Getting ready

Following are the ingredients required to execute this recipe:

- An Arduino board connected to a computer via USB.
- A small 8-ohm speaker.
- A 120-ohm resistor; larger values also work, but the sound will be less powerful. Don't use resistors under 100 ohms.

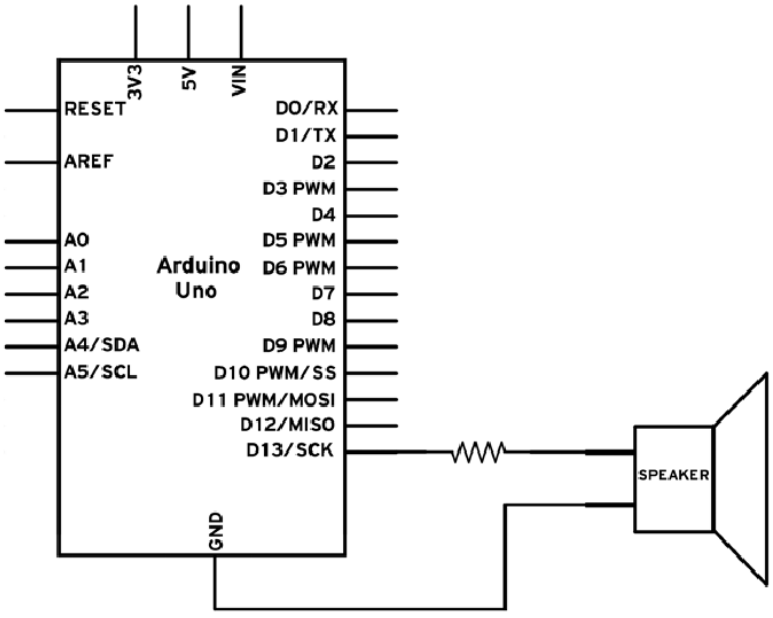
How to do it...

Follow these steps to connect a speaker to the Arduino:

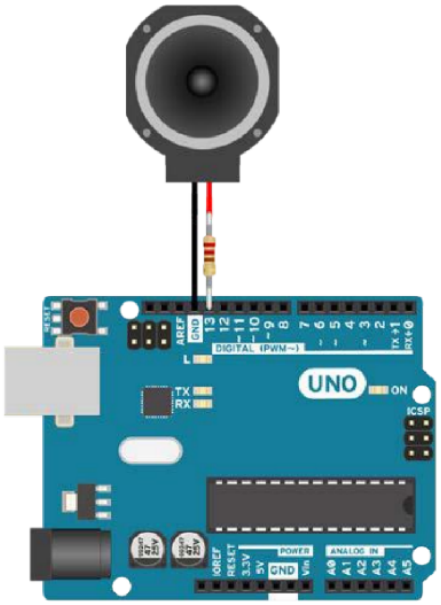
1. Connect one terminal of the speaker directly into the **GND** of the Arduino.
2. Using a 120-ohm resistor in series, connect the other terminal to an available digital pin; in this example, pin **12**.

Schematic

This is one possible implementation on the 13th digital pin. Other digital pins can also be used.



Here is an example of how to wire it in the air. No breadboard needed here:



Code

The following code will play the famous Solfeggio—Do Re Mi Fa Sol La Ti:

```
// Defining the 8 frequencies that make the 7 notes and one
repetition in the Solfeggio
#define Do  131
#define Re  147
#define Mi  165
#define Fa  175
#define Sol 196
#define La  220
#define Ti  247
#define Do2 262

// Defining the pin connected to the speaker
int tonePin = 13;

void setup(){
```

```
    // Tone pins don't need to be declared
}

void loop(){
  // Do
  tone(tonePin, Do, 125);
  delay(125);
  // Re
  tone(tonePin, Re, 125);
  delay(125);
  // Mi
  tone(tonePin, Mi, 125);
  delay(125);
  // Fa
  tone(tonePin, Fa, 125);
  delay(125);
  // Sol
  tone(tonePin, Sol, 125);
  delay(125);
  // La
  tone(tonePin, La, 125);
  delay(125);
  // Ti
  tone(tonePin, Ti, 125);
  delay(125);
  // Higher Do
  tone(tonePin, Do2, 125);
  delay(125);
}
```

If the speaker is connected to a different pin, simply change the `tonePin` value to the value of the pin that has been used.

How it works...

The `tone()` function is very easy to use. It generates a square wave of 50% duty cycle at the specified frequency. What does that mean? It means that the used pin will be HIGH half the time and LOW half the time. It will change between these two states at the specified frequency.

Every musical note has a specific frequency; in our case, Do, which is a C3, has the frequency of 131 Hz. This wave will make the speaker vibrate and generate sound. Arduino can only support monophonic sound using the Tone function. This means it can only generate one note at a time. Still, it is quite useful and fun. Now on to the code breakdown!

Code breakdown

The code simply uses the built-in tone function, which has the following parameters:

```
tone(pin, frequency, duration)
```

First, we declare the used pin:

```
int tonePin = 13;
```

Then, in `loop()`, we simply use the Tone function for each note, one after the other, with a duration of 125 milliseconds on the declared `tonePin`:

```
void loop(){
  // Do
  tone(tonePin, Do, 125);
  delay(125);
…
}
```

We need to make sure we are not calling the `tone()` function again in the following 125 milliseconds, as it will change the frequency.

It would be easier to declare an array containing all the notes and use a `for` loop to play them all:

```
// Array approach
  int solfege[] = {Do, Re, Mi, Fa, Sol, La, Ti, Do2};

  for (int i = 0; i < 8; i++){
    tone(tonePin, solfege[i], 125);
    delay(125);
  }
```

The `tone()` function cannot play sounds under 31 Hz, and on boards other than the Mega, it will interfere with PWM pins 3 and 11.

There's more...

There is a little more functionality in the Tone function. Here are a few more things we can do:

- Tone with no duration
- Tone on multiple pins

Let's see what they are.

Tone with no duration

The `tone()` function has two variants. The one we used plays the note until the time expires or until we use the `tone()` function again, whichever comes first. However, there is a simpler variant that doesn't have the duration parameter; it only contains the pin and the frequency. When we use that function, the note will start playing continuously. In order to stop the note, we need to use the `noTone(tonepin)` function. Here is an example:

```
tone(tonePin, Do);
delay(100);
noTone(tonePin);
```

The `noTone()` function has only one parameter: the pin number. We need to use the same pin number as the one used in our `tone()` function; otherwise, it will not stop the sound and it might interfere with our code.

Tone on multiple pins

The Tone function can only play one note on one pin at a time. However, we can stop playing on a pin and begin playing on another one. In the following example, we play Do on the 12th pin and then Re on the 13th pin. This, of course, requires two speakers:

```
// Do on pin 12
tone(12, Do, 125);
delay(125);
noTone(12);

// Re on pin 13
tone(13, Re, 125);
delay(125);
noTone(13);
```