|  |
| --- |
| [Type the company name] |
| SSE554 – Project 1: The Sample Pokedex |
| [Type the document subtitle] |

|  |
| --- |
| Andrew Robinson  [Pick the date] |

Table of Contents

[Background 4](#_Toc31045875)

[Development Process 4](#_Toc31045876)

[Searching the Pokedex By Number 10](#_Toc31045877)

[Reference 13](#_Toc31045878)

[FileReader Class 13](#_Toc31045879)

[FileReader.getPokemonFromFile Method 13](#_Toc31045880)

[Attributes Enumeration 14](#_Toc31045881)

[Pokedex Class 15](#_Toc31045882)

[Pokedex Constructor 15](#_Toc31045883)

[Pokedex.Dex Property 16](#_Toc31045884)

[Pokedex.getPokemonByName Method 16](#_Toc31045885)

[Pokedex.getPokemonByNumber Method 17](#_Toc31045886)

[Pokedex.getWeaknessVal Method 17](#_Toc31045887)

[dex Field 18](#_Toc31045888)

[weaknessesAndResistances Field 19](#_Toc31045889)

[PokedexMissingEntryException Class 19](#_Toc31045890)

[PokedexMissingEntryException Constructor 20](#_Toc31045891)

[Pokemon Class 20](#_Toc31045892)

[Pokemon Constructor 21](#_Toc31045893)

[Pokemon.Name Property 21](#_Toc31045894)

[Pokemon.PrimaryType Property 22](#_Toc31045895)

[Pokemon.SecondaryType Property 22](#_Toc31045896)

[Pokemon.Equality Method 23](#_Toc31045897)

[Pokemon.Inequality Method 24](#_Toc31045898)

[Pokemon.ToString Method 24](#_Toc31045899)

[Type Enumeration 25](#_Toc31045900)

[name Field 26](#_Toc31045901)

[primaryType Field 26](#_Toc31045902)

[secondaryType Field 27](#_Toc31045903)

[PokemonTypeNotRecognizedException Class 27](#_Toc31045904)

[PokemonTypeNotRecognizedException Constructor 27](#_Toc31045905)

[PokemonTypeNotRecognizedException(String) Constructor 28](#_Toc31045906)

[Program Class 28](#_Toc31045907)

[Program.Main Method 29](#_Toc31045908)

[Program.damageMenu Method 29](#_Toc31045909)

[Program.getInput Method 30](#_Toc31045910)

[Program.pokemonMenu Method 31](#_Toc31045911)

[Program.printMainMenu Method 32](#_Toc31045912)

[Program.searchByName Method 32](#_Toc31045913)

[Program.searchByNumber Method 33](#_Toc31045914)

[go Field 34](#_Toc31045915)

[kanto Field 34](#_Toc31045916)

[Index 35](#_Toc31045917)

Background

In the world of Pokemon, there is a device called the “Pokedex.” This device is referred to as an “encyclopedia” of all the different creature that reside in the world. Each Pokemon is referred to by a unique name, has a unique number, and has one to two types of the 18 that exist in the world. Each one of these types either resists, is unaffected, is susceptible to, or negates damage from attacks of the other 17 types.

To demonstrate my knowledge of Object Oriented Programming and Unit Testing, I have created a sample Pokedex Object and a supporting program using Test Driven Development. I accomplished this task by using **C#** to write my program and **MSTest** to create and run my unit tests. GitHub was used for version control.

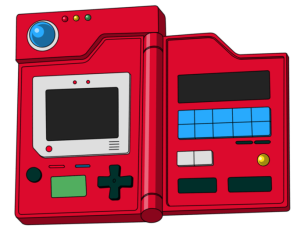


Figure A Pokedex from the Pokemon Video Game

*Figure from: rankedboost.com*

Development Process

My features were created using the Test Driven Development model in the following order:

1. Creation of the “Pokemon” class.
2. Importation of Pokemon from text file.
3. Calculate the damage weaknesses of any given Pokemon given any type of attack.
4. Adding equivalence operator overrides.
5. Adding ToString() method overrides.
6. Search the Pokedex by number.
7. Search the Pokedex by name.

For each one of these tasks the following happened:

1. Base code for the feature was added and threw an exception.
2. A test was created knowing what the expected outcome was.
3. The test was run to prove that it would fail.
4. Code was added to the features so they would pass the test.
5. The test was run again to prove that it would pass.

By making sure each of these tests was followed, I could ensure that each feature that I implemented had a test and that each feature produced the results that I expected.

Creation of the “Pokemon” Class

The characteristics that define each Pokemon include: their name, their primary type, and their secondary type (if they have one). I wanted to make sure that private variables were referenced correctly by their public accessor counterparts, so I made a unit test that checks that all the attributes of the new Pokemon object matches the values in the constructor.

