



## **THIS IS YOUR WORKBOOK**

This booklet contains the great bulk of what you will need throughout your bootcamp. Please, please, please don't lose it! In these glorious pages, you'll find information about the job assistance program, a glossary of important terms, information about Grand Circus and your program in general, and the labs that you'll need daily throughout the program.

This is your workbook! Write your name: \_\_\_\_\_

My Grand Circus Program Manager is: \_\_\_\_\_

Phone Number: \_\_\_\_\_

Email: \_\_\_\_\_

Get ready!



## **ABOUT GRAND CIRCUS**

Grand Circus is a training institute in the heart of downtown Detroit and Grand Rapids on a mission to elevate the tech community. We offer training, co-working and event hosting -- all under the same tent.

When it comes to training we dismiss that true skill comes with a degree. We focus instead on outcomes that matter. With project-based instruction our training delivers real world expertise. We call it training with a purpose.

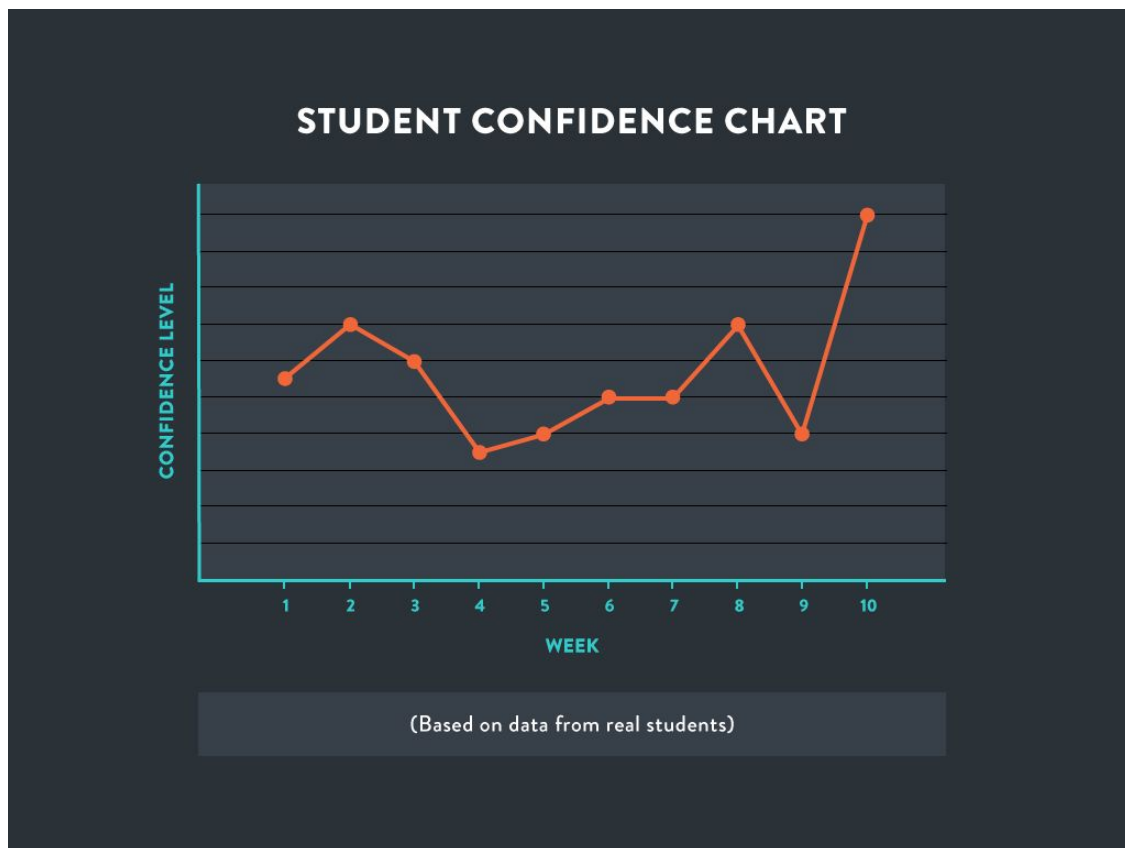
Our curriculum is built on the latest in technology, business and design and we have partnered with the best and brightest. Our instructors are real practitioners who are at the top of their fields and committed to the success of their students.



## STUDENT CONFIDENCE THROUGHOUT THE BOOTCAMP

The Bootcamp you are about to begin will qualify you with the confidence and technical skills to bring your tech career to the 21st century. You will soon experience the challenges that will put your patience and self-confidence to the test. Rest assured, **the emotional rollercoaster is completely normal**. The learning curve may be steep but you will come out the other side stronger and equipped for success.

Your journey over the next few weeks might look like the following graph.



*WHEN YOU FEEL LIKE QUITTING,  
THINK ABOUT WHY YOU STARTED.*



# **DURING YOUR BOOTCAMP**

## **GLOSSARY & LABS**



# LAB PROMPTS

There are a few labs set aside for each day of class. There are prompts for you to begin in class and others in a separate section marked **BONUS** for you to get extra practice if you are looking to challenge yourself. Most days of your bootcamp have one to three labs associated with it. Most of these labs have extended exercises to help you really hone your skills. Extended exercises can be completed in class once you complete your lab, or at home to continue practicing. Always push yourself with extended exercises before jumping into the **BONUS** materials.

We ask though that you don't skip ahead. If you need additional work because you are completing all the labs and their corresponding extended exercises in class, let your Program Manager and/or instructor know. We want you to be challenged!



## LAB NUMBER: 2

**Task:** Calculate the perimeter and area of various classrooms at Grand Circus.

### What will the application do?

- The application prompts the user to enter values of length and width of the classroom.
- The application displays the area and perimeter of that classroom.
- The application prompts the user to continue (keep measuring rooms!).

### Build Specifications

1. Assume that the rooms are perfect rectangles.
2. Assume that the user will enter valid numeric data for length and width.
3. The application should accept decimal entries.

### Hints:

- Don't mess up the formulas for area or perimeter.
- The Snug is 24' 6" x 20' 0". The Penthouse is 42' 6" x 16' 6".

### Extra Challenges:

- The application should continue only if the user agrees to.
- Calculate the volume of the rooms.

### Console Preview:

Welcome to Grand Circus' Room Detail Generator!

Enter Length: {user input here, for example: 24.5}

Enter Width: {user input here, for example: 20}

Area: {Answer will show up here}

Perimeter: { Answer will show up here }

Continue? (y/n): {user input here, for example: Y}

Enter Length: {3}

Enter Width: {4}

Area: {12}

Perimeter: {14}

...



## LAB NUMBER: 3

**Task:** Use conditional statements to automate the decision-making process.

### What will the application do?

- The application prompts the user to enter an integer between 1 and 100.
- Display the associated result based on the integer range entered.

### Build Specifications

- Use if/else statements to take different actions depending on user input.
- Given an integer entered by a user, perform the following conditional actions:
  - If the integer entered is odd, print the number entered and "Odd."
  - If the integer entered is even and in the inclusive range of 2 to 25, print "Even and less than 25."
  - If the integer entered is even and in the inclusive range of 26 to 60, print "Even."
  - If the integer entered is even and greater than 60, print the number entered and "Even."
  - If the integer entered is odd and greater than 60, print the number entered and "Odd."

### Extra Challenges:

- Include a set of parameters so that the program ends officially.
- Ask for user information (ex. name) at the beginning of the application, and use it to refer to the user throughout the application.
- Add validation to guarantee that a user enters a positive integer between 1 and 100.





## LAB NUMBER: 3

### Console Preview:

**Enter a number between 1 and 100:** {user input here, for example: 3}

**Output:** {output here, 3 and Odd }

[

Continue? (y/n): {user input here, for example: Y}

**Enter a number between 1 and 100:** {user input here, for example: 24}

**Output:** {output here, Even and less than 25 }

Continue? (y/n): {user input here, for example: Y}

**Enter a number between 1 and 100:** {user input here, for example: 75}

**Output:** {output here, 75 and Odd }

Continue? (y/n): {user input here, for example: N}

Bye!



## LAB NUMBER: 4

**Task:** Display a table of powers.

**What will the application do?**

- The application prompts the user to enter an integer.
- The application displays a table of squares and cubes from 1 to the value entered.
- The application prompts the user to continue.

**Build Specifications**

1. Assume that the user will enter valid data.
2. The application should continue only if the user agrees to.

**Hints:**

- Don't mess up the difference between squares and cubes!

**Extra Challenges:**

**Console Preview:**

Learn your squares and cubes!

Enter an integer: {user input here, for example: 5}

Number	Squared	Cubed
=====	=====	=====
1	1	1
2	4	8
3	9	27
4	16	64
5	25	125

Continue? (y/n): {user input here, for example: Y}

Enter an integer: ...



## LAB NUMBER: 5

**Task:** Calculate the factorial of a number.

### What will the application do?

- The application prompts the user to enter an integer from 1 to 10.
- The application displays the factorial of the number entered by the user.
- The application prompts the user to continue.

### Build Specifications

1. Use a *for loop* to calculate the factorial.
2. Assume that the user will enter valid data.
3. Use the *long* type to store the factorial.
4. The application should continue only if the user agrees to.

### Hints:

- A factorial is a number multiplied by each of the numbers before it. Factorials are denoted by the exclamation point (n!). Ex:
  - $1! = 1$  which equals 1
  - $2! = 1 \times 2$  which equals 2
  - $3! = 1 \times 2 \times 3$  which equals 6
  - $4! = 1 \times 2 \times 3 \times 4$  which equals 24

### Extended Challenges:

- Test the application and find the integer for the highest factorial that can be accurately calculated by this application, then modify the prompt so that it prompts the user for a number "from 1 to {the highest integer that returns accurate factorial calculation}".
- Use Recursion to implement the factorial.



## LAB NUMBER: 5

### Console Preview:

Welcome to the Factorial Calculator!

Enter an integer that's greater than 0 but less than 10: {user input here, for example: 3}  
The factorial of 3 is 6.

Continue? (y/n): {user input here, for example: Y}

Enter an integer that's greater than 0 but less than 10: {user input here, for example: 9}  
The factorial of 9 is 362880.

...



## LAB NUMBER: 6

**Task:** Create an application that simulates dice rolling.

### What will the application do?

- The application asks the user to enter the number of sides for a pair of dice.
- The application prompts the user to roll the dice.
- The application “rolls” two n-sided dice, displays the results of each, and then asks the user if he/she wants to roll the dice again.

### Build Specifications

1. Use static methods to implement the method(s) that generate the random numbers.
2. Experiment with different Random number generators that you can find on the internet.

### Hints:

- Use the Random class to generate a random number.

### Extended Challenges:

- Use the DiceRollerApp class to display special messages for craps, snake eyes, and box cars.

### Console Preview:

```
Welcome to the Grand Circus Casino! Roll the dice? (y/n): {user input here, for example:
y}
How many sides should each die have? { user entered 6 }

Roll 1:
2
5

Roll again? (y/n): ...
```



## LAB NUMBER: 7

**Task:** Write a program that will recognize invalid inputs using regular expressions.

**What will the application do?**

- The program will validate different kinds of input.

**Build Specifications**

1. Write a method that will validate names. Names can only have alphabets, they should start with a capital letter, and they have a maximum length of 30.
2. Write a method that will validate emails. An email should be in the following format: {combination of alphanumeric characters, with a length between 5 and 30, and there are no special characters}@{combination of alphanumeric characters, with a length between 5 and 10, and there is no special characters}.{domain can be combination of alphanumeric characters with a length of two or three}
3. Write a method that will validate phone numbers. A phone number should be in the following format: {area code of 3 digits} – {3digits} – {4 digits}
4. Write a method that will validate date based on the following format: (dd/mm/yyyy).

**Hints:**

- Use <https://regexr.com/> to try out the regular expressions before adding them to your C# code.

**Extended Challenges:**

- Write a method that validates HTML elements (Example: `<p> </p>` is a valid html element, and `<h1 <h1>` is not valid. Don't worry about special cases where you have self-contained HTML elements).

**Console Preview:**

```
Please enter a valid Name: James123
Sorry, name is not valid!

Please enter a valid email: james@james.com
Email is valid!

Please enter a valid phone number: 333443443434
Sorry, phone is not valid!
Please enter a valid date: 3/4/18
Sorry, date is not valid!
```



## LAB NUMBER: 8

**Task:** Write a program that will recognize invalid inputs when the user requests information about students in a class.

### What will the application do?

- Provide information about students in a class
- Prompt the user to ask about a particular student
- Give proper responses according to user-submitted information
- Ask if user would like to learn about another student

### Build Specifications

1. Account for invalid user input with exceptions
2. Try to incorporate `IndexOutOfRangeException` and `FormatException`

### Hints:

- Make it easy for the user – tell them what information is available

### Console Preview:

```
Welcome to our C# class. Which student would you like to learn more about? (enter a
number 1-20): 100

That student does not exist. Please try again. (enter a number 1-20): 10

Student 10 is Kim Driscoll. What would you like to know about Kim? (enter or
"hometown" or "favorite food"): age

That data does not exist. Please try again. (enter or "hometown" or "favorite food"):
hometown

Kim is from Detroit, MI. Would you like to know more? (enter "yes" or "no"): no

Thanks!
```



## LAB NUMBER: 9

**Task:** Improve your student information system from the previous lab by using a collection.

### Build Specifications

- Refactor your code to use Lists rather than arrays.
- Add another list to store another piece of information (so perhaps favorite color or favorite number) about each student. Update your validation to allow the user to select that third piece of information.
- Each iteration of the loop, ask first if they'd like to find out info about a student or add another student. If they choose to add another student, get the name and each piece of info and add them to the end of the list. Validate input--don't accept blanks for any of the questions.

### Extended Exercises

- Create the original lists in alphabetical order by student name. When a user adds a new student, insert them at the correct location alphabetically. Remember to put the information about a particular student at the same index in each list!
- If you already know about creating classes, go back and rewrite this:
  - Make a StudentInfo class with name and other info (hometown, food, whatever you chose) as data members.
  - Use a single List of StudentInfo instances to store the information.





## LAB NUMBER: 10

**Task:** Calculate a circle's circumference and area.

### What will the application do?

- The application prompts the user to enter a radius.
- The application displays an error if the user enters invalid data.
- When the user enters valid data, the application calculates the area and circumference of the circle and rounds to the nearest 2 decimal places.
- The application prompts the user to continue.
- The application displays a "goodbye" message that also indicates the number of circles the user built when the user chooses not to continue.

### Build Specifications

1. Create a class named *Circle* to store the data about this circle. This class should contain these constructors and methods:
  1. `public Circle(double radius)`
  2. `public double CalculateCircumference()`
  3. `public string CalculateFormattedCircumference()`
  4. `public double CalculateArea()`
  5. `public string CalculateFormattedArea()`
  6. `private string FormatNumber(double x)`
  7. Define a member called `radius`. This member should be private.
  8. Define a property to get access to the class member: `Radius`
2. For the value of pi, use the PI constant of the `System.Math` class.
3. In the Main method, get the user input, create a `Circle` object, and display the circumference and area.

### Hints:

- Don't mess up the formulas for circumference or area of a circle!

### Extended Challenge:

- Create a class named `Validator` and use its static methods to validate the data in this application.

### Console Preview:

```
Welcome to the Circle Tester
Enter radius: 3
Circumference: 18.85
Area:          28.27
Continue? (y/n): n
Goodbye. You created 2 Circle object(s).
```



## LAB NUMBER: 11

**Task:** List movies by category.

### What will the application do?

- The application stores a list of 10 movies and displays them by category.
- The user can enter any of the following categories to display the films in the list that match the category: animated, drama, horror, scifi.
- After the list is displayed, the user is asked if he or she wants to continue. If no, the program ends.

### Build Specifications

1. Each movie should be represented by an object of type Movie. The Movie class must provide two private fields: title and category. Both of these fields should be Strings. The class should also provide a constructor that accepts a title and category as parameters and uses the values passed to it to initialize its fields.
2. When the user enters a category, the program should read through all of the movies in the List and display a line for any movie whose category matches the category entered by the user.
3. Sufficient tests should be present.

### Extended Exercises:

- Standardize the category codes by displaying a menu of categories and having the user select the category by number rather than entering the name.
- Display the movies for the selected category in alphabetical order.

