Objects and Classes

Engineering 1D04, Teaching Session 9

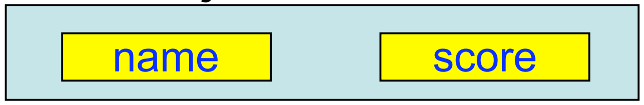
Recap - Classes & Objects

```
class hsRecord
{
    public string name;
    public int score;
}
```

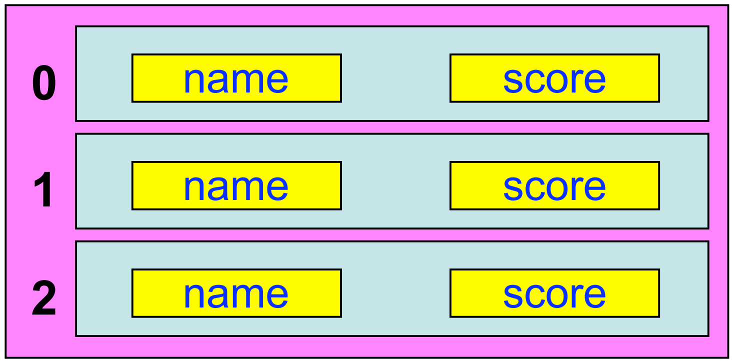
Class declaration defines a complex data structure

```
hsRecord myRef;
myRef = new hsRecord();
```

Instantiation of hsRecord is an *object*



```
hsRecord[] highScore = new hsRecord[3];
```



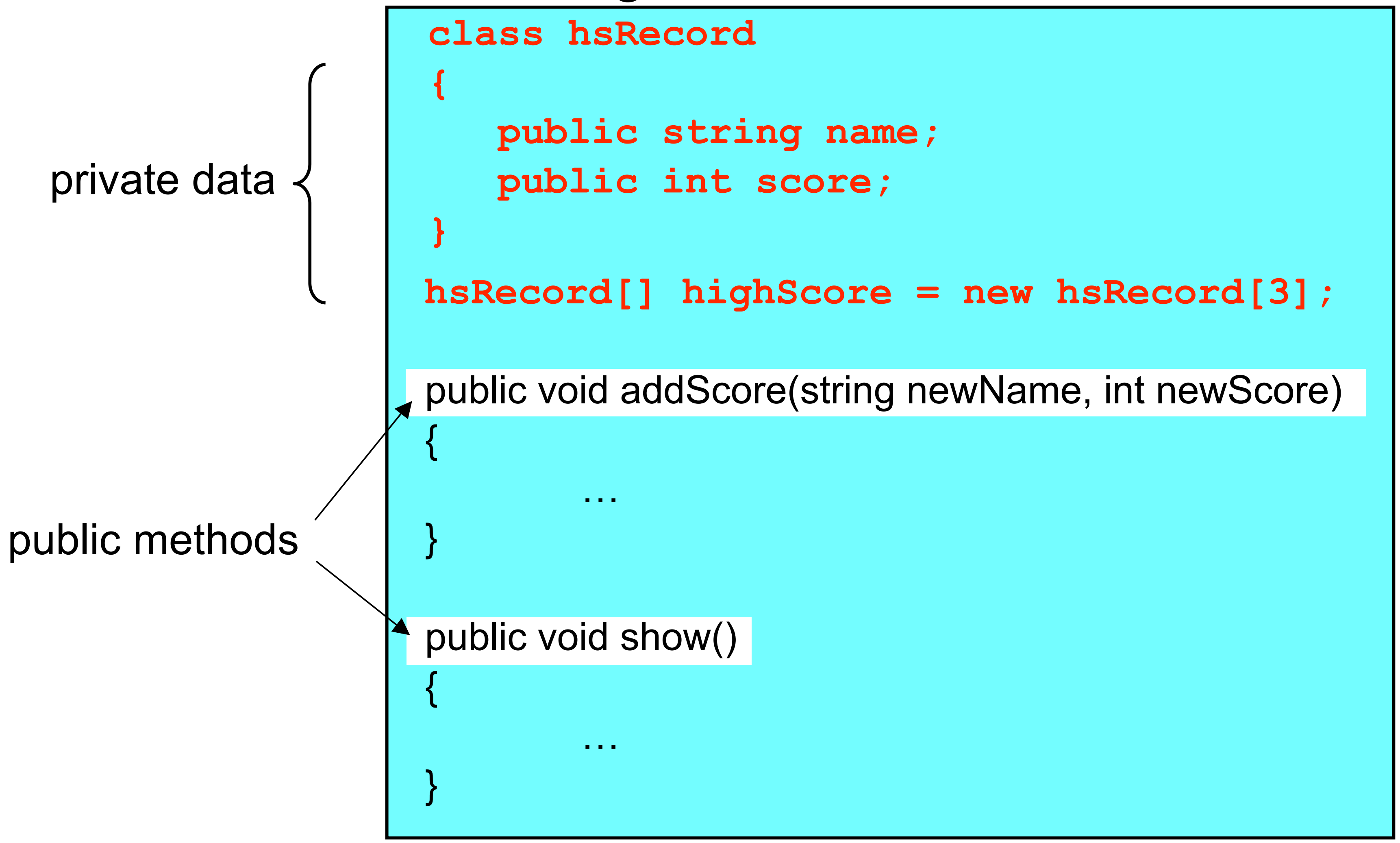
```
void addScore(string newName,
              int newScore)
{
…\\uses highScore etc
}
```

General Classes & Objects

- We have seen how classes can *encapsulate* data structures.
- This lets us group differently typed variables into a single record.
- Well - it's even better than that ☺
- Classes can encapsulate data structures and methods that work with those data structures, into a single entity.

General Classes & Objects

Class HighScore



```
class hsRecord
{
   public string name;
   public int score;
}
hsRecord[] highScore = new hsRecord[3];

public void addScore(string newName, int newScore)
{
        …
}

public void show()
{
        …
}
```

General Classes & Objects

Class HighScore

private data

cannot access from outside the class

```
class hsRecord
{
    public string name;
    public int score;
}
hsRecord[] highScore = new hsRecord[3];

public void addScore(string newName, int newScore)
{
        …
}

public void show()
{
        …
}
```

public methods

only way to access internal data structures from outside the class

General Classes & Objects

Class HighScore

So, what do users of the class see?

```
class hsRecord
{
    public string name;
    public int score;
}
hsRecord[] highScore = new hsRecord[3];

public void addScore(string newName, int newScore)
{
        …
}

public void show()
{
        …
}
```

General Classes & Objects

Class HighScore

This!

```
public void addScore(string newName, int newScore)

public void show()
```

General Classes & Objects

- Before we see how we can construct such a class, let’s discuss why it could be useful.
- So - why?