Source code

|  |
| --- |
| [TestMethod, TestCategory("Pokemon")]  public void CheckPokemonCreation()  {  P1.Pokemon pokemon = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);  Assert.IsTrue(pokemon.Name.Equals("Rhydon") && pokemon.PrimaryType.Equals(P1.Pokemon.Type.Ground) && pokemon.SecondaryType.Equals(P1.Pokemon.Type.Rock), "Test Pokemon Creation Successful");  } |

Results

|  |
| --- |
| Test Name: CheckPokemonCreation  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckPokemonCreation  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 19  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckPokemonCreation  Test Outcome: Passed |

Importation of Pokemon From a Text File

In the creation of this project, I created a database in a text file of the original 151 Pokemon. I wanted to make sure that all 151 entries were imported successfully. The method run in the FileReader class has an exception to make sure that all the types in the file were valid, so the test would fail if that exception was invoked.

Source code

|  |
| --- |
| [TestMethod, TestCategory("FileReader")]  public void CheckImport()  {  List<P1.Pokemon> pokemonList = new List<P1.Pokemon>();  pokemonList.AddRange(P1.FileReader.getPokemonFromFile(@"D:\Users\arobi\Desktop\pokemondatabase.txt"));  Assert.AreEqual(pokemonList.Count, 151, "Pokemon list not of expected size");  } |

Results

|  |
| --- |
| Test Name: CheckImport  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckImport  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 11  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckImport  Test Outcome: Passed |

Calculate the Damage Weaknesses

As mentioned earlier in this document, each Pokemon can have one or two types. Both types are either resistant, weak, very weak, or immune to attacks from different types of Pokemon. To showcase this interaction, a damage chart can be used.

A picture containing car, wall

Description automatically generated

Figure Pokemon Type Damage Chart

*From dexerto.com*

In order to make sure that this test worked, I wanted to use a test Pokemon to see what damage it should be weak to. Rhydon is a Rock-Ground type. Rock is 2x weak to Grass and Ground is 2x weak to grass. By multiplying these values together, we get that Rhydon is 4x weak to Grass type attacks. Given this knowledge, I created a unit test to check these values.

Source code

|  |
| --- |
| [TestMethod, TestCategory("Pokedex"), TestProperty("Test", "1")]  public void CheckWeaknessesAndResistances()  {  P1.Pokemon rhydon = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);  Assert.AreEqual(4.0, P1.Pokedex.getWeaknessVal(rhydon, P1.Pokemon.Type.Grass), "Incorrect Value: Rhydon is 4x times weak to grass");} |

Results

|  |
| --- |
| Test Name: CheckWeaknessesAndResistances  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckWeaknessesAndResistances  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 48  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckWeaknessesAndResistances  Test Outcome: Passed |

Adding Equivalence Operator Overrides

Normally, the “==” operator just checks if the variables are identical copies of each other. By overriding this operator, comparisons anywhere else in the code will be much easier than individually comparing all three different elements that make a Pokemon unique. To test this, I created two Pokemon objects with identical properties. When the expression “pokemon1==pokemon2” returns true, then the equivalence operator is implemented properly.

Source code

|  |
| --- |
| [TestMethod, TestCategory("Pokemon")]  public void CheckIsEqual()  {  P1.Pokemon pokemon1 = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);  P1.Pokemon pokemon2 = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);  Assert.IsTrue(pokemon1 == pokemon2);  } |

Results

|  |
| --- |
| Test Name: CheckIsEqual  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckIsEqual  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 25  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckIsEqual  Test Outcome: Passed |

Adding ToString() Method Overrides

Similarly to the “==” operator override, I wanted to be able to simply display the information about each Pokemon in the program. To do this, I created a test that looked for an identically formatted string to the one that the ToString() method would provide. If the strings matched, then the ToString() override was successful.

Source code

|  |
| --- |
| [TestMethod, TestCategory("Pokemon")]  public void CheckPokemonToString()  {  P1.Pokemon rhydon = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);  string testString = "Pokemon: Rhydon\nTypes: Ground-Rock\n";  Assert.IsTrue(testString.Equals(rhydon.ToString()));  } |

Results

|  |
| --- |
| Test Name: CheckPokemonToString  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckPokemonToString  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 32  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckPokemonToString  Test Outcome: Passed |

Searching the Pokedex By Number

Each Pokemon has a unique identifying number associated with it. I wanted to be able to return a Pokemon if the user provided that number. To test this, I loaded a sample Pokedex and created a test Pokemon. I then loaded that same Pokemon using the getPokemonByNumber method. If the returned Pokemon matched the sample Pokemon, the test was successful.

