

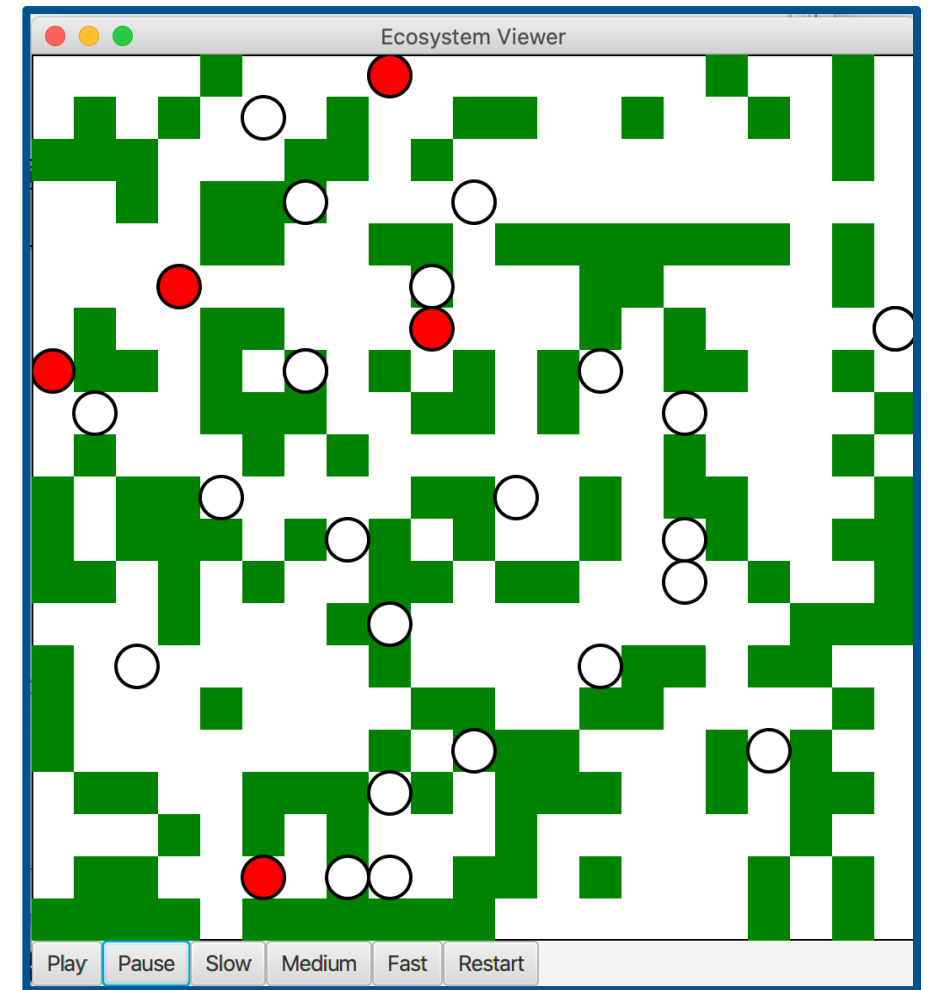
Arrays

Common problem: needing many objects in a program

Imagine you are writing a program to simulate an ecosystem.

- You want several “Fox” objects (red circles),
- A lot of “Rabbit” objects (white circles) that will be eaten by foxes,
- And a lot of “Grass” objects (green squares) that will be eaten by rabbits.

What does this look like in code?



Do we need
separate
variables for
each object?!

```
class Ecosystem
{
    int worldWidth;
    int worldHeight;

    Fox fox0;
    Fox fox1;
    Fox fox2;
    Fox fox3;
    Fox fox4;

    Rabbit rab0;
    Rabbit rab1;
    Rabbit rab2;
    Rabbit rab3;
    Rabbit rab4;
    Rabbit rab5;
    Rabbit rab6;
}
```

There must be a better way...