General Classes & Objects

- Before we see how we can construct such a class, let's discuss why it could be useful.
- ***Encapsulation*** - protecting data structures from being accessed from outside the class is crucial. Why?

General Classes & Objects

- Before we see how we can construct such a class, let’s discuss why it could be useful.
- ***Encapsulation*** - protecting data structures from being accessed from outside the class is crucial.
  - users cannot modify internal data
  - users cannot use information about how internal data structures work

General Classes & Objects

- Before we see how we can construct such a class, let's discuss why it could be useful.
- ***Multiple instances*** - the class can be instantiated multiple times and each instance is its own unique object. This way, it would require very little extra code to be able to store more than one set of high scores.

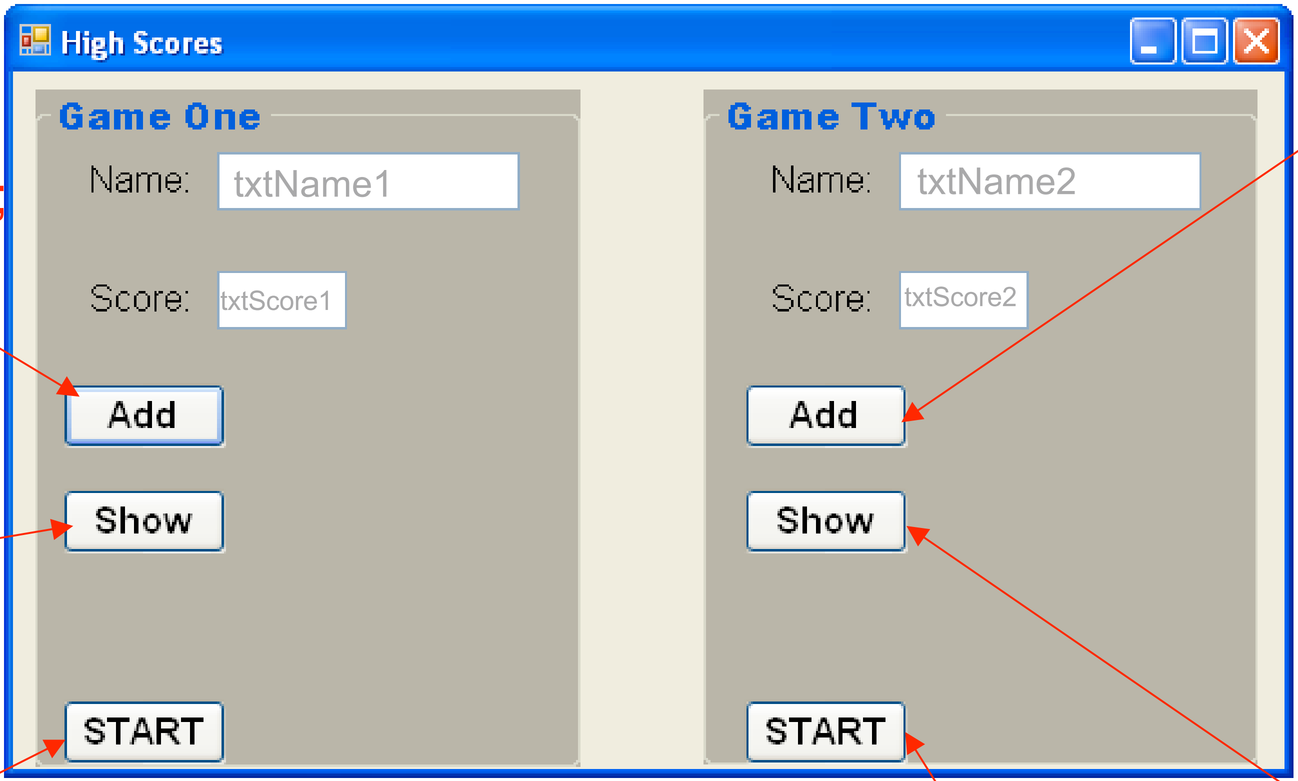
General Classes & Objects

- Example

hs2.add(n2,s2);

hs1.add(n1,s1);

hs1.show();



Instantiate first set:

hs1 = new HighScore();

Instantiate second set:

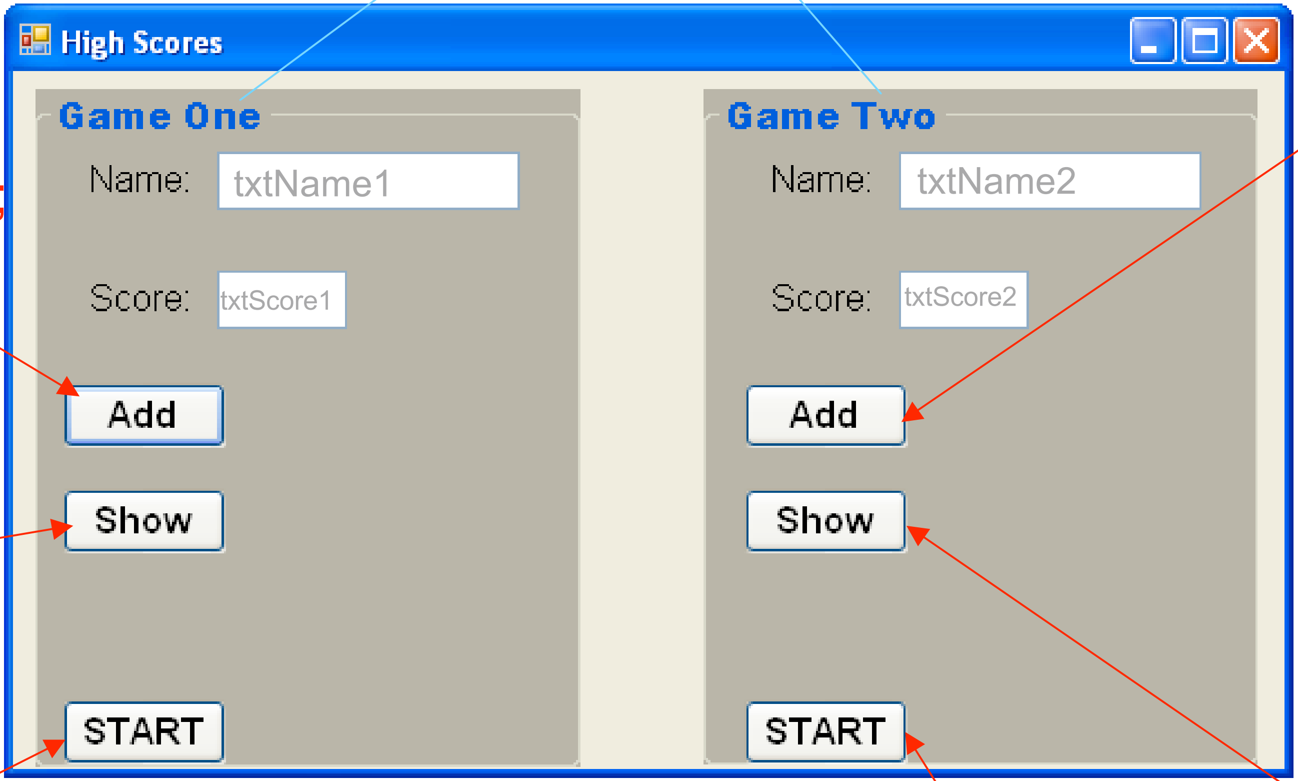
hs2 = new HighScore();