Source code

|  |
| --- |
| [TestMethod, TestCategory("Pokedex")]  public void CheckGetPokemonByNumber()  {  P1.Pokedex kanto = new P1.Pokedex(@"D:\Users\arobi\Desktop\pokemondatabase.txt");  P1.Pokemon rhydon = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);  Assert.IsTrue(rhydon == kanto.getPokemonByNumber(112));  } |

Results

|  |
| --- |
| Test Name: CheckGetPokemonByNumber  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckGetPokemonByNumber  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 40  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckGetPokemonByNumber  Test Outcome: Passed |

Searching the Pokedex By Name

Like the search by number, each Pokemon has a unique name. I wanted to be able to return a Pokemon if the user provided that name. To test this, I loaded a sample Pokedex and created a test Pokemon. I then loaded that same Pokemon using the getPokemonByName method. If the returned Pokemon matched the sample Pokemon, the test was successful.

Source code

|  |
| --- |
| [TestMethod, TestCategory("Pokedex")]  public void CheckGetPokemonByName()  {  P1.Pokedex kanto = new P1.Pokedex(@"D:\Users\arobi\Desktop\pokemondatabase.txt");  P1.Pokemon rhydon = new P1.Pokemon("Rhydon", P1.Pokemon.Type.Ground, P1.Pokemon.Type.Rock);    Assert.IsTrue(kanto.getPokemonByName("Rhydon") == rhydon);  } |

Results

|  |
| --- |
| Test Name: CheckGetPokemonByName  Test FullName: P1-UnitTest.P1\_UnitTest.UnitTest1.CheckGetPokemonByName  Test Source: C:\Users\arobi\source\repos\SSE554-P1\P1-UnitTest\UnitTest1.cs : line 55  Test Outcome: Passed  Test Duration: 0:00:00  Test Name: CheckGetPokemonByName  Test Outcome: Passed |

Reference

Classes

[FileReader](#topic_0000000000000006), [Pokedex](#topic_000000000000000B), [PokedexMissingEntryException](#topic_0000000000000004), [Pokemon](#topic_0000000000000013), [PokemonTypeNotRecognizedException](#topic_0000000000000001), [Program](#topic_0000000000000032)

FileReader Class

The file reader is a class that handles all File IO interactions. By keeping all these methods in a single class, they can be reference anywhere in the document and still use the same set of file readers.

[System.Object](https://docs.microsoft.com/en-us/dotnet/api/system.object)

**P1.FileReader**

|  |  |
| --- | --- |
| C# |  |
| public static class FileReader | |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

Methods

[getPokemonFromFile](#topic_000000000000000A)

Enumerations

[Attributes](#topic_0000000000000007)

FileReader.getPokemonFromFile Method

Prepares a list of Pokemon from the file provided. Each line of the text file should be formatted: "{NAME}:{TYPE1},{TYPE2};"

|  |  |
| --- | --- |
| C# |  |
| public static [List](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1)<[Pokemon](#topic_0000000000000013)> getPokemonFromFile(  [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) *filename* ) | |

Parameters

filename

|  |
| --- |
| File path that contains information to populate the Pokedex with. |

Returns

A List of Pokemon where each entry is populated with a line from the text file.

Exceptions

|  |  |
| --- | --- |
| Exception type | Condition |
| [PokemonTypeNotRecognizedException](#topic_0000000000000001) | Occurs when an unrecognized type is provided |

Source code

|  |
| --- |
| public static List<Pokemon> getPokemonFromFile(string filename)  {  List<Pokemon> toReturn = new List<Pokemon>();    string[] lines = System.IO.File.ReadAllLines(filename);  foreach (string line in lines)  {  string temp = line.Substring(0, line.Length - 1);    string[] attribs = temp.Split(':');  string[] types = attribs[(int)Attributes.types].Split(',');    try  {  toReturn.Add(new Pokemon(attribs[(int)Attributes.name], (Pokemon.Type)Enum.Parse(typeof(Pokemon.Type), types[0]), (Pokemon.Type)Enum.Parse(typeof(Pokemon.Type), types[1])));  }  catch (ArgumentException)  {  throw new PokemonTypeNotRecognizedException();  }  }    return toReturn;  } |

See Also

Applies to: [FileReader](#topic_0000000000000006)

Attributes Enumeration

|  |  |  |
| --- | --- | --- |
| Constant | Value | Description |
| name | 0 | The first attribute on the line of the text file. All text from the start of the line to the first ':' |
| types | 1 | The second attribute on the line of the text file. All text between the first ':' and the last character of the line |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

See Also

Applies to: [FileReader](#topic_0000000000000006)

Pokedex Class

Container class for the list of pokemon. This class also holds supporting search methods along with methods that calculate weaknesses and resistances to moves.