### Console Preview:

```
Welcome to the Movie List Application!

There are 100 movies in this list.
What category are you interested in? scifi
Star Wars
2001: A Space Odyssey
E.T. The Extra-terrestrial
A Clockwork Orange
Close Encounters of the Third Kind

Continue? (y/n): Y
What category are you interested in? ...
```

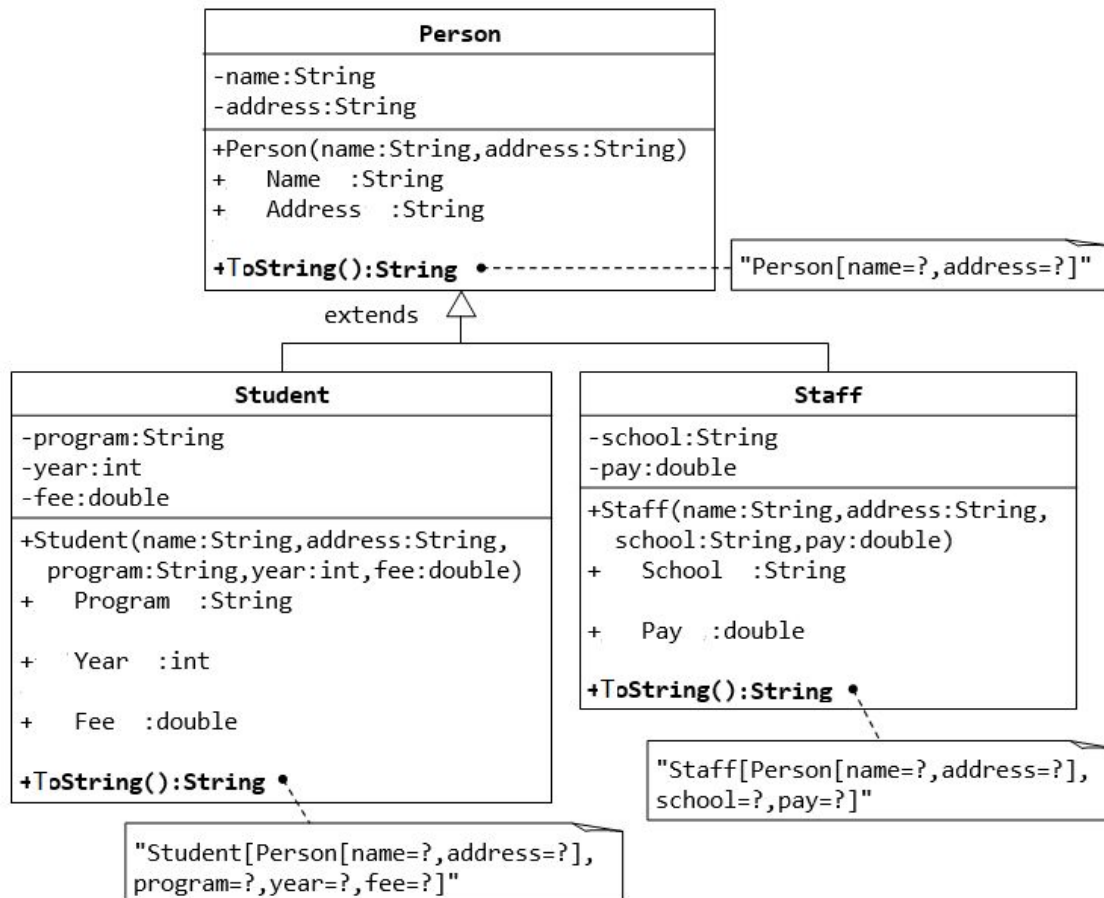


## LAB NUMBER: 12

**Task:** Design the following classes based on the provided UML diagram.

### Build Specifications

1. Each class has three parts, the class name, then the properties, and last is the methods. For each class, look at the provided UML diagram to know what properties and methods that you need to implement.
2. The properties are defined as private and the methods are defined as public.
3. Override the ToString method that is coming from the Object class. Each class will use the ToString method to print the values of its properties. For instance, the Person class, the ToString method will print the name and the address.
4. The Student and the Staff classes will extend the Person class. You can use "base" to call the Person class constructor from the constructors of those sub classes.
5. Add a default constructor to each class.



## LAB NUMBER: 13

**Task:** Create a rock, paper, scissors game.

### What will the application do?

- The application prompts the player to enter a name and select an opponent.
- The application prompts the player to select rock, paper, or scissors. Then, the application displays the player's choice, the opponent's choice, and the result of the match.
- The application continues until the user doesn't want to play anymore.
- If the user makes an invalid selection, the application should display an appropriate error message and prompt the user again until the user makes a valid selection.

### Build Specifications

1. Create an enumeration called Roshambo that stores three values: rock, paper, and scissors.
2. Create an abstract class named Player that stores a name and a Roshambo value. This class should include an abstract method named generateRoshambo that allows an inheriting class to generate and return a Roshambo value.
3. Create and name two player classes. One player should always select rock. The other player should randomly select rock, paper, or scissors (a 1 in 3 chance of selecting each).
4. Create a third player class that inherits the Player class and implements the generateRoshambo method. This method can return any value you choose.
5. Create a class called RoshamboApp that allows the user to play as either of your first two player classes.
6. Use a Validator class to validate the user's entries.

### Hints:

- Paper beats rock, rock beats scissors, scissors beats paper.

### Extended Challenges:

- Keep track of wins and losses, and display them at the end of each session.



## LAB NUMBER: 13

### Console Preview:

```
Welcome to Rock Paper Scissors!

Enter your name: Chioke

Would you like to play against Thejets or TheSharks (j/s)? : j

Rock, paper, or scissors? (R/P/S): r

Chioke: rock
Thejets: rock
Draw!

Play again? (y/n): Y

Rock, paper, or scissors? (R/P/S): p

Chioke: paper
Thejets: rock
Chioke wins!

Play again? (y/n): y

Rock, paper, or scissors? (R/P/S): s

Chioke: scissors
Thejets: rock
Thejets wins!

Play again? (y/n): N
```



## LAB NUMBER: 14

**Task:** Counting alligators, cloning sheep.

### What will the application do?

- The application uses an Alligator class that implements a Countable interface to display Alligator objects as shown in the console preview.
- Sheep class that implements a Countable interface and the Cloneable interface to display and clone Sheep objects as shown in the console preview.

### Build Specifications

1. Create an interface named Countable that can be used to count an object. This interface should include these methods:

```
void IncrementCount()
void ResetCount()
int GetCount()
String GetCountString()
```

2. Create a class named Alligator that implements the Countable interface. This class should include an instance variable that stores the count and a method that returns the formatted count.
3. Create a class named CountUtil. This class should include a static method that lets you count any Countable objects a specified number of times. For example:

```
public static void Count(Countable c, int MaxCount)
```

4. Create a class named CountTestApp that uses the CountUtil class to count an Alligator object 3 times as shown in the console preview.
5. Create a class named *Sheep* that implements the Countable and Cloneable interfaces. This class should include an instance variable that stores the count and the name of the sheep, and it should provide methods that can set and get the name of the sheep.
6. Modify the CountTestApp class so it (a) counts the first sheep 2 times, (b) clones the first sheep, changes the name, and counts it 3 times, and (c) counts the first sheep again 1 time.



## LAB NUMBER: 14

### Console Preview:

Counting Alligators...

1 alligator  
2 alligator  
3 alligator

Counting Sheep...

1 Blackie  
2 Blackie

1 Dolly  
2 Dolly  
3 Dolly

1 Blackie



## LAB NUMBER: 15

**Task:** Build Spoon pool.

**Goal:** Build a code that implements the Object pool design pattern.

### Build Specifications:

In this lab, *SingleSpoon* class holds one instance of *SingleSpoon* in "private static *SingleSpoon theSpoon;*" The *SingleSpoon* class determines the spoons availability using "private static bool *theSpoonIsAvailable* = true;"

After you create the class to implement a single spoon, create a spoon "pool" you would have the same basic logic, however multiple spoons would be distributed.

Note that this code is not thread safe. To make it thread safe all you would need is to make the *GetTheSpoon()* method synchronized.





## LAB NUMBER: 16

**Task:** Maintain a list of countries.

### What will the application do?

- The application begins by showing a menu with three options.
- If the user chooses option 1, the application displays a list of countries that have been saved in a file.
- If the user chooses option 2, the application prompts the user to enter a country and then it writes the country to the file of countries.
- If the user chooses option 3, the application displays a goodbye message and exits.

### Build Specifications

- Create a class named `CountriesTextFile` that contains one method that allows you to read a list of countries from a file and another method that allows you to write a list of countries to a file.
- Store the list of countries in a text files named `countries.txt` in the same directory as the `CountriesTextFile` class. If the `countries.txt` file doesn't exist, `CountriesTextFile` class should create it. The class should close all I/O streams when they are not needed.
- Create a class called `CountriesApp` that displays a menu and responds to user choices.

### Hint:

- Use a `Validator` class to validate user entries.

### Extended Exercises:

- Modify the `CountriesApp` class so it includes a menu choice that allows the user to delete a country from the file.



## LAB NUMBER: 16

### Console Preview:

Welcome to the Countries Maintenance Application!

1 - See the list of countries

2 - Add a country

3 - Exit

Enter menu number: 1

India

The United States

China

Rwanda

1 - See the list of countries

2 - Add a country

3 - Exit

Enter menu number: 2

Enter country: Brazil

This country as been saved!

1 - See the list of countries

2 - Add a country

3 - Exit

Enter menu number: 1

India

The United States

China

Rwanda

Brazil

1 - See the list of countries

2 - Add a country

3 - Exit

Enter menu number: 3

Buh-bye!



## LAB NUMBER: 17

**Task:** Using Test Driven Development practices, create an application that locates prime numbers.

### What will the application do?

- The application locates prime numbers in a sequence (ex: 1<sup>st</sup> prime number is 2, 10<sup>th</sup> prime number is 29, etc.).
- The application prompts the user which sequenced prime they want to locate.
- The application displays the prime number and its sequence number, and prompts the user to find another prime.
- When the user chooses to end his or her search, the application displays a goodbye message.

### Build Specifications

1. Use test driven development practices to complete your application.
2. Pair program (yes, really – close one of the computers).
3. Once you have a functioning program, check in your code with the instructor.
4. Refactor!
5. Push your completed application to GitHub.

### Console Preview:

Let's locate some primes!

This application will find you any prime, in order, from first prime number on.

Which prime number are you looking for? 6

The number 6 prime is 13.

Do you want to find another prime number? (y/n): y

Which prime number are you looking for? ...



## LAB NUMBER: 18

**Tasks:** Add methods to the LinkedList data structure, and design an algorithm.

1. Add the following methods to your LinkedList class:
  1. boolean RemoveAt (int index): Remove an object at a specified Index. Returns false if the index is out of range.
  2. void PrintReverse(): Prints the elements of the list in reverse.
  3. boolean InsertAt (int index, Object o): Inserts a new object at a specified index. Returns false if the index is out of range.

2. Design an algorithm that counts the occurrences of numbers in an array.

Example: If an array has the following content:

{1,2,3,2,2,4,5,5,7,8,4,4,1,0,10}, the frequency output should be:

[0]: 1

[1]: 2

[2]: 3

[4]: 3

[5]: 2

[7]: 1

[8]: 1

[10]: 1

1. Use an array to find the frequency.
2. Use a HashMap to find the frequency.

Estimate the Big-O for your code.



## LAB NUMBER: 19

**Task:** Practice writing SQL statements on the Northwind database.

Write SQL queries to do the following:

1. Select all the records from the "Customers" table.
2. Get distinct countries from the Customers table.
3. Get all the records from the table Customers where the Customer's ID starts with "BI".
4. Get the first 100 records of the orders table.
5. Get all customers that live in the postal codes 1010, 3012, 12209, and 05023.
6. Get all orders where the ShipRegion is not equal to NULL.
7. Get all customers ordered by the country, then by the city.
8. Add a new customer to the customers table. You can use whatever values.
9. Update all ShipRegion to the value 'EuroZone' in the Orders table, where the ShipCountry is equal to France.
10. Delete all orders from OrderDetails that have quantity of 1.
11. Calculate the average, max, and min of the quantity at the orderdetails table.
12. Calculate the average, max, and min of the quantity at the orderdetails table, grouped by the orderid.
13. Find the CustomerID that placed order 10290 (orderstable)
14. Do an inner join, left join, right join on orders and customers tables.
15. Get employees' firstname for all employees who report to no one.
16. Get employees' firstname for all employees who report to Andrew.



## LAB NUMBER: 20

**Task:** Create a personal website for yourself and to display your projects.

### Build Specifications

1. Create at least 3 pages: home, about, and portfolio
2. Your homepage should include 2-4 sections about you. For example: the first section focuses on blogging, the second focuses on developing, and the third focuses on your hometown.
3. Include a header with navigation as well as a footer for all pages.
4. Include links like LinkedIn and other professional social media.
5. Your Portfolio page should include a description and link to projects you want to highlight.
6. Your *About* page should include your bio and a way for visitors/employers to contact you.

### Extended Exercises

- Stick the header to the top of the page so that it always shows when you scroll down the page.
- Make your website responsive to device sizes by using Media Queries or a CSS framework like Bootstrap.
- Use the Google Maps API to map your hometown (or Detroit or Grand Circus).
- Learn about GitHub Pages and push there so your site is public.



## COFFEE SHOP APPLICATION

Through the next sequence of labs, you will build a simple web application for a website that will sell coffee supplies. The Coffee Web Application will have the following features at the end:

1. Register Users.
2. Login Users.
3. List products.
4. Add, delete, and update the products shown on the site.



## LAB NUMBER: 21

**Task:** Create a User Registration Form.

### What does the application do?

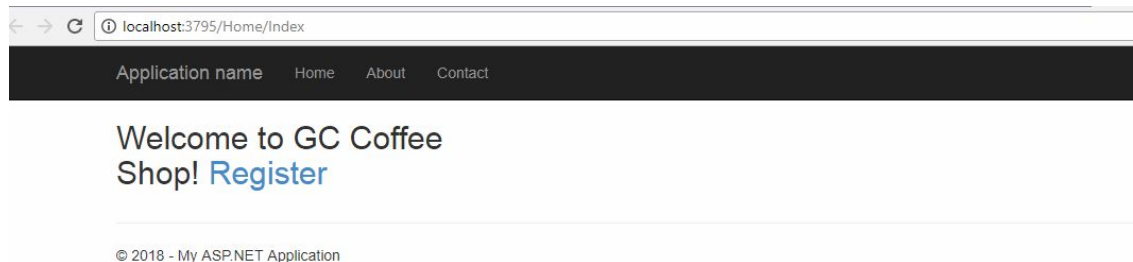
1. The Index View should have an action link pointing to the registration page.
2. The registration page will register a new user. The form should take in the following fields: User Name, Email, and Password.
3. When submitted, the page should redirect the user to a summary on a different page. This page will show a message => Welcome {UserName}
4. Go crazy! Design the form by adding different kinds of controls, such as text boxes, drop down lists, radio buttons, and check boxes.

### Build Specifications

1. Create the Registration form in HTML, then call an action to handle the input. Experiment with method="GET" and method="POST" to send the information from the form. Which do you think is more secure? Is there any sensitive data left out in the open?
2. Do form validation! Use HTML5, Controller C# code, or any other methods you can come up with to do validation.

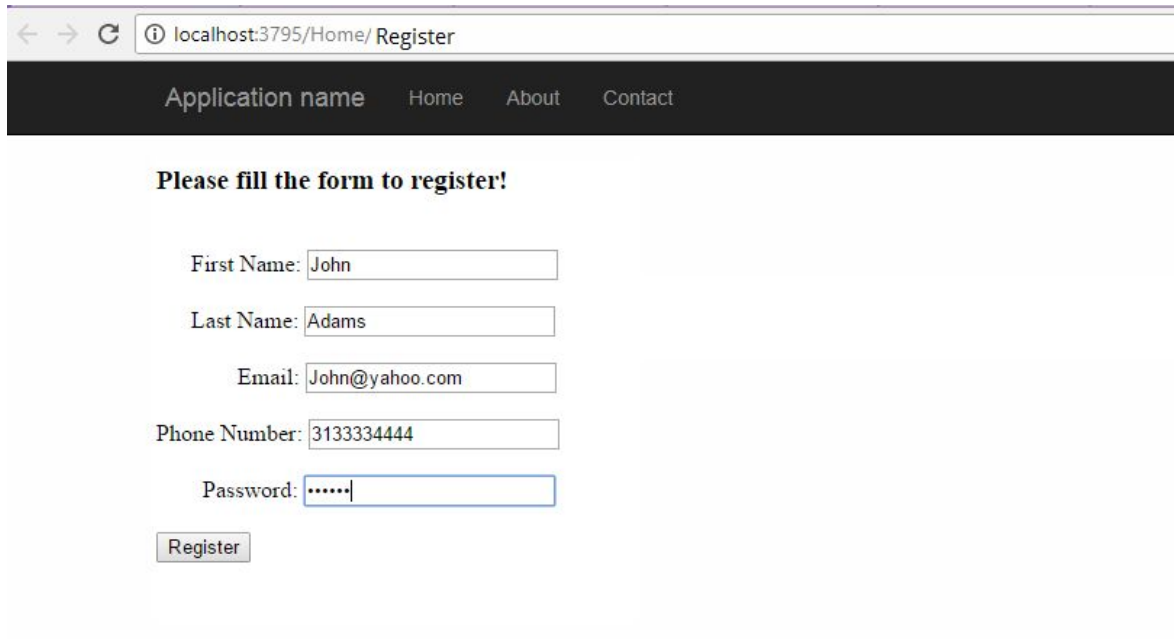
**The following are screenshots for a basic solution:**

### Index Page





## Registration Page



← → ↻ ⓘ localhost:3795/Home/Register

Application name Home About Contact

**Please fill the form to register!**

First Name:

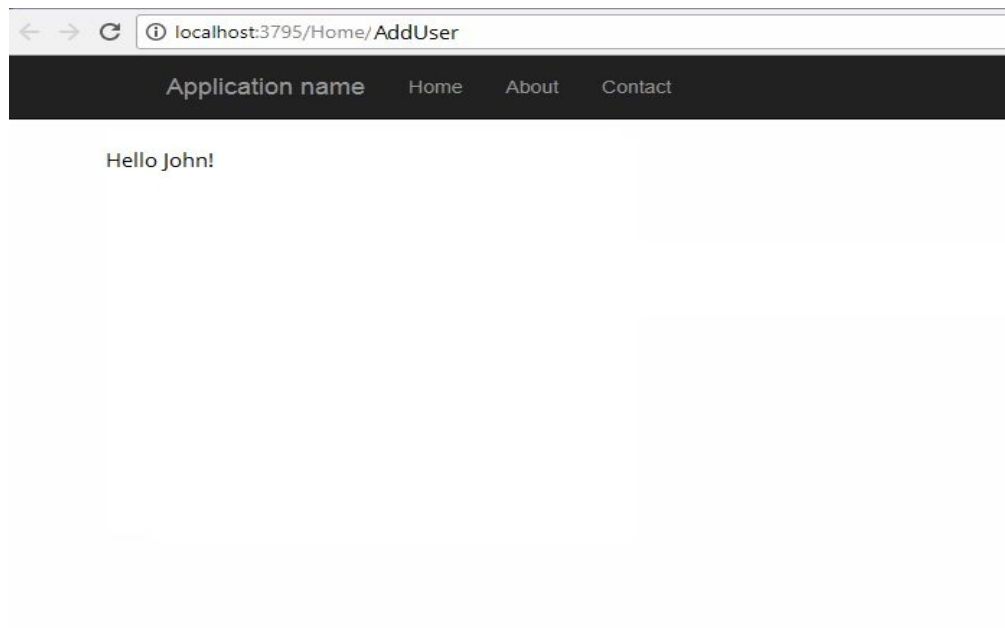
Last Name:

Email:

Phone Number:

Password:

## Summary Page



← → ↻ ⓘ localhost:3795/Home/AddUser

Application name Home About Contact

Hello John!



