfParts[i]).UnitPrice;    try  {  Console.Write("How many? ");  Qty = int.Parse(Console.ReadLine());  AnItem.Quantity = Qty;  }  catch(FormatException)  {  Console.WriteLine("Invalid Quantity!!!");  }  Choices.Add(AnItem);  break;  }  }  } while( PartID != "q" && PartID != "Q" );    DisplayReceipt(Choices);  }  public void DisplayReceipt(ArrayList lstItems)  {  decimal SubTotal = 0.00M,  TotalOrder = 0.00M;    Console.WriteLine("========================================================");  Console.WriteLine("=-= College Park Auto Parts =-= Receipt");  Console.WriteLine("------+---+-------------------------+-------+-----------");  Console.WriteLine("Part# Qty Description Price SubTotal");  Console.WriteLine("------+---+-------------------------+-------+-----------");    for(int i = 0; i < lstItems.Count; i++)  {  Part One = (Part)lstItems[i];  SubTotal = One.UnitPrice \* One.Quantity;  TotalOrder += SubTotal;  Console.WriteLine("{0} {1} {2} {3,6} {4,6}",  One.PartNumber, One.Quantity, One.PartName,  One.UnitPrice,SubTotal);  }    Console.WriteLine("------+---+-------------------------+-------+-----------");  Console.WriteLine("Total Order: {0:C}", TotalOrder);  Console.WriteLine("========================================================\n");  }  }  } |

1. Save, compile, and test the program. Here is an example:

|  |
| --- |
| =-= College Park Auto Parts =-=  How may I help you?  1. I want to process a customer's order  2. I want to see the current inventory  3. I want to add a new item to the inventory  Your choice (1, 2, or 3)? 1  Enter the part number (q to stop): GN780  How many? 1  Enter the part number (q to stop): CQ274  How many? 1  Enter the part number (q to stop): RL555  How many? 2  Enter the part number (q to stop): WA502  How many? 2  Enter the part number (q to stop): PY784  How many? 1  Enter the part number (q to stop): q  ========================================================  =-= College Park Auto Parts =-= Receipt  ------+---+-------------------------+-------+-----------  Part# Qty Description Price SubTotal  ------+---+-------------------------+-------+-----------  GN780 1 Bearing Wheel 40.15 40.15  CQ274 1 Transmitter Filter Kit 9.25 9.25  RL555 2 Crankshaft Seal 10.55 21.10  WA502 2 Spark Plug Seal 4.15 8.30  PY784 1 Spark Plug Platinum 145.85 145.85  ------+---+-------------------------+-------+-----------  Total Order: $224.65  ======================================================== |

1. Return to Notepad

|  |
| --- |
| Item Location |
|  |

Instead of the square brackets that allow you to retrieve an item based on its position, you can look for an item based on its complete definition. You have various options. You can first "build" an item and ask the compiler to check whether any item in the list matches your definition. To perform this search, you can call the **ArrayList.Contains()** method. Its syntax is:

public virtual bool Contains(object item);

The item to look for is passed as argument to the method. The compiler would look for exactly the item, using its definition, in the list. If any detail of the argument fails to match any item of the **ArrayList** list, the method would return false. If all characteristics of the argument correspond to an item of the list, the method returns true.

Another option to look for an item in a list consists of calling the **ArrayList.BinarySearch()** method. It is overloaded in three versions and one of them uses the following syntax:

public virtual int BinarySearch(object value);

The item to look for is passed argument to the method.

|  |
| --- |
| Item Deletion |
|  |

As opposed to adding an item to a list, you may want to remove one. To perform this operation, you have various options. You can ask the compiler to look for an item in the list and if, or once, the compile finds it, it would delete the item. To perform this type of deletion, you can call the **ArrayList.Remove()** method. Its syntax is:

public virtual void Remove(object obj);

This method accepts as argument the item that you want to delete from the list. To perform this operation, the list must not be read-only.

The **Remove()** method allows you to specify the exact item you want to delete from a list. Another option you have consists of deleting an item based on its position. This is done using the **RemoveAt()** method whose syntax is:

public virtual void RemoveAt(int index);

With this method, the position of the item is passed as argument. If the position is not valid because either it is lower or higher than the current **Count**, the compiler would throw an **ArgumentOutOfRangeException** exception.

To remove all items from a list at once, you can call the **ArrayList.Clear()** method. Its syntax is:

public virtual void Clear();