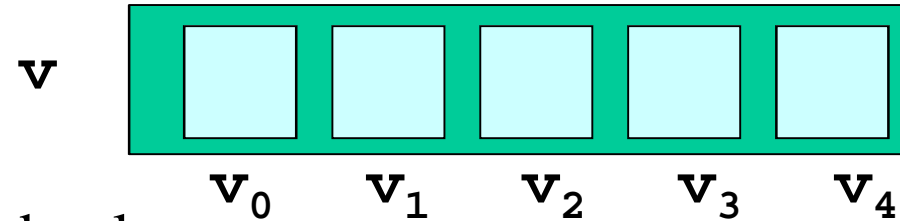


# Arrays

Arrays are objects.



Arrays are a **collection** of same typed values.

Individual values in the array are referenced with `<name>[<index>]`  
 where the index is an int value ranging from 0 to arraySize -1

```
<type> [] < array name > = new < type > [< length >];
```

```
<type> [] < array name > ; // size allocation deferrable
```

```
int[] xCoord = new int[MAX];
```

Or

```
int[] yCoord;
```

```
yCoord = new int[MAX]; // 0 <= subscript < MAX
```

```
yCoord[0] = 30;
```

```
yCoord[MAX] = MAX; // illegal out of bounds
```

Subscript or index, is the reference to a single value in the array  
`<array name>.length` – array object's length instance variable, its size

## Declaring and initializing arrays

```
Color[] colors = {Color.red, Color.green, Color.blue};  
double [] values = {5.0, 6.2, 8.3, 2.7, 3.1};
```

Declare first and later allocate and initialize (usually in constructor)

```
public class Sample {  
    int[] data;  
  
    public Sample(int max) {  
        Random aRandom = new Random();  
        data = new int[max]; // don't re-type  
        for(int i = 0; i < data.length; i++)  
            data[i] = aRandom.nextInt(100);  
    }  
    public boolean exists(int target) {....}  
  
    public static void main() {  
        Sample sample = new Sample(1000);  
        System.out.println("20 in sample " +  
            sample.exists(20));  
    }  
}
```

## Arrays && loops

For loops are often used to process each element in an array.

"iterate through the loop"

```
for(int i = 0; i < data.length; i++) data[i] = aValue ;
```

or the "for each" variation w/ collections

```
for ( <type> <variable> : <array> ) stmt ;
```

```
for(int value : data) value = aValue;
```

While loops can be used to search for the first occurrence of a value in an array. Can you write the following method with a while loop?

```
public boolean exists(int target) {  
    for (int i = 0; i < data.length; i++)  
        if (data[i] == target) return true;  
    return false; // not found condition  
}
```

```
public class Stats {
    static final int MAX = 5; // a constant
    int[] v;

    public static void main(String[] args) {
        Stats app = new Stats();
        app.show(); // call show() method
        app.sort(); // call sort() method
        System.out.println("min = " + app.v[0]);
        System.out.println("average = " + app.avg());
        System.out.println("max = " + app.v[MAX-1]);
        app.show();
    }

    public Stats() { // Constructor, allocate & initialize v
        int i;
        v = new int[MAX];
        for(i = 0; i < MAX; i++)
            v[i] = (int) (Math.random() * MAX + 1);
    }
}
```

```
public void show() {
    int i;
    System.out.println("The array is");
    for(i = 0; i < MAX; i++)
        System.out.print(v[i] + " ");
    System.out.println();
}

public double avg() {
    // How would you design and implement this method
    // assume the correct double is returned.
}

public void sort() {
    int i, j, min, temp;
    for(i = 0; i < MAX -1; i++) {
        min = i;
        for(j = i + 1; j < MAX; j++)
            if (v[j] < v[min]) min = j;
        temp = v[i]; // need temp variable to swap 2 values
        v[i] = v[min];
        v[min] = temp;
    }
}
}
```

The array is

5 5 5 3 3

min = 3

avg = 4.2

max = 5

The array is

3 3 5 5 5

```

for i = 0; i < MAX - 1; i++ {
    min = i
    for j = i + 1; j < MAX; j++
        if (v[j] < v[min]) min = j
    temp = v[i]
    v[i] = v[min]
    v[min] = temp
}

```

i	0											1		
j	1		2		3			4					2	
min	0						3							1
v[j] < v[min]		F		F		T			F					
temp									5					
v[0]	5									3		3		
v[1]	5											5		
v[2]	5											5		
v[3]	3										5	5		
v[4]	3											3		

## allExchangeSort

Algorithms have execution (performance) characteristics. Consider the following sorting algorithm.

