
Modifying Samples Using Ranges

Barb Ericson
Oct 2010

Learning Goals

- Computing concepts
 - Looping through a range
 - Returning a value from a method
 - Change more than one variable in a for loop
 - Overloading a method
- Processing ranges of Sound values
 - Increase volume in first half and decrease volume in the second half
 - Create a sound clip and splicing sounds together
 - Reversing and mirroring sounds
 - Creating echoes and changing frequencies

Working with Sample Values

- In the last chapter we got an array of SoundSample objects from the sound
 - and manipulated the objects in the array

```
SoundSample[] sampleArray = this.getSamples();
SoundSample sample = sampleArray[0];
sample.setValue(sample.getValue() * 2);
```
- In this chapter we can directly access the sound value from the sound
 - `sound.getSampleValue(int index)`
 - `sound.setSampleValue(int index, int value)`

Increase then Decrease the Volume

- If we want to increase the volume
 - In the first half of a sound
- And decrease it
 - in the second half
- We need to determine the index for halfway through the sound values
 - `int half = this.getLength() / 2;`
- We change the start and stop values on the for loop

Increase and Decrease Method

```
public void increaseAndDecrease()
{
    int half = this.getLength() / 2;
    int value = 0;

    // loop through the first half of the sound
    for (int i = 0; i < half; i++)
    {
        // get the current value
        value = this.getSampleValueAt(i);

        // set the value to 2x the original
        this.setSampleValueAt(i,value * 2);
    }
```

Decrease Second Half

```
// loop through the second half of the sound
for (int i = half; i < this.getLength(); i++)
{
    // get the current value
    value = this.getSampleValueAt(i);

    // set the value to half the original
    this.setSampleValueAt(i,(int) (value * 0.5));
}
}
```

Challenge

- Create a new method `increaseNoneDecrease` which will
 - Increase the volume for the first third of a sound
 - Leave the middle third alone
 - Decrease the volume for the last third

Creating a Sound Clip

- To clip the “This” out of “This is a test”.
 - Determine where the word "This" starts and stops in the sound
 - Using the sound explorer:

```
String file = FileChooser.getMediaPath  
("thisisatest.wav");  
Sound s = new Sound(file);  
s.explore();
```

Finding the End of the This

Play before bar to check

Play Entire Sound Play Before Play After Stop

Play Selection Clear Selection Start Index: N/A Stop Index: N/A

Position the bar

Current Index: 8500 Sample Value: -204

The number of samples between pixels: 100

Get the index

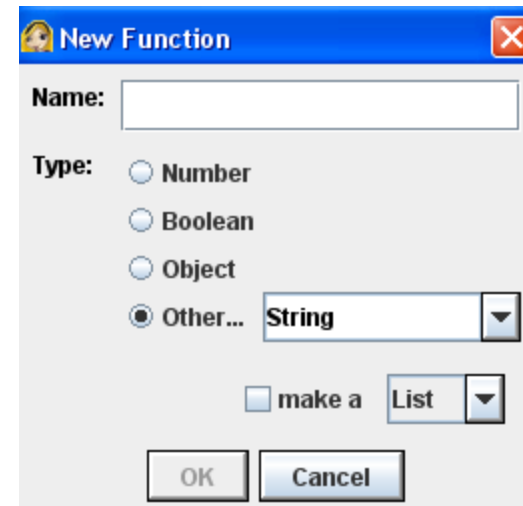
Zoom In

To Create a Sound Clip

- Create a new Sound object
 - Of the appropriate size
 - Ending value – starting value + 1
- Loop from start to end (inclusive)
 - for (int x = start; x <= end; x++)
- Use `getSampleValueAt(index)`
 - And `setSampleValueAt(index,value);`
- Return the new sound object
 - return target;

Returning a value in Alice

- When we created a function in Alice we had to specify the type of value it returned
 - Number, Boolean, Object, etc
- And, we had to use one or more return statements to return the value



```
public Number numberOfRevolutions ( Number 123 distance ) {  
  
    Do Nothing  
  
    return ( distance / ( 3.14 * subject = toyball .getWidth() ) ) ;  
}
```

Returning a Value from a Java Method

- To return a value from a method in Java
 - Include one or more return statements in the body of the method
 - The type of the thing being returned must match the declared return type
 - If the return type doesn't match the thing being returned you will get a compile error
- The clip method return type should be a Sound object
 - The return statement will return the target Sound object