## LAB NUMBER: 22

**Task:** Expand and improve The Coffee Shop App

### What does the application do?

- Offers more fields on the registration form from last lab.
- Provides additional validation.
- Displays all information from a user model

### Build Specifications

1. Add at least three additional fields of whatever types you want.
  - Experiment with the HTML <fieldset> tag for keeping related form elements together.
  - Experiment with dropdowns, radio buttons, check boxes, etc.
2. Add validation for at least two of those fields. (Others can be optional.) Think about:
  - Add a password confirmation text box.
  - Testing if an email field has a valid email address.
  - Checking if a phone number is properly formatted.
3. Create a new view to display all the properties of a User model. Experiment with the different ways to do this:
  - In the controller action, put each user property into a different viewbag, then in the view, print each viewbag into the page.
  - In the controller action, create a new User object, then pass it to the view. At the top of the view use an @model statement to import your model. What code do you need to print the properties?
  - Right click on the views folder and select add -> View. Experiment with the different templates .Net allows you to use.



## LAB NUMBER: 23

### This lab has three main tasks

1. Add Users into the database using Entity
2. Log in an existing user
3. If the user has money, let them purchase items off the shop page

### What does the application do?

- Save the user information from the User's registration page to the database.
- Enable the User to buy items on the Shop page. The items can be whatever products you wish to sell (Keep it work appropriate)

### Build Specifications

1. Create a database in SQL server and call it "ShopDB".
2. Next, create a table called "Users". In that table, store the Users' data when they register.
3. Using database first and Entity, create a User Model that will point to your table.
4. In your controller, create 2 actions Register() and MakeNewUser(User u)
5. Create a register view page with a form for each property present in the user model. In your form set method to post. Upon submitting invoke the MakeNewUser action.
6. In the MakeNewUser action, use your DB context to add the new user. Create a DB object and call the right methods to insert your new row. The syntax for this looks like so: DB.Users.Method().
7. In the same database, Create a table called "Items" with the following columns: Name, description, quantity, and price.
8. Create a view page call Shop that takes in a list of items and display them. If a user is stored in the session display BUY buttons for each item. Otherwise hide them
9. Pre-populate the table with sample data.
10. Create a Login page with a form with UserName and Password. If the inputted information matches a user in the database, store the user in a session, and redirect to the Shop page.
11. On the shop page, when the user selects buy if the user has enough money, subtract the price of the item from the user's funds. Update the user's funds column in the database. If the user is too poor, redirect to an error page stating the cost of the item and the user current funds.

**Hint:** After adding, altering, or deleting a table call DB.SaveChanges to save the updates to the database.



## LAB NUMBER: 24

**Task:** Adding onto the previous lab, once the User buys an item store the user's purchase and allow the User to see all the items they've bought.

### What does the application do?

- Lists the items associated with the logged in user.
- Allows the user to Add and Delete items on their own personal list.

### Build Specifications

- Return to SSMS and create a new table called UserItems. This table will have a Primary Key UserItemID and it will have two foreign keys: UserID and ItemID. Make sure the data types match the Primary Keys from the User and Item tables. Connect the User and Item tables to the UserItems tables.
- In your program, refresh your .edmx file to bring your new table into the program.
- Create a new controller called UserItemsController to manage the items a user has purchased.
- Go to the shop page and modify it to store a bought item in a table called PurchasedItems which track the items' ID along with the user's ID.

**Hint:** Remember in your controller, if you have more than one table, the same DB context can talk to either of them EX both `db.items.ToList()` and `db.users.ToList()` are usable anywhere the context is present.

### Views to Create

The following views should become available if a user is logged in:

- List - display all the items the logged in user has purchased, next to each item link to a Details page and a Delete page. Show only items attached to the user's ID in the UserItems table.
- Details - this page will display the full details for a specific item. In the controller, the action for this page will take an int id and try to find the item in the table with that id.
- Delete - Displays a message asking if the user is sure they wish to delete a row and showing all the data in that row. If the user selects yes, the item is deleted and it's cost is refunded to the user (*this is permanent so be careful*) and the user is redirected to the list page. If no, redirect back to the list page.



## LAB NUMBER: 25

**Task:** Validate Item information.

We need to add validation to the item add and edit forms that we did in the last lab. Use the entity framework's data annotations and HTML5 forms.

Display appropriate error messages so the user can re-enter a valid input.

Include the following validation rules:

- **Name:** Non-empty, only characters, up to 20 characters.
- **Price:** Non-empty, Float numbers, value from 1 to 150.
- **Quantity:** Non-empty, Integer numbers, value from 1 to 1000.
- **Description:** Non-empty, up to 2000 characters.



## LAB NUMBER: 26

**Task:** In this lab, you will create a login page using the ASP.NET Identity.

**Specifications:** You will have a main page that has the login text boxes (username and password). The user cannot go to any page unless he or she is authenticated. You can store the login information in the database.



## LAB NUMBER: 27

**Task:** Work with APIs that generate JSON data to display information on a web page.

### Specifications:

1. Work with the following API:  
<http://forecast.weather.gov/MapClick.php?lat=38.4247341&lon=-86.9624086&FcstType=json>
2. This API does not need a Key. It will give you weather forecast for a number of days.
3. If you want to see a tree representation of the JSON data, use this website:  
<http://jsonviewer.stack.hu/>
4. Only view the text part (Forecast) of all the days.
5. Try calling the API using different latitudes and longitudes. You can use text boxes to submit the data to the API.

### Extended Challenges:

- Try reading it in xml  
<http://forecast.weather.gov/MapClick.php?lat=38.4247341&lon=-86.9624086&FcstType=xml>
- You can try calling a different API from:  
<https://market.mashape.com/explore?page=1>



## LAB NUMBER: 28

**Task:** Work with an API that generates JSON data to display information on a web page.

### Build Specifications:

1. Work with the Deck of Cards API, documented at <https://deckofcardsapi.com/>
2. Make a call to the API to generate a new deck. Capture the deck ID returned.
3. Draw 5 cards from the deck and display their names and images inside a web page.

Notes: If you want to see a tree representation of the JSON data, use this website: <http://jsonviewer.stack.hu/>

### Extended Challenges:

1. On the page that displays the cards, have a button to draw the next 5 cards from the same deck. You'll need to pass the ID along, either coded into the URL or as a hidden field.
2. Add "Keep" functionality to the cards view. When the user clicks the draw again button, keep the cards they chose to keep and only draw to replace the others. (Checkboxes might be the easiest way to accomplish the keep.)





## LAB NUMBER: 29

**Task:** Create a Web API that will provide movie data.

### What will the application do?

Design an API that will provide the following services

1. Get a list of all movies
2. Get a list of all movies in a specific category
  - User specifies category as a query parameter
3. Get a random movie pick
4. Get a random movie pick from a specific category
  - User specifies category as a query parameter
5. Get a list of random movie picks
  - User specifies quantity as a query parameter
6. Get a list of all movie categories
7. Get info about a specific movie
  - User specifies title as a query parameter
8. Get a list of movies which have a keyword in their title
  - User specifies title as a query parameter

### Build Specifications:

- Use the Entity framework and your existing movie database.
- Set your API to include actions, and use the JSON Formatter for text/html requests but leave XML support in for text/xml etc.
- Note that if you use any special characters in queries to test your API, you need to encode them. See [https://www.w3schools.com/tags/ref\\_urlencode.asp](https://www.w3schools.com/tags/ref_urlencode.asp) for a list of encodings. For instance, if your database had Science Fiction as a category (with a space), your test query might look like /listByCategory?category=Science%20Fiction
- Complete the first 4 tasks above, then complete as many as you can tasks 5-7, starting with whatever you want.
- You can decide the key names for your query parameters.
- API queries which return no results should return properly formatted empty JSON (for instance, a search for a category that doesn't exist).



## LAB NUMBER: 30

**Task:** Create a user registration page and using Sessions persist the user throughout the app

### What will the application do?

Make a .Net web app that will have a user registration form. Once the form is completed, you will take their information and store it all in a session.

### Build Specifications:

You will need to build out these pages:

- User registration - Create and add a user to the database.
- User Login - if no user is logged in, input an email and password. If those credentials exist in the db, retrieve the match user and store that object in a session. If not, redisplay the login page and display a message saying incorrect login.
- User Details - Display all the properties for the user, and give an option to save the password.
- Logout - clear the current user from the session. What are different ways to do this?
- \_Layout - .Net builds this page for you, you only need to modify it. If the user is logged in, use Razor to display a link to the user details in the top right corner of the app. If no user is logged in, instead display a link to the login page.

Create a user model that stores the following information:

- User Name
- Email
- Password
- Age

If the user is logged in, using razor, a link to the user details page will appear in the top right corner of the app.

For the logout page, in the view display the message userName + " has logged out" on the page. In the action, clear the logged in user from the session. In the \_layout page, hide the logout link.

### Extended Exercise:

Using the entity framework and a code/model first connection, store a list of multiple users. Create a login page that checks if the user is in the database, if yes it retrieves and stores that user in a session object. If no, redirect the user to an error page stating "User not found in database".



## LAB NUMBER: 31

**Task:** Create a Data Access Layer for calling different end points on the API. This lab will have 3 parts:

1. In the DAL, create a method to call the API
2. In the DAL, create methods to parse data from different endpoints
3. Create views to display this information

### Build Specifications:

#### Part 1 - Calling the API:

1. Create a class called StarWarsDAL in your models folder.
2. Create a method with the following signature:  
`public static string GetData(String url)`
3. In the GetData() method, write code to get a response from the API, then pull the data into a string and return that string

#### Part 2 - Parsing JSON:

1. Create a Person model with the following properties:  
Name  
Species  
Gender  
HomePlanet
2. Create a constructor in Person:  
`Person(string APIText)`  
Inside the constructor parse the APIText into a JObject.  
For each property, initialize it with the correct data from the JObject
3. Create a Planet model with the following properties:  
Name
4. Create a constructor in Planet:  
`Planet(string APIText)`  
Inside the constructor parse the APIText into a JObject.  
For each property, initialize it with the correct data from the JObject
5. In the DAL create a method with the following signature:  
`public static Person GetPerson(int i)`
6. In GetPerson(), call GetData at the end point for people plus the id of the person you want to find. For example, the url for Luke Skywalker is <https://swapi.co/api/people/1/>
7. After you call GetData(), take the resulting string pass to the Person model constructor.
8. Next in the DAL create a method with the following signature:  
`Public static Planet GetPlanet(int i)`



## LAB NUMBER: 31

9. In GetPlanet(), call the end point for Planet you wish to find, For example Dagobah is <https://swapi.co/api/planets/5/>
10. Parse the data in the Planet constructor and return the new planet object.

### Part 3 - Views

1. On the index page create 2 forms: one to look up People by Id, the other to look up Planets by Id
2. Create an action for Person(int i) in the HomeController. Set the people form to call this action on submit. In the controller, use i to call the GetPerson() in the DAL.
3. Feed your newly created person into your Person view.
4. The Person view takes in a Person object and display all of its parameters.
5. Do the same, but for planets.



## LAB NUMBER: 32

**Task:** Use Linq to search the following collections.

### Build Specifications:

Assume we have the following collections:

```
int[] nums = { 10, 2330, 112233, 10, 949, 3764, 2942 };
```

```
List<Student> students = new List<Student>();  
    students.Add(new Student("Jimmy", 13));  
    students.Add(new Student("Hannah Banana", 21));  
    students.Add(new Student("Justin", 30));  
    students.Add(new Student("Sarah", 53));  
    students.Add(new Student("Hannibal", 15));  
    students.Add(new Student("Phillip", 16));  
    students.Add(new Student("Maria", 63));  
    students.Add(new Student("Abe", 33));  
    students.Add(new Student("Curtis", 10));
```

Answer the following questions:

For nums:

- 1) Find the Minimum value
- 2) Find the Maximum value
- 3) Find the Maximum value less than 10000
- 4) Find all the values between 10 and 100
- 5) Find all the Values between 100000 and 999999 inclusive
- 6) Count all the even numbers

For students:

- 1) Find all students age of 21 and over (aka US drinking age)
- 2) Find the oldest student
- 3) Find the youngest student
- 4) Find the oldest student under the age of 25
- 5) Find all students over 21 and with even ages
- 6) Find all teenage students (13 to 19 inclusive)
- 7) Find all students whose name starts with a vowel



# CAPSTONE PROJECTS

## TAKE A DEEPER DIVE INTO THE CODING POOL



## WEEK 1 CAPSTONE: PIG LATIN

**Intro:** Pig Latin is a children's word game in English where starting consonants are flipped to the ends of words and -ay added to each word. Hello World would be ellohay orldway in Pig Latin, for instance. Many languages have games similar to this--read more at <http://mentalfloss.com/article/50242/pig-latins-11-other-languages>

**Task:** Translate from English to Pig Latin.

### What Will the Application Do?

- The application prompts the user for a word.
- The application translates the text to Pig Latin and displays it on the console.
- The application asks the user if he or she wants to translate another word.

### Build Specifications

1. Convert each word to a lowercase before translating.
2. If a word starts with a vowel, just add "way" onto the ending.
3. If a word starts with a consonant, move all of the consonants that appear before the first vowel to the end of the word, then add "ay" to the end of the word.

### Hints

- Treat "y" as a consonant.

### Extended Challenges

- Keep the case of the word, whether its uppercase (WORD), title case (Word), or lowercase (word).
- Allow punctuation in the input string.
- Translate words with contractions. Ex: can't become an'tcay
- Don't translate words that have numbers or symbols. Ex: 189 should be left as 189 and hello@grandcircus.co should be left as hello@grandcircus.co.
- Check that the user has actually entered text before translating.
- Make the application take a line instead of a single word, and then find the Pig Latin for each word in the line.

### Console Preview

```
Welcome to the Pig Latin Translator!

Enter a line to be translated: {this sentence exists
here}

Isthay entencesay existsway erehay

Translate another line? (y/n): N
```



## WEEK 2 CAPSTONE: TASK LIST

**Intro:** Jill, a project manager on your team comes to you after you've been working for a few weeks. "So far you've been doing great work," she says, "so I'd like to ask you to take on an extra project. We've been having trouble meeting our goals on time and no app we've tried really fits our model. We'd like to build our own task management app. Can you start building us a task manager? We don't know if we want it to be a mobile app or a web app eventually, so just start with a console app. The senior devs tell me that it'll be easy to take the code and eventually build it into one or the other."

**Task:** Manage tasks through a menu system

### What Will the Application Do?

- Present a menu to the user and ask them to choose:
  1. List tasks
  2. Add task
  3. Delete task
  4. Mark task complete
  5. Quit
- If the user chooses list tasks:
  - Display all tasks.
  - Format the task so output is tabbed--it may be easiest to show the description last.
  - Show a task number, but start the task numbering with 1 not 0.
- If the user chooses to add task:
  - Prompt the user to input each piece of data (team member's name, task description, due date. (Tasks will always start incomplete--that is, completion status is false.)
  - Instantiate a new Task with this info, then add it at the end of your List.
- If the user chooses to delete task:
  - Ask the user which task number. Remember, they'll be using 1 through the size of the list, not 0 through size - 1, so shift their input accordingly.
  - Validate the number entered--make sure it's in range. Prompt them until they enter a number in range.
  - Display the task the user chose. Ask if they're sure they want to delete.
  - If they answer Y, remove that item from your list and return to the main menu. If they answer N, return to the main menu.
- If the user chooses mark task complete:
  - Ask the user which task number. Remember, they'll be using 1 through the size of the list, not 0 through size - 1, so shift their input accordingly.
  - Validate the number entered--make sure it's in range. Prompt them until they enter a number in range.
  - Display the task the user chose. Ask if they're sure they want to mark the task as complete.
  - If they answer Y, change the completion status within that item to true and return to the main menu. If they answer N, return to the main menu.
- If the user chooses quit, ask if they're sure. If they answer Y, exit the program; if they answer N, return to the main menu.
- Display the main menu options every time they return to the main menu.

### Build Specifications

1. Build a Task class.
  - As data members, the task should include
    - Team member's name





- Brief description
  - Due date
  - Whether it's been completed or not
- Include at least one constructor and properties for all data members
- 2. Store your Tasks in a List so you can easily add and delete.

### Hints

- Build the task class
- Since tasks always start incomplete, you could write a constructor that doesn't require that as an argument

### Extended Challenges

- Allow the user to display tasks for only one team member
- Allow the user to display tasks with a due date before a date they choose
- Allow the user to edit a task that has already been entered

### Console Preview

```
Welcome to the Task Manager!
  1. List tasks
  2. Add task
  3. Delete task
  4. Mark task complete
  5. Quit
What would you like to do? 2

ADD TASK
Team Member Name: Grant
Task Description: Update student workbooks
Due Date: 01/15/2018

Task entered!
  1. List tasks
  2. Add task
  3. Delete task
  4. Mark task complete
  5. Quit
What would you like to do? 1

LIST TASKS
Done?      Due Date      Team Member  Description
False      01/15/2018    Grant        Update student workbooks

  1. List tasks
  2. Add task
  3. Delete task
  4. Mark task complete
  5. Quit
What would you like to do? 5

Have a great day!
```



## WEEK 4 CAPSTONE: SHOPPING LIST

**Intro:** Terrance, your lead dev, comes to you this morning and says, “We’re really close to a deal with a new client but we need a mock-up to present tomorrow. Eventually the shopping cart will be coded in the backend, but can you please put together a Javascript mock-up today? The client has asked for a visual of how it would work, and I think with this we can close the deal. I know you’re a backend dev but you’re the only one with space on their schedule today and I know you can get it done.”

**Task:** Create a web page with Javascript to track and display an itemized shopping list and total.

### What Will the Application Do?

- Display a list of products and prices with Add buttons.
- When the user clicks a Check Out button, display all the items purchased and their prices along with a grand total.