### allExchangeSort

```
int [] v    // assume v is filled with random numbers
for (i = 0; i < v.length; i++)    // n times
    for (j = 0; j < v.length; j++) // n times
        if (v[i] < v[j]) {
            t = v[i]    // exchange values
            v[i] = v[j]
            v[j] = t }

```

allExchangeSort's performance is characterized by:

n	its size or v.length
comparisions	the count of comparisons made if (v[i] < v[j])
swaps	the count of times 2 value's exchange positions

One would expect around

$n^2$ comparisons	each n is compared n times
$n^2/4$ swaps	$1/2$ odds values are swapped, compared twice $\approx 1/4$

## exchangeSort

The allExchangeSort can be improved by reducing the number of duplicate comparisons.

### exchangeSort

```
int [] v    // assume v is filled with random numbers
for (i = 0; i < v.length - 1 ; i++)          // n - 1
    for (j = i + 1; j < v.length; j++)      // n/2
        if (v[i] < v[j])
            swap(v[i], v[j]) // exchange values
```

One would expect around

$n^2/2 - n/2$  comparisons

$n^2/4$  swaps       $1/2$  odd values are swapped, compared twice  $\approx 1/4$

## insertionSort

The exchangeSort can be improved by reducing the number of swaps

`insertionSort`

```
int [] v    // assume v is filled with random numbers
for (i = 0; i < v.length - 1 ; i++) {    // n - 1
    min = i
    for (j = i + 1; j < v.length; j++)    // n/2
        if (v[min] < v[j]) min = j
    swap(v[i], v[min])    // n - 1
}
```

One would expect around

$n^2/2 - n/2$  comparisons  
 $n - 1$  swaps

For 1,000 random values	comparisons	swaps (some duplicates)
allExchange	1,000,000	249,532
exchange	499,500	236,795
insertion	499,500	999

## main's arguments

Arrays have an instance variable length that holds the number of elements in the array.

```
public class SeeArgs {
    public static void main (String[] args) {
        int i, n;
        n = args.length;
        System.out.print("Arguments to main are:\n\t");
        for(i = 0; i < n; i++)
            System.out.print(args[i] + " ");
        System.out.println();
    } }
```

```
C:\ java SeeArgs the lazy brown fox
Arguments to main are:
    the lazy brown fox
```

Use `Integer.parseInt(aString)` or  
`Double.parseDouble((aString))` to convert string value to integer or double

## Deck of playing cards

Problem: represent a deck of 52 playing cards.

suit? value? shuffle?

deal? -- depends upon the game.

```
public class Cards {  
  
    public static final int MAX = 52;  
    int[] deck;  
  
    public static void main (String[] args) {  
        Cards app = new Cards();  
        app.show();  
        app.shuffle();  
        app.show();  
    }  
  
    public Cards() { // initialize card deck  
        int i;  
        deck = new int[MAX];  
        for (i = 0; i < MAX; i++) deck[i] = i;  
    }  
}
```

# Shuffle cards

```
private void shuffle() {
    int i, temp, rn;
    System.out.println("Shuffle deck");
    for (i = 0; i < MAX; i++) {
        // random value 0..51
        rn = (int) (Math.random() * MAX);
        // swap values
        temp = deck[i];
        deck[i] = deck[rn];
        deck[rn] = temp;
    }
}
```

trace of shuffle algorithm

array	i	rn
1 2 3 4 5	0	3
4 2 3 1 5	1	0
2 4 3 1 5	2	4
2 4 5 1 3	3	0
1 4 5 2 3	4	1
1 3 5 2 4		

## Card's value

```
void show() {
    int i, value;
    System.out.println("Card deck is ");
    for (i = 0; i < MAX; i++) {
        // determine card
        value = deck[i] % 13;
        switch (value) {
            case 9 : System.out.print("J"); break;
            case 10 : System.out.print("Q"); break;
            case 11 : System.out.print("K"); break;
            case 12 : System.out.print("A"); break;
            default : System.out.print(value + 2); // 0..8
        }
    }
}
```

The above representation is fine for printing out values and making comparisons .