Clip Method

```
public Sound clip(int start,
                  int end)
{
    // calc the num samples
    int numSamples =
        end - start + 1;
    Sound target =
        new Sound(numSamples);
    int value = 0;
    int targetIndex = 0;

    // copy from start to end
    for (int i = start; i <= end;
         i++, targetIndex++)
    {
        value = this.getSampleValueAt(i);
        target.setSampleValueAt(targetIndex,
                                value);
    }
    return target;
}
```

Testing the Clip Method

```
String file = FileChooser.getMediaPath(
    "thisisatest.wav");
Sound s = new Sound(file);
Sound s2 = s.clip(0,8500);
s2.write(
    FileChooser.getMediaPath("this.wav"));
s2.play();
s.explore();
s2.explore();
```

Challenge

- Create a clip of “is” from thisisatest.wav
- Determine where to start and end the clip
- Create the clip
- Write it to a file
 - `sound.write("c://clipls.wav");`

Splicing Sounds Together

- Originally meant cutting the sound tape into segments and then assembling them in the right order
- Easy to do digitally
- Copy more than one sound into a target sound
 - Track the source index and target index

Splice Method

```
public void splice()
{
    Sound sound1 = new Sound(FileChooser.getMediaPath("it.wav"));
    Sound sound2 = new Sound(FileChooser.getMediaPath("is.wav"));
    int targetIndex = 0; // the starting place on the target
    int value = 0;

    // copy all of sound 1 into the current sound (target)
    for (int i = 0;
        i < sound1.getLength();
        i++, targetIndex++)
    {
        value = sound1.getSampleValueAt(i);
        this.setSampleValueAt(targetIndex,value);
    }
}
```

Splice Method - Continued

// create silence between words by setting values to 0

```
for (int i = 0;  
    i < (int) (this.getSamplingRate() * 0.1);  
    i++, targetIndex++) {  
    this.setSampleValueAt(targetIndex,0);  
}
```

// copy all of sound 2 into the current sound (target)

```
for (int i = 0;  
    i < sound2.getLength();  
    i++, targetIndex++) {  
    value = sound2.getSampleValueAt(i);  
    this.setSampleValueAt(targetIndex,value);  
}  
}
```

Testing the Splice Method

```
String silence = FileChooser.getMediaPath(  
    "sec3silence.wav");  
Sound target = new Sound(silence);  
target.explore();  
target.splice();  
target.explore();
```

How does that work?

- There are three loops in the method splice
 - One copies to the target all of the first sound ("it")
 - One creates 0.1 seconds of silence between the sounds
 - One copies in all of the second sound ("is")
- Notice that `targetIndex` is set to 0 and never reset
 - It always keeps track of the next index in the target to copy to

Make Methods more Reusable

- Adding parameters to methods makes them easier to reuse
- Instead of putting the sounds to splice in the splice method
 - We can pass in the sound to add into the current sound
 - Pass in the start and stop (end) values to copy from in the passed sound
 - Pass the index to start the copy at in the current sound (target)

General Splice Method

```
public void splice(Sound source, int sourceStart,
                  int sourceStop, int targetStart)
{
    // loop copying from source to target
    for (int sourceIndex = sourceStart,
         targetIndex = targetStart;
         sourceIndex < sourceStop &&
         targetIndex < this.getLength();
         sourceIndex++, targetIndex++)
        this.setSampleValueAt(targetIndex,
                               source.getSampleValueAt(sourceIndex));
}
```

Method Overloading

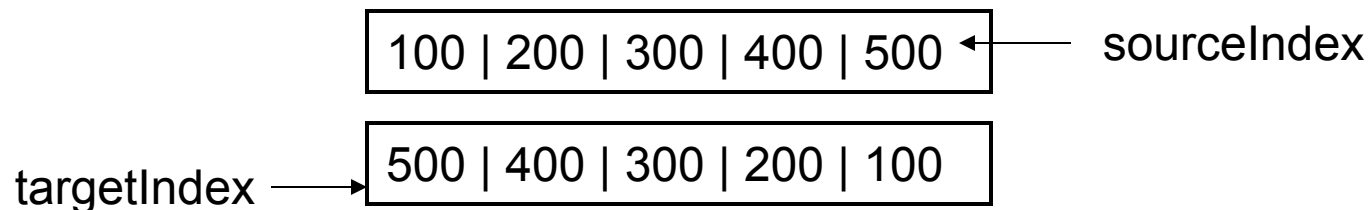
- You can have two methods with the same name like splice
 - As long as the parameter lists are different in some way
 - The number of parameters
 - The types of parameters
 - The order of the types
- The compiler figures out which method you are calling
 - Based on the order and types of arguments

Challenge

- Rewrite the first splice method to call the general splice method
- Use the general splice message to create a sound with three sounds spliced together to make a sentence
 - Like: "it is done".