[System.Object](https://docs.microsoft.com/en-us/dotnet/api/system.object)

**P1.Pokedex**

|  |  |
| --- | --- |
| C# |  |
| public class Pokedex | |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

Constructors

[Pokedex](#topic_000000000000000D)

Properties

[Dex](#topic_0000000000000012)

Methods

[getPokemonByName](#topic_0000000000000011), [getPokemonByNumber](#topic_0000000000000010), [getWeaknessVal](#topic_000000000000000F)

Fields

[dex](#topic_000000000000000C), [weaknessesAndResistances](#topic_000000000000000E)

Pokedex Constructor

Create a Pokedex using the data from a text file

|  |  |
| --- | --- |
| C# |  |
| public Pokedex(  [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) *filename* ) | |

Parameters

filename

|  |
| --- |
|  |

Source code

|  |
| --- |
| public Pokedex(string filename)  {  dex = FileReader.getPokemonFromFile(filename);  } |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

Pokedex.Dex Property

Returns the contents of the Pokedex as a List

|  |  |
| --- | --- |
| C# |  |
| public [List](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1)<[Pokemon](#topic_0000000000000013)> Dex {get;} | |

Source code

|  |
| --- |
| public List<Pokemon> Dex  {  get  {  return dex;  }  } |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

Pokedex.getPokemonByName Method

Returns a Pokemon based on its name in the Pokedex

|  |  |
| --- | --- |
| C# |  |
| public [Pokemon](#topic_0000000000000013) getPokemonByName(  [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) *s* ) | |

Parameters

s

|  |
| --- |
| Name of the Pokemon |

Returns

Pokemon object matching the provided name

Exceptions

|  |  |
| --- | --- |
| Exception type | Condition |
| [PokedexMissingEntryException](#topic_0000000000000004) | Occurs when an entry that referenced is not in the curret Pokedex |

Source code

|  |
| --- |
| public Pokemon getPokemonByName(string s)  {  for(int i = 0; i < dex.Count; i++)  {  if (dex[i].Name.Equals(s))  {  return dex[i];  }  }    throw new PokedexMissingEntryException();  } |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

Pokedex.getPokemonByNumber Method

Returns a Pokemon based on its number in the Pokedex

|  |  |
| --- | --- |
| C# |  |
| public [Pokemon](#topic_0000000000000013) getPokemonByNumber(  [int](https://docs.microsoft.com/en-us/dotnet/api/system.int32) *n* ) | |

Parameters

n

|  |
| --- |
| Pokedex Number |

Returns

Pokemon object matching the provided Pokedex number

Source code

|  |
| --- |
| public Pokemon getPokemonByNumber(int n)  {  return dex[n-1];  } |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

Pokedex.getWeaknessVal Method

The weakness of a Pokemon to a damage type.

|  |  |
| --- | --- |
| C# |  |
| public static [double](https://docs.microsoft.com/en-us/dotnet/api/system.double) getWeaknessVal(  [Pokemon](#topic_0000000000000013) *p*,  [Type](#topic_0000000000000014) *damageType* ) | |

Parameters

p

|  |
| --- |
| Pokemon to make the comparison to |

damageType

|  |
| --- |
| The type of the move used against the selected pokemon |

Returns

The damage multiplier of the attack. The value will be 0x, 0.5x, 1x, 2x, or 4x

Source code

|  |
| --- |
| public static double getWeaknessVal(Pokemon p, Pokemon.Type damageType)  {  double toReturn = 1;  toReturn \*= weaknessesAndResistances[(int)damageType][(int)p.PrimaryType];    if(p.SecondaryType != Pokemon.Type.None)  toReturn \*= weaknessesAndResistances[(int)damageType][(int)p.SecondaryType];    return toReturn;  } |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

dex Field

|  |  |
| --- | --- |
| C# |  |
| private [List](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1)<[Pokemon](#topic_0000000000000013)> dex | |

Source code

|  |
| --- |
| private List<Pokemon> dex; |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

weaknessesAndResistances Field

Weaknesses and resistances are defined by the game, this table makes referencing them much easier

|  |  |
| --- | --- |
| C# |  |
| new protected static [double](https://docs.microsoft.com/en-us/dotnet/api/system.double)[][] weaknessesAndResistances | |

Example

If you wanted to see the damage modifier for a Fire type move attacking a Grass type, it is as follows: weaknessesAndResistances[Pokemon.Type.Fire][Pokemon.Type.Grass] => 2