When you
need many
objects of a
single type,
use an array

```
class Ecosystem
```

```
{
```

```
    int worldWidth;
```

```
    int worldHeight;
```

```
    Fox[] foxes;
```

```
    Rabbit[] rabbits;
```

```
    Grass[] grass;
```

[] makes the datatype 'plural',
and defines an array of that type

```
Ecosystem(Fox[] foxes, Rabbit[] rabbits, Grass[] grass)
```

```
{
```

```
    this.worldWidth = 20;
```

```
    this.worldHeight = 20;
```

```
    this.foxes = foxes;
```

```
    this.rabbits = rabbits;
```

```
    this.grass = grass;
```

```
}
```

```
}
```

Arrays are lists
of a single type

```
class ArrayExamples
{
    int[] myNumbers = {1, 2, 2, 8, -7};

    double[] myOtherNumbers = {3.14, 0.999, 1.0};

    boolean[] quizAnswers = {true, true, false, true, false};

    String[] someWords = {"a", "aah", "aardvark", "aargh", "ab",
        "abacus", "abandon", "abase"};

    Fraction[] fractions = {new Fraction(1, 2), new Fraction(3, 5)};
}
```

Syntax for arrays

```
int[] variableName = {5, 6, 11};
```



Start with a datatype. Any kind of datatype may be used to make an array.

Syntax for arrays

```
int[] variableName = {5, 6, 11};
```

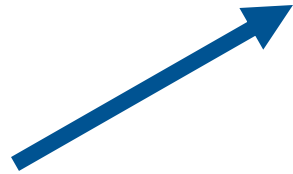


Square brackets indicate that we want to make an array of that datatype.

The brackets look like a box or container.
Think: "a box full of ints."

Syntax for arrays

```
int[] variableName = {5, 6, 11};
```



Any variable name may be used. It is best to use a descriptive plural noun.

Syntax for arrays

```
int[] variableName = {5, 6, 11};
```



Initialize the array with a comma-separated list enclosed in curly brackets.

Arrays may
not have a
mixture of
different
datatypes

```
int[] badArray = {5, 7.3, "hi"};
```



Not allowed!