Reversing a Sound

- To reverse a sound
 - Create a copy of the original sound
 - `Sound orig = new Sound(this.getFileName());`
 - Then loop starting the `sourceIndex` at the last index in the source and the `targetIndex` at the first index in the target
 - Decrement the `sourceIndex` each time
 - Increment the `targetIndex` each time



Reversing Method

```
public void reverse()
{
    Sound orig = new Sound(this.getFileName());
    int length = this.getLength();

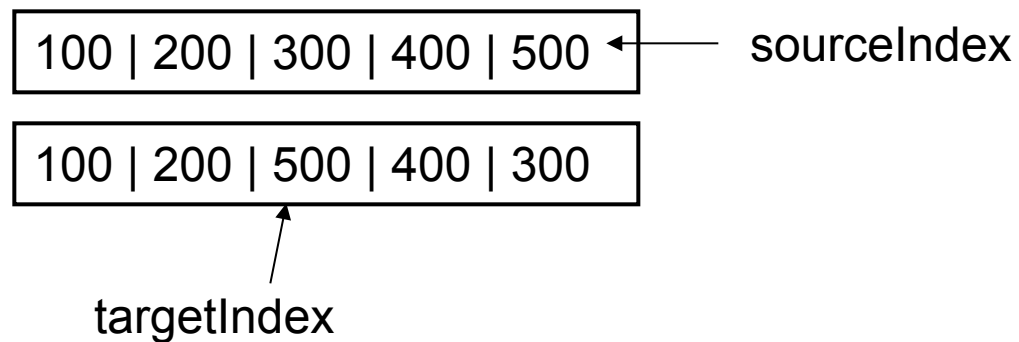
    // loop through the samples
    for (int targetIndex = 0, sourceIndex = length - 1;
        targetIndex < length && sourceIndex >= 0;
        targetIndex++, sourceIndex--)
        this.setSampleValueAt(targetIndex,
                               orig.getSampleValueAt(sourceIndex));
}
```

Testing the Reverse Method

```
String file = FileChooser.getMediaPath(  
    "thisisatest.wav");  
Sound s = new Sound(file);  
s.explore();  
s.reverse();  
s.explore();
```

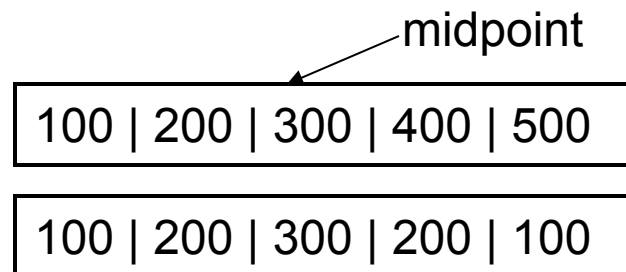
Challenge

- Reverse just the second half of a sound
 - Start the `targetIndex` at the `length / 2`
 - Start the `sourceIndex` at the `length - 1`
 - Loop while the `targetIndex < length`



Mirror a Sound

- Copy the first half of the sound to the second half
 - And reverse the sounds in the second half
 - Calculate the midpoint ($\text{length} / 2$)
 - Start the source index at 0 and copy from index to $\text{length} - \text{index} - 1$
 - While $\text{index} < \text{midpoint}$



