

Unit III: Advanced Techniques

Chapter 17: Arrays

An array is a list. When you declare a variable in Alice, you can select to make it an array. In this lesson, you will declare and use arrays. You will create arrays of words, numbers, or objects, then use loops to process the entire array.



IN the code above, the variable **greeting** is not an array; it is a **scalar**. It has a single value, “Hello World”. When the tortoise says **greeting**, he says “Hello World”

The variable **sayings** is an **array**. The items in an array are numbered starting with 0. That number is called the **index** or **subscript**. When you use an **array**, you need to use a **subscript** to specify which item, or element, in the list is to be used.

In this example, the tortoise says **sayings[0]**, which is “Ready”.

Objectives

- Declare a variable to be an array;
- Create code to assign initial values to an array;
- Use subscripts or index to access the elements of the array;
- Utilize loop control structures to process all the elements of the array;
- Create and process arrays of different data types, including arrays of objects;
- Understand the purpose of parallel arrays and ways to use them;
- Experiment with arrays of segments.

Ready, Set, Go

We start by declaring a String variable **sayings**, then check **is array**. When we click to initialize the array, we add the values one at a time.

Insert Variable

preview: String[] sayings = null ;

is variable: ☒ variable ☐ constant

value type: String ☒ is array

name: sayings

initializer: null

✖ initializer must be set.

Custom Array

preview: new String[] { }

type: String[]

value: add

We add the strings “Ready”, “Set”, “Go” one at a time. As we add elements to the array, we see the list of values with the subscript on the left. Notice that the subscript for “Ready” is [0]. Arrays always start numbering from 0. When you are finished, click OK at the bottom of the dialog window.

Custom Array

preview: new String[] { "Ready", "Set", "Go" }

type: String[]

value: [0] Ready

[1] Set

[2] Go

add

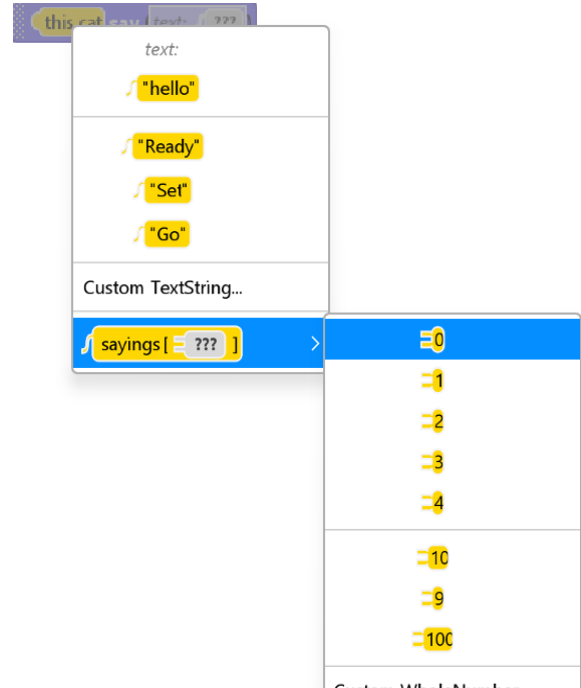
The code now looks like this:

```
void myFirstMethod ( )
do in order
    // Starting a race
    // Programmer: Janet Joy
    String[] sayings = new String[] { "Ready", "Set", "Go" } ;
```

The elements of the array are always referred to with a subscript. We want the cat to say “Ready”, “Set”, “Go” to start a race.

Drag the cat **say** command to the code window, then select **sayings[0]**

(Read this as “Sayings **sub** zero.” *Sub* stands for subscript.)



One of the reasons why arrays are so important is because we can use a variable as the subscript. In the code below, first we say each word using the subscripts. Then we say the same thing using a loop to say each word in the array.

```
void myFirstMethod ( )
do in order
    // Starting a race
    // Programmer: Janet Joy
    String[] sayings = new String[] { "Ready", "Set", "Go" } ;
    // The 3 statements below say "Ready", "Set", "Go"
    this.cat say ( sayings [ 0 ] add detail );
    this.cat say ( sayings [ 1 ] add detail );
    this.cat say ( sayings [ 2 ] add detail );
    // The loop below also says "Ready", "Set", "Go"
    for( Integer indexA = 0; indexA < sayings.length; indexA ++ ) {
        this.cat say ( sayings [ indexA ] add detail );
    }
    // When you run this, it will say "Ready" "Set", "Go" twice.
```

Array of Props

We start by declaring a variable **furniture**, then check **is array**.

