KEEPING IT
CLASSY
WITH
C#

Level 4

Loops

Wiring Up Musicians Logic

Our bands contain musicians, but we'll need to expand our code with loops to utilize them.

What our application will do:

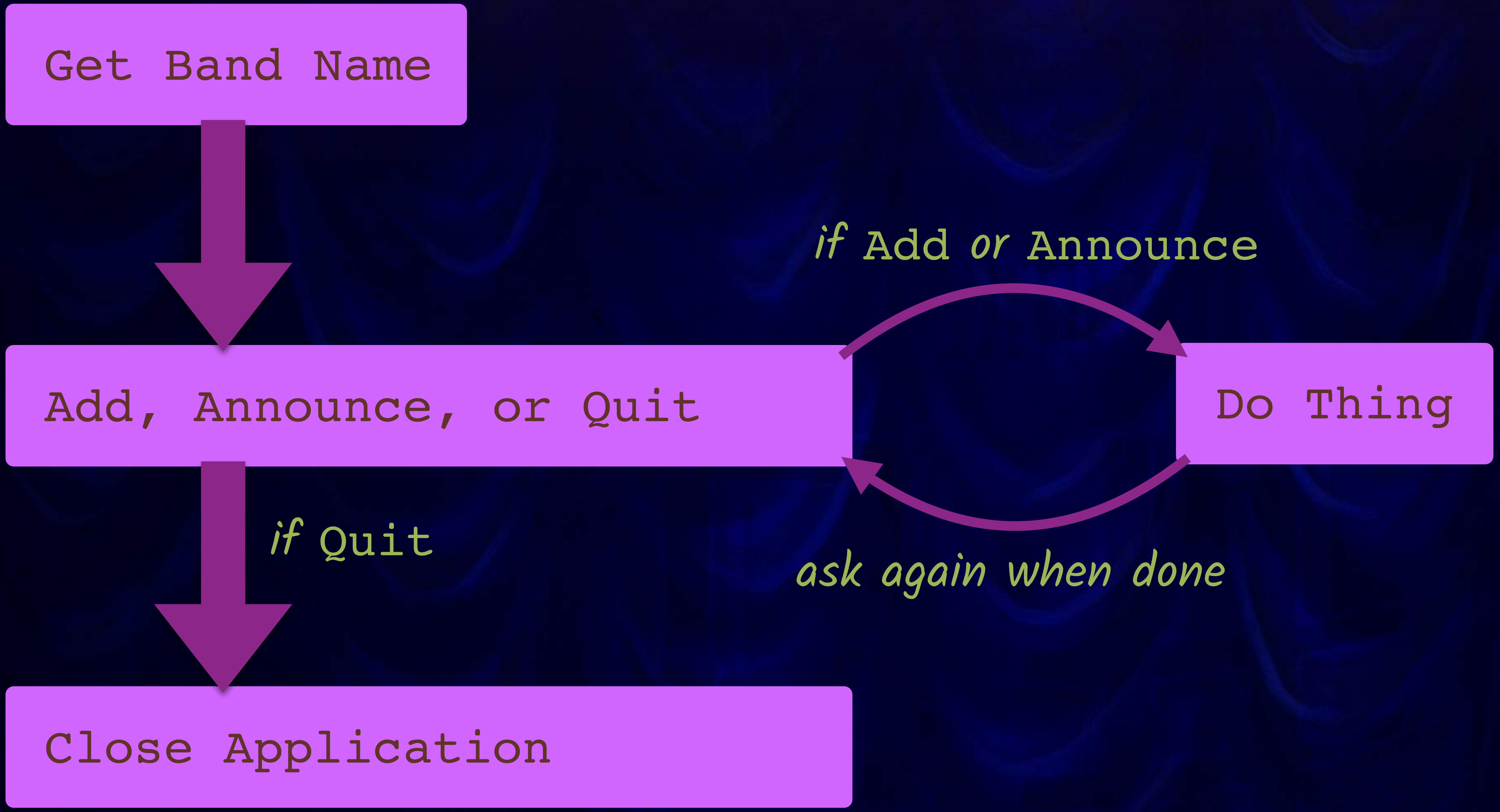
- Store information about a band and it's musicians
- Announce the band
- Announce the musicians

In this level:

- Create a Loop to add Musicians per band
- Create a Loop to announce musicians

Our Program Flow

Our application will get the band name, then ask to add a musician, announce the band, or quit.