Mirror Sound Method

```
public void mirrorFrontToBack()
{
    int length = this.getLength(); // save the length
    int mirrorPoint = length / 2; // mirror around this
    int value = 0; // hold the current value

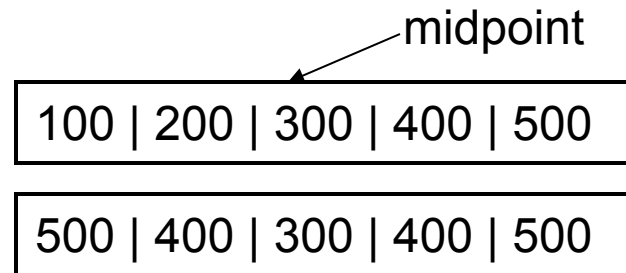
    // loop from 1 to mirrorPoint
    for (int i = 0; i < mirrorPoint; i++)
    {
        value = this.getSampleValueAt(i);
        this.setSampleValueAt(length - i - 1,value);
    }
}
```

Testing Mirror Method

```
Sound s = new Sound  
    (FileChooser.getMediaPath(  
        "croak.wav"));  
s.explore();  
s.mirrorFrontToBack();  
s.explore();
```

Challenge

- Write a method to mirror from the back to the front
 - Copy the back half of the sound reversed to the front



Blend Sounds

- Like blending pictures we can blend two sounds:
 - Copy the first 20,000 values of sound1 into this (the current sound)
 - Create a blend of sound1 and sound2 by ($0.5 * \text{sound1 value} + 0.5 * \text{sound2 value}$) and copying the result into this (the current sound)
 - Copy the next 20,000 values of sound 2 into this (the current sound)

Blend Sounds Method

```
public void blendSounds()
{
    Sound sound1 =
        new Sound(FileChooser.getMediaPath("aah.wav"));
    Sound sound2 =
        new Sound(FileChooser.getMediaPath("bassoon-
c4.wav"));
    int value = 0;

    // copy the first 20,000 samples from sound1 into target
    for (int index=0; index < 20000; index++)
        this.setSampleValueAt(index,
                               sound1.getSampleValueAt(index));
}
```

Blend Sounds - Continued

*// copy the next 20,000 samples from sound1 and blend that
// with the first 20,000 samples from sound2*

```
for (int index = 0; index < 20000; index++)  
{  
    value = (int) ((sound1.getSampleValueAt(index + 20000) *  
                    0.5) +  
                    (sound2.getSampleValueAt(index) * 0.5));  
    this.setSampleValueAt(index + 20000,value);  
}
```

// copy the next 20,000 samples from sound2 into the target

```
for (int index=20000; index < 40000; index++)  
    this.setSampleValueAt(index + 20000,  
                           sound2.getSampleValueAt(index));  
}
```

Testing Blend Sounds

```
String fileName =  
    FileChooser.getMediaPath(  
        "sec3silence.wav");  
Sound target = new Sound(fileName);  
target.explore();  
target.blendSounds()  
target.explore();
```

Challenge

- Write a method to blend two sounds together
 - Start with 10,000 samples from sound1
 - Then add 0.75 of sound1 to 0.25 of sound2 for 10,000 samples
 - Then do 0.5 times sound1 plus 0.5 times sound2 for 10,000 samples
 - Then do 0.25 times sound1 plus 0.75 of sound2 for 10,000 samples
 - Then have 10,000 samples from sound2

Echo

- By blending a sound with itself we can make an echo
 - Add the sound values to the same sound but after some delay

Method echo

```
public void echo(int delay)
{
    // make a copy of the original sound
    Sound s = new Sound(this.getFileName());
    int value = 0;

    // loop from delay to end of sound
    for (int i = delay; i < this.getLength(); i++)
    {
```

Method echo continued

```
/* get the value back by delay samples from the  
 * copy of the sound and make it fainter */  
value = (int) (s.getSampleValueAt(i-delay) * 0.6);
```

```
/* set the value at the current index to the sum  
 * of the current value and the echo */
```

```
this.setSampleValueAt(i,  
                      this.getSampleValueAt(i) +  
                      value);
```

```
}
```

```
}
```