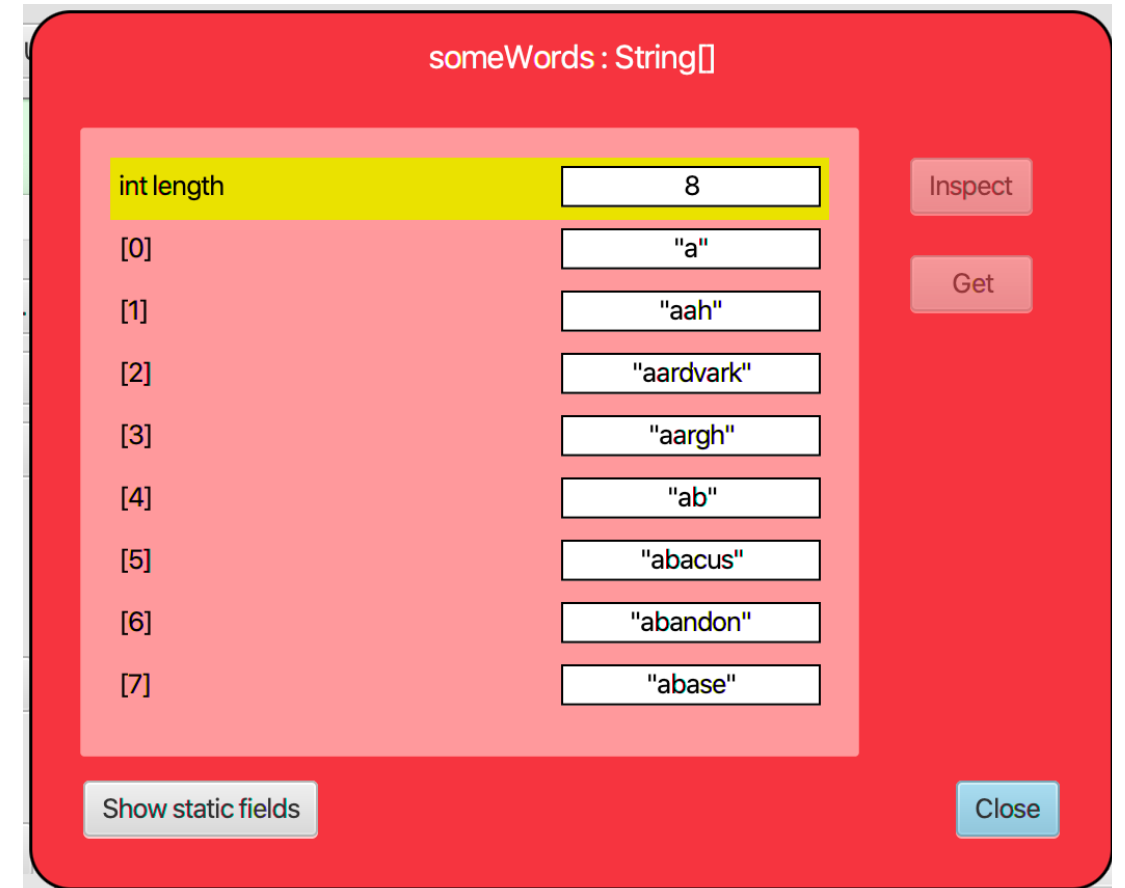
Arrays have “slots” that can be indexed

The **length** of an array is the number of slots it contains.

An **element** of an array is a piece of data inside the array in a particular slot.

An **index** is a number describing the position of the element (starting from zero).

If the length is 8, the last index must be 7 since we start counting from zero.



Arrays have “slots” that can be indexed

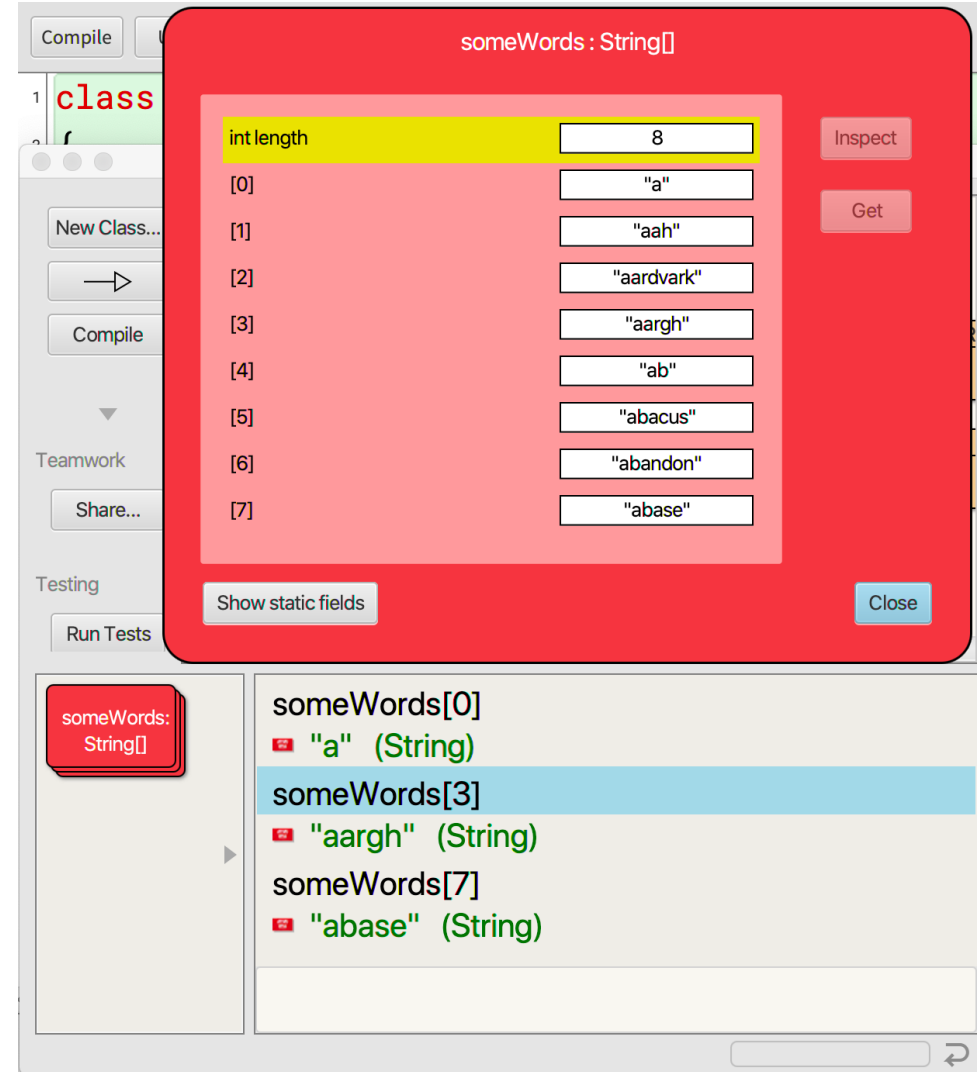
Example:

`someWords` is an array of `String` with **length** 8.