Source code

|  |
| --- |
| private static protected double[][] weaknessesAndResistances = new double[][]  {  new double[]{1,1,1,1,1,.5,1,0,.5,1,1,1,1,1,1,1,1,1}, //Normal  new double[]{2,1,.5,.5,1,2,.5,0,2,1,1,1,1,.5,2,1,2,.5 }, //Fighting  new double[]{1,2,1,1,1,.5,2,1,.5,1,1,2,.5,1,1,1,1,1}, //Flying  new double[]{1,1,1,.5,.5,.5,1,.5,0,1,1,2,1,1,1,1,1,5}, //Poison  new double[]{1,1,0,2,1,2,.5,1,2,2,1,.5,2,1,1,1,1,1}, //Ground  new double[]{1,.5,2,1,.5,1,2,1,.5,2,1,1,1,1,2,1,1,1}, //Rock  new double[]{1,.5,.5,.5,1,1,1,.5,.5,.5,1,2,1,2,1,1,2,.5}, //Bug  new double[]{0,1,1,1,1,11,1,2,1,1,1,1,1,2,1,1,.5,1}, //Ghost  new double[]{1,1,1,1,1,2,1,1,.5,.5,.5,1,.5,1,2,1,1,2}, ///Steel  new double[]{1,1,1,1,1,.5,2,1,2,.5,.5,2,1,1,2,.5,1,1}, //Fire  new double[]{1,1,1,1,2,2,1,1,1,2,.5,.5,1,1,1,.5,1,1}, //Water  new double[]{1,1,.5,.5,2,2,.5,1,.5,.5,2,.5,1,1,1,.5,1,1}, //Grass  new double[]{1,1,2,1,0,1,1,1,1,1,2,.5,.5,1,1,.5,1,1}, //Electric  new double[]{1,2,1,2,1,1,1,1,.5,1,1,1,1,.5,1,1,0,1}, //Psychic  new double[]{1,1,2,1,2,1,1,1,.5,.5,.5,2,1,1,.5,2,1,1}, //Ice  new double[]{1,1,1,1,1,1,1,1,.5,1,1,1,1,1,1,2,1,0}, //Dragon  new double[]{1,.5,1,1,1,1,1,2,1,1,1,1,1,2,1,1,.5,.5}, //Dark  new double[]{1,2,1,.5,1,1,1,1,.5,.5,1,1,1,1,1,2,2,1}, //Fairy  }; |

See Also

Applies to: [Pokedex](#topic_000000000000000B)

PokedexMissingEntryException Class

Exception to handle missing entries in the Pokedex

[System.Object](https://docs.microsoft.com/en-us/dotnet/api/system.object)

[System.Exception](https://docs.microsoft.com/en-us/dotnet/api/system.exception)

**P1.PokedexMissingEntryException**

|  |  |
| --- | --- |
| C# |  |
| public class PokedexMissingEntryException : [Exception](https://docs.microsoft.com/en-us/dotnet/api/system.exception) | |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

Constructors

[PokedexMissingEntryException](#topic_0000000000000005)

PokedexMissingEntryException Constructor

|  |  |
| --- | --- |
| C# |  |
| public PokedexMissingEntryException() | |

Source code

|  |
| --- |
| public PokedexMissingEntryException()  {    } |

See Also

Applies to: [PokedexMissingEntryException](#topic_0000000000000004)

Pokemon Class

The base Pokemon class that defines each entry. This class holds an enum for the Pokemon class, variables to hold the Pokemon name and types, and some overriden methods to make comparisons easier.

[System.Object](https://docs.microsoft.com/en-us/dotnet/api/system.object)

**P1.Pokemon**

|  |  |
| --- | --- |
| C# |  |
| public class Pokemon | |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

Constructors

[Pokemon](#topic_000000000000002B)

Properties

[Name](#topic_000000000000002C), [PrimaryType](#topic_000000000000002D), [SecondaryType](#topic_000000000000002E)

Methods

[Equality](#topic_000000000000002F), [Inequality](#topic_0000000000000030), [ToString](#topic_0000000000000031)

Enumerations

[Type](#topic_0000000000000014)

Fields

[name](#topic_0000000000000028), [primaryType](#topic_0000000000000029), [secondaryType](#topic_000000000000002A)

Pokemon Constructor

Creates a Pokemon. Pokemon can have either one or two types, the second type is optional in the constructor

|  |  |
| --- | --- |
| C# |  |
| public Pokemon(  [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) *name*,  [Type](#topic_0000000000000014) *primaryType*,  [Type](#topic_0000000000000014) *secondaryType* = Type.None ) | |

Parameters

name

|  |
| --- |
| Name of the Pokemon |

primaryType

|  |
| --- |
| The primary type of the Pokemon |

secondaryType

|  |
| --- |
| The secondary type of the Pokemon, if it has one |

Source code

|  |
| --- |
| public Pokemon(string name, Type primaryType, Type secondaryType = Type.None)  {  this.name = name;  this.primaryType = primaryType;  this.secondaryType = secondaryType;    } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Pokemon.Name Property

Name of the Pokemon

|  |  |
| --- | --- |
| C# |  |
| public [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) Name {get; set;} | |

Source code

|  |
| --- |
| public string Name  {  get  {  return name;  }  set  {  name = value;  }  } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Pokemon.PrimaryType Property

Primary type of the Pokemon

|  |  |
| --- | --- |
| C# |  |
| public [Type](#topic_0000000000000014) PrimaryType {get; set;} | |

Source code

|  |
| --- |
| public Type PrimaryType  {  get  {  return primaryType;  }  set  {  primaryType = value;  }  } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Pokemon.SecondaryType Property

