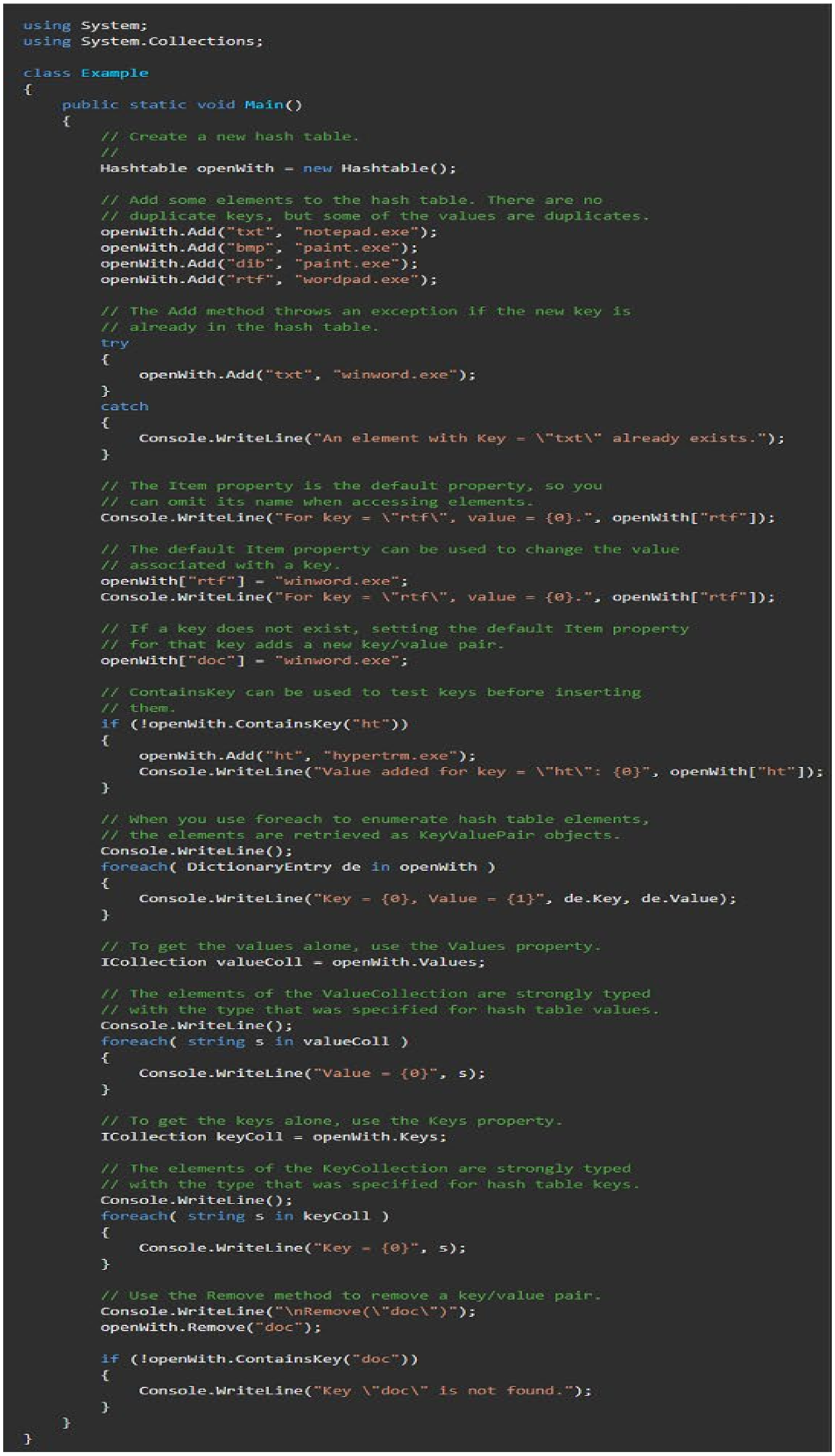
Hash Table

A hash table is a data structure where data is stored in key/value pairs. The hash table uses a hash function, which is a mathematical operation that computes an index from the key. A corresponding value to the key is then stored in that index. The value can be accessed and manipulated through the assigned key. An advantage of using a hash table is that it is very efficient for accessing, adding, and deleting data. A disadvantage is that a hash function can sometimes generate the same index for two different keys, resulting in a collision. There are various ways to handle a collision. One way to accommodate collisions is to have each index or "bucket" be a linked list that can store data elements that happen to have the same hash code. The following example shows how to initialize a hash table in C# along with some common methods.



```csharp
using System;
using System.Collections;

class Example
{
    public static void Main()
    {
        // Create a new hash table.
        //
        Hashtable openWith = new Hashtable();

        // Add some elements to the hash table. There are no
        // duplicate keys, but some of the values are duplicates.
        openWith.Add("txt", "notepad.exe");
        openWith.Add("bmp", "paint.exe");
        openWith.Add("dib", "paint.exe");
        openWith.Add("rtf", "wordpad.exe");

        // The Add method throws an exception if the new key is
        // already in the hash table.
        try
        {
            openWith.Add("txt", "winword.exe");
        }
        catch
        {
            Console.WriteLine("An element with Key = \"txt\" already exists.");
        }

        // The Item property is the default property, so you
        // can omit its name when accessing elements.
        Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

        // The default Item property can be used to change the value
        // associated with a key.
        openWith["rtf"] = "winword.exe";
        Console.WriteLine("For key = \"rtf\", value = {0}.", openWith["rtf"]);

        // If a key does not exist, setting the default Item property
        // for that key adds a new key/value pair.
        openWith["doc"] = "winword.exe";

        // ContainsKey can be used to test keys before inserting
        // them.
        if (!openWith.ContainsKey("ht"))
        {
            openWith.Add("ht", "hypertrm.exe");
            Console.WriteLine("Value added for key = \"ht\": {0}", openWith["ht"]);
        }

        // When you use foreach to enumerate hash table elements,
        // the elements are retrieved as KeyValuePair objects.
        Console.WriteLine();
        foreach( DictionaryEntry de in openWith )
        {
            Console.WriteLine("Key = {0}, Value = {1}", de.Key, de.Value);
        }

        // To get the values alone, use the Values property.
        ICollection valueColl = openWith.Values;

        // The elements of the ValueCollection are strongly typed
        // with the type that was specified for hash table values.
        Console.WriteLine();
        foreach( string s in valueColl )
        {
            Console.WriteLine("Value = {0}", s);
        }

        // To get the keys alone, use the Keys property.
        ICollection keyColl = openWith.Keys;

        // The elements of the KeyCollection are strongly typed
        // with the type that was specified for hash table keys.
        Console.WriteLine();
        foreach( string s in keyColl )
        {
            Console.WriteLine("Key = {0}", s);
        }

        // Use the Remove method to remove a key/value pair.
        Console.WriteLine("\nRemove(\"doc\")");
        openWith.Remove("doc");

        if (!openWith.ContainsKey("doc"))
        {
            Console.WriteLine("Key \"doc\" is not found.");
        }
    }
}
```