These "values" don't follow normal rankings for “numbered cards”  
where: values 2..10 integer value instead of 0..8

# Card's suite

```
// determine suit
switch (deck[i] / 13) {
    case 0 : System.out.print("-C"); break;
    case 1 : System.out.print("-D"); break;
    case 2 : System.out.print("-H"); break;
    case 3 : System.out.print("-S");
}
// format println
if (i == 12 || i == 25 || i == 38 || i == 51)
    System.out.println();
else
    System.out.print(" ");
}
}
}
```

## run of program

```
> java Cards
```

```
Card deck is
```

```
2-C 3-C 4-C 5-C 6-C 7-C 8-C 9-C 10-C J-C Q-C K-C A-C
```

```
2-D 3-D 4-D 5-D 6-D 7-D 8-D 9-D 10-D J-D Q-D K-D A-D
```

```
2-H 3-H 4-H 5-H 6-H 7-H 8-H 9-H 10-H J-H Q-H K-H A-H
```

```
2-S 3-S 4-S 5-S 6-S 7-S 8-S 9-S 10-S J-S Q-S K-S A-S
```

```
Shuffle deck
```

```
Card deck is
```

```
9-C 2-S 5-H 6-C 5-D A-H 2-D Q-C 6-S A-D 8-C K-H 7-C
```

```
7-H J-D Q-H A-S K-C A-C 10-S 8-S Q-S 6-H 4-D 10-H 9-S
```

```
2-H 5-C 6-D 7-S Q-D 3-C 7-D 10-D J-C 3-D 5-S 9-D 8-H
```

```
4-H 4-C 9-H 3-H K-S 8-D 3-S J-H 4-S J-S 10-C K-D 2-C
```

```
>
```

## Multidimensional Arrays

A table or spreadsheet is an example of a 2 dimensional array.  
row and columns represent the dimensions

```
public static final int ROWS = 3, COLUMNS = 2;
int [][] table = new int[ROWS][COLUMNS];
...
for (i = 0; i < ROWS; i++)
    for (j = 0; j < COLUMNS; j++) table[i][j] = 0;
```

OR

```
int[][] table = {{0, 1}, {2, 3}, {4, 5}};
// an array of 3 cells, each an array of 2 ints
...
for (i = 0; i < ROWS; i++) {
    for (j = 0; j < COLUMNS; j++)
        System.out.print(table[i][j] + " ");
    System.out.println();
}
```

run:

0	1
2	3
4	5

2 D array is a 1 D array where each cell is a 1 D array

3 D array is a 1 D array where each cell is a 2 D array

3D arrays can be used to represent points in 3D space (x, y, z).

1D arrays can represent vectors.

Multidimensional arrays can represent matrices.

Assume you wanted to use arrays to represent squares that will be drawn in a 2D graphics program. The squares could have an x and y position, a size, and a color.

```
import java.awt.Graphics; // graphics
int[] x, y, size;
Color[] color;
Graphics g; // graphics context
... // assume g exists
// draw nSquare squares
for(i = 0; i < nSquares; i++) {
    g.setColor(color[i]);
    g.fillRect(x[i], y[i], size, size);
}
```

The index "links" values in the arrays together. To define attributes of a square.