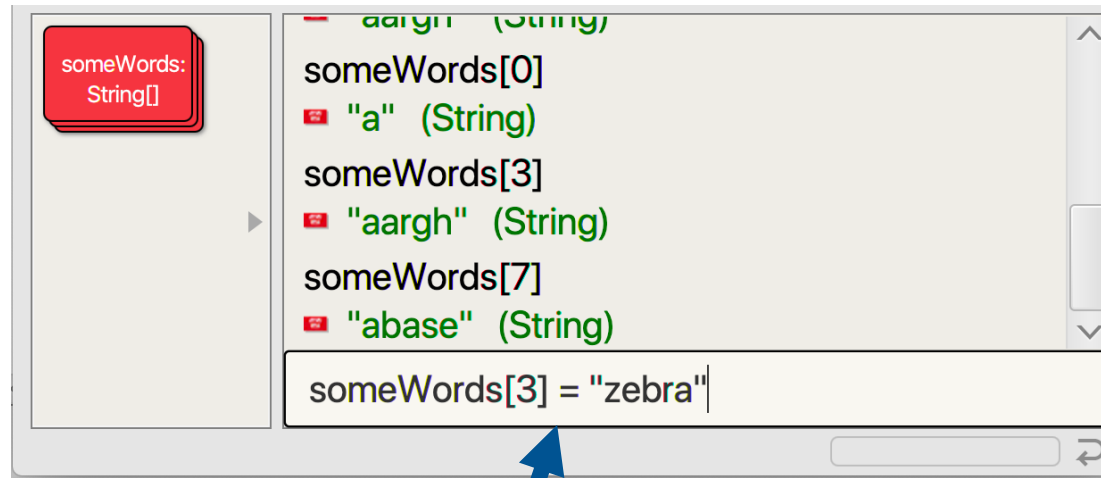
“aargh” is an **element** of `someWords`.

The **index** of “aargh” is 3, although it is the fourth `String` in the array.

“aargh” is bound to the array variable `someWords[ 3 ]`.

