KF7012 Week 5 MVP lab

Good morning everybody, today we will be looking at implementing the theory we covered in this week's lecture.  An important concept regarding patterns in the presentation layer of an application was covered but we need to really put into practice what we discussed in the lecture to make any real sense of the theory. Additionally you are expected to be able to do this on the assignment therefore it is vital that you fully versed in the application of the pattern. The Solution is available both as a project and within the appendix of this document. DO NOT go to the solution unless you have spent significant time trying to complete the exercise. You will learn nothing if you take the easy route and look at the solution before attempting the work. If you are stuck ask someone! If they cannot help only only look at the part of the solution you’re having problems with. Do not use it as an excuse not to do the work.

Separation of concern is at the heart of this pattern and it will significantly enhance the ability to be able to unit test the application in its entity, interfaces have always been problematic to test and often were not tested well, due to the cost involved in testing. Manual testing of interfaces therefore does not support agile, we need techniques which support incremental depelopment and full regression testing. Manual testing is too slow and expensive!

Step 1 –

Get the zip file called ApplicationNonMVP, unzip the application and run it. Before you delve into the code get a clear understanding of the external behavior of the application.



Application background

The application was written for a little bit of fun a few years ago. It is very simple in that it does not require external data sources, it is only one single screen and has few controls, much of the logic is already placed within a model. Therefore it is ideal for the proposes of our lab, in that it should be straight forward to implement MVP and make the form passive.

The application shows time complexity of random character passwords. Note these are many order of magnitude more difficult to crack than ones which people choose for themselves which are normally based on words and numbers and changes to case!

The number of unique values for is biased on length and available number of characters. For example if we can only use lower case letters then and we have to have a password of three characters then the possible combinations is 26 \* 26 \* 26 or 263 = 17,576 unique combinations. If we brute force the password it should be found on average once we have gone though half the possible values or 8,788 = (263/2)

Hex values have only 0-F for each possible character. Therefore cannot be used in conjunction with the others.

Step 2 –

Now you have a clear understanding of the applications behavior it is time to look at the code which underpins the functionality.

The MODEL

The project uses a class which was developed independently of the project. DoubleConverter is needed because of the way that large numbers are displayed in the most programming languages. For example 1.2345E-02 would mean 0.012345. This is a very nice handy piece of code that was used to format the number of years. For example some passwords would take far longer than the life of the universe in years to crack. We do not want the result to be displayed in scientific notation, or a poor representation with lots of trailing zeros, so the class was used to covert accurately.

KeyGen.cs also contains the class TimeToCrack which is acting like a struct, in that the data is publically accessible. An enumeration keySpace is used to represent all the different types of key types which can be used such as lowercase letters, uppercase letters etc. The class KeyGen is main model class which the form interacts with. It uses static constant arrays to hold the each of the key spaces. The constructor is used in order to set the key space, it can be passed in up to 4 key spaces via its overloaded constructor, if there are two or more keyspaces and one is Hex then an exception is generated. This is purely defensive and there so be no need to catch this exception within this simple application as it just stops the generator being placed into an inconsistent state.

The FORM

The form holds and instantiates the model, this is not a multi user application so this is ‘kind’ of ok for a naive application.

The load event uses the model to set up the initial state of the components for the attempts per second and the key length. It also then sets the check boxes. It then uses a method which is called when the model updates updateView()

updateView()uses the values which have just been placed into the test boxes as parameters into the KeyGen, and calls the method to generate the key and also get the time complexity. It then uses this to update the time and key values for the GUI.

Event for clicking the generate key button, calls updateView() to create a new key.

KeyLength text box has a leave event which needs to validate that a number has been entered. It then passes the new keylengh to the model and once again calls updateView().

ProcessCheckBoxes() is called ever time there is an event on a check box. A list of the enumeration keySpace is built. A check is made to make sure that if the HEX checkbox is selected then others have not also been selected. If none are selected key and duration are set to empty strings. The list of keySpace is used to create a new KeyGen object and the updateView() is called again.

Step 3 –

Create an interface which will allow you to control the form passively. You need to look at what components you need to read and also which need to be read and set. The solution has an interface called IKeyGUI

SEE APPENDIX A for the Solution to this part. But please try this first else you will never learn how to do it yourself.

Step 4 - This will be the harder part you how new to create a presenter class you can call this what you want however the one used in the solution is called KeyGenPresenter.

The class diagram for MVP pattern has the Presenter between the GUI and the Model. And that the Presenter must have references to each of these. Since we created an interface for the GUI it should be the interface which you are referencing.