x	y	s	c
10	10	10	red
40	50	20	blue
70	20	40	green

## Simulate rolls of 2 dice – 1000 rolls

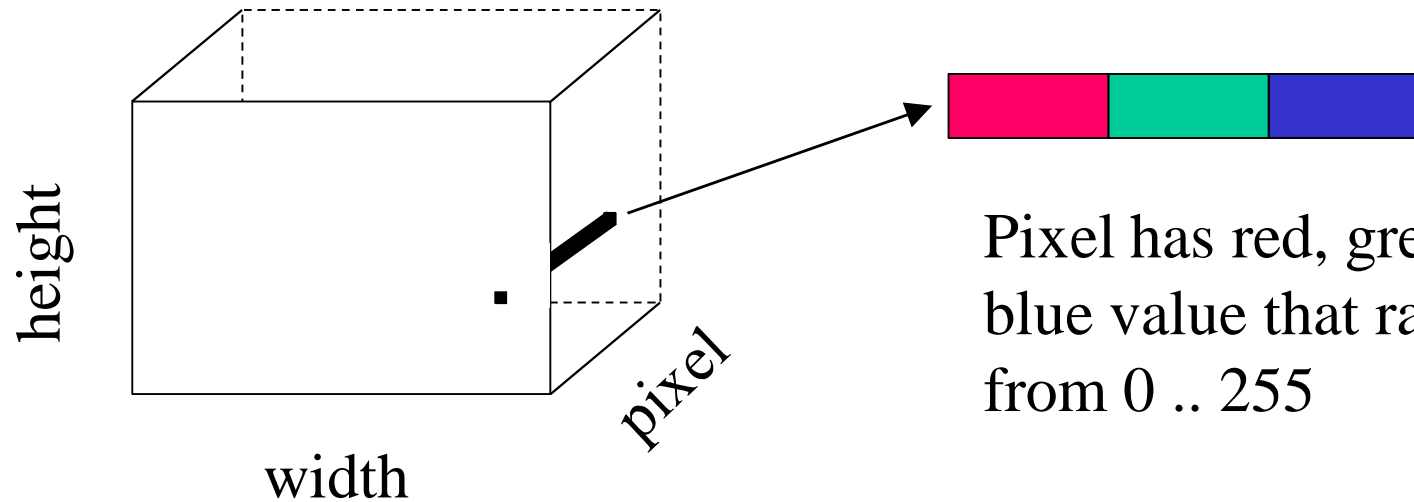
```
public class Dice {  
    public static void main (String[] args) {  
        int i, j, die1, die2;  
        int[][] rolls = new int[6][6];  
  
        for (i = 0; i < 6; i++)  
            for (j = 0; j < 6; j++) rolls[i][j] = 0;  
  
        for(i = 0; i < 1000; i++) {  
            die1 = (int) (Math.random() * 6);  
            die2 = (int) (Math.random() * 6);  
            rolls[die1][die2]++; }  
  
        System.out.println("\t 1 \t 2 \t 3 \t 4 \t 5 \t 6");  
        for (i = 0; i < 6; i++) {  
            System.out.print(i+1 + "\t");  
            for (j = 0; j < 6; j++)  
                System.out.print(rolls[i][j]/1000.0 + "\t");  
            System.out.println(); }  
    }  
}
```

	1	2	3	4	5	6
1	0.039	0.032	0.026	0.03	0.026	0.021
2	0.031	0.027	0.023	0.032	0.027	0.02
3	0.024	0.023	0.034	0.027	0.027	0.028
4	0.023	0.033	0.036	0.029	0.026	0.023
5	0.021	0.022	0.033	0.024	0.026	0.024
6	0.038	0.028	0.025	0.03	0.033	0.029

How to compute the probability for 2, 3, .. 12 outcomes ??

2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10
6	7	8	9	10	11
7	8	9	10	11	12

# Pictures and Pixels



Pixel has red, green, and blue value that range from 0 .. 255

Digital pictures have a collection of Pixels in a 2 dimensional array.

<b>red</b>	<b>green</b>	<b>blue</b>	<b>color</b>
0	0	0	black
255	255	255	white
255	0	0	red
0	0	255	blue
128	128	128	grey
255	255	0	yellow

## Picture classes

Using some interesting classes in bookClasses:

<b>FileChooser</b>	get/set media directory and paths
pickAFile()	browser to search for a file
<b>Pixel</b>	references a picture's picture element
getRed()	returns int of a Pixel's red value also getGreen() and getBlue()
setRed(value)	sets a Pixel's red int value also setGreen(v) and setBlue(v)
<b>Picture</b>	digital picture object (extends SimplePicture)
Picture()	constructs empty picture
Picture(name)	constructs picture from file name (*.png, *.jpg)
getPixel(x, y)	returns reference to the Pixel at location (x,y)
getPixels()	returns a Pixel[] <b>referencing</b> Picture's Pixels
getWidth()	returns Picture's int number of width pixels
getHeight()	returns Picture's int number of height pixels