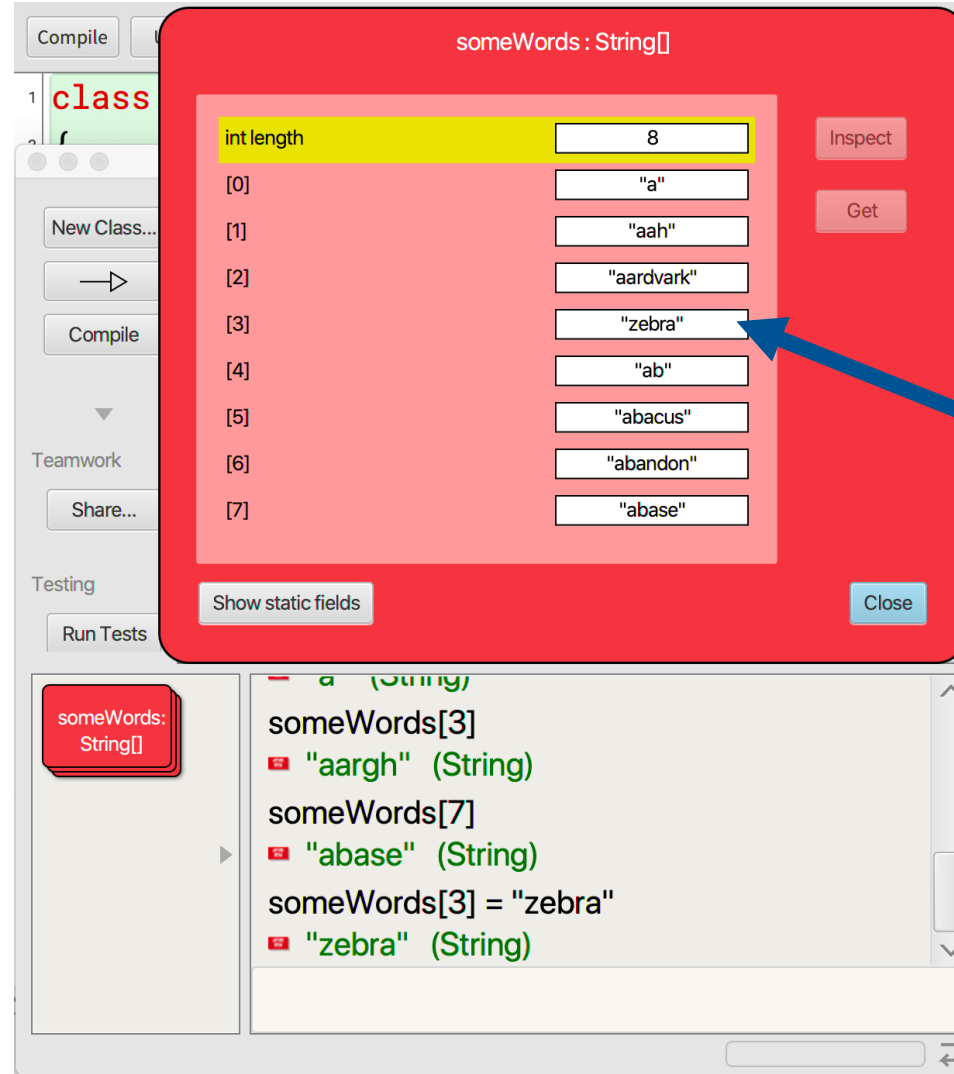


Changing an element in an array



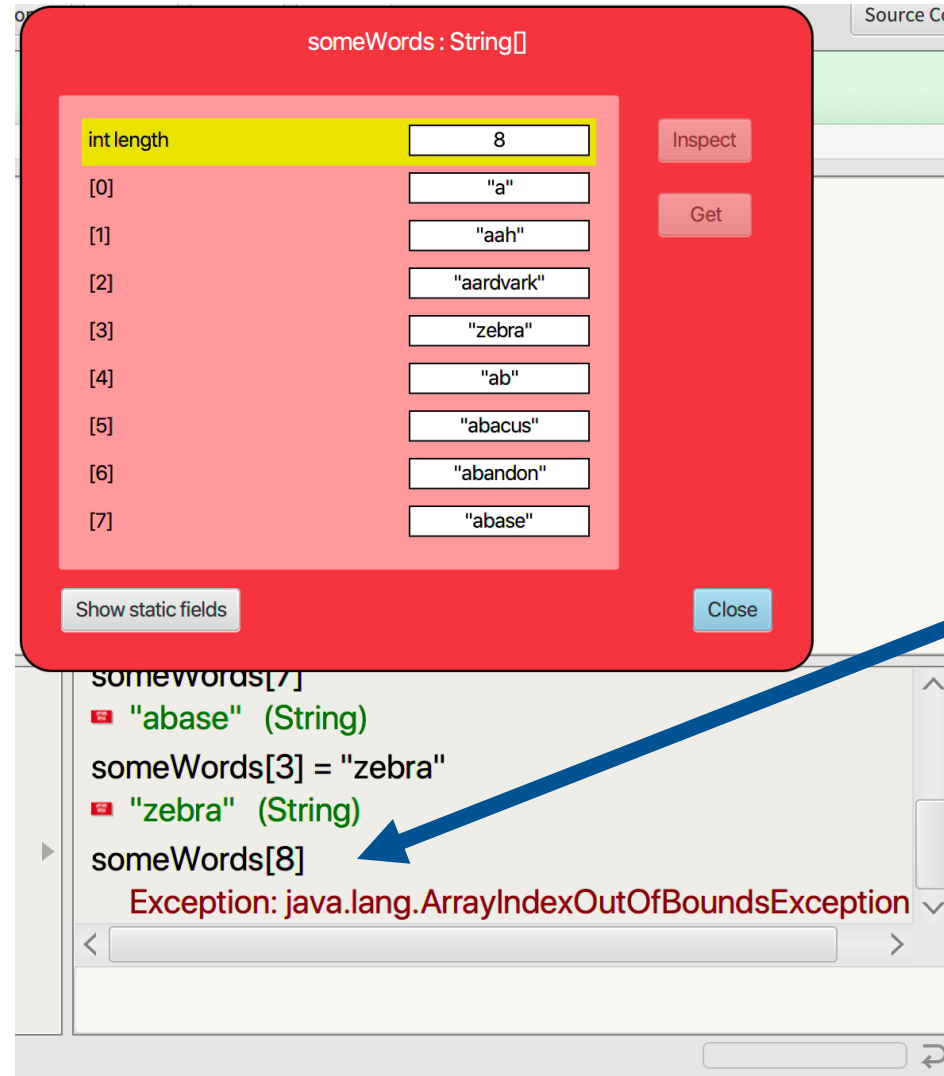
Change what is in an array by setting the element's array variable to a new value

