**Monday, 4/18 –**

* **Character data type** – individual character; delimited with single quotes.
* **String data type** – set of characters; delimited with double quotes.
* **[]** – indexer operator; gets an individual character out of the string**.**
* **First character in a string** – has a position of 0
* **// use foreach loop**

foreach (char letter in name)

{

letter = name[index]; **// Error; point of “foreach” is to get 1 element out of each space.**

MessageBox.Show(letter.ToString());

}

* **Totals.Length –** totals is the object, length is the property; not static
* **Array.Sort(totals) – ,**static method
* **MessageBox.Show -** ,static method
* **Static method –**
* **IndexOf –** Shows the index of starting point.
* **Trim, TrimStart, TrimEnd -**  will eliminate leading and trailing whitespace; always trim data from user.
* **Insert –** modifies a copy of original string and puts it in a new variable.
* **Split –**
* **EXTRA CREDIT FOR 8-1 – Write a method; bool HasUpperCase(string str)**

**Wednesday, 4/20 –**

* **Reverse** – apple, start with last character in string and concatenate to new string, work backward and add each character to the new string. Can’t use foreach because they start at beginning.
* Input > output
* **Shift Cypher** – abcdefghijklmnop…….z (charsToShift) 1 cab = dbc/4 cab=gef
  + for each char c in input
    - index = find the index of char in alphabet

**Monday, 4/25 –**

* **Concentration Extra Credit:** Don’t do regular cards, colors, pics and word numbers, etc.
* **#region - #endregion** – when you have lots of code, can make it expandable and collapsible.
* **bin > debugger > cards** – folder with card images
  + card2c.jpg – 2=value, c=suit (FYI: values=2-9, t(ten), j(jack), q(queen), a(ace)/suit=c(club), d(diamonds), s(spades), h(hearts))
* **Shuffle** – Pick two random numbers in array and swap them twenty times.
  + **Algorithm: For loop around outside X 10 if you want more shuffle.**
    - For index = 1 to 20
      * rndIndex = a random # between 1 & 20
      * string temp = card at index
      * card at index = card at rndIndex
      * card at rndIndex = temp
* Every form has an array of “Controls”

**Wednesday, 4/27 –**

* **Concentration:**
* **sender** – first parameter of event handler is always an object “sender;” it’s what triggered the event, thing you click on. In this case, it is a picture box. Generic data type used as placeholder.
* **int cardIndex = int.Parse(card.Name.Substring(4)); -** get the 4th character in the string, parse to int and save in variable.
* **IsMatch –** Looks at filenames in array; not PB – index in PB and index in filename are the same. Matches with numbers not suits.
  + **Substring(4, 1);**

**Monday, 5/9 –**

* **Zen** – Object-oriented programming; teaching something how to be itself.
* **Class Diagram –** set of rectangles divided into 3 places
* When you design a thing you are really creating a class… Not an object.
* **Class** – Blueprint or design
* **Object** – actual implementation of class
* **Properties –** takes attributes and ???
* **Getting a property** – receives value
* **Setting a property –** changes value
* **Forward Engineering –**
* **Reverse Engineering –**
* **Fields in VS** – variables in class diagram (instance variables)
* **Constructor** – special method whose name is the same as the name of the class
* **(+1 overload) –**
* **Getter –** Read value (GetSuit/SetSuit)
* **Setter –** Change value (SetSuit/SetValue)
* **Instantiation –** when you create an object
* **Encapsulation –** when we build a class we take all of the attributes and characteristics, methods and behaviors and wrap them up in class
* **Data hiding –** hiding data from the outside world when we build classes, protecting data that makes the object what it is from tampering from others.
* **DLL** – is file, in a windows environment, that contains code that you can link to dynamically at runtime, library of classes.
* **N tier? Applications –**
* Business Object = Domain object = tictactoe board = card class
* **3-layer application** – GUI, Business, database
* **Product class –** 3 attributes, code (string that contains a code that uniquely identifies each product.) > Description (string that contains a description of the product.) > Price (decimal that contains the product’s price.)
* **Class names always start with a capital letter**
* **The Product class: Fields and constructors**

using System;

namespace ProductMaint

{

public class Product // **Class names start with capital letter**

{

// **Instance variables or fields; “private” for data hiding.**

private string code;

private string description;

private decimal price;

public Product(){} // **(default) constructor, know this because it has the same name as the class**

public Product(string code, string description, // **another (overloaded) constructor**

decimal price)

{

this.Code = code; // **parameters set equal to instance variables**

this.Description = description;

this.Price = price;

}

* **this –** refers to the calling object
* The Product class: The Code property //**writing a property**
* public string Code
* {
* get
* {
* return code;
* }
* set
* {
* code = value;
  + }
* **Lab 4 – refer to slides**

**Wednesday, 5/11 –**

* **Testing a class –** call one of your methods, then print something to screen, compare to what you expect to see
* **Override –** to override the tostring that’s part of the .net framework
* **Console application –** command line
* **Main –** is the entry point method for every c# app., part of app that actually runs, for GUI: creates form and then runs it.