hs2.show();

General Classes & Objects

- Example

aside: we can set each game up in a *group*. This groups components together as shown.



hs1.add(n1,s1);

hs1.show();

hs2.add(n2,s2);

hs2.show();

Instantiate first set:

hs1 = new HighScore();

Instantiate second set:

hs2 = new HighScore();

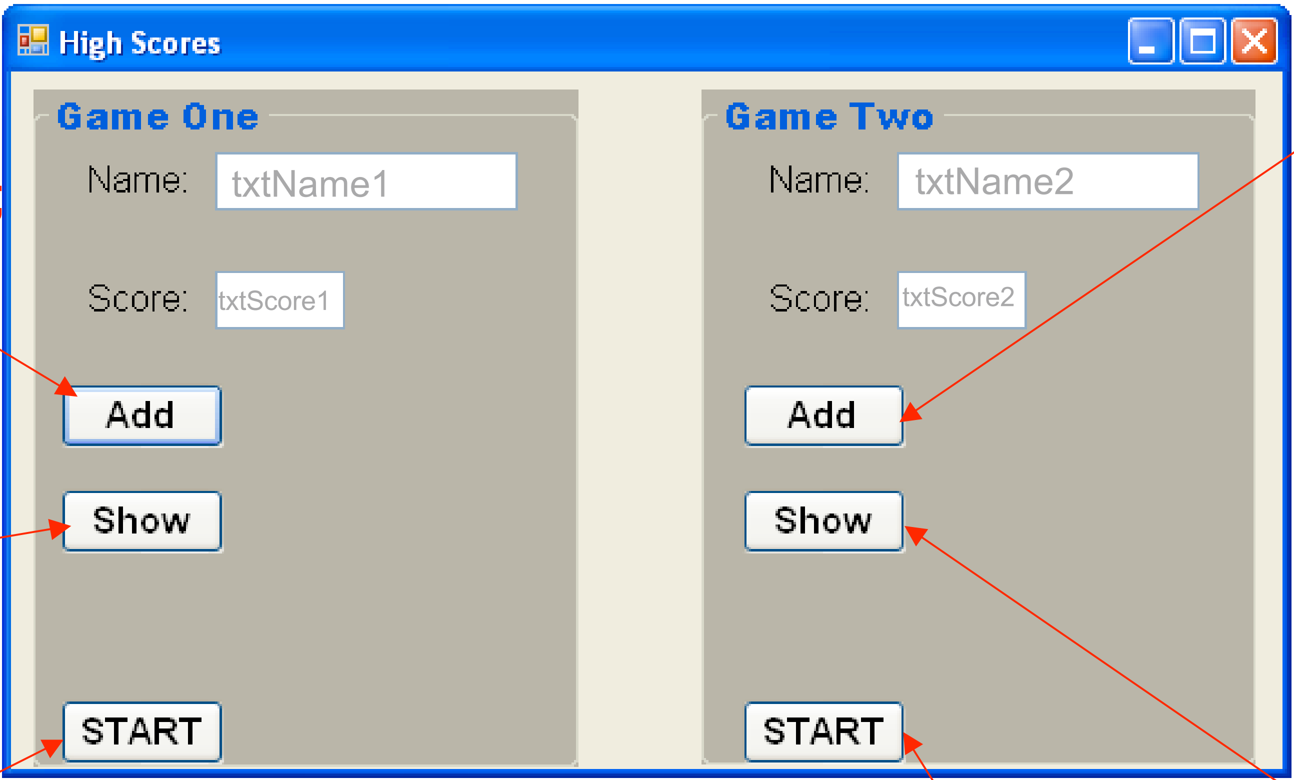
General Classes & Objects

- Example

what else do we need
to know/do to get this to work?

hs2.add(n2,s2);

hs1.add(n1,s1);



hs1.show();

Instantiate
first set:
hs1 = new HighScore();

Instantiate second set:
hs2 = new HighScore();

hs2.show();

General Classes & Objects

- Example

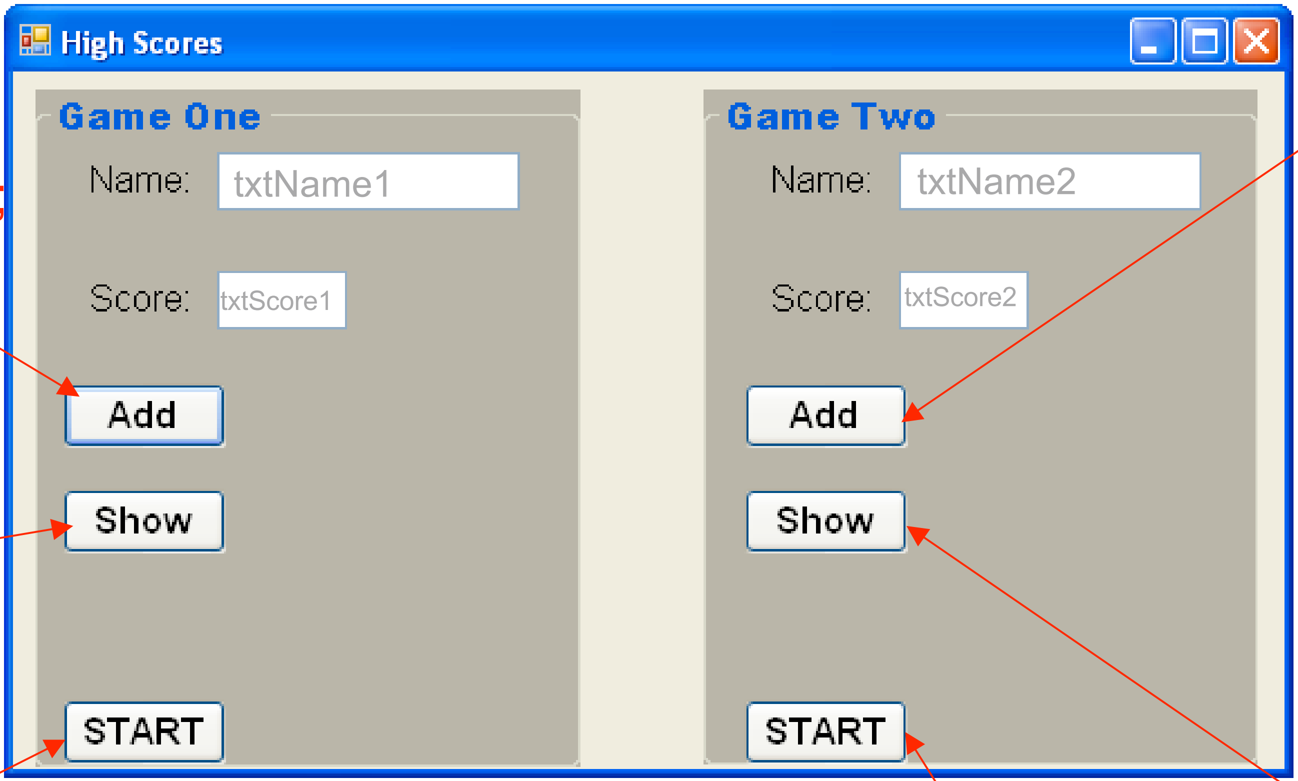
declare:
HighScore hs1, hs2;