### Build Specifications

1. You need to have a list of items on the page to select from. When you click on the “add item” button, this item should be added to your shopping list.
2. Create an array for the items and an array for the corresponding prices.
3. Loop through the arrays, printing out the name and price of each item on a new line.
4. Total up the combined cost of all of your items with the amount’s label being “total”.

### Hints

- In Javascript, arrays are enclosed in square brackets. [ ]
- Remember to keep track of the total of all the prices and print this separately.
- To keep it simple at first, you can display your output in an alert box. When that’s working, write a results page or display it in a separate part of this page.

### Extended Challenges

- Use CSS to dress up the page
- Include quantities of items--either have a field next to the item for quantity to add, or have every button click add another one of that item, or have + and - buttons (but don’t let the user have a negative number of items!)
- Calculate and display sales tax (Michigan 6%) and grand total. Round all numbers to two digits.



## WEEK 6 CAPSTONE: TASK LIST REVISITED

**Intro:** Several months into your first developer position, Jill the project manager comes back to you. “Remember that task list we had you start as a console app a while back? We’ve finally decided--we want it as a web app. Think you can dust it off and make it for us? This time we only want users to see their own tasks, but it should basically work the same as the console app.”

**Task:** Create a web-based task manager

### What Will the Application Do?

- A user can create a new account with email address and password
- Once logged in, the user can create a new task. Tasks consist of:
  - The ID of the user who owns it (user doesn’t get to edit this field)
  - A task description
  - The due date
  - Whether it’s complete or not
- The user should see a list of all tasks they own
  - There should be a button or checkbox to mark the task complete
  - There should be a button to delete the task

### Build Specifications

1. Build this as an MVC web application
2. Store users and tasks in separate SQL database tables
3. Minimum views: Welcome/login page, task list, add task view

### Hints

- When making the SQL call for which tasks to display, show only the tasks which have this user’s ID

### Extended Challenges

- Let the user search for a task by a word or words in the description
- Let the user sort or filter by due date
- Let the user sort or filter by completion status



## WEEK 7 CAPSTONE: PRODUCT API

**Intro:** Terrance, the lead dev, comes back to you. “Great job on the shopping list the other day! The client loved it and signed the contract. You really came through for us. Now I’ve got a production task for you: One of our clients, Northwind--I think you’ve done some work with their database before?--needs their product catalog accessible by API for a web app.”

**Task:** Build an API to return details of products from the Northwind Database

### What Will the Application Do?

- If the API is accessed with a product id, it will return all the details from the Northwind Database for that specific product.
- If the API is accessed without a product id, it will return an array of product IDs and product names.

### Build Specifications

1. Return data from the API as JSON
2. When *{your machine’s web server url including port}/api/product* is accessed, return an array of product IDs and names from the database
3. When *{your machine’s web server url including port}/api/product?id={productID}* is accessed, return an object with all the product info (all the columns in that database table for that specific product). Return a null object if the productID is invalid.

### Hints

- You might want to make a new class to hold just product ID and product name.

### Extended Challenges

- Look at the Northwind Database Product table and add more functionality. (*Hint: each of these could use another parameter.*)
  - Allow search by category ID
  - Allow search by supplier ID
  - Allow search with a maximum price
  - Allow users to include or exclude discontinued items



# ENTITY FRAMEWORK CAPSTONE: GRAND CIRCUS UNIVERSITY

**Intro:** Grand Circus wants to track its students, courses, and enrollment. To do that, you need to help build a website to help Grand Circus staff manage all of this.

**Task:** Build an Entity Framework CRUD application to track students, courses, and enrollments in Grand Circus University.

## What Will the Application Do?

- The administrator can see a list of all the courses offered by Grand Circus.
- The administrator can see a list of all the students in any course.
- The administrator can perform CRUD operations on both courses and students in any of the courses.
- A student can see a list of all the courses he/she took.
- A student can get information about any class he/she took, such as the final grade, semester, and the min, max, and average of the final grade for all students who enrolled in that class with the student.
- A student can search for classes to enroll in.

## Build Specifications

- Start by building the database. You mainly need a table to store student information (first name, last name, phone number, address), and a table to store course information (course ID, name, category (full time, or part time)). Those tables have a many-to-many relationship. This relationship needs to capture the semester on when the student registered for the class, and the final grade for that student.
- Build the application using ASP.NET MVC with Entity Framework. You can use Identity framework to add roles (Administrator, Student).

## Extended Challenges

- Once a course is deleted, you can archive the course's information by copying the information to a table called: Archived classes.
- Store the picture for each student and show it on the student's main page. Also, the administrator should be able to see the pictures for all students enrolled in any class.



## FINAL CAPSTONE PROJECT: GRAND CIRCUS CAR DEALERSHIP

**Intro:** Grand Circus wants to start a new venture by making an API to show its wide selection of cars. This API can then be used by any web application that wants to show this information to everyday customers.

### Task:

1. Build an API that will provide information about cars. This information will be stored in a database.
2. Build an Entity Framework CRUD application access the car API. The application will access the data and search it.

### What Will the Application Do?

- Car API:
  - The API can show information about cars by using make, model, year, color.
  - The API should be able to send the information back to the client using JSON.
- Web application: The web application that you need to build will be a client for the Car API. The website should provide search capabilities to the user. The user can search for cars by make, model, year, color, or a combination on any.

### Build Specifications

- Start by building the database. You mainly need a table to store information about cars (make, model, year, color). The Car API will utilize this database.
- Build the application using ASP.NET MVC with Entity Framework. The application will access the Car API, then use the API to provide search capabilities to the user.

### Extended Challenges

- Find another API that you can use to get dealerships for the cars you find.



# EXTRA EXERCISES

## GET YOUR FUNDAMENTALS PRACTICE ON



## EXERCISE 1

### Description

Prompt the user to enter a string. After the user enters a string, output the same string back to the console.

### Example

```
>>Enter some text: <<Hello, World! ECHOOOOOO!  
>>Hello, World! ECHOOOOOO!
```

---

## EXERCISE 2

### Description

Prompt the user to enter a number. After the user enters a number, add 1 to the number and output it back to the console.

### Example

```
>>Enter a number: <<52  
>>53
```

---

## EXERCISE 3

### Description

Prompt the user to enter a number. After the user enters a number, add .5 to the number and output it back to the console.

### Example

```
>>Enter a number: <<17.3  
>>17.8
```

---

## EXERCISE 4

### Description

Prompt the user to enter two numbers. After the user enters the numbers, add them together and output the sum back to the console.

### Example

```
>>Enter a number: <<12.2  
>>Enter another number: <<17.3  
>>The sum is 29.5
```

---





## EXERCISE 5

### Description

Prompt the user to enter two numbers. After the user enters the numbers, multiply them and output the product back to the console.

### Example

```
>>Enter a number: <<10.2
>>Enter another number: <<13.4
>>The product is 136.68
```

---

## EXERCISE 6

### Description

Use a do-while loop to output "Hello, World!" in a loop. Each time you output "Hello, World!" ask the user whether they would like to continue.

### Example

```
>>Hello, World!
Would you like to continue (y/n)? <<y
>>Hello, World!
Would you like to continue (y/n)? <<y
>>Hello, World!
Would you like to continue (y/n)? <<y
>>Hello, World!
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 7

### Description

Use a do-while loop to run exercise 1 in a loop. Each time you run the code ask the user whether they would like to continue.

### Example

```
>>Enter some text: <<Hello, World! ECHOOOOOO!  
>>Hello, World! ECHOOOOOO!  
Would you like to continue (y/n)? <<y  
>>Enter some text: <<Hello, World! ECHOOOOOO again!  
>>Hello, World! ECHOOOOOO again!  
Would you like to continue (y/n)? <<y  
>>Enter some text: <<Hello, World! ECHOOOOOO againnnn!  
>>Hello, World! ECHOOOOOO againnnn!  
Would you like to continue (y/n)? <<n  
>>Goodbye!
```

---

## EXERCISE 8

Make exercises 2-5 run in a loop. Use a do-while loop to run the code in a loop. Each time you run the code ask the user whether they would like to continue.

---

## EXERCISE 9

### Description

Prompt the user to enter a language. Based on the language the user input, display "Hello, World!" in that language. Use a switch statement to choose what to display.

### Example

```
>>Enter a language: <<English  
>>Hello, World!  
Would you like to continue (y/n)? <<y  
>>Enter a language: <<Spanish  
>>Hola Mundo!  
Would you like to continue (y/n)? <<y  
>>Enter a language: <<Dutch  
>>Hallo wereld!  
Would you like to continue (y/n)? <<n  
>>Goodbye!
```

---



## EXERCISE 10

### Description

Determine whether the user is tall enough to ride a roller coaster. Prompt the user to enter her height in inches. If she is less than 54 inches tall, notify her that she cannot ride the Raptor. If she is at least 54 inches tall, notify her that she can ride the Raptor.

### Example

```
>>Enter your height in inches: 52.3
>>Sorry, you cannot ride the Raptor. You need 1.7 more inches.
Would you like to continue (y/n)? <<y
>>Enter your height in inches: 55.9
>>Great, you can ride the Raptor!
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 11

### Description

Use a for loop to output all the numbers from 0 to 9.

### Example

```
>>0 1 2 3 4 5 6 7 8 9
Would you like to continue (y/n)? <<y
>>0 1 2 3 4 5 6 7 8 9
Would you like to continue (y/n)? <<y
>>0 1 2 3 4 5 6 7 8 9
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 12

### Description

Output all the numbers from 9 to 0. Use a for loop to output all the numbers from 9 to 0.

### Example

```
>>9 8 7 6 5 4 3 2 1 0
Would you like to continue (y/n)? <<y
>>9 8 7 6 5 4 3 2 1 0
Would you like to continue (y/n)? <<y
>>9 8 7 6 5 4 3 2 1 0
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 13

### Description

Prompt the user for a number. Use a for loop to output all the numbers from that number to 0.

### Example

```
>>Enter a number: <<5
>>5 4 3 2 1 0
Would you like to continue (y/n)? <<y
>>Enter a number: <<12
>>12 11 10 9 8 7 6 5 4 3 2 1 0
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 14

### Description

Prompt the user for a number. Use a for loop to output the squares of all the numbers from 1 to that number.

### Example

```
>>Enter a number: <<2
>>1 4
Would you like to continue (y/n)? <<y
>>Enter a number: <<7
>>1 4 9 16 25 36 49
Would you like to continue (y/n)? <<n
```

---

## EXERCISE 15

### Description

Prompt the user for a number. Use a for loop to output the cubes of all the numbers from 1 to that number.

### Example

```
>>Enter a number: <<2
>>1 8
Would you like to continue (y/n)? <<y
>>Enter a number: <<7
>>1 8 27 64 125 216 343
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 16

### Description

Use a for loop to output a triangle of asterisks with a height of ten.

### Example

```
*
* *
* * *
* * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

---

## EXERCISE 17

### Description

Use a for loop to output a triangle of asterisks with a height of ten.

### Example

```
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * *
* * * * *
 * * * * *
  * * * * *
   * * * * *
    * * * * *
     * * * * *
      * * * * *
```

---



## EXERCISE 18

### Description

Prompt the user to enter a number. Use a for-loop to calculate the sum of all the numbers from 1 to the number entered.

### Example

```
>>Enter a number: <<100
>>5050
Would you like to continue (y/n)? <<y
>>Enter a number: <<20
>>210
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---

## EXERCISE 19

### Description

Prompt the user to enter two numbers. Use a for-loop to calculate the sum of all the numbers from the first number entered to the second.

### Example

```
>>Enter a number: <<12
>>Enter another number: <<21
<<The sum of all the numbers from 12 to 21 is 165.
Would you like to continue (y/n)? <<y
>>Enter a number: <<3
>>Enter another number: <<5
<<The sum of all the numbers from 3 to 5 is 12.
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---



## EXERCISE 20

### Description

Prompt the user to enter a number. Use a for-loop to calculate the product of the number and the two preceding numbers.

### Example

```
>>Enter a number: <<6
>>The product of 6, 5, and 4 is 120.
Would you like to continue (y/n)? <<y
>>Enter a number: <<8
>>The product of 8, 7, and 6 is 120.
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---

## EXERCISE 21

### Description

Prompt the user to enter a series of words. Once the user is done entering the words, output a sentence containing all the words.

### Example

```
>>Enter a word: <<The
>>Would you like to enter another word (y/n)? <<y
>>Enter a word: <<cow
>>Would you like to enter another word (y/n)? <<y
>>Enter a word: <<jumped
>>Would you like to enter another word (y/n)? <<y
>>Enter a word: <<over
>>Would you like to enter another word (y/n)? <<y
>>Enter a word: <<the
>>Would you like to enter another word (y/n)? <<y
>>Enter a word: <<moon.
>>Would you like to enter another word (y/n)? <<n
>>The cow jumped over the moon.
Would you like to continue (y/n)? <<y
>>Enter a word: <<Hello,
>>Would you like to enter another word (y/n)? <<y
>>Enter a word: <<World!
>>Would you like to enter another word (y/n)? <<
>>Hello, World!
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 22

### Description

Prompt the user to enter two numbers that will determine a range. Then prompt the user to enter another number and check if it is in the range.

### Example

```
>>Enter a number: <<7
>>Enter another number: <<25
>>Your range is 7-25.
Enter a number to verify it is in the range: <<20
>>20 is in the range.
Would you like to continue (y/n)? <<y
>>Enter a number to verify it is in the range: <<32
>>32 is outside the range.
Would you like to continue (y/n)? <<y
>>Enter a number to verify it is in the range: <<7
>>7 is in the range.
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---

## EXERCISE 23

### Description

Prompt the user to enter a string. Extract and output the first ten characters of the string.

### Example

```
>>Enter some text: <<abcdefghijklmnp
<<The first ten characters were: abcdefghij
Would you like to continue (y/n)? <<y
>>Enter some text: <<Hello, World!
<<The first ten characters were: Hello, Wor
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---





## EXERCISE 24

### Description

Prompt the user to enter a string. Extract and output the last ten characters of the string.

### Example

```
>>Enter some text: <<abcdefghijklmnop
<<The last ten characters were: ghijklmnop
Would you like to continue (y/n)? <<y
>>Enter some text: <<Hello, World!
<<The last ten characters were: lo, World!
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 25

### Description

Prompt the user to enter a sentence. Split the sentence into individual words and display each word on its own line.

### Example

```
>>Enter a sentence: <<The cow jumped over the moon.
>>The
cow
jumped
over
the
moon.
Would you like to continue (y/n)? <<y
>>Enter a sentence: <<Hello, World!
>>Hello,
World!
Would you like to continue (y/n)? <n
<<Goodbye!
```

---



## EXERCISE 26

### Description

Prompt the user to enter text. Count and output how many vowels were in the string.

### Example

```
>>Enter some text: <<abcdefghijklmnopqrstuvwxyz
>>There were 5 vowels.
Would you like to continue (y/n)? <<y
>>Enter some text: <<Hello, World!
>>There were 3 vowels.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 27

### Description

Prompt the user to enter text. Count and output how many consonants were in the string.

### Example

```
>>Enter some text: <<abcdefghijklmnopqrstuvwxyz
>>There were 19 consonants.
Would you like to continue (y/n)? <<y
>>Enter some text: <<Hello, World!
>>There were 7 consonants.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 28

### Description

Prompt the user to enter text. Remove all the vowels and output the text.

### Example

```
>>Enter some text: <<abcdefghijklmnopqrstuvwxyz
>>bcd fghjklmnpqrstvwxyz
Would you like to continue (y/n)? <<y
>>Enter some text: <<Hello, World!
>>Hll, Wrld!
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 29

### Description

Prompt the user to enter text. Remove all the vowels in the middle of the word, but leave any that start or end the word.

### Example

```
>>Enter some text: <<Elephants are wonderful!
>>Elphnts are wndrfl!
Would you like to continue (y/n)? <<y
>>Enter some text: <<Is every flake edible?
>>Is evry flke edble?
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 30

### Description

Prompt the user to enter text. Reverse the text.

### Example

```
>>Enter some text: <<abcdefghijklmnopqrstuvwxyz
>>zyxwvutsrqponmlkjihgfedcba
Would you like to continue (y/n)? <<y
>>Enter some text: <<Hello, World!
>>!dlroW ,olleH
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 31

### Description

Create an array of size 5 and fill it with the following numbers: 2, 8, 0, 24, 51. Prompt the user to enter an index. Display the element in the array at that index.

### Example

```
>>Enter an index of the array: <<2
>>The value at index 2 is 0.
Would you like to continue (y/n)? <<y
>>Enter an index of the array: <<7
<<That is not a valid index.
Would you like to continue (y/n)? <<y
>>Enter an index of the array: <<0
>>The value at index 0 is 2.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 32

### Description

Create an array of size 5 and fill it with the following numbers: 2, 8, 0, 24, 51. Prompt the user to enter a number. If the number is in the array, display the index at which it is located.

### Example

```
>>Enter a number: <<12
>>That value cannot be found in the array.
Would you like to continue (y/n)? <<y
>>Enter an index of the array: <<24
<<The value 24 can be found at index 3.
Would you like to continue (y/n)? <<y
>>Enter an index of the array: <<abc
>>That value cannot be found in the array.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



### EXERCISE 33

#### Description

Create an array of size 5 and fill it with the following numbers: 2, 8, 0, 24, 51. Let the user change the array by specifying an index and a replacement number.

#### Example

```
>>Enter an index of the array: <<2
>>The value at index 2 is 0. Would you like to change it (y/n)? <<y
>>Enter the new value at index 2: <<17
>>The value at index 2 is 17. Would you like to continue (y/n)? <<y
>>Enter an index of the array: <<2
>>The value at index 2 is 17. Would you like to change it (y/n)? <<n
>>Would you like to continue (y/n)? <<y
>>Enter an index of the array: <<7
<<That is not a valid index. Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

### EXERCISE 34

#### Description

Create an array of size 5 and fill it with the following numbers: 16, 32, 64, 128, 256. Prompt the user to enter a command, 'half' or 'double'. If the user enters 'half', half all the elements in the array. If the user enters 'double', double all the elements in the array.

