****

Software Development: Data Structures

(H16Y 35)

Outcomes 2 (part 2), 3 and 4, Exercise 4

Source code – use as a reference / crib

using System;

using System.Collections;

using System.Collections.Generic;

//using System.ComponentModel;

//using System.Data;

//using System.Drawing;

//using System.Linq;

using System.Text;

//using System.Threading.Tasks;

using System.Windows.Forms;

namespace Ex4StudentHashtable

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

// \*\*\*\*\*\*\*

// TASK 1 – insert code here to

// Declare and create an appropriate instance variable (a Hashtable object

// called StudentNames) to store the student details as a key / value pair

// \*\*\*\*\*\*\*

private Hashtable StudentNames = new Hashtable(); // create Hashtable object

private void btnAdd\_Click(object sender, EventArgs e) // add last name (the key) and Student object to hashtable

{

/\* This event handler reads the first name and last name of a student

from the user interface, creates an

object of class Student and inserts into the Hashtable

This method receives two arguments - a key object, and a value object

In this example, the key is the last name of the Student (a string),

and the value is the corresponding Student object

An ArgumentException is thrown if the Hashtable already contains the key

\*/

Student student = new Student(txtFirstName.Text, txtSurname.Text);

// \*\*\*\*\*\*\*

// TASK 2, ADD AN ENTRY – insert code here to

// \* Create a new Hashtable entry

// \* Inform the user if the key already exists

// \* Clear text boxes as appropriate

// \*\*\*\*\*\*\*

lblDisplay.Text = "";

try // add new key/value pair

{

StudentNames.Add(txtSurname.Text, student);

lblStatus.Text = student.ToString() + " object added to hashtable, with default hash function returning " + txtSurname.Text.GetHashCode().ToString();

txtFirstName.Text = txtSurname.Text = "";

}

catch (ArgumentException argumentException) // if key already in table, output message

{

{

lblStatus.Text = argumentException.ToString();

}

}

}

private void btnFind\_Click(object sender, EventArgs e)

{

/\* This event handler retrieves the object associated with a specific key,

using the Hashtable’s subscript operator

The expression in square brackets is the key for which the Hashtable

should return the corresponding object

If the key is not found, the result is null

\*/

lblDisplay.Text = "";

object result = StudentNames[txtSurname.Text]; // Known as 'boxing'

// Boxing is the act of converting a value type into the System.Object (alias object) reference

type

// \*\*\*\*\*\*\*

// TASK 3, FIND AN ENTRY – insert code here to

// \* Find an existing Hashtable entry and inform the user accordingly

// \*\*\*\*\*\*\*

if (result != null)

lblStatus.Text = result.ToString() + " object found";

else

lblStatus.Text = txtSurname.Text + " key not found";

}

private void btnRemove\_Click(object sender, EventArgs e)

{

/\* This event handler invokes Hashtable method Remove to delete a key

and its associated object from the Hashtable

If the key does not exist in the table, nothing happens

\*/

lblDisplay.Text = "";

// \*\*\*\*\*\*\*

// TASK 4, DELETE AN ENTRY – insert code here to

// \* Delete a current entry associated with a key provided

// \* Clear the text boxes as appropriate

// \*\*\*\*\*\*\*

StudentNames.Remove(txtSurname.Text);

lblStatus.Text = "Object at key " + txtSurname.Text + " removed from hashtable";

txtFirstName.Text = txtSurname.Text = "";

}

private void btnRemoveAll\_Click(object sender, EventArgs e)

{

/\* This event handler invokes Hashtable method Clear()

to delete all Hashtable entries

\*/

if (MessageBox.Show("Are you sure you wish to delete all the hashtable entries?", "Confirm Delete", MessageBoxButtons.YesNo) == DialogResult.Yes)

{

// \*\*\*\*\*\*\*

// TASK 5, DELETE ALL ENTRIES – insert code here to

// \* Remove all entries in the collection

// \* Clear text boxes as appropriate

// \*\*\*\*\*\*\*

StudentNames.Clear();

lblDisplay.Text = "";

lblStatus.Text = "The hashtable has been emptied";

txtFirstName.Text = txtSurname.Text = "";

}

}

private void btnDisplayObjects\_Click(object sender, EventArgs e)

{

/\* Class Hashtable provides method GetEnumerator that returns an enumerator

of type IDictionaryEnumerator (which derives from IEnumerator)

Such enumerators provide properties Key and Value to access the information

for a key/value pair

This event handler uses the Value property of the enumerator to output

the objects in the Hashtable

This event handler also uses Hashtable's Count property to establish whether

the Hashtable is empty (i.e. Count is zero)

\*/

if (StudentNames.Count == 0)

{

lblStatus.Text = "The hashtable is empty";

}

else

{

// \*\*\*\*\*\*\*

// TASK 6, DISPLAY ALL ENTRIES – insert code here to

// Display all of the objects currently in the Hashtable

// by iterating through the collection and appending each value

// \*\*\*\*\*\*\*

/\* IDictionaryEnumerator enumerator = StudentNames.GetEnumerator();

StringBuilder buffer = new StringBuilder();

while (enumerator.MoveNext())

buffer.Append(enumerator.Value + "\r\n");

lblDisplay.Text = buffer.ToString();

\*/

// An alternative approach - using C#'s foreach() control structure

foreach(DictionaryEntry student in StudentNames)

lblDisplay.Text += student.Value + "\n"; }

}

private void btnQuit\_Click(object sender, EventArgs e)

{

Application.Exit();

}

}

}