The screenshot shows the Alice software interface with three main panels:

- Filtering:** Under the 'Contains' tab, a list of assignable items includes 'myScene', 'room', 'camera', 'bed', 'chair', and 'dresser'. The last three are checked.
- Selection:** A class hierarchy tree is shown. The path 'SThing' > 'SScene' > 'Scene' > 'STurnable' > 'SMovableTurnable' > 'SModel' > 'SProp' > 'Prop' is selected. Other visible classes include 'BedSingle', 'Chair', 'Dresser', 'SCamera', and 'SRoom'.
- Available Procedures, Functions, and Properties:** This panel shows the methods for the selected 'Prop' class:
 - class Prop**
 - class SProp (inherit)**
 - class SJointedModel (inherit)**
 - procedures: • straightenOutJoints
 - class SModel (inherit)**
 - procedures: • say, • think, • setVehicle, • setPaint, • setOpacity, • setWidth, • setHeight, • setDepth, • resize, • resizeWidth, • resizeHeight, • resizeDepth

After selecting **is Array** and then **Gallery Class**, **Prop**, we click to initialize the array, we can add the furniture to the array:

The 'Custom Array' dialog box shows the following configuration:

- preview:** `new Prop[] { this.bed, this.chair, this.dresser }`
- type:** **Prop[]**
- value:** A list of three elements:
 - [0] `this.bed`
 - [1] `this.chair`
 - [2] `this.dresser`
- add** button with a dropdown arrow.

The code is shown below:

```
void myFirstMethod ( )
```

do in order

```
// Array of props disappear
// Programmer: Janet Joy

Prop[] furniture = new Prop[] { this.bed, this.chair, this.dresser } ;
```

Next, we will write code to make all of the furniture invisible. (Opacity is set to 0)

This is a very useful tool: when we change scenes, we may make one set of furniture invisible and a different set of furniture visible to make it look like a different room. We might also want to make treasures invisible at the beginning of the movie and then make them visible when the character gets rich at the end of the movie.

Each in Together

There is a special control structure to do everything in an array together:

```
void myFirstMethod ( )
do in order
  // Array of props disappear
  // Programmer: Janet Joy
  Prop[] furniture = new Prop[] { this.bed, this.chair, this.dresser } ;
  ThreadUtilities.eachInTogether( ( Prop item ) -> {
    item.setOpacity(=0.0, SetOpacity.duration(=0.0) add detail);
  }, furniture );
```

After you drag in the **eachin _Together** tile, and add **item** as the variable, you can find **item** in the drop down window under the top left window that shows the scene. *Set the duration for the setOpacity to 0.0 to make all of the furniture disappear instantly.*

The screenshot shows the 'this' object in the Alice programming environment. It lists several properties: 'this', 'this.room', 'this.camera', 'this.bed', 'this.chair', and 'this.dresser'. Each property is accompanied by a small icon representing the object (e.g., a room, a camera, a bed, a chair, and a dresser). Below this list, there is a section labeled 'myFirstMethod constants:' which shows a blue box with a briefcase icon and the label 'item'.

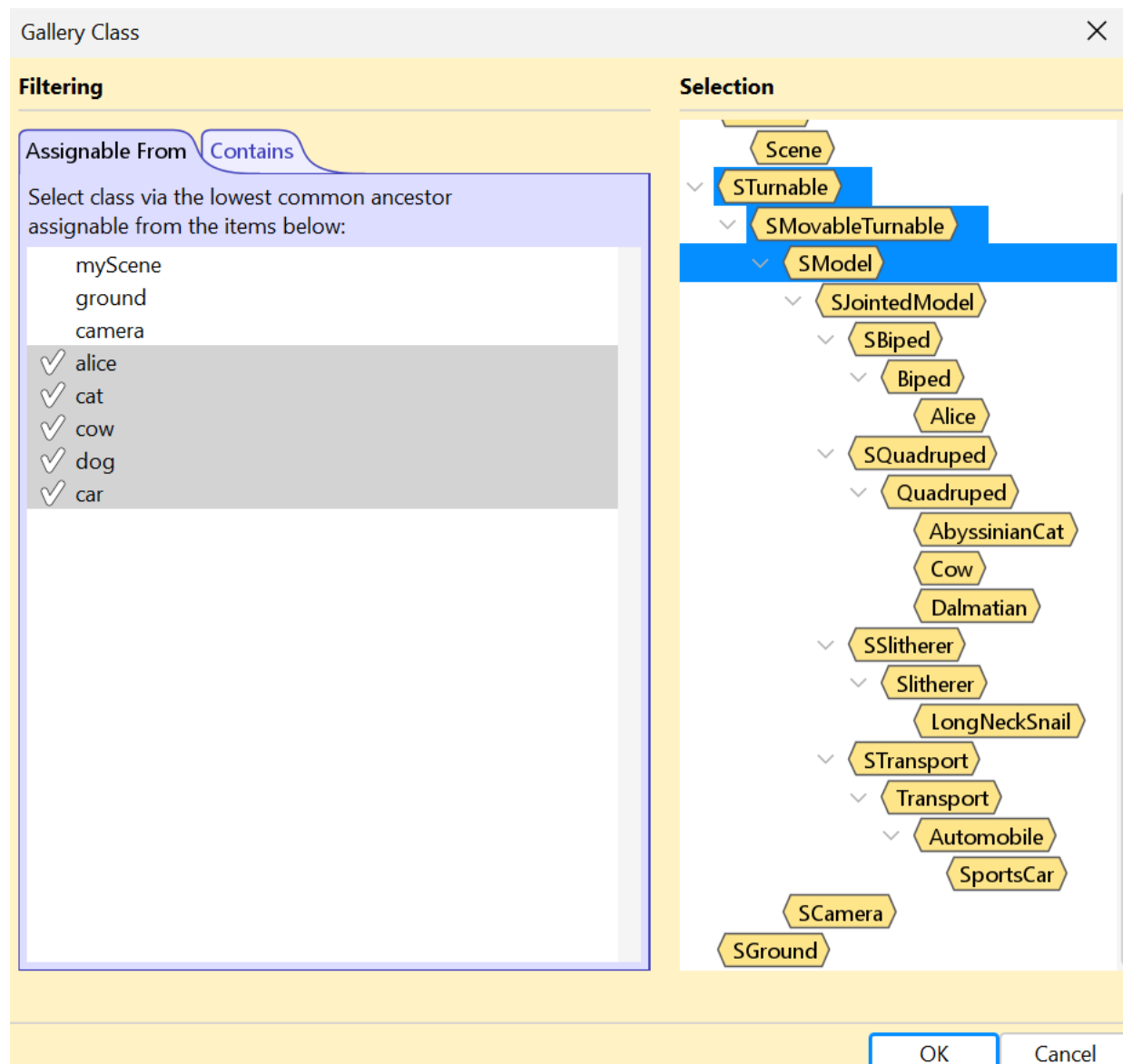
Experiment: Create another set of fancy furniture right on top of the other furniture. (*Don't worry, you are going to make sure that only one set of furniture is visible at a time.*)

