Task 1: Sprite Animation

- Declare an Animator class.
- Create an object (or instance) of the Animator class.
- Call the animator's *LoadImages()* method.

You may remember as a child playing with a "flip book". The pages have a series of small images, each of which differed slightly from the preceding one. As you flipped through the pages of the book, the image appeared to move, like a cartoon.

Rapid presentation of a series of changing images is an extremely common technique for producing animation. It works because the human visual system retains a "ghost" of its visual input for a very short time (a few milliseconds) after the input is removed. Thus if a sequence of images is presented quickly enough, the eye perceives continuous movement.

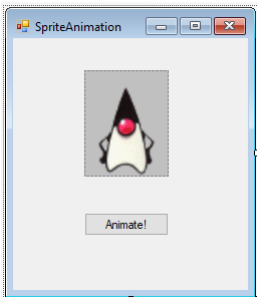
In computer graphics, this style of animation is often used to produce movement. Characters in computer games, for example, are generally rendered in this way. The individual images of the character in motion are called "sprites", and the technique is called "sprite animation". Generating complex animation sequences without flicker involves delicate mathematical computations, image drawing in memory, and well-timed screen repainting. Fortunately, we can produce a simple sprite animation just by loading a series of images into a PictureBox control.

Shown below is a ten image sprite sequence:

If these images are displayed rapidly using a single PictureBox it will appear as though the little guy is waving to you.

Creating the Form

1. Open a new application called *SpriteAnimation*.
2. Add the images from the **04-classes-objects/sprite-animation** directory into to the Resource folder.
3. Add a PictureBox and a Button to the form. Change the Text properties of the Form and Button. Load the first image into the PictureBox.



Declaring the Animator Class

1. In the **Solution Explorer**, RightClick on the *SpriteAnimation* namespace, and then select ***Add a Class***. Name the class ***Animator***. This will create an *Animator.cs* file which can be seen in the Solution Explorer window.

2. Open the *Animator.cs* file from the Solution Explorer.
   Give the *Animator* class public scope.
   The Animator class needs two **fields**: an array of images that holds the image files and the PictureBox into which to load the images.
   The *Animator* class needs two **methods**: its **constructor** called *Animator* so that it can construct/create itself and a method called *Loadimages()* which controls the animation of the images.

   Your *Animator* class should look like this:

```
namespace SpriteAnimation
{
    public class Animator
    {
        private Image[] images;
        private PictureBox pictureBox;

        public Animator()
        {
        }

        public void LoadImages()
        {
        }
    }
}
```

   Note that when we declare the *fields*, we say that we will be using an array of *Image*, but we have not stipulated how big the array is. We will set the size of the array in the constructor.

3. The **constructor** is a special method used to create an object by initialising its fields. It has the same name as the class and no return type. This is a C# convention. The constructor is used to initialise the fields to appropriate starting values.
   Our constructor is therefore called *Animator*. It sets the size of the array to the number of images that we want to use, loads the image files into the images array from the Resources folder and passes a reference to the PictureBox on the form.

```
public Animator(PictureBox pictureBox)
    {
      images = new Image[11];

      for (int i = 0; i < images.Length; i++)
      {
          images[i] = (Bitmap) Properties.Resources.ResourceManager.GetObject
                      ("T" + i.ToString());
      }

        this.pictureBox = pictureBox;
    }
```

By writing our constructor method this way, we ensure that any instance of our class is properly initialised at runtime. This means that no instance can be created without the provision of legal values for *all* its fields.

4. The *LoadImages()* method will load each image in turn from the images array into the PictureBox where it is immediately displayed.

```
public void LoadImages()
{
    for (int i = 0; i < images.Length; i++)
    {
       pictureBox.Image = images[i];
    }
}
```

Now if we run this, all the images will be loaded so quickly that we only see the last image. To slow things down, we need to add:

```
Application.DoEvents();
```

which will process all Windows messages as they occur.

We also need to add:

```
System.Threading.Thread.Sleep(100);
```

which suspends the current thread for 100 milliseconds, long enough for the image to be visible to the human eye.

Alternatively, to make your code more readable, add

```
using System.Threading;
```

to the list at the top of the class, and then shorten the command to

```
Thread.Sleep(1000);
```

Creating an Animator Object

To create our Animator object (an instance of the Animator class), we must **declare** it as a field in the Form.
In the Form1's constructor, we **create** the animator object by calling its constructor. Note we need to pass a reference to the PictureBox across to the Animator so that it can be used from within the Animator class.

```
public partial class Form1 : Form
{
    private Animator animator;

    public Form1()
    {
        InitializeComponent();

        animator = new Animator(pictureBox1);
    }
```

Using the Animator Object's Methods

We can now use the Animator object (that we have just created) in the `button1_Click` handler for the *Animate!* Button.
When someone clicks the *Animate!* Button, the Animator object should call its *LoadImages()* method. Think of this as though you were sending a message to the animator object: "animator, please run through your animation process by loading each image in turn."

```
animator.LoadImages();
```

Refactoring/Refining your code

1. Remember the Block Comment at the top of Form1.

2. Add informative comments for every class, every field and every method. These comments are designed to help someone else to navigate and read your code. Do *not* try to be funny or cryptic, you should be professional and communicative.

3. Replace any numbers in your code with constants. Constants are listed before the fields and are always written in capital letters.

```
private const int NIMAGES = 11;
```

Task 2: Deal A Card

********* CHECKPOINT 3 *********

- Make a Card class.
- Create 1 Card object (an instance of the Card class).
- Use the Card object, by calling the Card's DealACard() method.
- Create an array of 5 cards.

Your game should provide a "*Deal a card"* button. Each time the user clicks the "*Deal a card"* button, a single card is selected randomly from the 52 possible playing cards. Its corresponding image is loaded into the PictureBox and a description is displayed in a Label. The 52 images are.jpg files and available on Teams

A possible screen layout is shown below:

or

Implementation Suggestions:

1. **Creating the Form**
   Open a new application called *DealACard*.
   Add the images from the **04-class-objects/deal-a-hand** directory to the Resources folder (Project/DealACard properties).
   Add a PictureBox and a Button to the form. Change the Text properties of the Form and Button.

2. **Make a Card class.**

   In the **Solution Explorer**, RightClick on the *DealACard* namespace, and then select ***Add a Class***. Name the class ***Card***. This will create a *Card.cs* file which can be seen in the Solution Explorer window.

   Open the *Card.cs* file from the Solution Explorer.
   Give the *Card* class public scope.

   What data does a Card need to hold or know? This data is stored in the **fields**. What will the data type for each field be?

```
private int rank;
private int suit;
private Image image;
private PictureBox pictureBox;
private Random random;
```

What **constants** will you use?

You always need a **constructor**. What is its name?

Use constructor is used to initialise all the field values.
What parameter values do you need to pass into the constructor?
Do you need to use the keyword **this**?

You will need a **method** called *DealACard*(). This method should call 3 **sub-methods**, one to randomly choose the suit, rank and image for the card, another to load the image into the pictureBox and the third to build the description string that is displayed in the Label.

```
private void chooseACard()
private void loadPictureBox()
private string writeToLabel()
```

3. **Create one Card object.**
   In the Form1 class, create one Card object (I called this *card*).

4. **Use the Card object's methods.**
   In the Form1 class, create a *Click* handler for the "*Deal a card"* button. This should call the three methods in the Card class that randomly choose a card, load the image and build the description string.

5. Add the block comment in the Form, class comment in Card class and in-line comments as required.

Task 3: Deal A Hand

1. Modify your application so that you deal a **hand of five cards**.

2. Now check that you **do not have duplicate cards**.