# Picture.explore()

```
public class ExplorePicture {
    Picture aPicture;
    Pixel[] pixel;
    int numberOfPixels;

    public ExplorePicture() {
        FileChooser fc = new FileChooser();
        String fileName = fc.pickAFile();
        aPicture = new Picture(fileName);
        System.out.println("Create picture " + fileName);
        pixel = aPicture.getPixels();
        numberOfPixels = pixel.length;
        System.out.println("Picture has " + pixel.length + " pixels.");
        System.out.println("Explore " + aPicture);
        aPicture.explore();
        for(int i = numberOfPixels/4; i < numberOfPixels/4 * 3; i += 2) {
            pixel[i].setRed(255);
            pixel[i].setBlue(255);
            pixel[i].setGreen(255); }
        aPicture.show();
    }
}
```

Dr Java Interactions pane (*edited to fit on slide*)

```
ExplorePicture ep = new ExplorePicture();
```

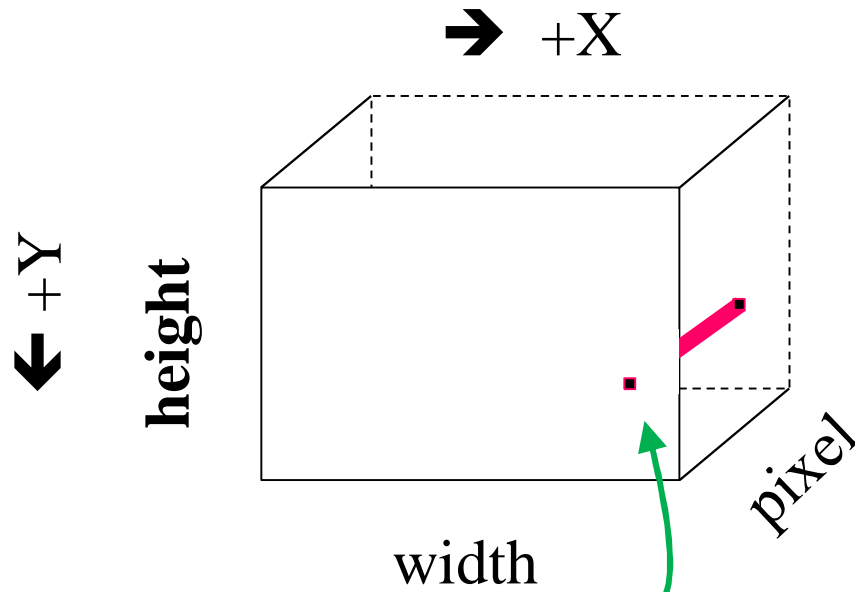
```
Create picture chase.jpg
```

```
Picture has 60000 pixels.
```

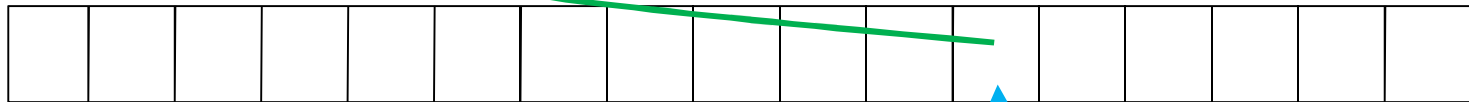
```
Explore chase.jpg height 250 width 240
```



# Reference variables



An assignment to an array of pixels  
 retrieved from a picture  
 is an assignment to the  
 corresponding picture's pixel  
 Pictures are 2D **(not 1D)**  
 arrays of pixels...



```
Picture aPicture;
Pixel[] pixel;
...
pixel = aPicture.getPixels();
...
pixel[i].setRed(255);
```

## Pixel – color methods

The Pixel class (bookClasses) has several Color methods

Color is a class defined in Java's API – the `java.awt.Color.*` package  
Java does not have a Pixel class.

`Color getColor()` returns the color object of the pixel

`void setColor(aColor)` sets the Pixel's rgb values using `aColor`

`double colorDistance(Color aColor)`

returns the distance between the current color and `aColor`  
in the cartesian color 3 dimension space

$$\text{distance} = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$

`double getAverage()`

returns the average of the rgb values  
=  $(r + g + b) / 3.0$

## Sorting pixels

Lets examine sorting an array of pixels

```
Picture picture = new Picture(...);
Pixel [] pixel = picture.getPixels();
...
int min, swaps = 0;
Pixel temp;
for(int i = 0; i < pixel.length -1; i++) {
    min = i;
    for (int j = i + 1; j < pixel.length; j++)
        if (lessThan(pixel[j], pixel[min]) min = j;
if (min != i)
    swapPixel(i, min);
...

