Learn C#: Intro to Classes (Abstraction)

C# Class Instance

In C#, an object is an *instance* of a class. An object can be created from a class using the `new` keyword.

```
Burger cheeseburger = new Burger();
// If a class is a recipe, then an object is a single meal made from that recipe.

House tudor = new House();
// If a class is a blueprint, then an object is a house made from that blueprint.
```

C# Members

In C#, a class contains *members*, which define the kind of data stored in a class and the behaviors a class can perform.

```
class Program {
  static void Main(string[] args)
  {
    string greeting = "hello";

    // Console is a class
    // and WriteLine is one of its
members
    Console.WriteLine(greeting.Length);
// Prints 5

    Forest f = new Forest("Amazon
Rainforest");

    // access members using dot "."
notation
    f.AnnounceForest();
    // Prints "This forest is known as
the Amazon Rainforest!"
  }
}

class Forest
{
  public string name;
  public Forest(string name)
  {
    this.name = name;
  }

  public AnnounceForest()
  {
    Console.WriteLine($"This forest is
known as the {name}!")
  }
}
```

C# this Keyword

In C#, the `this` keyword refers to the current instance of a class.

```
// We can use the this keyword to refer to the current class's members hidden by similar names:
public NationalPark(int area, string state)
{
  this.area = area;
  this.state = state;
}

// The code below requires duplicate code, which can lead to extra work and errors when changes are needed:
public NationalPark(int area, string state)
{
  area = area;
  state = state;
}
public NationalPark(int area)
{
  area = area;
  state = "Unknown";
}

// When "this" is used as a method
// it instructs one constructor to call another
// the following achieves the same as the preceding 2 constructors
public NationalPark(int area) : this(are, "Unknown")
{ }
```

C# Field

In C#, a *field* stores a piece of data within an object. It acts like a variable and may have a different value for each instance of a type.

```
public class Person
{
   public string firstName;
   public string lastName;
}

// In this example, firstName and
lastName are fields of the Person class.
```

C# Parameterless Constructor

In C#, if no constructors are specified in a class, the compiler automatically creates a parameterless constructor.

```
public class Freshman
{
  public string firstName;
}

public static void Main (string[] args)
{
  Freshman f = new Freshman();
  // name is null
  string name = f.firstName;
}

// In this example, no constructor is
defined in Freshman, but a parameterless
constructor is still available for use in
Main(). All fields will be set to a
default value according to their type and
remain unchanged until updated manually.
```

C# Constructor

In C#, whenever an instance of a class is created, its *constructor* is called. It must have the same name as the enclosing class. Like other methods, a constructor can be overloaded. This is useful when you may want to define an additional constructor that takes a different number of arguments.

```
// Takes two arguments
public Forest(int area, string country)
{
  this.area = area;
  this.country = country;
}

// Takes one argument
public Forest(int area)
{
  this.area = area;
  this.country = "Unknown";
}
```

```
// Typically, a constructor is used to
set initial values and run any code
needed to "set up" an instance.

// A constructor looks like a method, but
it does not include a return type and it
must have the same name as the enclosing
type.
```

C# Classes

In C#, *classes* are used to create custom types. The class defines the kinds of information and methods included in a custom type.

Abstraction

In programming, the process of pulling related data and methods into a logical, reusable implementation is called *abstraction*. In C#, classes are frequently used for abstraction, by modeling real-world objects based on their attributes and behaviors.

Print Share