n2=txtName2.Text
s2=txtScore2.Text

hs2.add(n2,s2);

n1=txtName1.Text
s1=txtScore1.Text

hs1.add(n1,s1);



hs1.show();

Instantiate
first set:
hs1 = new HighScore();

Instantiate second set:
hs2 = new HighScore();

hs2.show();

General Classes & Objects

- Back to the class - HighScore
- We can use declarations we already developed for the earlier version.
- We also developed the algorithms for *add* and *show* - and we can use them also.

General Classes & Objects

```
public class HighScore
{
        private class hsRecord
        {
                public string name;
                public int score;
        }

        private const int maxElements = 10;
        private hsRecord[] hsArray = new hsRecord[maxElements];
        private int length;

        public bool add(string newName, int newScore)
        {
                . . .
        }

        public void show()
        {
                . . .
        }
}
```

no room to show this here

no room to show this here

General Classes & Objects

```
public class HighScore
{
        private class hsRecord
        {
                public string name;
                public int score;
        }

        private const int maxElements = 10;
        private hsRecord[] hsArray = new hsRecord[maxElements];
        private int length;

        public bool add(string newName, int newScore)
        {
                . . .          no room to show this here
        }

        public void show()
        {
                . . .          no room to show this here
        }
}
```

`public`	available outside the class
`private`	NOT available outside the class
`public`	what about this one?

General Classes & Objects

```
public class HighScore
{
        private class hsRecord
        {
                public string name;
                public int score;
        }

        private const int maxElements = 10;
        private hsRecord[] hsArray = new hsRecord[maxElements];
        private int length;

        public bool add(string newName, int newScore)
        {
                . . .
        }

        public void show()
        {
                . . .
        }
}
```

available outside hsRecord but not outside HighScore

no room to show this here

no room to show this here

General Classes & Objects

```
public bool add(string newName, int newScore)
{
    int j, mark;
    hsRecord newRecord;

    newRecord = new hsRecord();
    newRecord.score = newScore;
    newRecord.name = newName;
    if (length < maxElements)
    {
        hsArray[length] = newRecord;
        length++;
    }

    mark = 0;
    while (mark <= length - 1 &&
           hsArray[mark].score >= newScore) mark++;

    if (mark <= length - 1)
    {
        for (j = length - 1; j > mark; j--)
            hsArray[j] = hsArray[j - 1];
        hsArray[mark] = newRecord;
    }
    return ((length < maxElements) || (mark <= length - 1));
}
```

a small change:
add returns a boolean value.

True $\Rightarrow$ record was added because score was good enough

False $\Rightarrow$ record was not added because score was not good enough

General Classes & Objects

```
public void show()
{
    const int nameLength = 15;

    string displayString, s;

    displayString = "High Scores\n";
    for (int i = 0; i <= length-1; i++)
    {
        s = hsArray[i].name;
        while (s.Length < nameLength) s += " ";
        s += "\t" + hsArray[i].score + "\n";
        displayString += s;
    }
    MessageBox.Show(displayString);
}
```

show displays the current high scores in a message box

General Classes & Objects

- Once we have the class set up, we can concentrate on using it to implement the behaviour on our form.
- If we want two instances of the class (we need them to manage each of the two high score lists we want to maintain) we have to declare two variables of that type:
  - HighScore hs1, hs2;

General Classes & Objects

For Game One Scores:

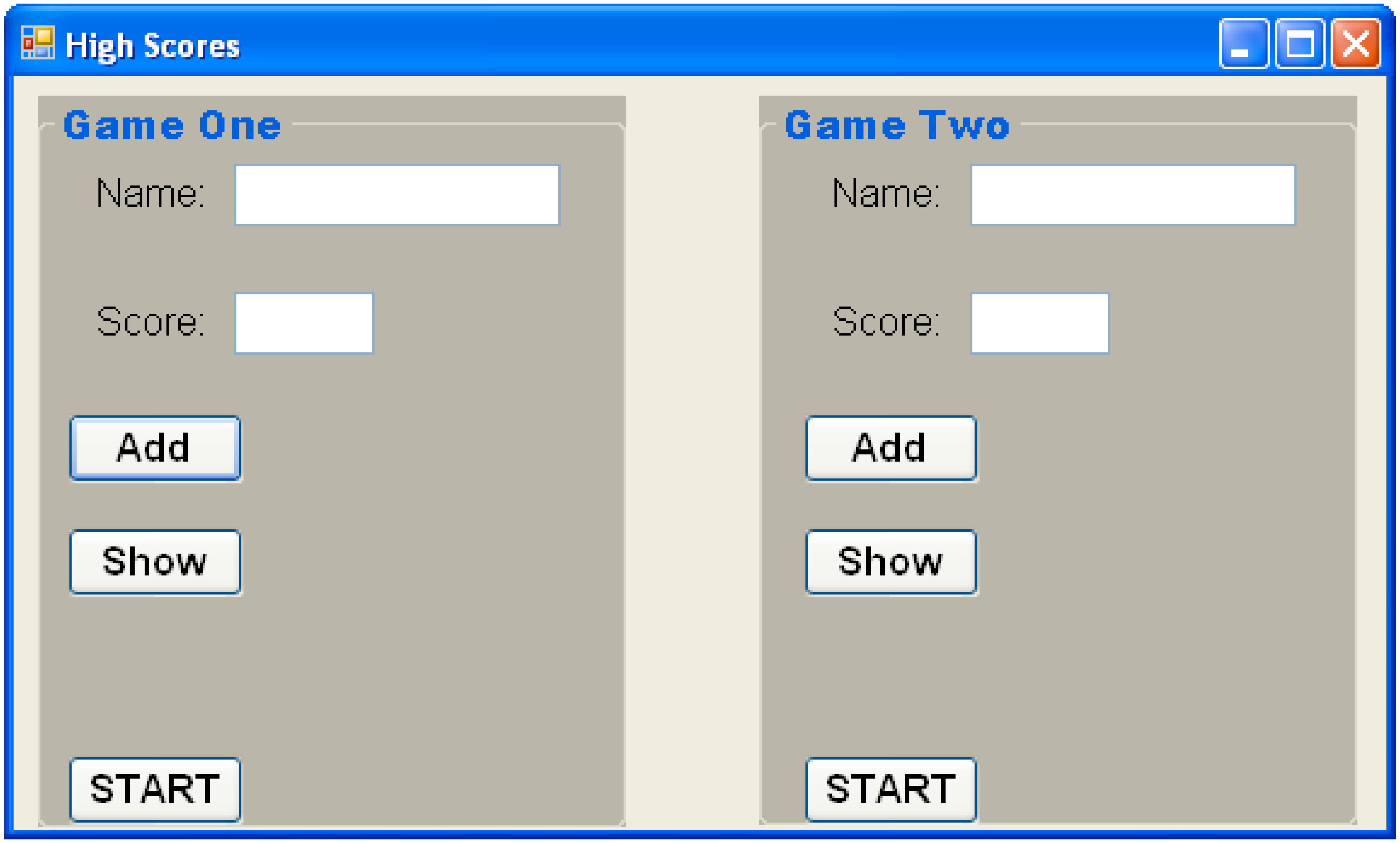
```
private void btnStart1_Click(object sender, EventArgs e)
{
    hs1 = new HighScore();
}

private void btnAdd1_Click(object sender, EventArgs e)
{
    if (hs1.add(txtName1.Text,
        Convert.ToInt32(txtScore1.Text)))
           lblAdded1.Text = txtName1.Text + " added";
    else lblAdded1.Text = txtName1.Text + " not added";
}

private void btnShow1_Click(object sender, EventArgs e)
{
    hs1.show();
}
```

General Classes & Objects

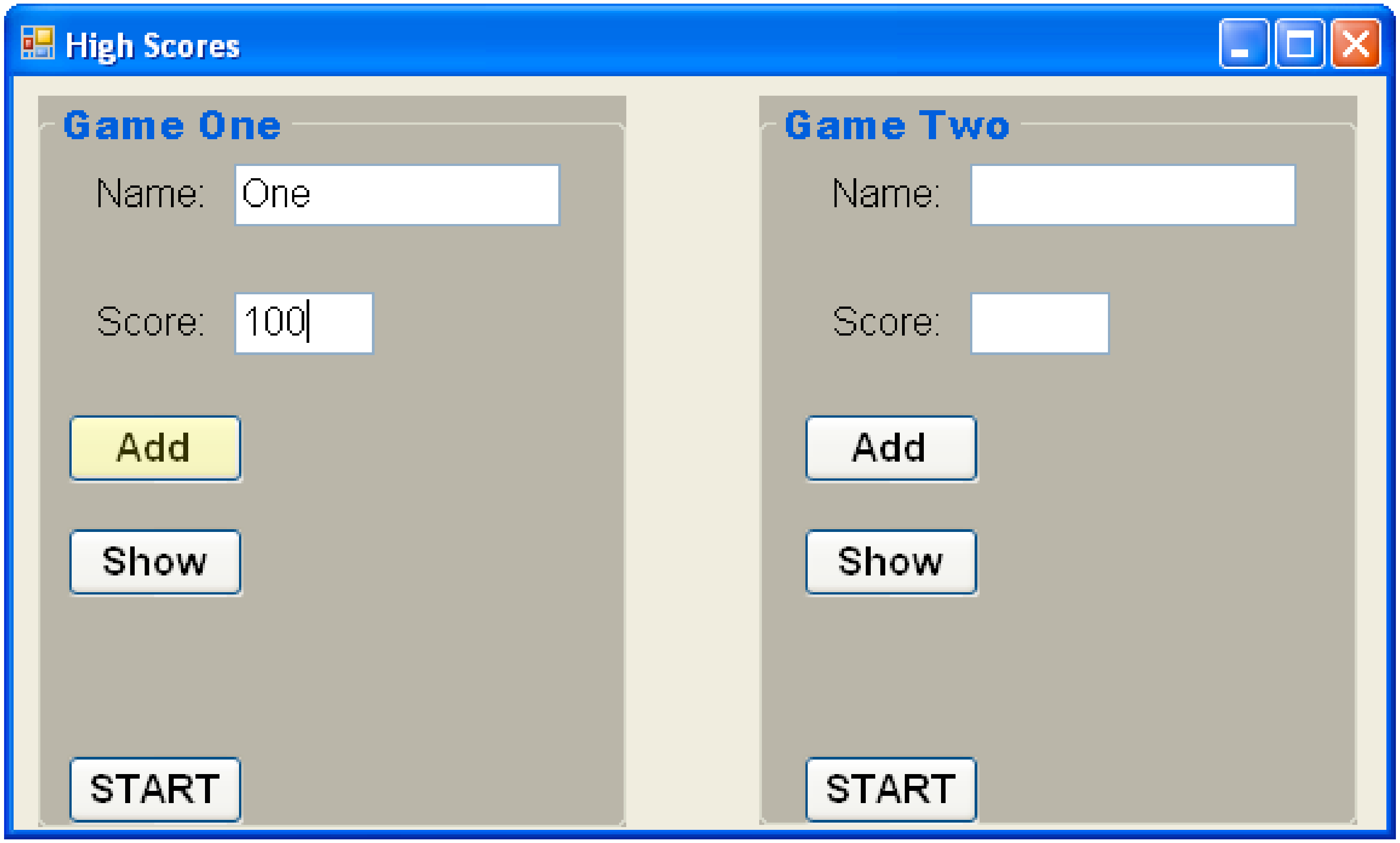
For Game Two Scores:

```
private void btnStart2_Click(object sender, EventArgs e)
{
    hs2 = new HighScore();
}

private void btnAdd2_Click(object sender, EventArgs e)
{
    if (hs2.add(txtName2.Text,
        Convert.ToInt32(txtScore2.Text)))
           lblAdded2.Text = txtName2.Text + " added";
    else lblAdded2.Text = txtName2.Text + " not added";
}

private void btnShow2_Click(object sender, EventArgs e)
{
    hs2.show();
}
```