Think about this interface and how you are going to replicate the same behavior you have in the form. So start with the Load event. Look at the method in the form and determine how you are going to implement this within the presenter.

It should look something like this.

Original code from the Form

txt\_keysPerSecond.Text = keygen.getAttempsPerSecond().ToString();

txt\_keyLength.Text = keygen.getKeyLength().ToString();

chk\_hex.Checked = true;

chk\_LowerCase.Checked = false;

chk\_number.Checked = false;

chk\_uppercase.Checked = false;

chk\_otherchars.Checked = false;

updateView();

New code from within the Presenter. Note Screen data type is IKeyGUI

screen.setKeysPerSecond(keygen.getAttempsPerSecond().ToString());

screen.setKeyLength(keygen.getKeyLength().ToString());

screen.hexOnly = true;

screen.lowercase = false;

screen.number = false;

screen.otherChars = false;

screen.uppercase = false;

updateView();

The majority of the code you have in the Form can be easily refactored to work in the Presenter.

Step 5

Alter the GUI you need to make the form so that it implements the IKeyGUI interface. Events need to be sent to the presenter, you will have to create suitable method names in order to deal with the events in the presenter.

The solutions for the Presenter and the Form are in Appendix B

Step 6

Depending on how you attempted the exercise you will most likely have to modify the entry point into the application

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

**// Application.Run(new Form1()); //removed**

**Form1 f1 = new Form1(); //added**

**KeyGenPresenter kgp = new KeyGenPresenter(f1); //added**

**Application.Run(f1); //added**

}

}

}

APPENDIX A

If you are looking here and you have not really tried to do the work then you will not learn!

Solutions are useful only to validate and check. Not to find out how it was done, that’s what example are for.

public interface IKeyGUI

{

Boolean lowercase { set; get; }

Boolean uppercase { set; get; }

Boolean otherChars { set; get; }

Boolean number { set; get; }

Boolean hexOnly { set; get; }

void setNewKey(String key);

void setYears(String years);

void setDays(String days);

void setHours(String hours);

void setSeconds(String seconds);

void setKeysPerSecond(String s);

String getKeysPerSecond();

void setKeyLength(String keyLength);

String getKeyLength();

void register(KeyGenPresenter KGP);

void message(String message);

}

Appendix **B**

public class KeyGenPresenter

{

private KeyGen keygen;

private IKeyGUI screen;

public KeyGenPresenter(IKeyGUI screen)

{

keygen = new KeyGen();

this.screen = screen;

screen.register(this);

intialiseForm();

}

private void intialiseForm()

{

screen.setKeysPerSecond(keygen.getAttempsPerSecond().ToString());

screen.setKeyLength(keygen.getKeyLength().ToString());

screen.hexOnly = true;

screen.lowercase = false;

screen.number = false;

screen.otherChars = false;

screen.uppercase = false;

updateView();

}

public void chk\_LowerCase\_CheckedChanged()

{

ProcessCheckBoxes();

}

public void chk\_uppercase\_CheckedChanged()

{

ProcessCheckBoxes();

}

public void chk\_number\_CheckedChanged()

{

ProcessCheckBoxes();

}

public void chk\_otherchars\_CheckedChanged()

{

ProcessCheckBoxes();

}

public void chk\_hex\_CheckedChanged()

{

ProcessCheckBoxes();

}

private void ProcessCheckBoxes()

{

List<keySpace> selectedKeySpace = new List<keySpace>();

if (screen.lowercase)

selectedKeySpace.Add(keySpace.lowercase);

if (screen.number)

selectedKeySpace.Add(keySpace.numbers);

if (screen.uppercase)

selectedKeySpace.Add(keySpace.uppercase);

if (screen.otherChars)

selectedKeySpace.Add(keySpace.otherChars);

if (screen.hexOnly)

{

if (selectedKeySpace.Count != 0)

{

screen.message("Hex cannot be selecte with other\n Removing Hex");

screen.hexOnly = false;

}

else

{

selectedKeySpace.Add(keySpace.hex);

}

}

if (selectedKeySpace.Count == 0)

{

screen.setNewKey("");

screen.setYears("");

screen.setDays("");

screen.setHours("");

screen.setSeconds("");