When you make one array invisible (opacity=0) make the other set visible. (opacity=1)

Parallel Arrays

Parallel arrays are arrays that have a one-to-one correspondence between the arrays. For instance, you might have names in one array and phone numbers in the other. If **names[5]** is “Mary” then **phone[5]** is Mary’s phone number. This enables you to look up phone numbers or print a directory.

In the program below, one array has an assortment of objects, the other has the sounds they make. Our scene has a girl (alice), a cat, a cow, a dog, and a car. We select Gallery class, SModel to include all of those objects. When you select a class, it shows you what objects can be included, as seen on the left, there is a check mark for each of those objects.



The variable declaration of **objects** as an SModel array is shown below:

```
SModel[] objects = new SModel[] { this.alice, this.cat, this.cow, this.dog, this.car } ;
```

The second array is just strings, but it is important to put the sounds in the same order as the objects.

```
void myFirstMethod ( )
do in order
  // What sound do things make?
  // Programmer: Janet Joy
  SModel[] objects = new SModel[] { this.alice, this.cat, this.cow, this.dog, this.car } ;
  String[] sounds = new String[] { "hello", "meow", "moo", "bark-bark", "beep-beep" } ;
  // The loop will do each item in the objects array, but we need a counter to know which sound to use.
  Integer index = 0 ;
  for( SModel item : objects ) {
    item say ( sounds [ index ] ) add detail ;
    index = index + 1 ;
  }
}
```

The for loop will do each item in the array **object** one at a time, but we need **index** as a counter in order to know which sound to use. And we need to increment index or all the characters will say “hello” instead of their own sound.

Note: One disadvantage of using an array of SModel is that we can't make them do things like wave, because the car can't wave.

The code below has an array of fish. When the shark appears, they all turn to face the cave and move to it.

```
void myFirstMethod ( )
do in order
  // Fish are swimming, but all go to the cave when the shark appears.
  // Programmer: Janet Joy
  SSwimmer[] fish = new SSwimmer[] { this.blueTang, this.blueTang2, this.blueTang3, this.blueTang4 } ;
  ThreadUtilities.eachInTogether( ( SSwimmer one ) -> {
    one turnToFace ( this.cave ) add detail ;
    one moveAndOrientTo ( this.cave ) add detail ;
  }, fish ) ;
}
```

Experiment: Add the shark and make the fish swim to the cave when the shark appears.

Array of Segments: Elephant and Ostrich

The elephants trunk, the ostriches neck, and a few other things are arrays. In this illustration, the array of joints in the elephants trunk and the ostriches neck uses each in together to curve the trunk and neck.



```
void myFirstMethod ( )
```

do in order

```
// Elephant trunk segments are an array
// The neck of the ostrich is also an array.
// Programmer: Janet Joy

ThreadUtilities.eachInTogether( ( SJoint joint ) -> {
    joint turn ( TurnDirection.BACKWARD, 0.125, add detail );
    joint turn ( TurnDirection.FORWARD, 0.125, add detail );
}, this.elephant.getTrunkArray ( ) );

ThreadUtilities.eachInTogether( ( SJoint neckJoint ) -> {
    neckJoint turn ( TurnDirection.FORWARD, 0.125, add detail );
    neckJoint turn ( TurnDirection.BACKWARD, 0.125, add detail );
}, this.ostrich.getNeckArray ( ) );
```

We drag in the **eachInTogether** tile, then create an empty array of **SJoint** (*select other types*). Then drag the elephant function **getTrunkArray** to replace the empty array at the end of the control structure. It's a bit complicated, but shows some of the things you can do with arrays.