Examples with Game One



Examples with Game One



Game One

Name: One

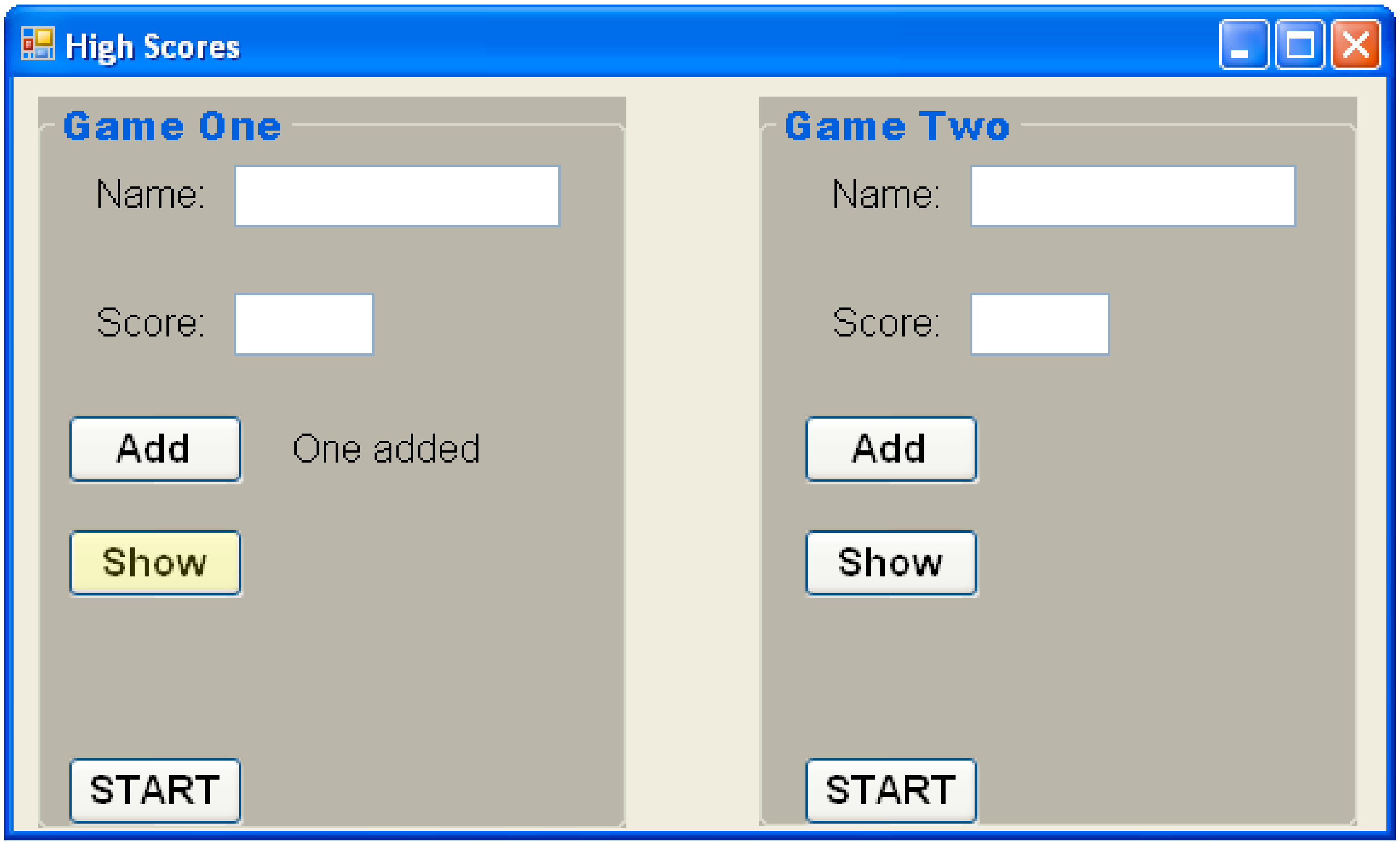
Score: 100

Game Two

Name:

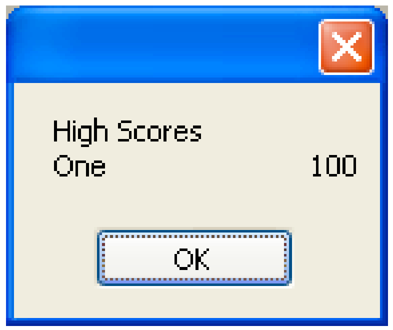
Score:

Examples with Game One



One added

Examples with Game One


```
High Scores
One                 100
```

Using the Class

```
namespace High_Score_Class
{
    public partial class Form1 : Form
    {
        HighScore hs1, hs2;

        public Form1() {
            InitializeComponent();
        }
        private void btnStart1_Click(object sender, EventArgs e) {
            . . .
        }
        private void btnAdd1_Click(object sender, EventArgs e) {
            . . .
        }
        private void btnShow1_Click(object sender, EventArgs e) {
            . . .
        }
        private void btnStart2_Click(object sender, EventArgs e) {
            . . .
        }
        private void btnAdd2_Click(object sender, EventArgs e) {
            . . .
        }
        private void btnShow2_Click(object sender, EventArgs e) {
            . . .
        }
    }
    public class HighScore
    {
      . . .
    }
}
```