}

else if (selectedKeySpace.Count == 1)

keygen = new KeyGen(selectedKeySpace[0]);

else if (selectedKeySpace.Count == 2)

keygen = new KeyGen(selectedKeySpace[0], selectedKeySpace[1]);

else if (selectedKeySpace.Count == 3)

keygen = new KeyGen(selectedKeySpace[0], selectedKeySpace[1], selectedKeySpace[2]);

else if (selectedKeySpace.Count == 4)

keygen = new KeyGen(selectedKeySpace[0], selectedKeySpace[1], selectedKeySpace[2], selectedKeySpace[3]);

else

screen.message("Error: processing the selected key type");

if (selectedKeySpace.Count != 0)

updateView();

}

public void updateView()

{

keygen.setAttempsPerSecond(UInt64.Parse(screen.getKeysPerSecond().Trim()));

keygen.setKeyLength(int.Parse(screen.getKeyLength().Trim()));

screen.setNewKey(keygen.getNewKey());

TimeToCrack timeToCrack = keygen.timeToCrack();

screen.setYears(timeToCrack.years);

screen.setDays(timeToCrack.days);

screen.setHours(timeToCrack.hours);

screen.setSeconds(timeToCrack.seconds);

}

public void txt\_keysPerSecond\_Leave()

{

string text = screen.getKeysPerSecond().Trim();

UInt64 attempsPerSec = 0;

if (UInt64.TryParse(text, out attempsPerSec))

{

keygen.setAttempsPerSecond(attempsPerSec);

updateView();

}

else

{

attempsPerSec = keygen.getAttempsPerSecond();

screen.setKeysPerSecond(attempsPerSec.ToString());

}

}

public void txt\_keyLength\_Leave()

{

string text = screen.getKeyLength().Trim();

int keyLength = 0;

if (int.TryParse(text, out keyLength))

{

keygen.setKeyLength(keyLength);

updateView();

}

else

{

keyLength = keygen.getKeyLength();

screen.setKeyLength(keyLength.ToString());

}

}

}

public partial class Form1 : Form, IKeyGUI

{

private KeyGenPresenter presenter;

public Boolean lowercase

{

set { chk\_LowerCase.Checked = value; }

get { return chk\_LowerCase.Checked; }

}

public Boolean uppercase

{

set { chk\_uppercase.Checked = value; }

get { return chk\_uppercase.Checked; }

}

public Boolean otherChars

{

set { chk\_otherchars.Checked = value; }

get { return chk\_otherchars.Checked; }

}

public Boolean number

{

set { chk\_number.Checked = value; }

get { return chk\_number.Checked; }

}

public Boolean hexOnly

{

set { chk\_hex.Checked = value; }

get { return chk\_hex.Checked; }

}

public Form1()

{

this.InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

public void setNewKey(String key)

{

txt\_generatedKey.Text = key;

}

public void setYears(String years)

{

txt\_years.Text = years;

}

public void setDays(String days)

{

txt\_days.Text = days;

}

public void setHours(String hours)

{

txt\_hours.Text = hours;

}

public void setSeconds(String seconds)

{

txt\_seconds.Text = seconds;

}

public void setKeysPerSecond(String s)

{

txt\_keysPerSecond.Text = s;

}

public void setKeyLength(String keyLength)

{

txt\_keyLength.Text = keyLength;

}

public String getKeysPerSecond()

{

return txt\_keysPerSecond.Text;

}

public String getKeyLength()

{

return txt\_keyLength.Text;

}

public void register(KeyGenPresenter KGP)

{

presenter = KGP;

}

public void message(String message)

{

MessageBox.Show(message);

}

private void chk\_LowerCase\_CheckedChanged(object sender, EventArgs e)

{

presenter.chk\_LowerCase\_CheckedChanged();

}

private void chk\_uppercase\_CheckedChanged(object sender, EventArgs e)

{

presenter.chk\_uppercase\_CheckedChanged();

}

private void chk\_number\_CheckedChanged(object sender, EventArgs e)

{

presenter.chk\_number\_CheckedChanged();

}

private void chk\_otherchars\_CheckedChanged(object sender, EventArgs e)

{

presenter.chk\_otherchars\_CheckedChanged();

}

private void chk\_hex\_CheckedChanged(object sender, EventArgs e)

{

presenter.chk\_hex\_CheckedChanged();

}

private void btn\_generateKey\_Click(object sender, EventArgs e)

{

presenter.updateView();

}

private void txt\_keysPerSecond\_Leave(object sender, EventArgs e)

{

presenter.txt\_keysPerSecond\_Leave();

}

private void txt\_keyLength\_Leave(object sender, EventArgs e)

{

presenter.txt\_keyLength\_Leave();

}

}