#### Example

```
>>Enter a command (half/double): half
>>The array now contains: 8, 16, 32, 64, 128.
Would you like to continue (y/n)? <<y
>>Enter a command (half/double): half
>>The array now contains: 4, 8, 16, 32, 64.
Would you like to continue (y/n)? <<y
>>Enter a command (half/double): double
>>The array now contains: 8, 16, 32, 64, 128.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 35

### Description

Create an array of size 5 and fill it with the following strings: "cow", "elephant", "jaguar", "horse", "crow". Prompt the user to enter two indices separated by a space. The first index will specify the word, and the second will specify a letter in that word. Display the corresponding word and letter.

### Example

```
>>Enter two indices: <<2 0
>>The value at index 2 is jaguar. The letter at index 0 is j.
Would you like to continue (y/n)? <<y
>>Enter two indices: <<4 2
>>The value at index 4 is crow. The letter at index 2 is o.
Would you like to continue (y/n)? <<y
>>Enter two indices: <<0 1
>>The value at index 0 is cow. The letter at index 1 is o.
Would you like to continue (y/n)? <<y
>>Enter two indices: <<7 0
>>Those are not valid indices.
Would you like to continue (y/n)? <<y
>>Enter two indices: <<2 17
>>Those are not valid indices.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 36

### Description

Create two arrays, each of size 5. Fill the first array with the numbers: 12, 11, 10, 9, 8. Fill the second array with the strings: "Drummers Drumming", "Pipers Piping", "Lords a-Leaping", "Ladies Dancing", "Maids a-Milking". Combine both arrays to display a piece of the Christmas song.

### Example

```
>>Enter a command (sing/quit): sing
>>12 Drummers Drumming
11 Pipers Piping
10 Lords a-Leaping
9 Ladies Dancing
8 Maids a-Milking
Enter a command (sing/quit): quit
>>Goodbye!
```

---



## EXERCISE 37

### Description

Prompt the user to enter five numbers. Store these numbers in an array and output their sum.

### Example

```
>>Enter a number: <<7
>>Enter a number: <<12
>>Enter a number: <<45
>>Enter a number: <<29
>>Enter a number: <<12
>>7 + 12 + 45 + 29 + 12 = 105
<<Would you like to continue (y/n)? <<y
>>Enter a number: <<11
>>Enter a number: <<78
>>Enter a number: <<5
>>Enter a number: <<0
>>Enter a number: <<30
>>11 + 78 + 5 + 0 + 30 = 124
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 38

### Description

Prompt the user to enter five numbers. Store these numbers in an array and output their average.

### Example

```
>>Enter a number: <<7
>>Enter a number: <<12
>>Enter a number: <<45
>>Enter a number: <<29
>>Enter a number: <<12
>>(7 + 12 + 45 + 29 + 12)/5 = 21
<<Would you like to continue (y/n)? <<y
>>Enter a number: <<11
>>Enter a number: <<78
>>Enter a number: <<5
>>Enter a number: <<0
>>Enter a number: <<30
>>(11 + 78 + 5 + 0 + 30)/5 = 24.8
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 39

### Description

Prompt the user to enter five numbers. Store these numbers in an array and output them in ascending order.

### Example

```
>>Enter a number: <<7
>>Enter a number: <<12
>>Enter a number: <<45
>>Enter a number: <<29
>>Enter a number: <<12
>>7 12 12 29 45
<<Would you like to continue (y/n)? <<y
>>Enter a number: <<11
>>Enter a number: <<78
>>Enter a number: <<5
>>Enter a number: <<0
>>Enter a number: <<30
>>0 5 11 30 78
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 40

### Description

Prompt the user to enter five numbers. Store these numbers in an array and output the median.

### Example

```
>>Enter a number: <<7
>>Enter a number: <<12
>>Enter a number: <<45
>>Enter a number: <<29
>>Enter a number: <<12
>>The median of (7, 12, 12, 29, 45) is 12.
<<Would you like to continue (y/n)? <<y
>>Enter a number: <<11
>>Enter a number: <<78
>>Enter a number: <<5
>>Enter a number: <<0
>>Enter a number: <<30
>>The median of (0, 5, 11, 30, 78) is 11.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---





## EXERCISE 41

### Description

Prompt the user to enter two numbers. Divide the two numbers and show only two decimal places.

### Example

```
>>Enter a number: <<1
>>Enter another number: <<8
>>1/8 is approximately .13.
Would you like to continue (y/n)? <<y
>>Enter a number: <<7
>>Enter another number: <<3
>>7/3 is approximately 2.33
Would you like to continue (y/n)? <<y
>>Enter a number: <<5
>>Enter another number: <<0
>>You cannot divide by 0.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 42

### Description

Create a class called Point, that has two properties, X and Y. Let the user enter an X and Y value to create a Point object.

### Example

```
>>Enter an X coordinate: <<7
>>Enter a Y coordinate: <<12
>> You have created a point object (7,12).
Would you like to continue (y/n)? <<y
>>Enter an X coordinate: <<16
>>Enter a Y coordinate: <<-4
>> You have created a point object (16,-4).
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 43

### Description

Enhance the point class in the previous exercise by adding a CalculateDistance method. The distance method will calculate the distance of a point from the origin, (0,0).

### Example

```
>>Enter an X coordinate: <<3
>>Enter a Y coordinate: <<4
>> You have created a point object (3,4). It has a distance of 5.
Would you like to continue (y/n)? <<y
>>Enter an X coordinate: <<5
>>Enter a Y coordinate: <<12
>> You have created a point object (5,12). It has a distance of 13.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 44

### Description

Enhance the point class in the previous exercise by adding a CalculateMidpoint method. The CalculateMidpoint will take as an argument another point and return the midpoint between the current ('this') object and the point passed in. Its method signature will look like: Point CalculateMidpoint(Point other)

### Example

```
>>Enter coordinates for a point: 4 7
>>Enter coordinates for another point: 8 13
>>The midpoint between (4,7) and (8,13) is (6, 10).
Would you like to continue (y/n)? <<y
>>Enter coordinates for a point: 12 12
>>Enter coordinates for another point: 20 20
>>The midpoint between (12,12) and (20,20) is (16, 16).
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 45

### Description

Create a class Square, that has one property, SideLength. The class should have two methods, CalculatePerimeter, and CalculateArea. Let the user enter the length of the square's side. Create an object based on that length and output details of the square.

### Example

```
>>Enter a side length: <<2
>>The square has side length 2. Its area is 4 and its perimeter is
8.
Would you like to continue (y/n)? <<y
>>Enter a side length: <<12
>>The square has side length 12. Its area is 144 and its perimeter
is 48.
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---

## EXERCISE 46

### Description

Create a class Triangle, that has three properties, Side1Length, Side2Length, Side3Length. The class should have two methods, CalculatePerimeter, and CalculateArea. Let the user enter the length of the triangle's sides. Create an object based on those lengths and output the triangle's details. Hint: Google Heron's Formula.

### Example

```
>>Enter the side lengths of a triangle: <<3 4 5
>>The triangle has side lengths 3, 4, and 5. Its area is 6 and its
perimeter is 12.
Would you like to continue (y/n)? <<y
>>Enter the side lengths of a triangle: <<5 12 13
>>The triangle has side lengths 5, 12, and 13. Its area is 30 and
its perimeter is 30.
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---



## EXERCISE 47

### Description

Prompt the user to enter a string. Store the string in a list and display the contents of the list with each element separated by a space.

### Example

```
>>Enter some text: <<The
<<You have entered: The
Would you like to continue (y/n)? <<y
>>Enter some text: <<cow
<<You have entered: The cow
Would you like to continue (y/n)? <<y
>>Enter some text: <<jumped...
<<You have entered: The cow jumped...
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 48

### Description

Prompt the user to enter as many numbers as she wants. Store these numbers in a list and output their sum.

### Example

```
>>Enter a number (q to quit): <<7
>>Enter a number (q to quit): <<12
>>Enter a number (q to quit): <<45
>>Enter a number (q to quit): <<29
>>Enter a number (q to quit): <<12
>>Enter a number (q to quit): <<q
>>7 + 12 + 45 + 29 + 12 = 105
<<Would you like to continue (y/n)? <<y
>>Enter a number (q to quit): <<11
>>Enter a number (q to quit): <<78
>>Enter a number (q to quit): <<5
>>Enter a number (q to quit): <<0
>>Enter a number (q to quit): <<30
>>Enter a number (q to quit): <<42
>>Enter a number (q to quit): <<17
>>11 + 78 + 5 + 0 + 30 + 42 + 17 = 183
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 49

### Description

Create a list of squares. Use the Square class you built in Lab 9. Then display some statistics for the collection of squares.

### Example

```
>>Enter a side length (q to quit): <<1
>>Enter a side length (q to quit): <<5
>>You created 2 squares.
Largest: 5
Smallest: 1
Average Area: 13
Average Perimeter: 12
Would you like to continue (y/n)? <<y
>>Enter a side length (q to quit): <<7
>>Enter a side length (q to quit): <<4
>>Enter a side length (q to quit): <<9
>>Enter a side length (q to quit): <<16
>>Enter a side length (q to quit): <<2
>>Enter a side length (q to quit): <<7
>>Enter a side length (q to quit): <<q
>>You created 6 squares.
Largest: 16
Smallest: 2
Average Area: 75.83
Average Perimeter: 30
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 50

### Description

Create a list of triangles. Use the Triangle class you built in Lab 9. Then display some statistics for the collection of Triangles.

### Example

```
>>Enter the side lengths of a triangle (q to quit): <<3 4 5
>>Enter the side lengths of a triangle (q to quit): <<5 12 13
>>Enter the side lengths of a triangle (q to quit): <<q
>>Average Area: 18
Average Perimeter: 21
Would you like to continue (y/n)? <<n
<<Goodbye!
```

---



## EXERCISE 51

### Description

Create a dictionary with the following key value pairs: hello => hola, food => comida, world => mundo, computer => computadora, exercise => ejercicio. Prompt the user to enter a word and output the translation.

### Example

```
>>Enter a word in English: <<hello
>>hello in Spanish is hola.
Would you like to continue (y/n)? <<y
>>Enter a word in English: <<world
>>world in Spanish is mundo.
Would you like to continue (y/n)? <<y
>>Enter a word in English: <<playing
>>playing cannot be translated.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 52

### Description

Prompt the user to enter data for a collection of shapes. Display aggregate information about the shapes.

### Example

```
>>Enter a new shape. Square (1), Triangle (2), Circle(3), Quit(q):
<<1
>>Enter a side length: >>4
>>Enter a new shape. Square (1), Triangle (2), Circle(3), Quit(q):
<<2
>>Enter the side lengths of a triangle: <<3 4 5
>>Enter a new shape. Square (1), Triangle (2), Circle(3), Quit(q):
<<3
>>Enter the radius: 4
>>Enter a new shape. Square (1), Triangle (2), Circle(3), Quit(q):
<<q
>>Average Area: 24.09
Average Perimeter: 17.71
Would you like to continue (y/n)? <<n
```

---

## EXERCISE 53

Use the Square, Triangle, and Circle class from Lab 9 to extract an abstract class Shape. Find the methods they all have in common and put them into a base class. All three classes should then inherit from this base class.

---



## EXERCISE 54

Enhance the shape hierarchy you built with different kinds of shapes. Try building in rectangles, pentagons, hexagons, trapezoids, parallelograms, rhombi, ellipses, etc.

---

## EXERCISE 55

Create a class hierarchy using animals. Think about: Which properties and methods do all animals share? Which properties and methods do only some animals share? Which properties and methods are unique to a single animal?

---

## EXERCISE 56

Here's a short guide to help you get started on Lab 11. Start with the Student class. Create all the properties including name and address. Then create the Staff class with all the properties including name and address. Once you have completed both classes examine them and note what is different and what is similar. Take the similar components and move them into a base class. This process should be just like extracting the Shape class.

---

## EXERCISE 57

### Description

Prompt the user for rock, paper, scissors input and display the winner. Player 2 can see Player 1's choice so it's not a very fun game yet. Try hiding the user's input with an asterisk, like when you type a password. Then you'll be able to play with a friend.

### Example

```
>>>Player 1, enter rock (r), paper (p), or scissors(s): <<r
>>>Player 2, enter rock (r), paper (p), or scissors(s): <<p
>>>Player 2 wins.
>>>Would you like to continue (y/n)? <<y
>>>Player 1, enter rock (r), paper (p), or scissors(s): <<s
>>>Player 2, enter rock (r), paper (p), or scissors(s): <<r
>>>Player 2 wins.
>>>Would you like to continue (y/n)? <<n
>>>Goodbye!
```

---



## EXERCISE 58

### Description

Create the following Enumerations to practice the syntax:

Suit => Diamonds, Hearts, Clubs, Spades  
Browser => Chrome, Firefox, IE, Opera  
Brand => Polo, Gucci, Armani, Athleta  
Day => Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday

Think about the contexts these enums would be used in.

---

## EXERCISE 59

Define a card class with two properties, Rank and Suit. For these properties, define two enums, Rank (Ace, King Queen, etc.) and Suit (Spades, Diamonds, Hearts, Clubs).

---

## EXERCISE 60

### Description

Use the class defined in the previous exercise to make a deck of 52 cards combining the 4 suits with the 13 ranks. Use a random number generator to shuffle the deck and display the topmost card.

### Example

```
>>Shuffling...You drew a King of Spades.  
Would you like to continue (y/n)? <<y  
>>Shuffling...You drew a 3 of Clubs.  
Would you like to continue (y/n)? <<n  
>>Goodbye!
```

---

## EXERCISE 61

Change Lab 11's Shape class from an abstract class into an interface. Why were you able to do this? What are the differences between an interface and an abstract class? Can you convert Lab 12's Player class into an interface? Why or why not?

---





## EXERCISE 62

### Description

Enhance your Card class by having it implement the IComparable interface. Now draw two cards and determine which is higher. Spades > Hearts > Diamonds > Clubs.

### Example

```
>>The computer drew a King of Spades.  
Draw your card. Press Enter...<<  
>>You drew a 3 of Clubs. You lose.  
Would you like to continue (y/n)? <<y  
>>Shuffling...The computer drew a 7 of Hearts.  
Draw your card. Press Enter...<<  
>>You drew a 7 of Clubs. You lose.  
Would you like to continue (y/n)? <<n  
>>Goodbye!
```

---

## EXERCISE 63

### Description

The String class in C# is like any other class you have written. It has just been written for you by someone else. Using your knowledge of the String class start writing the code for the string class. You don't have to fill out the methods. Just get a sense of what the class looks like.

### Example

```
namespace System  
{  
    public class String  
    {  
        public String Substring(int startIndex, int  
        substringLength)  
        {  
            ...  
        }  
    }  
}
```

---



## EXERCISE 64

### Description

Define a Validator class. Add a method IsInt that takes a string and returns true if the string can be parsed as an int.

### Example

```
>>Enter a number: <<17
>>17 can be parsed as an integer.
Would you like to continue (y/n)? <<y
>>Enter a number: <<hello
>>hello cannot be parsed as an integer.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 65

### Description

Enhance the Validator class. Overload the method IsInt to take an input string and a min and max value. Return true if the string can be parsed as an integer and is between the min and max value. For this application, pass IsInt 3 and 12 as the min and max value.

### Example

```
>>Enter a number: <<5
>>5 can be parsed as an integer between 3 and 12.
Would you like to continue (y/n)? <<y
>>Enter a number: <<hello
>>hello cannot be parsed as an integer between 3 and 12.
Would you like to continue (y/n)? <<y
>>Enter a number: <<17
>>17 cannot be parsed as an integer between 3 and 12.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 66

### Description

Enhance the Validator class to determine whether the user entered a yes or no. Create a method ParseYesNo that takes a string and returns an enum YesNoResponse. The enum can be one of three values: YES, NO, NOT\_RECOGNIZED.

### Example

```
>>Enter y/n: <<y
>>The user entered yes.
>>Enter y/n: <<n
>>The user entered no.
>>Enter y/n: <<abc
>>The user's response was not recognized.
>>Enter y/n: <<Y
>>The user entered yes.
>>Enter y/n: <<No
>>The user entered no.
```

---

## EXERCISE 67

### Description

Read a list of numbers from a file, and output their sum.  
Create a file named numbers.txt with the following contents:

```
7
12
13
15
12
11
```

### Example

```
>>Enter a file to read numbers from: <<numbers.txt
>>7 + 12 + 13 + 15 + 12 + 11 = 70
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 68

### Description

Read translations from a file, and store them in a dictionary. Create a file named dictionary.txt with the following contents:

hello,hola  
food,comida  
world,mundo  
computer,computadora  
exercise,ejercicio

### Example

```
>>Enter a file to read translations from: <<dictionary.txt
>>Enter a word in English: <<hello
>>hello in Spanish is hola.
Would you like to continue (y/n)? <<y
>>Enter a word in English: <<world
>>world in Spanish is mundo.
Would you like to continue (y/n)? <<y
>>Enter a word in English: <<playing
>>playing cannot be translated.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 69

### Description

Create a list of numbers and output them to a file.

### Example

```
>>Enter a number (q to quit): <<1
>>Enter a number (q to quit): <<5
>>Enter a number (q to quit): <<7
>>Enter a number (q to quit): <<q
>>Enter the name of the output file: <<numbers1.txt
>>Successfully output numbers to numbers1.txt.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 70

### Description

Create a dictionary of translations and append them to an existing file.

### Example

```
>>Enter a word and its translation (q to quit): woman mujer
>>Enter a word and its translation (q to quit): man hombre
>>Enter a word and its translation (q to quit): q
>>Enter the name of the output file: <<dictionary.txt
>>Successfully added translations to dictionary.txt.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 71

### Description

Read a list of triangles from a file.

