Reflection

Libraries

A library is a collection of classes in C# that can be used in any project as long as you import them, this means that when you have created a class say “Person” you can access any public function in that class.

The way you do this is by adding it as a dependency:

```
using People;
using Register;
using System;
using System.Collections.Generic;
```

These dependencies are declared like that at the top of any C# document and then can be used, you will still need to add the library to you project solution but that can be a simple copy past job.

Then all you need to do to use a function form that library is create a new var with that class as an object and then when you access the function with the formatting:

```
var peoples = new RegisterClass(new List<Person>());

peoples.AddToList(new Person("bob", 20, 293749, "somewhere", "janitor", 700d,
"merc", "here"));
```

In this case the function name is AddToList, and the variable that the object is stored in is named peoples.

UML Class Diagram

A class diagram is a visual representation of what the code dose/will do, this means that it can either be used for planning a project or documenting an already existing project.

The reason why you would want to use a class diagram to plan out your projects is that it can give you a good overview of what you need to write later, it also forces you to think about what a class should contain.

The Formatting that is sued for a box is that at the top you have the class name, and under that you have the variable names and their type formatted like name : type. Here you also indicate weather or not the variable is public or private, this indicates how you can access them.

Public meaning that it can be accessed from anywhere and can there for also be modified form anywhere.

Private indicates that it can only be accessed in that class and can not be modified by anyone but that class.

After one declares the variables the next and last thing that needs to be added is the functions, this is done mostly in the same way. name(input) : output type. Here you also indicate what protection level the functions should have.

Dictionaries

Dictionaries are another way of storing information like a with a List or an Array, but the way that it is different is how you use it. It uses a key and that key is allocated a string, object, integer or what ever else you can have in a variable.

```
Dictionary<Basicperson, string> classDict = new Dictionary<Basicperson, string>
{
       { new Basicperson("bob", 25), "test" }
};
```

In this case the key is an object named Basicperson and the thing that key is referencing is a string.

(I must admit I don't really have a grasp on how to access the value stored in the dictionary and that this is something that I need to work on.)

Delegates

Delegates are functions stored in variables, the concept is simple to understand but slightly more difficult to use. This is the best and most common used example I can give.

```
public delegate int MyDelegate(int x, int y);
class Program
{
    static int Sum(int x, int y) { return x + y; }

    static void Main()
    {
        var d = new MyDelegate(Sum);

        int result = d.Invoke(12, 15);

        Console.WriteLine(result);

        Console.ReadLine();

    }
}
```

You start off but declaring it in the name space where it is to be used, first the access level in this case public. Then you go on to say that it is a delegate, and what its return type is. The return type is the same as the function that is going to be stored using this delegate, after that you give it a name. Then come the input types that also must match the functions input types that is going to be stored in it.

Then to use it you create a var, in this var you will create a new instance of the delegate and give it the name of the function as input. After that you can invoke the function like you would a function in a class. But with the function now in a variable you can use the output of that function directly in the input of another function, you can also add the variable to an array, list or even dictionary.