```

How could we compare pixels?

How could we swap pixels?

## swapping pixels

```
public void swapPixel(int index, int min) {  
    Pixel temp = null;  
    temp = pixel[index];  
    pixel[index] = pixel[min];  
    pixel[min] = temp;  
}
```

The above swapPixel(...) method will not "sort" the picture's pixels.

Why ?

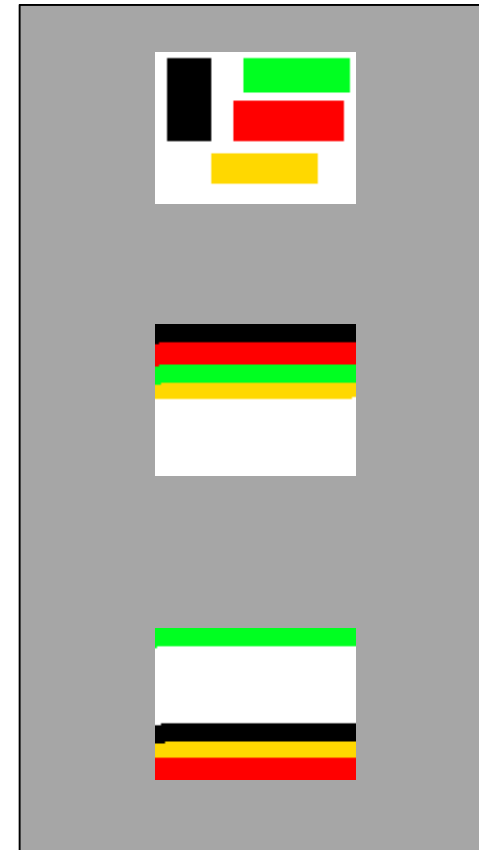
How can we write swapPixel(...) so that the picture's pixels are sorted?

## sorting pixels

Consider sorting the image test.png

using `Pixel.getAverage()`

using `Pixel.getColorDistance(Color aColor)`  
where `aColor` is `java.awt.Color.cyan`



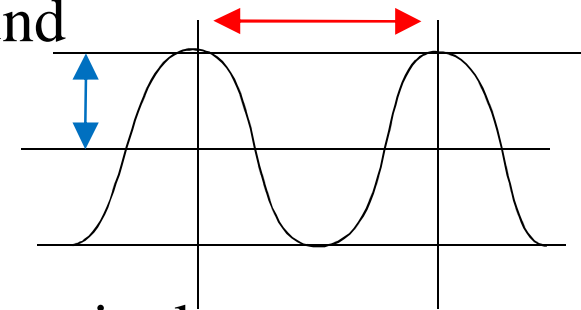
# Sound

Sound is another multimedia data collection that can be processed with algorithms (chapter 8).

Sound is a wave – it has a frequency, Hz (**cycles** / second)

The frequency is the note or pitch of the sound

256 hz is middle C on a piano



Sound has a format (\*.wav, \*.mp3, \*.au, ...)

Monophonic sound (1 channel) stores its data in a single array

Stereophonic sound (2 channels) uses two arrays.

Analog (continuous) sound is "sampled"