How do we implement repeating to ask until `Quit` *is used?*

Add Command List

Before we create a loop, we'll inform the user of the accepted commands.

Program.cs

```
…
static void Main(string[] args)
{
  Console.WriteLine("What is the name of your band?");
  Band band = new Band();
  band.Name = Console.ReadLine();

  Console.WriteLine("Type 'Add' to add a musician.");
  Console.WriteLine("Type 'Announce' to announce the band.");
  Console.WriteLine("Type 'Quit' to quit the application.");
}
…
```

Now we can start implementing our loop!

KEEPING IT CLASSY WITH C#

while Loop

A while loop will continue to run until the break keyword is used or it's condition is false.

Program.cs

```
…
static void Main(string[] args)
{
    …
    var repeat = true;
    while(repeat)
    {

    }
}
…
```

A new variable repeat will be used in our while loop's condition

This will run forever until loop is false or the break keyword is used

Be careful using loops, the above example has no way to exit the loop creating what's known as an infinite loop

Looping Conditions

Add our conditions that will handle adding a musician, announcing the band, or quitting the application.

Program.cs

```
while(repeat)
{
  Console.WriteLine("Add, Announce, or Quit?");
  var action = Console.ReadLine();
  if(action == "Add"){…}
  else if(action == "Announce"){…}
  else if(action == "Quit") {…}
  else
  {
    Console.WriteLine(action + " is not a valid command");
  }
}
```

We'll also handle when the input doesn't match any of our commands

Add and Announce Commands

Add the appropriate calls to AddMusician **and** Announce **methods from** Band.

Program.cs

```
…
while(repeat)
{
   …
   if(action == "Add")
   {
      band.AddMusician();    AddMusician will be run when "Add" is entered
   }
   else if(action == "Announce")
   {
      band.Announce();       Announce will be run when "Announce" is entered
   }
   …
}
```

break keyword

The break keyword escapes the loop at the point it's called.

Program.cs

```
…
while(repeat)
{
  Console.WriteLine("Add, Announce, or Quit?");
  …
  else if(action == "Quit")
  {
    break;
  }
  …
}
…
```

The loop will exit at this point skipping any remaining code in the loop

What would this look like if we changed the while condition instead of using break?

KEEPING IT CLASSY WITH C#

Alternative: Escape Using while Condition

When the while condition is false, the loop will escape upon reaching the end of the while block.

Program.cs

```
…
var repeat = true;
while(repeat)
{
  Console.WriteLine("Add, Announce, or Quit?");
  …
  else if(action == "Quit")
  {
    repeat = false;
  }
  …
}
…
```

When loop is set to false, the while loop will escape once it finishes it's current loop

With that done we need to update our Announce method to include our musicians

Foreach Loop

A foreach loop iterates through a group of objects one by one and runs code for each item.

Band.cs

```
…
void Announce()
{
  Console.WriteLine("Welcome " + Name + " to the stage!");

  foreach(var musician in Musicians)
  {
    musician.Announce();
  }
}
…
```

foreach will loop through each musician and in Musicians and run their Announce method

foreach will run until it's run on every item in a group of objects, an unhandled exception is thrown, or the break keyword is used

KEEPING IT CLASSY WITH C#

Our Working Application

Our application now allows us to do everything we set out to do when we started.

Application features include:

- Stores information about a band and it’s musicians
- Announces the band
- Announces the musicians

Our Running Application

Our loops will allow users to repeat actions until they use the Quit command.

```
      Add, Announce, or Quit?
>>>   $ Add
      What is the name of the musician to be added?
>>>   $ Robert
      What instrument does Robert play?
>>>   $ Guitar
      Add, Announce, or Quit?
>>>   $ Quit
```

KEEPING IT CLASSY WITH C#

A Quick Recap on Loops

Loops allow us to repeat code logically without rewriting it again and again.

- All loops escape immediately when the break keyword is used
- Always make sure there is a way to escape the loop!
  (Infinite loops can be really bad)
- while loops will escape before running the first line in their block when their condition is false
- foreach loops will escape when they've run their code for every item in the collection