Changing an element in an array



The element at index 3 is now "zebra"

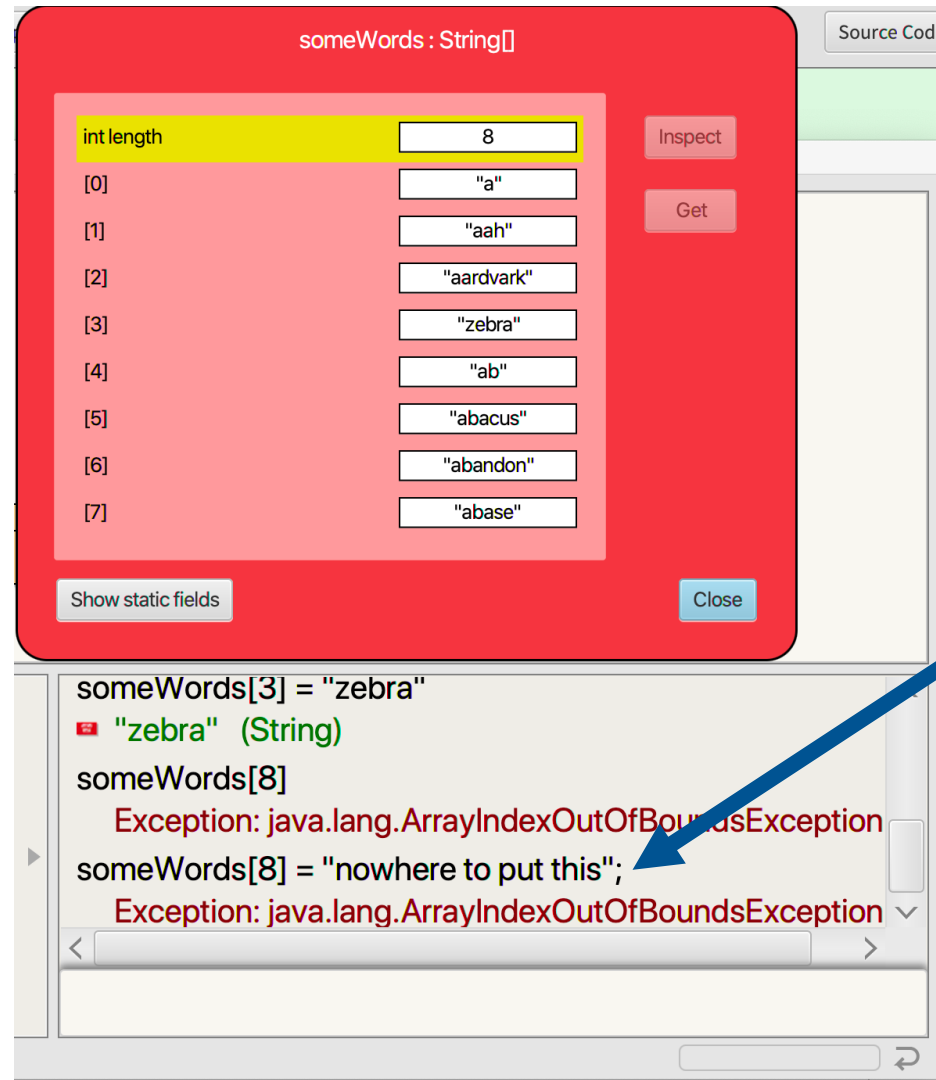
Watch out for
indexing out of
bounds



No element
with index 8
exists!