**Wednesday, 5/18 –**

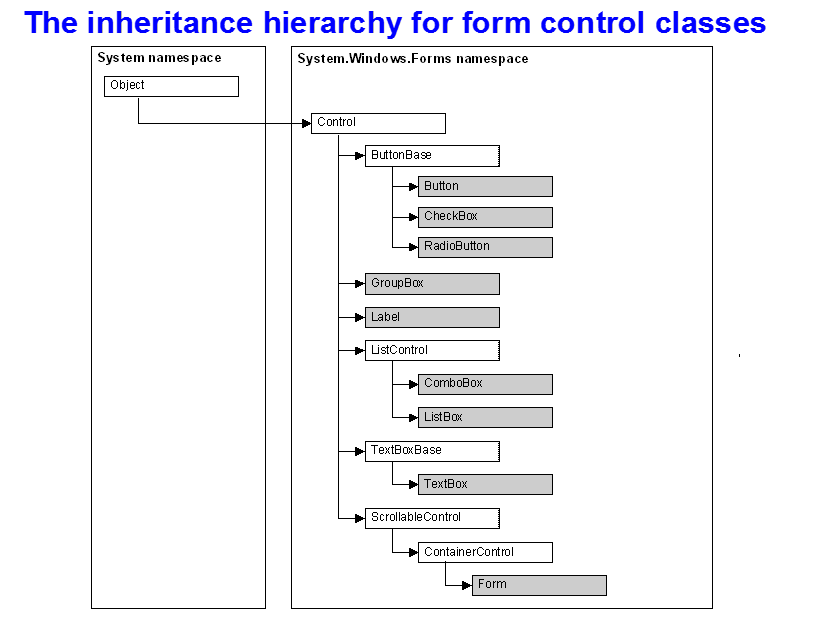
* Skip delegates and events – Tell Mari if there are questions about it on the quiz.
* What things should you be able to do with a “List” – add, remove, sort, check something off, search, edit item, constructors, count
* Product ProductList
  + Code << has a “list of >> count
  + Desc Add
  + Price remove

Fill

Save

* List is an easier data structure to work with than arrays if you are adding and removing things, they grow and shrink automatically as opposed to arrays that you have to say exactly how big it is.
* There is a list class that can store any kind of object and a list class that stores specific objects
* Specific
  + private List<Product> “*data type”* products; *“list name variable”*
* Instance Variable?
* When there is a getter, and no setter it is a read-only property… Because you don’t want to change the value.
* Static means belongs to the class as a whole don’t call it on an object
* Find files to test fill and save on CustomerList file = “Customers.txt”
* Lab 6 ignore 2-3 and 8, have already done 5
  + Focus on deck and hand class
  + GetCard use get product by index as reference
  + Discard is like deal only you say which index to remove from, return and take out of list
  + Index of returns integer, -1 if not in there otherwise returns an index
  + Has card returns Boolean

**Monday, 05/23 –**

* **Indexer that uses an integer as an index** –
  + Product p = products[0]; >>> *getter*
    - Gets first element from products list
      * P = product data type, products = product list, 0 = integer
  + Product p2 = new Product(…); >>> *setter*
  + Products[0] = p2;
    - P2 = value
* **this** – refers to the calling object
* **Extra Credit**: use an indexer to fan out cards and let user pick one, add indexer to deck, hand, or both classes.
* **Inheritance** - Get all data in previous class(es) as well as add it’s own; related classes; one class drives the other(s).
* **Every class in C# is derived from a class called “Object.”**
  + ****
* **Private -**
* **Public –**
* **Protected –** “in between,” can use it in class and derived classes
* **virtual -**  tells compiler that “it’s ok with me if the derived class overrides this.”
* **: -** Operator means inheritance
  + public class Book : Product >>> *means book is a product*
* **: base -**  refers to base class, constructor, to set the parameters that follow.
* **Lab 7 –**
  + WS – Company
  + Retail – Home Phone
* **To relate classes type : and name of class.**