Create a file named triangles.txt with the following contents:

```
3 4 5
5 12 13
```

### Example

```
>>Enter a file to read triangles from: <<triangles.txt
>>Average Area: 18
Average Perimeter: 21
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---



## EXERCISE 72

### Description

Determine whether a number is prime or not.

### Example

```
>>Enter a number: <<17
>>17 is prime
Would you like to continue (y/n)? <<y
Enter a number: <<34
>>34 is not prime
Would you like to continue (y/n)? <<y
Enter a number: <<101
>>101 is not prime
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

## EXERCISE 73

### Description

Output the first 100 prime numbers

### Example

```
>>2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, ...
```

---

## EXERCISE 74

Output the first 500 prime numbers. Measure how long it takes. Try optimizing the algorithm and tracking your progress. Every time you make a change that lowers your time, check in the code and put the statistics in your commit comment. Just Google It if you run out of ideas to optimize your algorithm.

---

## EXERCISE 75

### Description

Write an algorithm that calculates the square root of a number. Just Google It if you are stuck. Do not use the built in Math.Sqrt method. The algorithm initially only needs to work for perfect squares.



### Example

```
>>Enter a number: <<25
>>The square root of 25 is 5.
Would you like to continue (y/n)? <<y
Enter a number: <<100
>>The square root of 100 is 10.
Would you like to continue (y/n)? <<n
Goodbye!
```

---

### EXERCISE 76

#### Description

Determine whether a number is even or odd.

### Example

```
>>Enter a number: <<17
>>17 is odd.
Would you like to continue (y/n)? <<y
Enter a number: <<34
>>34 is even.
Would you like to continue (y/n)? <<y
Enter a number: <<73737373
>>73737373 is odd.
Would you like to continue (y/n)? <<n
>>Goodbye!
```

---

### EXERCISE 77

Create a Movie Entity using a Database-First approach. The first step will be to create a Movie Table using SQL Server Management Studio. Add all the necessary columns.

---

### EXERCISE 78

Using the table you created in the previous exercise, create entities. In Visual Studio, add Entities using a Database-First approach.

---

### EXERCISE 79

Now that you have entities created, start building the CRUD (Create, Read, Update, Delete) pages.

---



## EXERCISE 80

With us now using OOP to create classes and objects, we want you to work with some design patterns. Create either a Singleton class(1 instance of object), or Factory class(can create multiple instances of objects). You can choose to build just one or both types. Both design patterns have their uses, and by building and implementing these patterns will help you to discover their uses.

[C# Factory pattern:](https://www.c-sharpcorner.com/article/factory-method-design-pattern-in-c-sharp/)

<https://www.c-sharpcorner.com/article/factory-method-design-pattern-in-c-sharp/>

[C# Singleton pattern:](https://www.c-sharpcorner.com/UploadFile/8911c4/singleton-design-pattern-in-C-Sharp/)

<https://www.c-sharpcorner.com/UploadFile/8911c4/singleton-design-pattern-in-C-Sharp/>

---

## EXERCISE 81

Write tests for the RemoveAt and the InsertAt methods in your Data Structure. We will use the built in NUnit testing that we can do in Visual Studio. Start slow with your test, and be as thorough as you can. Be sure to test for all edge cases.

---

## EXERCISE 82

You will create a view with a form that takes at least two pieces of data, text, and digits. You will then validate the user's input with JavaScript. Not only will you have to validate the user's input, but you will also have to display to the user if they have given valid input. You can use JavaScript to manipulate HTML and CSS, so have fun with it and see what you can come up with.

---

## EXERCISE 83

Let's build a calculator using our JavaScript skills. We can add as many features to this calculator, but let's start with the basics. Addition, Subtraction, Multiplication, and Division. Let us add a clear button as well, to make sure that we don't have to reload the website to start over with a new calculation.

---

## EXERCISE 84

This program will use cookies to save a date for a certain amount of time. You will need to use Cookies, and not Sessions, to ensure that the data will remain. Use the DateTime class, and HttpCookie class, to get this exercise complete.

---





## EXERCISE 85

Take what you have learned previously with JavaScript and Cookies and now use it to personalize the your site for the user. Create a view that will ask the user to make a font choice and a background color choice. You can choose to show examples of each or not. Then take the user's choices and implement them into the view. You will also save their choice into cookies so that their options will remain even after they close the browser.

---



## Exercise 86

Let's take our program from the other day that utilized the Weather API. We will now pull out more data from the API, and display it. Experiment with different ways to display the data. You can use tables, but you can also mess with div's, and don't forget about Bootstrap classes!

---

## Exercise 87

Take the Entity program that you were working on, and modify the delete customer action. We will be using the DbContextTransaction class to make sure that there is no issues with the system during a delete command. At least three methods will need to be used, BeginTransaction(), RollBack(), and Commit().

---

## Exercise 88

Lets make our web page more responsive by adding Ajax. Ajax allows us to have a more dynamic web page, and keeps the page from having to reload everytime we want to execute a simple action/command. Use a project that has been connected to a DataBase with Entity Framework, and add Ajax to it. Use Ajax to do the same queries that you were doing before, and watch how the data is reached quicker, and you don't have to reload the page each time that you make a query.

---



# BONUS EXERCISES



## BONUS-1

**Task:** Convert given number grades into letter grades.

### What will the application do?

- The user will enter a numerical grade from 0 to 100.
- The application will display the corresponding letter grade.
- The application will prompt the user to continue.

### Build Specifications

1. Assume that the user will enter valid integers for the grades.
2. The application should only continue if the user enters "y" or "Y".

### Hints:

- Grade Ranges:
  - A : 100 - 88
  - B : 87 - 80
  - C : 79 - 67
  - D : 66 - 60
  - F : 60 - 0

### Extended Challenge:

Edit your grade ranges to include pluses and minuses (ex: 99-100 = A+).

### Console Preview:

```
Welcome to the Letter Grade Converter!
```

```
Enter a numerical grade: 90
```

```
Letter Grade: A
```

```
Continue? (y/n): Y
```

```
Enter a numerical grade: 82
```

```
Letter Grade: B
```

```
Continue? (y/n): y
```

```
Enter a numerical grade: 51
```

```
Letter Grade: F
```

```
....
```



## BONUS-2

**Task:** Calculate the age of a person!

**What will the application do?**

- Ask the user to enter their birthday. You can ask the user to enter the year, month, then day, then store it in a DateTime variable.
- Calculate the Age of the user (Show the output in years and days).

**Extra Challenges:**

- If the current day is the user's birthday, print a nice message!



## BONUS-3

**Task:** Guess a number from 1 to 100. You should complete this by pair programming. Yes, *really*. Buddy up, and close one of your computers.

### What will the application do?

- The application prompts the user for an *int value* from 1 to 100 until the user guesses the random number that the application has generated.
- The application displays messages that indicate whether the user's guess is too high or too low.
- When the user finally guesses the number, the application displays the number of guesses it took the user along with a rating.
- The application prompts the user to play again.
- When the user chooses to end the game, the application displays a goodbye message.

### Build Specifications

1. If the user's guess is more than 10 higher or more than 10 lower than the random number, the user should be notified as such. (ex: "Way too high!")
2. If the user's guess is less than 10 higher or less than 10 lower than the random number, then the user should be notified as such. (ex: "Too low!")
3. The message that's displayed when the user successfully gets the number should change based on the number of guesses it took him or her.
4. The application should only accept numbers from 1 to 100.
5. When the user is prompted to Play Again, the application should only accept a value of "y" or "n".
6. If the user enters invalid data, the application should display an error message and prompt the user for data again.
7. The code used to validate the data should be stored in separate methods.

### Hints:

- Use the Random class to generate a random number.
- Hint for number 7:
  - `public static double GetDoubleWithinRange(string Prompt, double Min, double Max)`
  - `public static int GetIntWithinRange(string Prompt, int Min, int Max)`



## BONUS-3

### Console Preview:

```
Welcome to the Guess a Number Game!
+++++

I'm thinking of a number between 1 and 100.
Try to guess it, n00b.

Enter your lousy guess: 8
You got it in 1 tries.
You must be seriously amazing slash telepathic!

Try again? (y/n): y

I'm thinking of a number between 1 and 100.
Try to guess it, n00b.

Enter your lousy guess: 8
You're crazy low, bro. Try again.

Enter a number: 49
Too low, Joe. Try again.

Enter a number: 56
Too high! Try again.

Enter a number: 51
You got it in 4 tries.
Pretty good, I guess. I bet you can do better, though.

Try again? (y/n): ....
```



## BONUS-4

**Task:** Create an object-oriented validation class.

### What will the application do?

- The application prompts the user to enter a valid within a specified range, a valid double within a specified range, a required String, and one of two Strings.
- If an entry isn't valid, the application displays an appropriate error message.

### Build Specifications

1. Create a class named `OOValidator` that contains these constructors and methods:

```
public int GetInt(string prompt)
public int GetIntWithinRange(string prompt, int min, int max)
public double GetDouble(string prompt)
public double GetDoubleWithinRange(string prompt,
    double min, double max)
```

2. Create a class named `MyValidator` that extends the `OOValidator` class. This class should add two new methods:

```
public String GetRequiredString(string prompt)
public String GetChoiceString(string prompt,
    string s1, string s2)
```

3. Create a class named `ValidatorTestApp` and use it to test the methods in the `Validator`, `OOValidator`, and `MyValidator` classes.





## BONUS-4

### Console Preview:

Welcome to the Validation Tester application

Int Test

Enter an integer between -100 and 100: x

Error! Invalid integer value. Try again.

Enter an integer between -100 and 100: -101

Error! Number must be greater than -101

Enter an integer between -100 and 100: 101

Error! Number must be less than 101

Enter an integer between -100 and 100: 100

Double Test

Enter any number between -100 and 100: x

Error! Invalid decimal value. Try again.

Enter any number between -100 and 100: -101

Error! Number must be greater than -101.0

Enter any number between -100 and 100: 101

Error! Number must be less than 101.0

Enter any number between -100 and 100: 100

...



## BONUS-5

**Task:** Display a sorted list of student scores.

**What will the application do?**

- The application accepts last name, first name, and score for one or more students and stores the results in an array.
- The application prints the student name and their score alphabetically by last name.

**Build Specifications**

1. The program should implement a class called Student that stores last name, first name, and score for each student. The class should implement the Comparable interface so that students can be sorted by name.
2. The program should you an Array to store Student objects. It should sort the array prior to printing the student list.
3. Validate user input so that the number of students can only be a positive integer, the last name and first name values cannot be empty Strings, and the score is an integer from 0 and 100.

**Hints:**

- Remember to plan for two students who have the same last name. How will the program alphabetize these students?

**Extended Challenges:**

- Add the ability to store the list by score.

**Console Preview:**

```
Welcome to the Student Scores Application!
```

```
Enter number of students: 2
```

```
Student 1 last name: Mayonnaise
```

```
Student 1 first name: Patty
```

```
Student 1 score: 94
```

```
Student 2 last name: Funny
```

```
Student 2 first name: Doug
```

```
Student 2 score: 83
```

```
Funny, Doug: 83
```

```
Mayonnaise, Patty: 94
```



## BONUS-6

**Task:** Create, package, and document the GameConsole class.

### What will the application do?

- The application prompts the user to enter a valid within a specified range, a valid double within a specified range, a required String, and one of two Strings.
- If an entry isn't valid, the application displays an appropriate error message.

### Build Specifications

1. Create a class named GameConsole that can be used to display output to the user and get input from the user. Feel free to reuse your best code from any previous exercises or projects. At a minimum, this class should include these methods:

```
// for output
public void Print(String s);
public void Println(String s);
public void Println();

// for input
public String GetRequiredString(String prompt);
public String GetChoiceString(String prompt, String s1, String s2);
public int GetInt(String prompt);
public int GetIntWithinRange(String prompt, int min, int max);
public double GetDouble(String prompt);
public double GetDoubleWithinRange(String prompt, double min, double max);
```

2. Create a class named GameConsoleTest that tests the GameConsole application to make sure it's working correctly as shown in the console output. Feel free to reuse your best code from any previous exercises or projects.
3. Store the GameConsole class in an assembly named  
YourLastName.util
  - a. Then, add a using statement for this package to the GameConsole class.
4. Add xml comments to the GameConsole class. These comments should document the purpose, author, and version of the class. It should also document the function of each method, including any parameters accepted by the method and any value it returns.
5. Create a release build for the project and store it in the release subdirectory of this project.
6. Generate the documentation for this project and store it in the doc subdirectory of the release directory.



## BONUS-6

### Console Preview:

Welcome to the Console Test Center!

Int Test

Enter an integer between -100 and 100:

Error! This entry is required. Try again.

Enter an integer between -100 and 100: z

Error! Invalid integer value. Try again.

Enter an integer between -100 and 100: -900

Error! Number must be greater than -100. Try again.

Enter an integer between -100 and 100: 500

Error! Number must be less than 100. Try again.

Enter an integer between -100 and 100: 50

String Choice Test:

Select one (x/y):

Error! This entry is required! Try again.

Select one (x/y): K

Error! Entry must be either x or y. Try again.

...



## BONUS-7

**Task:** Simulate a race between the Tortoise and the Hare.

### What will the application do?

- The runners differ in their speed and how often they need to rest. One of the runners, named "Tortoise," is slow but never rests. The other runner, named "Hare," is ten times as fast but rests 90% of the time.
- There is a random element to the runners' performance, so the outcome of the race is different each time the application is run.
- The race is run over a course of 1000 meters. Each time one of the runners moves, the application displays the runner's new position on the course. The first runner to reach 1000 wins the race.
- When one of the runners finishes the race, the application declares that runner to be the winner and the other runner concedes.

### Build Specifications

1. Each runner should be implemented as a separate thread using a class named `ThreadRunner`. The `ThreadRunner` class should include a constructor that accepts three parameters: a string representing the name of the runner, an int value from 1 to 100 indicating the likelihood that on any given move the runner will rest instead of run, and an int value that indicates the runners speed—that is, how many meters the runner travels in each move.
2. The `run` method of the `ThreadRunner` class should consist of a loop that repeats until the runner has reached 1000 meters. Each time through the loop, the thread should decide whether it should run or rest based on a random number and the percentage passed to the constructor. If this random number indicates that the runner should run, the class should add the speed value passed to the constructor. The `run` method should sleep for 100 milliseconds on each repetition of the loop.
3. If the `run` method is interrupted, it should display a message that concedes the race and quits.
4. The `main` method of the application's main class should create two runner threads and start them. One of the threads should be named "Tortoise." It runs only 10 meters each move, but plods along without ever resting. The other thread should be named "Hare." It should run 100 meters each move, but should rest 90% of the time.
5. This class should also include a method named `finished` that one of the threads can call when it finishes the race. That method should declare the thread that calls it to be the winner and should interrupt the other thread so it can concede the race.
6. The `finished` method should provide for the possibility that the two threads will finish the race at almost the same time. If that happens, it should ensure that only one of the threads is declared the winner. (There are no ties!)



## BONUS-7

### Hint:

- To determine whether a thread should run or rest, calculate a random number between 1 and 100. Then, have the thread rest if the number is less than or equal to the percentage of time that the thread rests. Otherwise, the thread should run.
- The finished method in the main application class will need to know which thread called it.

### Extended Exercises:

- Modify the application so that it can support 9 runners.
- Modify the main applications class so that it runs the race 100 times and reports how many times each runner wins.
- Add an additional random element to the runner's performance. For example, have a "clumsiness percentage" that indicates how often the runner trips and hurts himself. When the runner trips, he sprains his or her ankle and can run only at half speed for the next five moves.
- Add the ability for runners to interfere with each other. For example, have an "orneriness percentage" that indicates how likely the runner is to trip another runner who is passing him. This will require additional communication among the threads.

### Console Preview:

```
Get set...go!
Tortoise: 10
Tortoise: 20
Tortoise: 30
Tortoise: 40
Hare: 100
Tortoise: 50
...//output lines omitted//...
Hare: 500
Tortoise: 970
Tortoise: 980
Tortoise: 990
Tortoise: 1000
Tortoise: I finished!

The race is over! The Tortoise is the winner!

Hare: You beat me fair and square, this time!
```



## BONUS-8

**Task:** Using the design parameters below and your newly-earned HTML/CSS knowledge, create a document from scratch.

### Build Specifications

- Use test driven development practices to complete your application.
- Implement elements of HTML/CSS to create a simple webpage.
- Incorporate different elements, such as headings, buttons, etc.

### Extended Exercises

- Add even more stuff! Link to Facebook/Twitter/ etc.
- Feel free to use JavaScript, Bootstrap, or other resources.

Sample:

The screenshot shows the level eleven website. The navigation bar includes the level eleven logo, links for SOLUTION, COMPANY, SALES LIBRARY, and BLOG, and a GET A DEMO button. The main headline reads "We Saw \$2.1 Million In New Sales" by Jim Smith - DMN Media. Below this, a testimonial states: "Modern sales leaders, meet your sales performance platform. Get ready to transform your team into a data-driven sales machine." followed by four bullet points: "Identify & track the metrics that matter in your sales process.", "Keep your team focused on those metrics, & spike performance.", "Motivate your team with visibility into their own sales metrics.", and "Do it all within Salesforce, without help from IT." At the bottom of the testimonial are two buttons: WATCH VIDEO and GET A DEMO. To the right, a tablet and smartphone display the level eleven dashboard. The tablet screen shows a "NEW BUSINESS BUMP-UP" report with a table of sales data:

Rank	Name	Amount
1	MONICA THOMPSON	\$1,663,000
2	JIMMY WALTER	\$1,200,000
3	HEATHER O'DONNELL	\$1,100,800
4	SERGIO PHILLIPS	\$850,000
5	KIYOMI YOSHIMOTO	\$600,000
6	GILBERT DUNCAN	\$500,650

The smartphone screen shows a sidebar with metrics: PROSPECTING CALLS (55), NEW BIZ MEETINGS (24), NEW PIPELINE (\$64K), and CLOSED DEALS.

## WHO'S USING LEVELEVEN?

VPs of SALES



VPs of MARKETING



SALES OPS



Now, challenge yourself! Don't just steal the code from [leveleven.com](http://leveleven.com)!