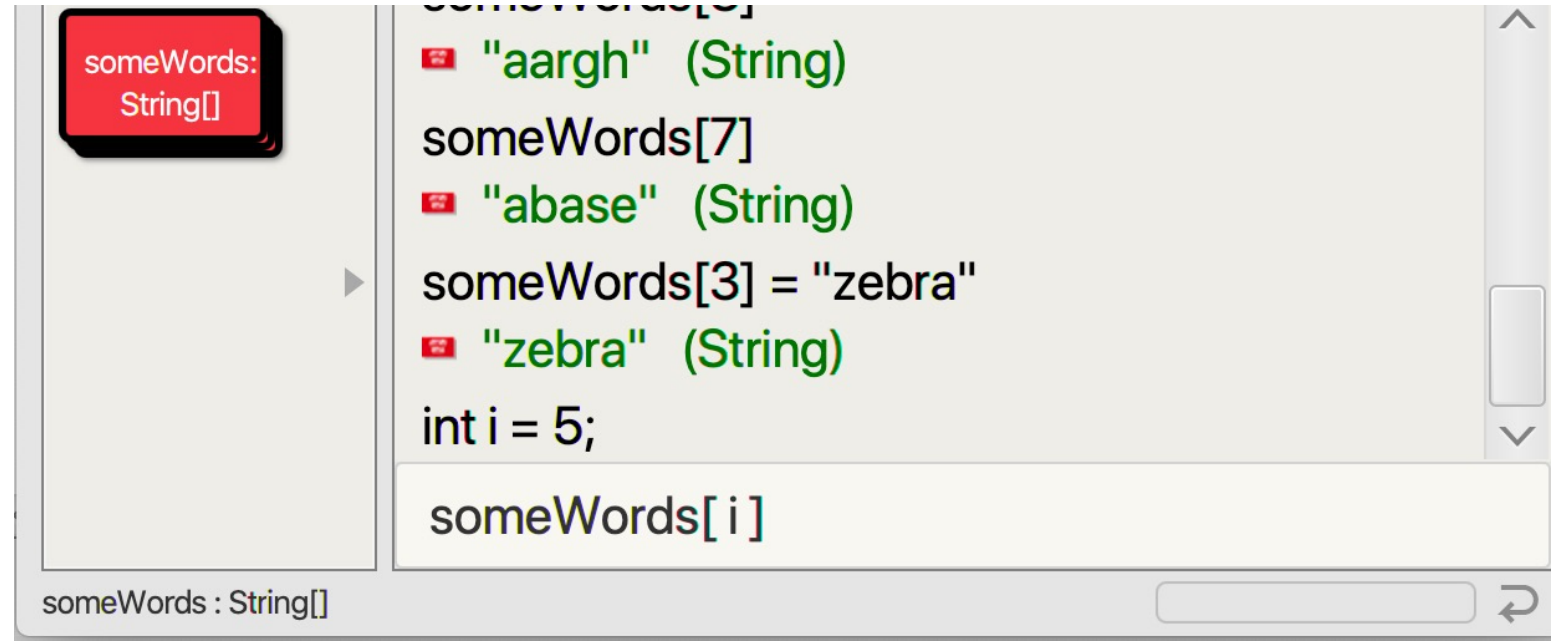
Arrays have a fixed size.

They do not expand or shrink after creation.



Not possible to expand an array or add elements beyond the end.

Useful trick:
you can index
with a variable



The screenshot shows an IDE window with a variable declaration and several array accesses. On the left, a red tooltip displays 'someWords: String[]'. The main editor area contains the following code: 'someWords[0]' (partially visible), '"aargh" (String)', 'someWords[7]', '"abase" (String)', 'someWords[3] = "zebra"', '"zebra" (String)', and 'int i = 5;'. Below the code, a text box shows 'someWords[i]'. The status bar at the bottom indicates 'someWords : String[]'.

```
someWords[0]  
"aargh" (String)  
someWords[7]  
"abase" (String)  
someWords[3] = "zebra"  
"zebra" (String)  
int i = 5;  
someWords[i]
```

someWords : String[]