digitized (16 bits per sample) at a sample /second rate

for CD quality sound (22Khz) you need to sample at 44.1 Khz

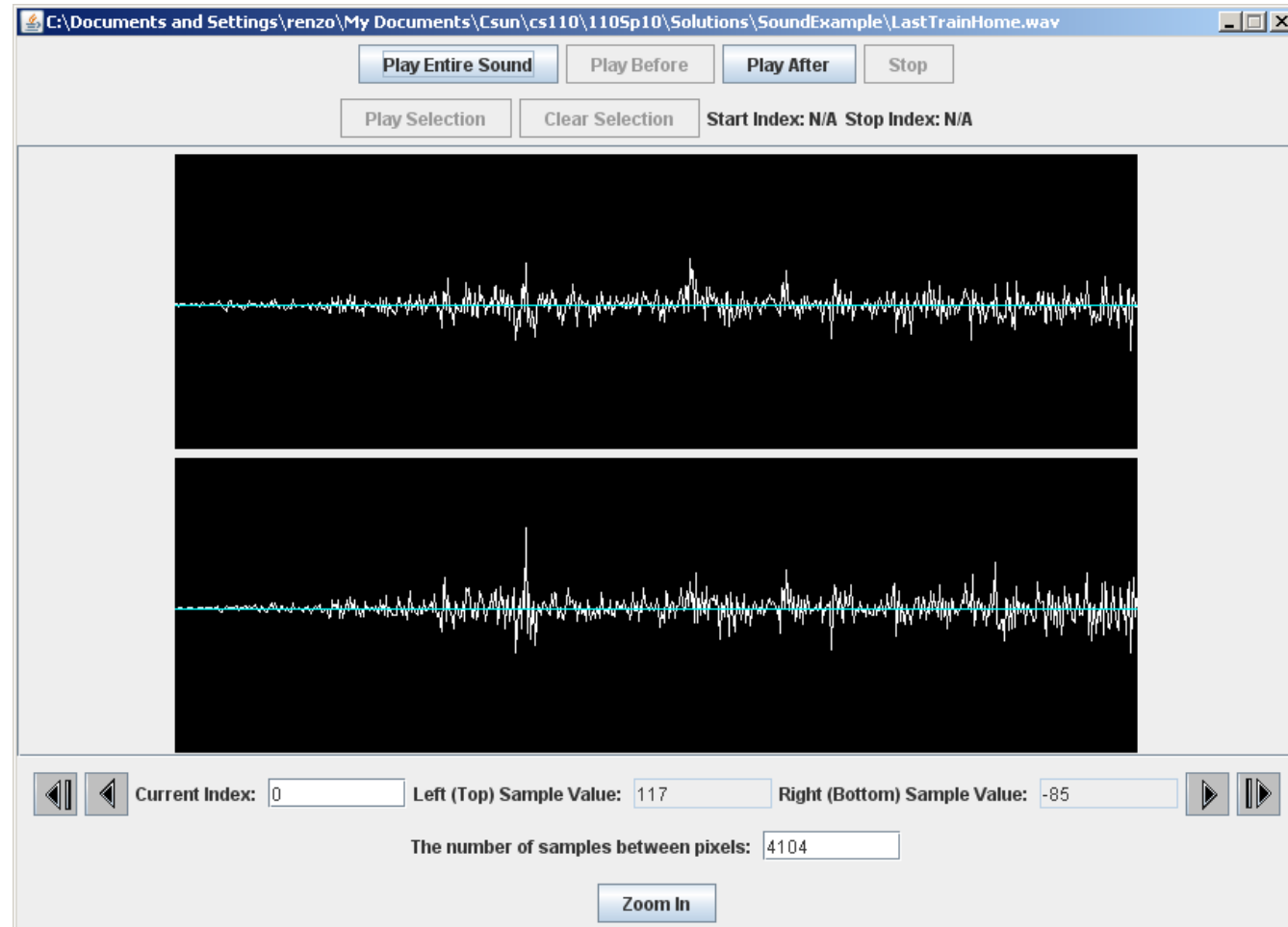
Sound loudness is measured in decibels (db) a logarithmic scale based on the **amplitude** of the sound.

```
import java.text.*; // for DecimalFormat

public class SoundExample {

public SoundExample() {
    String path, soundFile = null;
    path = System.getProperty("user.dir");
    FileChooser.setMediaPath(path);
    soundFile = FileChooser.pickAFile();
    Sound sound = new Sound(soundFile);
    sound.play();
    int length = sound.getLength();
    double rate = sound.getSamplingRate();
    double seconds = length / rate;
    DecimalFormat df = new DecimalFormat("#.##");
    System.out.println("Sound Properties:");
    System.out.println("\t number of channels is " +
        sound.getChannels());
    System.out.println("\t number of samples is " + length);
    System.out.println("\t sampling rate is " + df.format(rate)
        + " times a second");
    System.out.println("\t time to play is " +
        df.format(seconds) + " seconds");
    sound.explore();
}
}
```

# SoundExample



```
> SoundExample se = new SoundExample();
```

```
...
```

```
Sound Properties:
```

```
    number of channels is 2
```

```
    number of samples is 2627001
```

```
    sampling rate is 44100 times a second
```

```
    time to play is 59.57 seconds
```