Test the echo method

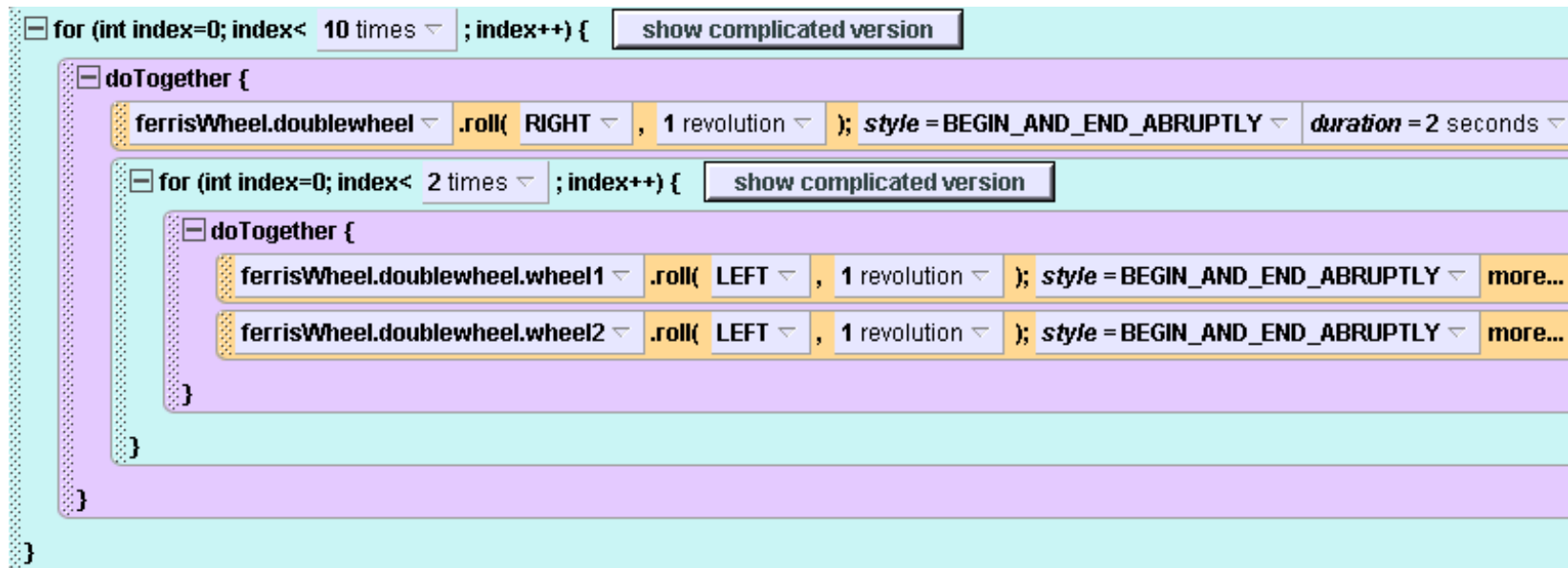
```
String fileName =  
    FileChooser.getMediaPath(  
        "thisisatest.wav");  
Sound sound = new Sound(fileName);  
sound.explore();  
sound.echo(20000);  
sound.explore();
```

How does that work?

- The echo method takes an integer delay
 - The number of samples to wait before starting the echo
- It makes a copy of the current sound
 - So that we copy without the echo
- Loop starting from delay to the end of the sound
 - Get the value at i-delay and multiply it by 0.6 to make it quieter
 - Set the current sound value at i to the current value at i plus the calculated value

Creating Multiple Echoes

- You can use a nested loop to create multiple echoes
 - Here is an example nested loop in Alice



Creating Multiple Echoes Method

```
public Sound echo(int delay, int numEchoes)
{
    int soundLength = this.getLength();
    Sound echoSound = new Sound(numEchoes * delay + soundLength);
    int value = 0;
    int echoIndex = 0;
    int echoValue = 0;
    double echoAmplitude = 1; // to start

    // copy the original sound
    echoSound.splice(this,0,soundLength,0);

    /* loop starting with 1 to create the first echo at the
       * right place and end when = the number of echoes
       */
}
```

Multiple Echoes - continued

```
for (int echoCount = 1; echoCount <= numEchoes; echoCount++)  
{  
    // decrease the volume (amplitude) of the echo  
    echoAmplitude = echoAmplitude * 0.6;  
  
    // echo the whole sound  
    for (int i = 0; i < soundLength; i++)  
    {  
        echoIndex = i + (delay * echoCount);  
        echoValue = (int) (this.getSampleValueAt(i) * echoAmplitude);  
        echoSound.setSampleValueAt(echoIndex,echoValue +  
            echoSound.getSampleValueAt(echoIndex));  
    }  
}  
return echoSound;  
}
```

Try Multiple Echoes

```
Sound s = new Sound(  
    FileChooser.getMediaPath("croak.wav"));  
Sound e = s.echo(8000,5);  
e.play();
```