Secondary type of the Pokemon, if it has one

|  |  |
| --- | --- |
| C# |  |
| public [Type](#topic_0000000000000014) SecondaryType {get; set;} | |

Source code

|  |
| --- |
| public Type SecondaryType  {  get  {  return secondaryType;  }  set  {  secondaryType = value;  }  } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Pokemon.Equality Method

See's if a Pokemon's Name and Type(s) match

|  |  |
| --- | --- |
| C# |  |
| public static [bool](https://docs.microsoft.com/en-us/dotnet/api/system.boolean) operator ==(  [Pokemon](#topic_0000000000000013) *a*,  [Pokemon](#topic_0000000000000013) *b* ) | |

Parameters

a

|  |
| --- |
| Pokemon To Compare |

b

|  |
| --- |
| Pokemon To Compare |

Returns

True if the pokemon name and type(s) match. False otherwise

Source code

|  |
| --- |
| public static bool operator ==(Pokemon a, Pokemon b)  {  if (a.Name.Equals(b.Name) && ((a.PrimaryType == b.PrimaryType && a.SecondaryType == b.SecondaryType) || (a.PrimaryType == b.SecondaryType && a.SecondaryType == b.PrimaryType)))  return true;  else  return false;  } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Pokemon.Inequality Method

See's if a Pokemon's Name and Type(s) match

|  |  |
| --- | --- |
| C# |  |
| public static [bool](https://docs.microsoft.com/en-us/dotnet/api/system.boolean) operator !=(  [Pokemon](#topic_0000000000000013) *a*,  [Pokemon](#topic_0000000000000013) *b* ) | |

Parameters

a

|  |
| --- |
| Pokemon To Compare |

b

|  |
| --- |
| Pokemon To Compare |

Returns

False if the pokemon name and type(s) match. True otherwise

Source code

|  |
| --- |
| public static bool operator !=(Pokemon a, Pokemon b)  {  if (a.Name.Equals(b.Name) && ((a.PrimaryType == b.PrimaryType && a.SecondaryType == b.SecondaryType) || (a.PrimaryType == b.SecondaryType && a.SecondaryType == b.PrimaryType)))  return false;  else  return true;  } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Pokemon.ToString Method

Prints the Pokemon's name and Type(s)

|  |  |
| --- | --- |
| C# |  |
| public override [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) ToString() | |

Returns

Formatted string containing the Pokemon's Name and Type(s)

Source code

|  |
| --- |
| public override string ToString()  {  string toReturn = "Pokemon: " + Name + "\n";  if (primaryType == Type.None || secondaryType == Type.None)  {  toReturn += "Type: " + primaryType.ToString() + "\n";  }  else  {  toReturn += "Types: " + primaryType.ToString() + "-" + secondaryType.ToString() + "\n";  }    return toReturn;  } |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

Type Enumeration

A simple enum to keep track of Pokemon types

|  |  |  |
| --- | --- | --- |
| Constant | Value | Description |
| Normal | 0 | Normal Type |
| Fighting | 1 | Fighting Type |
| Flying | 2 | Flying Typep |
| Poison | 3 | Poison Type |
| Ground | 4 | Ground Type |
| Rock | 5 | Rock Type |
| Bug | 6 | Bug Type |
| Ghost | 7 | Ghost Type |
| Steel | 8 | Steel Type |
| Fire | 9 | Fire Type |
| Water | 10 | Water Type |
| Grass | 11 | Grass Type |
| Electric | 12 | Electric Type |
| Psychic | 13 | Psychic Type |
| Ice | 14 | Ice Type |
| Dragon | 15 | Dragon Typpe |
| Dark | 16 | Dark Type |
| Fairy | 17 | Faiy Type |
| None | 18 | No Type |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

See Also

Applies to: [Pokemon](#topic_0000000000000013)

name Field

|  |  |
| --- | --- |
| C# |  |
| private [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) name | |

Source code

|  |
| --- |
| private string name; |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

primaryType Field

|  |  |
| --- | --- |
| C# |  |
| private [Type](#topic_0000000000000014) primaryType | |

Source code

|  |
| --- |
| private Type primaryType; |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

secondaryType Field

|  |  |
| --- | --- |
| C# |  |
| private [Type](#topic_0000000000000014) secondaryType | |

Source code

|  |
| --- |
| private Type secondaryType; |

See Also

Applies to: [Pokemon](#topic_0000000000000013)

PokemonTypeNotRecognizedException Class

Exception to handle invalid Pokemon types

[System.Object](https://docs.microsoft.com/en-us/dotnet/api/system.object)

[System.Exception](https://docs.microsoft.com/en-us/dotnet/api/system.exception)

**P1.PokemonTypeNotRecognizedException**

|  |  |
| --- | --- |
| C# |  |
| public class PokemonTypeNotRecognizedException : [Exception](https://docs.microsoft.com/en-us/dotnet/api/system.exception) | |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