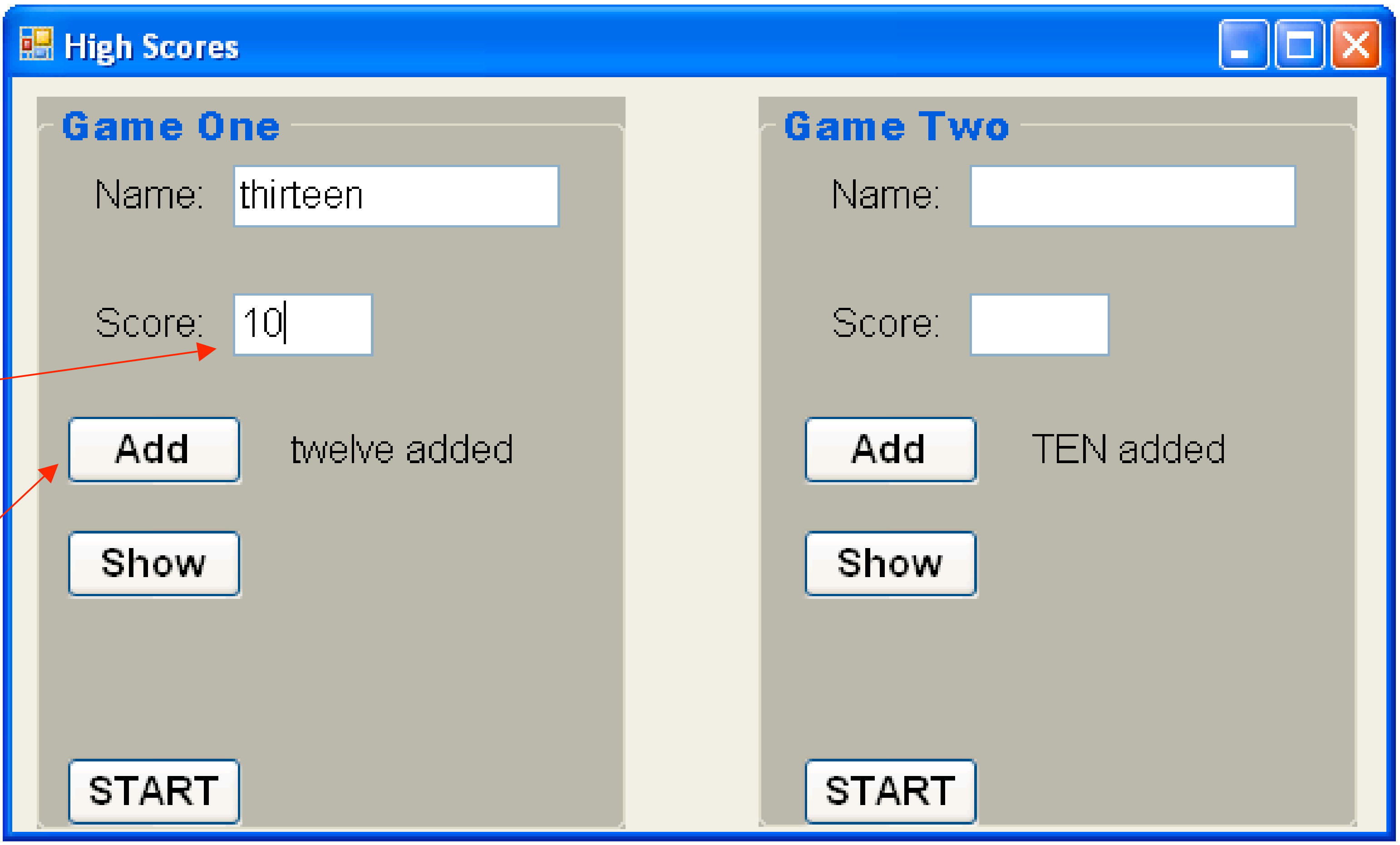
references to objects declared

Form uses HighScore class

HighScore

More of the Example

Two separate high score systems operating



Name: thirteen
Score: 10
twelve added
TEN added

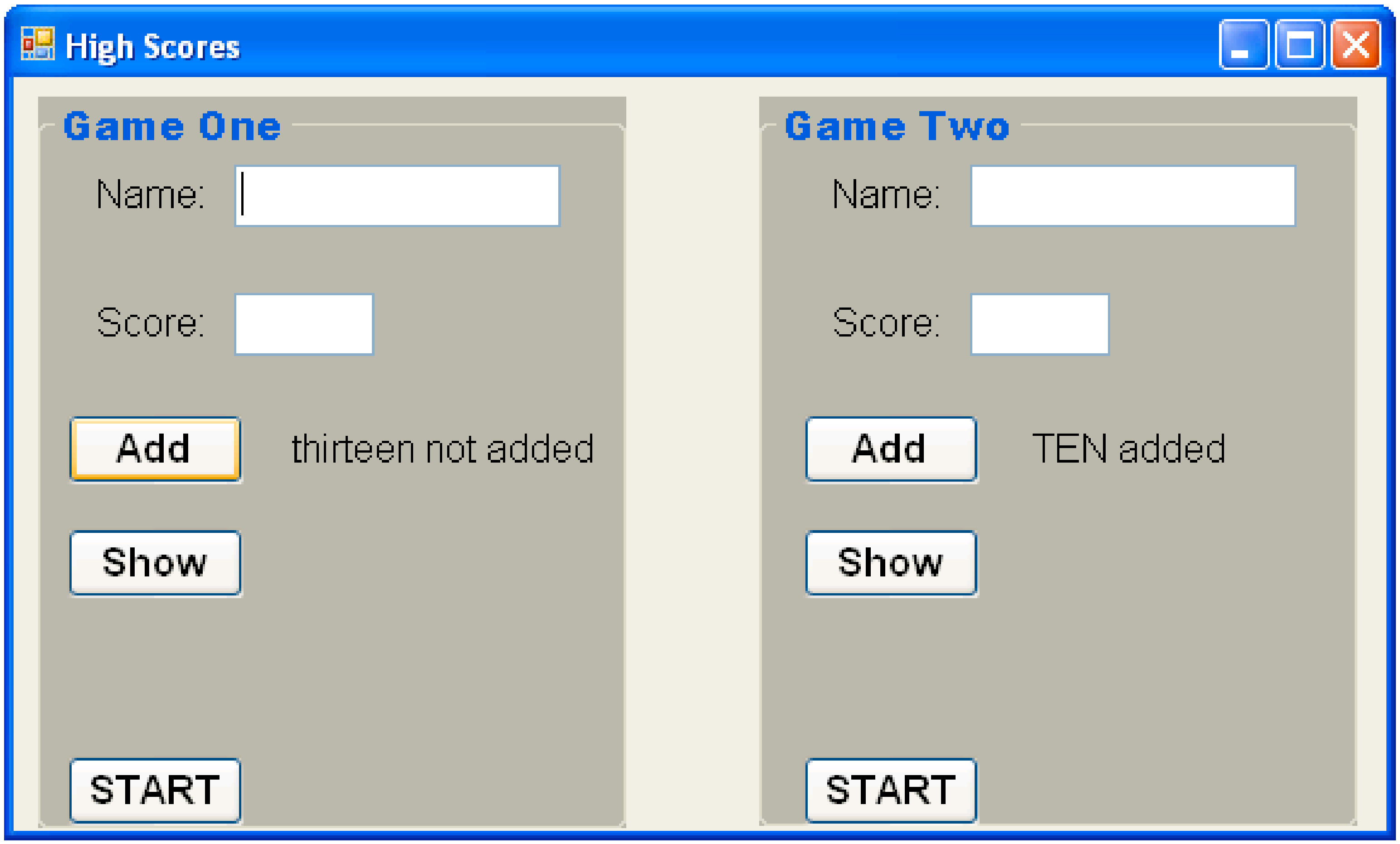
trying to add a score of 10

now click Add

More of the Example

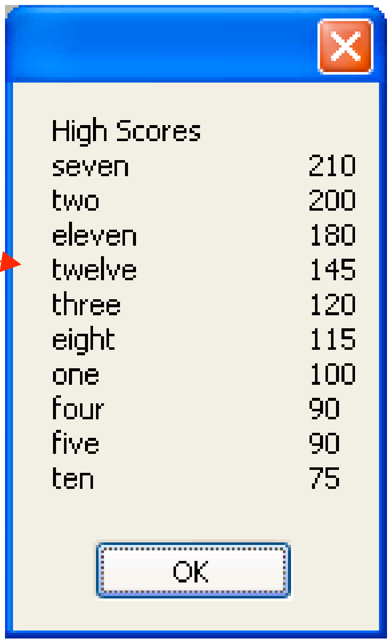
Two separate high score systems operating

a score of 10 was not enough

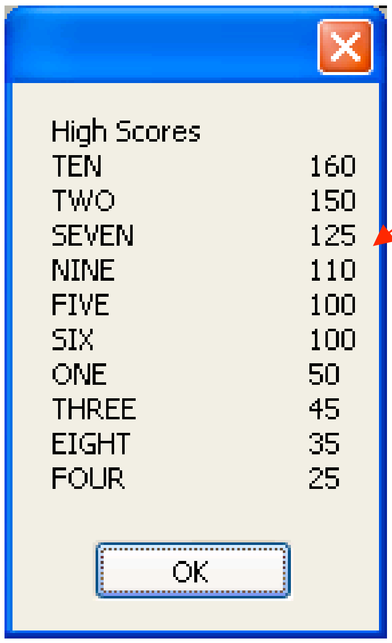


More of the Example

after clicking Show for Game One



High Scores	
seven	210
two	200
eleven	180
twelve	145
three	120
eight	115
one	100
four	90
five	90
ten	75



High Scores	
TEN	160
TWO	150
SEVEN	125
NINE	110
FIVE	100
SIX	100
ONE	50
THREE	45
EIGHT	35
FOUR	25

after clicking Show for Game Two

User Interface Considerations

User Interface

- There are some fine points concerning the user interface we should consider.
- Note - this is just another aspect of algorithms. The user interface has to be considered in our solution algorithms.

1. When we click on a button we need to consider what component should have the *focus* (i.e. what should be “live”).
2. How do we help the user not to click an inappropriate button?

User Interface

1. We can specify that a component has the focus by using the method Focus(). So, after clicking *Add* in the Game One group, we could set the focus to the name text box by **`txtName1.Focus()`**
2. A better set up of components would be the following: In each group, make all components not visible initially - except *Start*. If *Start* is clicked, make *Start* not visible, make all others visible.

User Interface

- Example for the button *Add*

```
private void btnAdd1_Click(object sender, EventArgs e)
{
        if (hs1.add(txtName1.Text,
             Convert.ToInt32(txtScore1.Text)))
                lblAdded1.Text = txtName1.Text + " added";
        else lblAdded1.Text = txtName1.Text + " not added";
        txtName1.Text = "";    } clear text boxes
        txtScore1.Text = "";   }
        txtName1.Focus();  <-- move focus
}
```

Class Methods and Properties

Class Methods & Properties

- We have already seen *methods* of classes. In our example we used two methods: add and show.
- The methods give users of the class access to internal variables, but protects those variables from unauthorized modification.
- We have also seen *properties*. Where?

Class Methods & Properties

- We have seen *properties* in all the forms and components we have worked with.
- Examples:

  object

  - txtName1.Text
  - txtName1.Visible

  property

  object

- So, how do we create and code properties? What do you think a property is (in relation to a class)?

Class Properties

- A property of a class is simply the value of an internal variable of the class.
- Sometimes we want to set the value of that variable from outside the class.
- At other times we want to get the value of that variable from outside the class.
- Examples
  - txtName1.Text = “A.N. Other”;
  - myString = txtName1.Text;