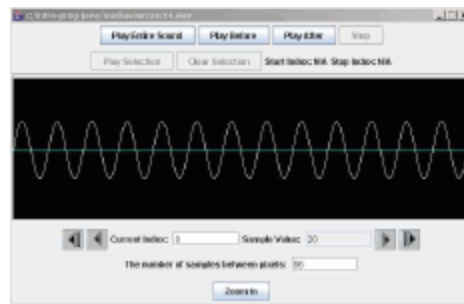
Changing the Sound Frequency

- The *frequency* of a wave is the number of cycles per second (cps), or *Hertz (Hz)*
 - (Complex sounds have more than one frequency in them.)
- Our perception of pitch is related (logarithmically) to changes in frequency
 - Higher frequencies are perceived as higher pitches
 - We can hear between 5 Hz and 20,000 Hz (20 kHz)
 - A above middle C is 440 Hz

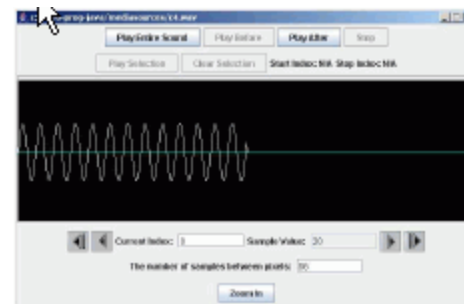


Double the Frequency

- If we take every other sample we double the frequency of the sound
 - Completes two cycles instead of one in the same time
 - It will sound higher



100 | 200 | 300 | 400 | 500



100 | 300 | 500 | 0 | 0

Double Frequency Method

```
public void doubleFreq()
{
    // make a copy of the original sound
    Sound s = new Sound(this.getFileName());

    /* loop and increment target index
    * by one but source index by 2,
    * and set target value
    * to the copy of the original sound
    */
```

Double Frequency - Continued

```
for (int sourceIndex=0, targetIndex = 0;  
    sourceIndex < this.getLength();  
    sourceIndex=sourceIndex+2, targetIndex++)  
    this.setSampleValueAt(targetIndex,  
                           s.getSampleValueAt(sourceIndex));
```

```
// clear out the rest of this sound to silence (0)
```

```
for (int i = this.getLength() / 2;  
     i < this.getLength();  
     i++)  
    this.setSampleValueAt(i,0);
```

```
}
```

Test Double Frequency

```
Sound s = new Sound  
    (FileChooser.getMediaPath(  
        "c4.wav"));  
s.explore();  
s.doubleFreq();  
s.explore();
```

Challenge

- Create a method that will take every third sample
 - Will this sound higher or lower?
- Can you make this method more general?
 - By passing in the amount to add to the source index each time?

Halving the Frequency

- We can copy each source value twice to half the frequency
 - Only get through half a cycle in the same time we used to get through a full cycle
 - It will sound lower

100		200		300		400		500
-----	--	-----	--	-----	--	-----	--	-----

100		100		200		200		300
-----	--	-----	--	-----	--	-----	--	-----

Halve Frequency Method

```
public void halveFreq()
{
    // make a copy of the original sound
    Sound s = new Sound(this.getFileName());

    /* loop through the sound and increment target index
     * by one but source index by 0.5 and set target value
     * to the copy of the original sound
     */
    for (double sourceIndex=0, targetIndex = 0;
        targetIndex < this.getLength();
        sourceIndex=sourceIndex+0.5, targetIndex++)
        this.setSampleValueAt((int) targetIndex,
            s.getSampleValueAt((int) sourceIndex));
}
```

Testing Halve Frequency

```
Sound s = new Sound  
    (FileChooser.getMediaPath(  
        "c4.wav"));  
s.explore();  
s.halveFreq();  
s.explore();
```

Challenge

- Write a method that will copy each sound value 4 times to the target
 - Will the new sound be higher or lower?
- Can you make this more general?
 - By passing in the number of times to copy the source value
 - Try it with 3 times and check the index values to make sure that you are doing it right

Summary

- You can work with ranges by changing the start and end value of an index in a loop
- You can return a value from a method
 - Declare the return type to match the thing being returned and use one or more return statements
- You are not required to change a variable in a for loop
 - But, you can change more than 1
- You can have more than one method with the same name (overloading)
 - As long as the parameter list is different