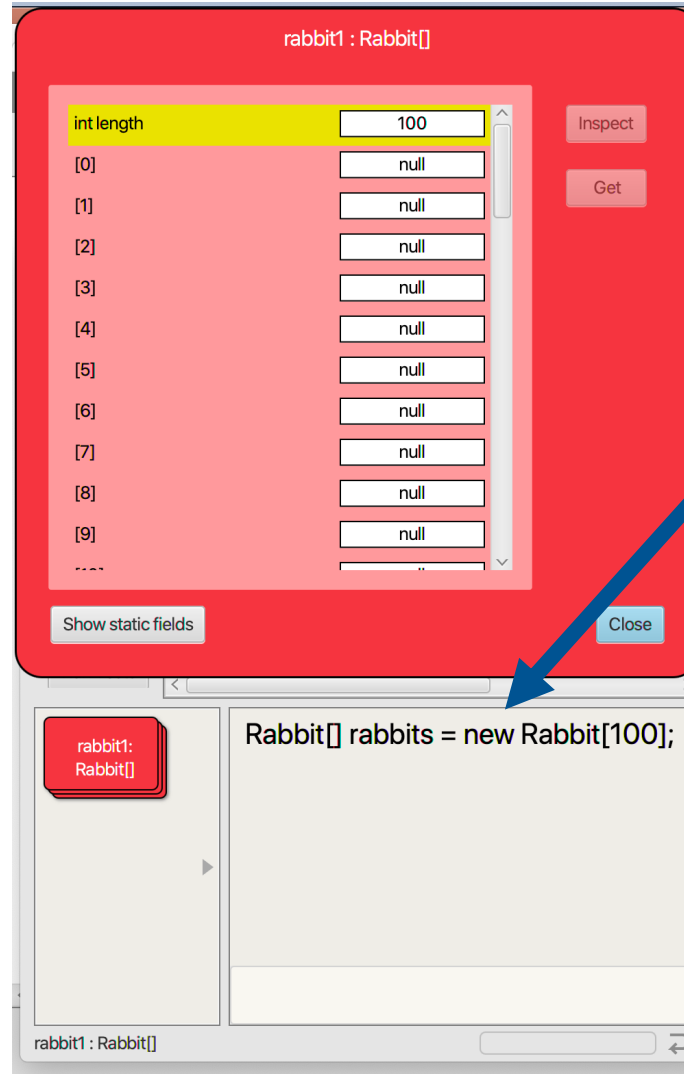
If we declare a variable `i = 5`,
`someWords[i]` is the same as typing
`someWords[5]`

Useful trick:
you can index
with a variable

What does the following code do?

```
String[] animalNoises = {"Moo", "Quack",  
"Meow", "Oink", "Baa", "Squeak", "Ribbit"};  
int i = 2;  
while (i < 7)  
{  
    System.out.println(animalNoises[i]);  
    i++;  
}
```

Another useful feature: create an empty array of a given size



`new Datatype[100]` can create an empty array with 100 slots to hold a particular datatype.

In general, the default value in each slot is `null`, which we will interpret as “no available object yet”.

“Empty” arrays
of primitive
datatypes
actually have a
default value

The screenshot displays three inspection windows for arrays in an IDE. The first window, titled 'numbers : int[]', shows an array of 25 elements, all with the value 0. The second window, titled 'moreNumbers : double[]', shows an array of 50 elements, all with the value 0.0. The third window, titled 'someBools : boolean[]', shows an array of 75 elements, all with the value false. Each window has an 'Inspect' button, a 'Get' button, and a 'Close' button. Below the windows, there are three tabs labeled 'numbers: int[]', 'moreNumbers: double[]', and 'someBools: boolean[]'. To the right of these tabs, the following code is visible:

```
int[] numbers = new int[25];  
double[] moreNumbers = new double[50];  
boolean[] someBools = new boolean[75];
```

The default value of numeric types is zero.
The default for booleans is false.

Working with empty arrays

What does the following code do?

```
int[] nums = new int[50];  
int i = 0;  
while (i < 50)  
{  
    nums[i] = i * 2;  
    i++;  
}
```

Working with empty arrays

What does the following code do?

```
int[] nums = new int[50];  
int i = 0;  
while (i < 50)  
{  
    nums[i] = i * 2;  
    i++;  
}
```

Loops are very useful for filling arrays with values and for working with arrays in general.

We will learn more useful loop types in the next lessons.