## BONUS-9

**Task:** Create a personal website for yourself and to display your projects.

### Build Specifications

- Create at least 3 pages: home, about, and portfolio
- Your homepage should include 2-4 sections about you, for example: the first section focuses on blogging, the second focuses on developing, and the third focuses on your hometown.
- Include a header with navigation as well as a footer for all pages.
- Include social media links.
- Your portfolio page should include a description and link to projects you want to highlight.
- Your about page should include your bio and a way for visitors/employers to contact you.

### Extended Exercises


- Stick the header to the top of the page so that it always shows when you scroll down the page.
- Make your website responsive to device sizes by using Media Queries or a CSS framework like Bootstrap
- Use the Google Maps API to map your hometown (or Detroit or Grand Circus).






## BONUS-9

Sample:


[About](#) [My Work](#) [Contact](#)



# Jeseekia Vaughn


Detroit, MI

Hi! I'm a student, blogger, tech nerd, and always a  
Detroiter.




### The Commentator

I recently started my own personal blog with no focus in particular. I have been told that I am very insightful and decided to share my views on various experiences I have with society. As the National Society of Black Engineers' National Publications Chair, I also write a column for the NSBE mag. Recently I wrote a blog for Teach for America about why I chose to study Engineering. You should check it out!



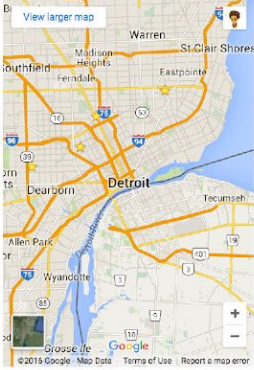
### The Tech Nerd

Along with being a Mechanical Engineering student at Wayne State University, I recently took an in-depth course on front-end web development at [Grand Circus Detroit](#). Because of this, I am now experienced in HTML(5), CSS(3), Javascript, and JQuery. I am new to TÁing for [Girl Develop It, Detroit](#)







### The Detroiter

Check out my city!



[My TFA Blog](#) [Learn with me](#) [Detroiter](#)

I'm pretty social too. We should be friends.



## BONUS-10

**Task:** Create a site demonstrating jQuery goodness.

### Build Specifications

- Create a webpage with the basic HTML needed and some HTML elements that can be manipulated.
- Create a list of jQuery topics. Each item should use jQuery to demonstrate what the feature does or how it works.
- You should list the following topics:
  1. Selectors
  2. Effects
    - Slide actions
    - Fade actions
    - Show/ hide
  3. Events
    - click
    - hover
    - dblClick

### Extended Exercises

- Implement 4 other jQuery events
- Incorporate some jQuery into a previous HTML/CSS/JS lab



## BONUS-11

**Task:** List artists and albums.

### What does the application do?

The application reads an XML file and displays a list of artists, albums, and albums by artist.

### Build Specifications

- Create a class named MusicArtistsApp that reads an XML file named music\_artists.xml and displays a list similar to that shown in the console below.
- Get the music\_artists.xml file from your instructor!

### Console Preview:

```
Artist and Album Listings
```

```
Artists
```

```
-----
```

```
The Beatles  
Elvis Presley  
John Prine
```

```
Albums
```

```
-----
```

```
Rubber Soul  
Revolver  
Sgt. Pepper's Lonely Hearts Club Band  
The White Album  
Elvis at Sun  
Elvis 30 #1 Hits  
John Prine  
Sweet Revenge
```

```
Artists and Albums
```

```
-----
```

```
The Beatles  
Rubber Soul  
Revolver  
Sgt. Pepper's Lonely Hearts Club Band  
The White Album  
Elvis Presley  
Elvis at Sun  
Elvis 30 #1 Hits  
John Prine  
John Prine  
Sweet Revenge
```



## BONUS-12

- **Task:** Design a parking lot using object-oriented principles. You need to specify the classes and the relationships between them.
- **Build Specifications**
  1. The parking lot has multiple levels. Each level has multiple rows of spots.
  2. The parking lot can park motorcycles, cars, and buses.
  3. The parking lot has motorcycle spots, compact spots, and large spots.
  4. A motorcycle can park in any spot.
  5. A car can park in either a single compact spot or a single large spot.
  6. A bus can park in five large spots that are consecutive and within the same row. It cannot park in small spots.

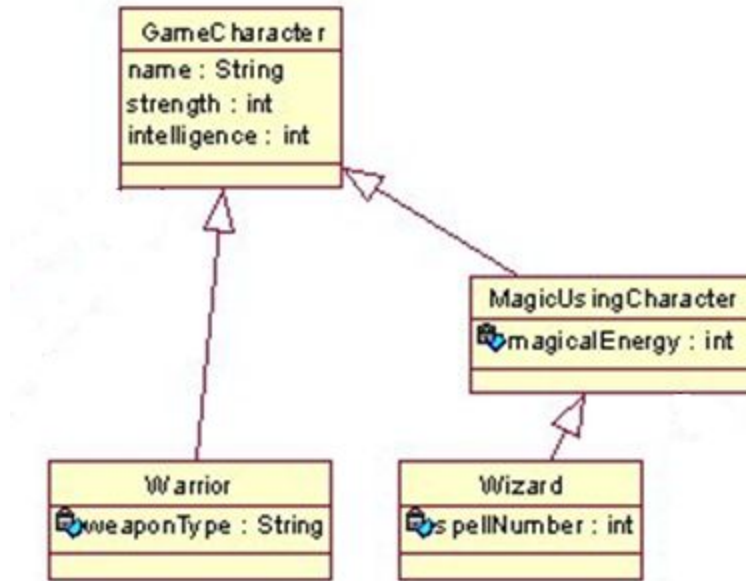


## BONUS-13

**Task:** Implement polymorphism

### Build Specifications

Consider the following class structure:



1. Implement each one of the classes you see in the figure. You need to code the setters and getters, constructors. Add a method called "Play" at the **GameCharacter** class, which basically prints the Name of the Character and shows the strength and the intelligence.
2. You need to override the Play method at the **Warrior**, **MagicUsingCharacter**, and the **Wizard** classes. Each method should print all the properties of that class (including the inherited ones).
3. In the main, create an array of type **GameCharacter** and call it "gameCharacters", and then add: Two Warriors and three Wizards. After that, do a for loop to invoke the Play method for all the objects in that array.



## BONUS-14

**Task:** Create an extensive movie database

**What does the application do?**

- The application lists and categorizes movies.
- The application is on a webpage.

**Build Specifications**

1. Find a partner. Each you and your partner should locate your code from Lab #10 (list movies). Create a repository on GitHub and push both of your projects, then reconcile any differences.
2. Once you're both working from the same code, create a database to hold all of the existing movies.
3. Expand your movie offerings to at least 40 movies.
4. Add at least one new category.
5. Put together a beautiful front end so that users can easily interact with your movie database.

**Extended Challenges:**

- Allow users to rate movies and store these ratings within the database.
- Sort the movies alphabetically which a category is displayed.
- Allow users to choose more than one category to display at a time.



## BONUS-15

**Task:** Maintain customer data (Use the Customers table in the Northwind database)

### What does the application do?

- The application begins by displaying a menu with four options: list, add, delete, and modify.
- If the user enters “list,” the application shows the customer data in the database.
- The other options can be used to modify the Customers table.

### Build Specifications

1. You need to use the Entity framework to link to the database, and then use the SQL commands to get and modify data.
2. Use prepared statements! Don't use concatenation to construct the SQL commands.
3. Build a DAO (Data access object) class to have all the SQL commands there.
4. Put the connection string in a configuration file.



## BONUS-16

**Task:** Hold details about cars in a Car class.

### What will the application do?

- Ask the user how many cars they want to enter.
- For each car, take input for the details.
- Print out a table of the cars at the end.

### Build Specifications

- Create a class named Car to store the data about this car. This class should contain:
  - Data members for car details
    1. A String for the make
    2. A String for the model
    3. An int for the year
    4. A double for the price
  - A no-arguments constructor that sets data members to default values (blanks or your choice)
  - A constructor with four arguments matching the order above
  - Properties for all data members
  - Override the ToString() method returning a formatted string with the car details.
- Create a class named CarApp that gets the user input, creates a Car object, and displays the car information.

### Hints:

- Use the right visibility modifiers!
- Use either an array or a List to hold the cars.
- Make sure to match the signature of ToString() from Object.
- You can just use \t tab escape characters to line things up, or if you want to get fancier, look at for text formatters.

### Extended Challenge:

- Modify or create a class named Validator with static methods to validate the data in this application.





## BONUS-16

### Console Preview:

Welcome to the Grand Circus Motors admin console!

How many cars are you entering: {4}

Enter Car #1 Make: {*Nikolai*}

Enter Car #1 Model: {*Model S*}

Enter Car #1 Year: {*2017*}

Enter Car #1 Price: {*54999.90*}

...

Current Inventory:

Nikolai	Model S	2017	\$54,999.90
Fourd	Escapade	2017	\$31,999.90
Chewie	Vette	2017	\$44,989.95
Hyonda	Prior	2017	\$23,795.50



## BONUS-17

**Task:** Extend your Car class with a UsedCar subclass.

### What will the application do?

- Display a list of at least 6 cars (at least 3 new and 3 used).
- Let the user select one of the cars.
- Re-display the information for car selected.
- Ask the user if they want to buy the car. If they do, remove the car.
- Re-display the list.

### Build Specifications

1. Create a subclass of Car named UsedCar. UsedCar has additionally:
  - a. Data member: A double for mileage.
  - b. Constructor: Takes five arguments (same order as constructor from last lab with the mileage last).
  - c. ToString: overrides Car's ToString() to include (Used) and the mileage.
2. For this lab, you can instantiate your Car and UsedCar instances in the code—you don't need to take input for them.
3. Make your menu system as you'd like (for choosing which car from the list—numbers, letters, any other way)
4. Store all Car and UsedCar instances together in the same List. (This time, don't use an array.)

### Hints:

- Let polymorphism work for you.
- Remember casting.

### Extended Challenge:

- Again use a Validator class (see previous lab).



## BONUS-17

### Console Preview:

1. Nikolai	Model S	2017	\$54,999.90
2. Fourd	Escapade	2017	\$31,999.90
3. Chewie	Vette	2017	\$44,989.95
4. Hyonda	Prior	2015	\$14,795.50 (Used) 35,987.6 miles
5. GC	Chirpus	2013	\$8,500.00 (Used) 12,345.0 miles
6. GC	Witherell	2016	\$14,450.00 (Used) 3,500.3 miles
7. Quit			

Which car would you like? {6}

GC            Witherell      2016            \$14,450.00 (Used) 3,500.3 miles

Would you like to buy this car? {y}

Excellent! Our finance department will be in touch shortly.

1. Nikolai	Model S	2017	\$54,999.90
2. Fourd	Escapade	2017	\$31,999.90
3. Chewie	Vette	2017	\$44,989.95
4. Hyonda	Prior	2015	\$14,795.50 (Used) 35,987.6 miles
5. GC	Chirpus	2013	\$8,500.00 (Used) 12,345.0 miles
6. Quit			

Which car would you like? {6}

Have a great day!



## BONUS-18

**Task:** Create a CarLot class to store your new and used cars. This is a more open-ended and design-focused lab.

### What will the application do?

- Display a list of at least 6 cars (at least 3 new and 3 used).
- Let the user select one of the cars.
- Display and price and (if a used car) mileage for the selected car.
- Ask if they want to select again and repeat if they do.

### Build Specifications

1. Pair with a classmate for this lab. Choose one person's Car and UsedCar to use, or merge your two together.
2. Discuss with your partner how a CarLot class could best store information. In what cases would each of these make more sense?
  - A two-dimensional array of Cars
  - A List of Cars
  - Any other option?
3. Plan out your methods for CarLot. Decide if any other information is needed in the CarLot class or in Car/UsedCar.
4. Write a CarLotApp class which instantiates and puts cars in your CarLot class. It should invoke CarLot methods to let a user:
  - List all cars.
  - Add a car.
  - Remove a car.
  - Look up a car in a given position.
  - Replace a car in a given position.

### Extended Challenge:

Provide search features:

- View all cars of an entered make.
- View all cars of an entered year.
- View all cars of an entered price or less.
- View only used cars or view only new cars.

Your output will vary based on decisions you make with your partner.



## BONUS-19 (COLLECTIONS)

Task: Make a shopping list application which uses collections to store your items.

What will the application do?

- Display a list of at least 8 item names and prices.
- Ask the user to enter an item name
  - If that item exists, display that item and price and add that item and its price to the user's order.
  - If that item doesn't exist, display an error and re-prompt the user. (Display the menu again if you'd like.)
- Ask if they want to add another. Repeat if they do. (User can enter an item more than once; we're not taking quantity at this point.)
- When they're done adding items, display a list of all items ordered with prices in columns.
- Display the average price of items ordered.

Build Specifications

- Use a Dictionary <string, double> to keep track of the menu of items. You can code it with literals.
- Use parallel ArrayLists (one of strings, one of doubles) to store the items ordered and their prices.
- Write 3 methods to find 1) the average of the items ordered and the indexes of the 2) highest and 3) lowest cost items.

Extended Challenges:

- Implement a menu system so the user can enter just a letter or number for the item.
- Make a third ArrayList for quantities of items ordered and allow the user to order more than one at a time.
  - Extended extended: If they order an item already in their cart, increase the quantity rather than adding another entry.
- Display the most and least expensive item ordered.

Console Preview:

<b>Welcome to Guenther's Market!</b>	
<b>Item</b>	<b>Price</b>
=====	
apple	\$0.99
banana	\$0.59
cauliflower	\$1.59
dragonfruit	\$2.19
Elderberry	\$1.79
figs	\$2.09
grapefruit	\$1.99
honeydew	\$3.49



```
What item would you like to order? apple
Adding apple to cart at $0.99

Would you like to order anything else (y/n)? y

{menu redisplays}

What item would you like to order? watermelon
Sorry, we don't have those. Please try again.

{menu redisplays}

What item would you like to order? dragonfruit
Adding dragonfruit to cart at $2.19

Would you like to order anything else (y/n)? y

{menu redisplays}

What item would you like to order? honeydew
Adding honeydew to cart at $3.49

Would you like to order anything else (y/n)? y

{menu redisplays}

What item would you like to order? figs
Adding figs to cart at $2.09

Would you like to order anything else (y/n)? n

Thanks for your order!
Here's what you got:
apple          $0.99
dragonfruit    $2.19
honeydew       $3.49
figs           $2.09
Average price per item in order was 2.19
Press any key to continue . . .
```



## BONUS-20

Write SQL Queries for the following:

1. Select all records from the customers table.
2. Find all customers living in London or Paris
3. Make a list of cities where customers are coming from. The list should not have any duplicates or nulls.
4. Show a sorted list of employees' first names.
5. Find the average of employees' salaries
6. Show the first name and last name for the employee with the highest salary.
7. Find a list of all employees who have a BA
8. Find total for each order
9. Get a list of all employees who got hired between 1/1/1994 and today
10. Find how long employees have been working for Northwind (in years!)
11. Get a list of all products sorted by quantity (ascending and descending order)
12. Find all products that are low on stock (quantity less than 6)
13. Find a list of all discontinued products.
14. Find a list of all products that have Tofu in them.
15. Find the product that has the highest unit price.
16. Get a list of all employees who got hired after 1/1/1993
17. Get all employees who have title : "Ms." And "Mrs."
18. Get all employees who have a Home phone number that has area code 206



## BONUS-21

**Task:** Work with an API that generates JSON data to display information on a web page.

### Build Specifications:

1. Work with Mashape API at <https://market.mashape.com/divad12/numbers-1>. To use this API, you need to signup for an account at <https://market.mashape.com> . The site will provide you with a Key that you will need for connecting with the API.
2. Read the documentation to know how to use the Key. The Key should be added to the header of the Http request in your code.
3. For the API, you need to create an Html form that has two text fields: Day and month. When you submit the form, you need to call the numbers API **Get a fact** (go to <https://market.mashape.com/divad12/numbers-1> to read the documentation ) to get a random fact for that date, then display it on a view.
4. On another view, create a view that will show a random fact by calling the **Get random fact** API. The API will require a type for the fact, which can be trivia, math, date, or year.

Notes: If you want to see a tree representation of the JSON data, use this website: <http://jsonviewer.stack.hu/>

### Extended Challenges:

1. Try to use the other APIs under the number API in other views of your making.





## BONUS-22

**Task:** Calculate batting statistics.

### What will the application do?

- The application calculates the batting average and slugging percentage for one or more baseball/softball players.
- For each player, the application first asks for the number of at bats. For each at bat, the application asks for a result.
- To enter an at-bat result, the user enters the number of bases earned by the batter.
- After all the at-bats are entered, the application displays the batting average and slugging percentage of the batter.

### Build Specifications

1. Use an Array to store the at-bat results for each player.
2. Validate the input so that the user can only enter positive integers. For the at-bat results, the user can only enter 0, 1, 2, 3, or 4.
3. Validate the user's response to the question "Another batter?" so that the user can only enter Y, y, N, or n. If the user chooses N or n, end the program.
4. Format the batting average and slugging percentages such that three decimal places are shown.

### Hints:

- Batting average is total number of at-bats for which the player earned at least one base divided by the total number of at-bats.
- Slugging percentage is the total number of bases earned divided by the total number of at-bats.

### Extended Challenges:

- At the start of the program, prompt the user for the number of batters to enter, then save the statistics in a two-dimensional array. The program won't have to prompt the user whether to enter data for another batter since it will know how many batters are to be entered. After all batters have been entered, print a one line summary for each batter.
  - Batter 1 average: 0.357                      slugging percentage: 0.650
  - Batter 2 average: 0.238                      slugging percentage: 0.540
- Instead of storing an Array of integers, create a class named AtBat and store instances of this class in the array. This class should define an enumeration named Result with members OUT, SINGLE, DOUBLE, TRIPLE, and HOMERUN. The class should have a constructor that accepts a Result parameter and a method named basesEarned that returns an int representing the number of bases earned for the at bat.