set string value as text box property

get string value from text box property

Class Properties

- We don’t have to always both *set* and *get* property values. It just works out that most of the time we want to do both.
- C# has treated this situation in a really nice and consistent way.
- We use *get* and *set* code blocks to get and set property values.

Class Properties

- Example

name is private so not available outside the class

```
public class DemoExample
{
    private string name;

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

by convention we use caps for the starting character - the name is otherwise same as the private variable

just returns the current value of the internal string

value is a magic value - it is automatically the value the user puts after the equals sign

Class Properties

- Example

name is private so not available outside the class

```
    public class DemoExample
    {
        private string name;

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

by convention we use caps for the starting character - the name is otherwise same as the private variable

just returns the current value of the internal string

value is a magic value - it is automatically the value the user puts after the equals sign

Class Properties

class DemoExample

```
private string name;

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

```
DemoExample v;
string s;

s = v.Name;
v.Name = "demo";
```

Back to High Scores

Back to HighScores

- There are at least two changes we may think about making to our HighScores class.
- Any ideas?

Back to HighScores

- There are at least two changes we may think about making to our HighScores class.
- The method *Show()* should probably be a property *Result*. Why?
- When we instantiate the object (with Start), maybe we should be able to give it a string that informs the object the name of the game for which it is keeping the scores.

Back to HighScores

- The way we implemented Show in the current version of HighScores is too rigid.
- Show was implemented as a method, and the method formats the result string and then displays it using a message box.
- What if we don’t want to display the results in a message box? We have no choice. The result is available only that way.

Back to HighScores

- It is much more versatile to implement Show as a property - let's call it *Result*.
- It requires no set, just a get.

```
public string Result
{
    get
    {
        const int nameLength = 15;
        string display, s;
        display = "High Scores\n";
        for (int i = 0; i <= length-1; i++)
        {
            s = hsArray[i].name;
            while (s.Length < nameLength) s += " ";
            s += "\t" + hsArray[i].score + "\n";
            display += s;
        }
        result = display;
        return result;
    }
}
```

private string result;
declared in the class

Back to HighScores

- What about including a title for the name of the game at the time of instantiation?
- Think about it - we’ll do it next time.