Constructors

[PokemonTypeNotRecognizedException](#topic_0000000000000002)

PokemonTypeNotRecognizedException Constructor

|  |  |
| --- | --- |
| C# |  |
| public PokemonTypeNotRecognizedException() | |

Source code

|  |
| --- |
| public PokemonTypeNotRecognizedException()  {    } |

See Also

Applies to: [PokemonTypeNotRecognizedException](#topic_0000000000000001)

PokemonTypeNotRecognizedException(String) Constructor

|  |  |
| --- | --- |
| C# |  |
| public PokemonTypeNotRecognizedException(  [string](https://docs.microsoft.com/en-us/dotnet/api/system.string) *type* ) | |

Parameters

type

|  |
| --- |
|  |

Source code

|  |
| --- |
| public PokemonTypeNotRecognizedException(string type) : base("\"" + type + "\" is not a valid Pokemon Type")  {    } |

See Also

Applies to: [PokemonTypeNotRecognizedException](#topic_0000000000000001)

Program Class

[System.Object](https://docs.microsoft.com/en-us/dotnet/api/system.object)

**P1.Program**

|  |  |
| --- | --- |
| C# |  |
| internal class Program | |

Requirements

**Namespace:**[P1](#topic_0000000000000000)

**Assembly:** P1 (in P1.exe)

Methods

[Main](#topic_0000000000000035), [damageMenu](#topic_000000000000003B), [getInput](#topic_0000000000000037), [pokemonMenu](#topic_000000000000003A), [printMainMenu](#topic_0000000000000036), [searchByName](#topic_0000000000000039), [searchByNumber](#topic_0000000000000038)

Fields

[go](#topic_0000000000000034), [kanto](#topic_0000000000000033)

Program.Main Method

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) Main(  [string](https://docs.microsoft.com/en-us/dotnet/api/system.string)[] *args* ) | |

Parameters

args

|  |
| --- |
|  |

Source code

|  |
| --- |
| static void Main(string[] args)  {  kanto = new Pokedex(@"D:\Users\arobi\Desktop\pokemondatabase.txt");    while (go)  {  printMainMenu();  getInput();  Console.Clear();  }  } |

See Also

Applies to: [Program](#topic_0000000000000032)

Program.damageMenu Method

Handles the damage menu

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) damageMenu(  [Pokemon](#topic_0000000000000013) *p* ) | |

Parameters

p

|  |
| --- |
| Pokemon returned from search |

Source code

|  |
| --- |
| private static void damageMenu(Pokemon p)  {  Console.Clear();  Console.Write(p);  Console.WriteLine("--------------------------------------------");  Console.WriteLine("What kind of damage would you like to do?");  for (int i = 0; i < (int)Pokemon.Type.None; i++)  {  Console.WriteLine((i + 1) + "). " + (Pokemon.Type)i);  }    int testType;  try  {  testType = int.Parse(Console.ReadLine());    if (testType < 0 || testType > 18)  throw new IndexOutOfRangeException();    Console.WriteLine(p.Name + "is affected " + Pokedex.getWeaknessVal(p, (Pokemon.Type)(testType - 1)) + "x by " + ((Pokemon.Type)(testType - 1)).ToString() + " type moves.");  }  catch (FormatException)  {  Console.Clear();  Console.WriteLine("That is an invalid input");  }  catch (IndexOutOfRangeException)  {  Console.Clear();  Console.WriteLine("Please select a value between 1 and 18");  }  finally  {  Console.WriteLine("Press Enter to Continue");  Console.Read();  }  } |

See Also

Applies to: [Program](#topic_0000000000000032)

Program.getInput Method

Gets the input for the main menu

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) getInput() | |

Source code

|  |
| --- |
| private static void getInput()  {  string input = Console.ReadLine();    switch (input)  {  case "1":  searchByNumber();  break;  case "2":  searchByName();  break;  case "3":  go = false;  break;  default:  break;  }  } |

See Also

Applies to: [Program](#topic_0000000000000032)

Program.pokemonMenu Method

Handles the menu once a Pokemon has been searched for

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) pokemonMenu(  [Pokemon](#topic_0000000000000013) *p* ) | |

Parameters

p

|  |
| --- |
| Pokemon returned from search |

Source code

|  |
| --- |
| private static void pokemonMenu(Pokemon p)  {  while (true)  {  Console.Clear();  Console.Write(p);  Console.WriteLine("--------------------------------------------");  Console.WriteLine("1.) Damage Type Calculation");  Console.WriteLine("2.) Main Menu");    string input = Console.ReadLine();    switch (input)  {  case "1":  damageMenu(p);  break;  case "2":  return;  default:  return;  }  }  } |

See Also

Applies to: [Program](#topic_0000000000000032)