## BONUS-22

### Console Preview:

Welcome to Batting Average Calculator!

Enter number of times at bat: 5

0=out, 1=single, 2=double, 3=triple, 4=home run

Result for at-bat 0: 0

Result for at-bat 1: 1

Result for at-bat 2: 0

Result for at-bat 3: 2

Result for at-bat 4: 3

Batting average: 0.600

Slugging Percentage: 1.200

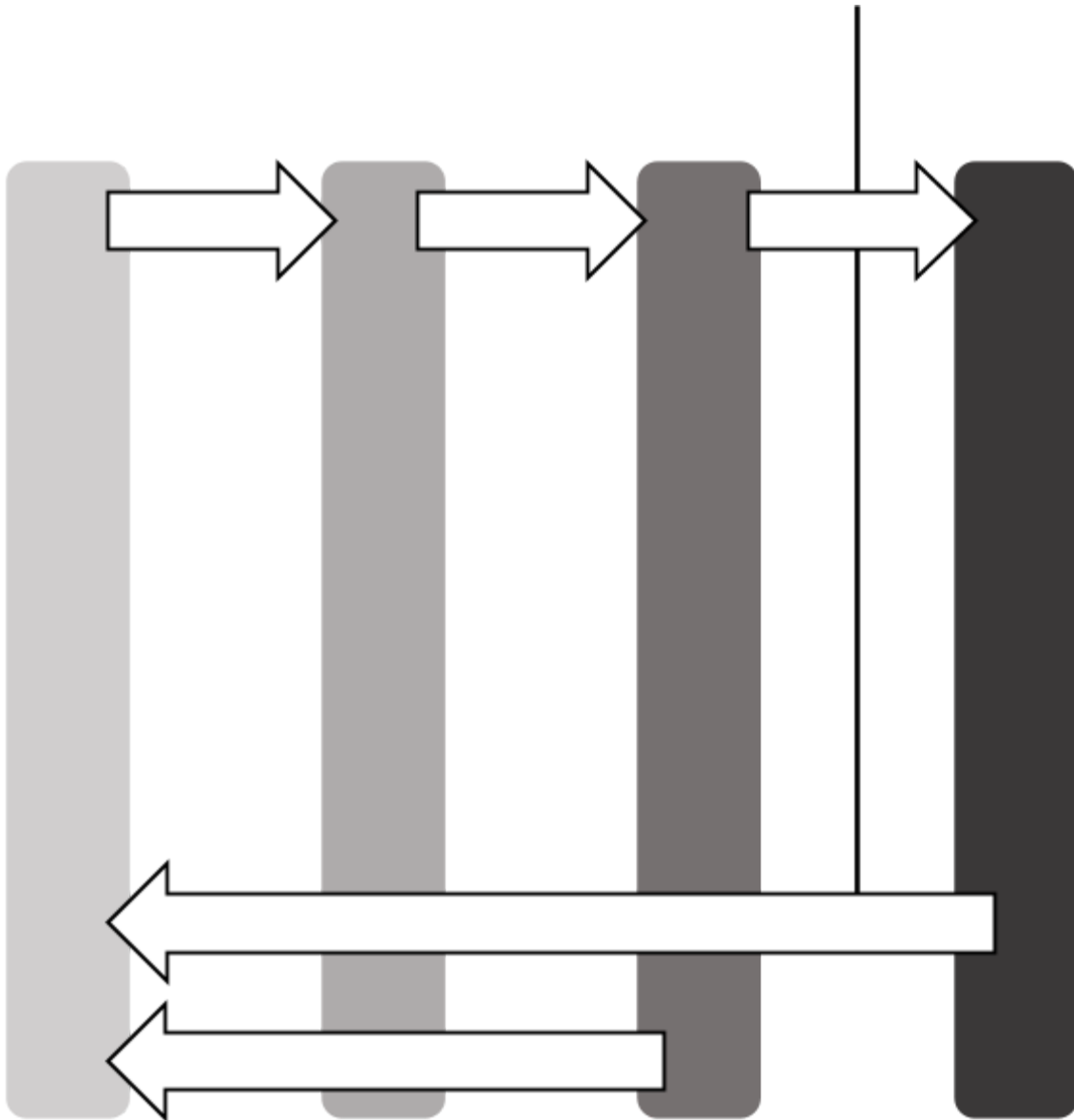
Another batter? (y/n): N



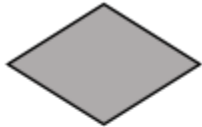
# DIAGRAMS

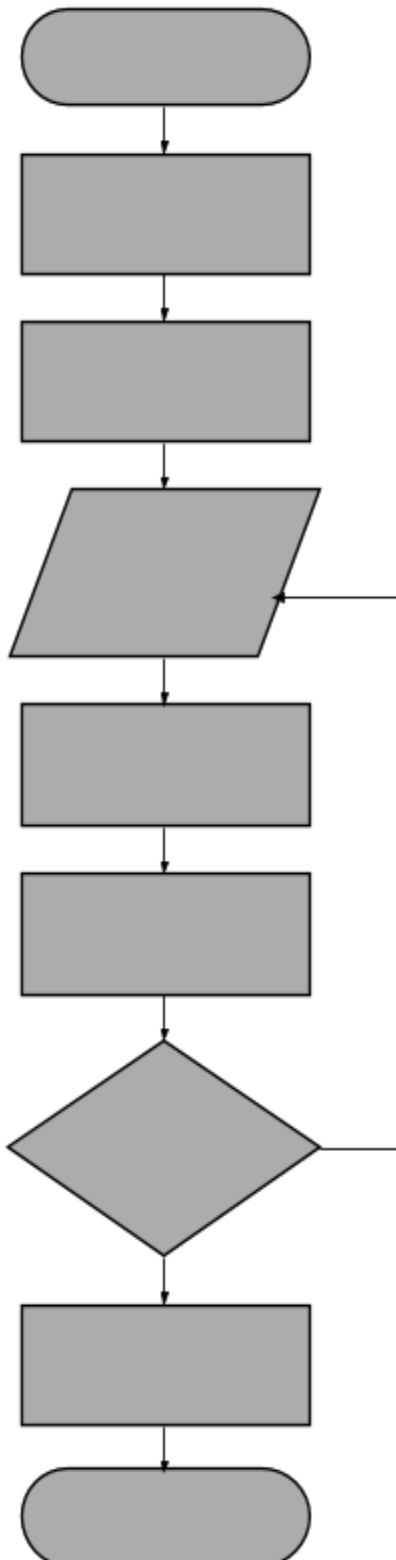


# GIT & GITHUB



# COMMON FLOWCHARTING SYMBOLS





# SQL REFERENCE

## SELECT STATEMENTS

**SELECT \* FROM tbl**

Select all records, all columns from table named tbl

Example: **SELECT \* FROM Students**

**SELECT col1, col2 FROM tbl**

Select col1 and col2 only from all records from table named tbl

Example: **SELECT FirstName, LastName, ID FROM Students**

**SELECT \* FROM tbl WHERE condition**

Select all columns from records in table named tbl which meet condition

Example: **SELECT \* FROM Students WHERE ID > 2000**

**ORDER BY colname ASC, DESC**

Add these qualifiers to the end of a SQL query to sort it.

Examples: **SELECT \* FROM Students ORDER BY LastName ASC**

**SELECT \* FROM Students ORDER BY LastName ASC, ID DESC**

**SELECT DISTINCT colname FROM tbl**

Retrieve unique values for column colname in table tbl.

Examples: **SELECT DISTINCT FirstName FROM Students**

Note that these building blocks can be combined.

Example: **SELECT DISTINCT LastName FROM Students WHERE ID > 2000 ORDER BY ID ASC**

**SELECT \* FROM tbl1 INNER JOIN tbl2 ON join-condition**

Pull information from table 1 and table 2, matching records based on condition.

Example: **SELECT \* FROM Students INNER JOIN Bootcamps WHERE  
Students.CampID = Bootcamps.ID**



## TABLE STATEMENTS

```
CREATE TABLE tbl (  
    col1 datatype (length) ,  
    col2 datatype (length), ...  
    PRIMARY KEY col1  
)
```

Create a table named tbl with the columns indicated. Set the primary key to the first column.

Example: 

```
CREATE TABLE Students (  
    FirstName VARCHAR(55) ,  
    LastName VARCHAR(55) ,  
    ID INT(11) NOT NULL AUTO_INCREMENT ,  
    PRIMARY KEY ID  
)
```

```
DROP TABLE tbl
```

Delete the table tbl from the database.

Example: 

```
DROP TABLE Students
```

```
INSERT INTO tbl (col1, col2) VALUES (val1, val2)
```

Add a new record into table tbl, putting val1 into col1, val2 into col2, etc.

Example: 

```
INSERT INTO Students (FirstName, LastName) VALUES ("Jane",  
    "Smith")
```

(Note that because ID is autoincremented, we don't provide a value.)

```
DELETE FROM tbl WHERE condition
```

Remove records where condition is met.

Example: 

```
DELETE FROM Students WHERE ID=102
```

```
UPDATE tbl SET col1=val1 WHERE condition
```

Change the value stored in col1 for records where condition is met.

Example: 

```
UPDATE Students SET LastName="Williams" WHERE ID=101
```

```
ALTER TABLE tbl ADD COLUMN col1 datatype (length)
```

Add a column of datatype (and length) named col1 to table tbl.

Example: 

```
ALTER TABLE Students ADD COLUMN Email VARCHAR(60)
```

```
ALTER TABLE tbl DROP COLUMN col1
```

Remove the column col1 from table tbl. All values are lost.

Example: 

```
ALTER TABLE Students DROP COLUMN Email
```

Syntax Notes:

- On most SQL servers, SQL keywords like SELECT, FROM, ORDER BY, DISTINCT, ASC, DESC, etc. are not case sensitive. Many reference materials follow the convention of capitalizing them for clarity, but many developers don't do this in actual practice.
- Closing semicolon is optional unless the query includes more statements.





# GLOSSARY



# GLOSSARY OF TERMS

**ARRAY:** a series of elements, all of which are of the same size and type

```
int[] anArray;  
anArray = new int[3];  
anArray[0]=5;  
anArray[1]=10;
```

**ARRAY ELEMENT:** an entry within an array

```
anArray[0]=5; // 5 is an element in anArray
```

**LIST:** a dynamic series of objects; used instead of an array if total number of objects for the array is unknown at the time of the ArrayList's creation.

```
List<int> aList = new List<int>(10);  
// creates an ArrayList of integers of size 10
```

**CLASSES:** a blueprint from which individual objects are made (like a template – think of a cookie cutter)

```
public class Cupcake  
{  
    private String flavor;  
    private String frosting;  
    private boolean sprinkles;  
  
    public void Cupcake()  
    {  
        flavor = "vanilla";  
        frosting = "pink";  
        sprinkles = true;  
    }  
  
    public void Cupcake(String bestFlavor,String bestFrosting,  
                        boolean addSprinkles)  
    {  
        flavor = bestFlavor;  
        frosting = bestFrosting;  
        sprinkles = addSprinkles;  
    }  
  
    public void EatCupcake()  
    {  
        Console.WriteLine("Yum!");  
    }  
}
```



**COMPOSITION:** describes a relationship between two or more objects, it is an “has a/an” relationship; an object made up of other objects

```
public class HungryPerson
{
    private Cupcake cupcake;
    //a HungryPerson “has a” cupcake;

    public void devourCupcake()
    {
        cupcake.EatCupcake();
    }
}
```

**CONCATENATION:** a way of joining strings one after another; most frequently executed with the + operator, though can be invoked with the .concat() method.

```
“Grand” + “Circus”; //if printed, would be GrandCircus

String s = “Grant”;
s =s.Concat(“Chirpus”); //if printed, would be GrantChirpus
```

**CONDITIONALS:** statements that break up the flow of execution by employing decision making, looping, and branching, enabling the program to execute conditional lines of code

```
char response = 'Y';

if (response == n)
{
    Console.WriteLine (“womp womp”);
}
else
{
    Console.WriteLine (“Hooray!!”);
}
```

**IF STATEMENTS:** executes a certain segment of code only if a specified test evaluates as true.

**IF -ELSE STATEMENTS:** provides a secondary path in the instance that the if-then statement returns false.

**SWITCH STATEMENT:** works with primitive data types, enums to provide a number of different execution paths. Uses a switch block with one or more cases or default labels to decide which path to execute.

```
char response = Y;
switch (response)
{
    case n:      Console.WriteLine (“womp womp”);
    case maybe: Console.WriteLine (“Decide!”);
    default:     Console.WriteLine (“Hooray!!”);
}
```



**CONSTRUCTOR:** a method that shares the same names as its class that assigns initial field values to objects when invoked

```
public class Cupcake
{
    /*member variables, other member function code omitted for brevity*/

    public Cupcake(String bestFlavor,String bestFrosting,
        bool addSprinkles)
    {
        flavor = bestFlavor;
        frosting = bestFrosting;
        sprinkles = addSprinkles;
    }
}
```

**INHERITANCE:** describes a relationship between two or more objects, it is an “is a/an” relationship; an object derived from another object; can be modified or extended

```
public class SuperCupcake : Cupcake
{
    public bool cherryOnTop;
    /*SuperCupcake inherits all the fields of Cupcake, and
        adds one of its own*/
}
```

**LOOPS:** used in C# to execute a statement repeatedly; C# has three main types:

**FOR LOOPS:** used to execute a statement repeatedly for a given number of times when the programmer knows the number of times to repeat the statement

```
for (int i = 0; i < 12; ++i)
{
    Console.WriteLine ("I think I'll have a cupcake!\n");
}
//“I think I'll have a cupcake!” prints 12 times
```

**WHILE LOOPS:** used to execute a statement repeatedly until a specific condition is fulfilled

```
int dozenCupcakes = 12;
while (iAmHungry > 0)
{
    Console.WriteLine ("I think I'll have a cupcake!\n");
    --dozenCupcakes;
}
//“I think I'll have a cupcake!” prints 12 times
```

**DO WHILE LOOPS:** used to execute a statement repeatedly until a specific condition is met and then checks the condition

```
int dozenCupcakes = 12;
do
{
    Console.WriteLine ("I think I'll have a cupcake!\n");
    --dozenCupcakes;
} while (iAmHungry > 0);
```



**METHOD:** a set of rules that describe the behavior of an object or class. In a statement, a modifier can be called by using the period [squares.getArea()]. Ex:

```
modifier returnType NameOfMethod (parameter list)
{
    method body;
}
```

**MODIFIER:** defines the access type of a method (public/private); optional to use

**RETURN TYPE:** type of return expected by the method (ex: String, int, double)

**NAME OF METHOD:** must begin with lowercase, followed by camel case (ex: getArea)

**PARAMETER LIST:** type, order, and number of parameters the method needs; it is optional to use these (ex: (int a, String name)).

**METHOD BODY:** defines what the method does with statements

**NEW/NEW KEYWORD:** used to create new instances of objects

```
Object Foo = new Object();
```

**OBJECT:** is created from a class, all objects have states (fields) and behaviors (methods); objects must be *declared*, *instantiated*, and *initialized* before they can be used

**DECLARATION:** associates a variable name with a specified object type

```
int dozenCupcakes;
//creates an integer named "dozenCupcakes"
```

**INITIALIZATION:** establishes the properties for a newly instantiated object

```
int dozenCupcakes = 12;
//assigns a value to dozenCupcakes
```

**INSTANTIATION:** creates an instance of an object, often using the "new" keyword (think of a cookie cutter stamping out an actual cookie – the cookie is an "instance" of the cookie cutter)

```
Cupcake yummyCupcake = new Cupcake();
```

**OPERATOR:** special symbols that perform specific operations on one, two, or three *operands* and then return a result. Frequently used operators are:

**\* , / , %:** multiplicative

```
[Ex: double area=(.5)*this.base*this.height;]
```

**+ , - :** additive

```
[Ex: String myIntroduction="Hello, I'm"+name;]
```

**< , > , <= , >= , instanceof:** relational

```
[Ex: while(count<=8) {
    Console.WriteLine (count);
}
Console.WriteLine ("Done!"); ]
```

**== :** equality

**&& :** logical AND.

**|| :** logical OR.



**OVERLOADING:** when more than one method in a class have the same name; can only happen when their argument lists are different;

```
public class Cupcake {
/*member variables, other member function code omitted for brevity*/
    public Cupcake() {
        //default constructor, has no parameters
        flavor = "vanilla";
        frosting = "pink";
        sprinkles = true;
    }
    public Cupcake(String bestFlavor,
String bestFrosting,
boolean addSprinkles){
/*shares the same name as default constructor; takes various
parameters*/
        flavor = bestFlavor;
        frosting = bestFrosting;
        sprinkles = addSprinkles;
    }
}
```

**OVERRIDING:** when a different implementation of a super method is used in a child class; when a method of the same name as a parent method is executed with different parameters in the child class' method

**POLYMORPHISM:** In object oriented programming, it is the ability to redefine methods (override) for derived classes; a programming language's ability to process objects differently based on their data type or class

```
//two versions of the StartBaking method

    public String StartBaking() {
        return "Let's bake!";
/*this would go in the BakedGood superclass, which Cupcake extends*/
    }

public String StartBaking() {
    return "Let's bake cupcakes!";
/*this would go in the Cupcake class, which extends BakedGood*/
}
```

---

```
BakedGood b = new BakedGood();
Console.WriteLine(b.StartBaking());
//prints "Let's bake!"
```

```
BakedGood c = new Cupcake();
Console.WriteLine(c.StartBaking());
```



```
//prints "Let's bake cupcakes!"
```

**STRINGS:** a series of characters (char), denoted by `" "`; an object. Ex:

```
String name = "Bootcamper Debbie";
```

**THIS KEYWORD:** used within an instance method or a constructor to reference the current object (as opposed to a super). Ex:

```
public Person(String name) {  
    this.name=name;  
}
```



## ADDITIONAL TERMS TO KNOW

Got a term you just can't seem to remember? Write it out here for quick reference.

[illegible]