Program.printMainMenu Method

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) printMainMenu() | |

Source code

|  |
| --- |
| private static void printMainMenu()  {  Console.WriteLine("Welcome to the Kanto Pokedex!\nPlease choose from the following options:");  Console.WriteLine("--------------------------------------------");  Console.WriteLine("1.) Search By Number");  Console.WriteLine("2.) Search by Name");  Console.WriteLine("3.) Exit");  } |

See Also

Applies to: [Program](#topic_0000000000000032)

Program.searchByName Method

Handles the user wanting to serach by the Pokemon name

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) searchByName() | |

Source code

|  |
| --- |
| private static void searchByName()  {  Console.Clear();  Console.WriteLine("Enter the name of your desired Pokemon:");  string toSearch;  try  {  toSearch = Console.ReadLine();  pokemonMenu(kanto.getPokemonByName(toSearch));  }  catch (PokedexMissingEntryException)  {  Console.Clear();  Console.WriteLine("That name is not in the Pokedex");  }  finally  {  Console.WriteLine("Press Enter to Continue");  Console.Read();    }  } |

See Also

Applies to: [Program](#topic_0000000000000032)

Program.searchByNumber Method

Handles the user wanting to search by the Pokedex number

|  |  |
| --- | --- |
| C# |  |
| private static [void](https://docs.microsoft.com/en-us/dotnet/api/system.void) searchByNumber() | |

Source code

|  |
| --- |
| private static void searchByNumber()  {  Console.Clear();  Console.WriteLine("Enter the number of your desired Pokemon:");  int toSearch;  try  {  toSearch = int.Parse(Console.ReadLine());    if (toSearch < 0 || toSearch > 151)  throw new IndexOutOfRangeException();    pokemonMenu(kanto.getPokemonByNumber(toSearch));  }  catch (FormatException)  {  Console.Clear();  Console.WriteLine("That is an invalid input");  }  catch (IndexOutOfRangeException)  {  Console.Clear();  Console.WriteLine("Please select a value between 1 and 151");  }  finally  {  Console.WriteLine("Press Enter to Continue");  Console.Read();  }      } |

See Also

Applies to: [Program](#topic_0000000000000032)

go Field

|  |  |
| --- | --- |
| C# |  |
| private static [bool](https://docs.microsoft.com/en-us/dotnet/api/system.boolean) go | |

Source code

|  |
| --- |
| private static bool go = true; |

See Also

Applies to: [Program](#topic_0000000000000032)

kanto Field

|  |  |
| --- | --- |
| C# |  |
| private static [Pokedex](#topic_000000000000000B) kanto | |

Source code

|  |
| --- |
| private static Pokedex kanto; |

See Also

Applies to: [Program](#topic_0000000000000032)

# Index

[Attributes Enumeration](#topic_0000000000000007)

[Dex Property](#topic_0000000000000012)

[Equality Operator](#topic_000000000000002F)

[FileReader Class](#topic_0000000000000006)

[Inequality Operator](#topic_0000000000000030)

[Main Method](#topic_0000000000000035)

[Name Property](#topic_000000000000002C)

[P1 Namespace](#topic_0000000000000000)

[P1 Reference](#topic_000000000000003C)

[Pokedex Class](#topic_000000000000000B)

[Pokedex Constructor](#topic_000000000000000D)

[PokedexMissingEntryException Class](#topic_0000000000000004)

[PokedexMissingEntryException Constructor](#topic_0000000000000005)

[Pokemon Class](#topic_0000000000000013)

[Pokemon Constructor](#topic_000000000000002B)

[PokemonTypeNotRecognizedException Constructor](#topic_0000000000000002)

[PokemonTypeNotRecognizedException (String) Constructor](#topic_0000000000000003)

[PokemonTypeNotRecognizedException Class](#topic_0000000000000001)

[PrimaryType Property](#topic_000000000000002D)

[Program Class](#topic_0000000000000032)

[SecondaryType Property](#topic_000000000000002E)

[ToString Method](#topic_0000000000000031)

[Type Enumeration](#topic_0000000000000014)

[damageMenu Method](#topic_000000000000003B)

[dex Field](#topic_000000000000000C)

[getInput Method](#topic_0000000000000037)

[getPokemonByName Method](#topic_0000000000000011)

[getPokemonByNumber Method](#topic_0000000000000010)

[getPokemonFromFile Method](#topic_000000000000000A)

[getWeaknessVal Method](#topic_000000000000000F)

[go Field](#topic_0000000000000034)

[kanto Field](#topic_0000000000000033)

[name Field](#topic_0000000000000028)

[pokemonMenu Method](#topic_000000000000003A)

[primaryType Field](#topic_0000000000000029)

[printMainMenu Method](#topic_0000000000000036)

[searchByName Method](#topic_0000000000000039)

[searchByNumber Method](#topic_0000000000000038)

[secondaryType Field](#topic_000000000000002A)

[weaknessesAndResistances Field](#topic_000000000000000E)