Unit II: Creating Code

Chapter 8: Scene Procedures in Alice 3

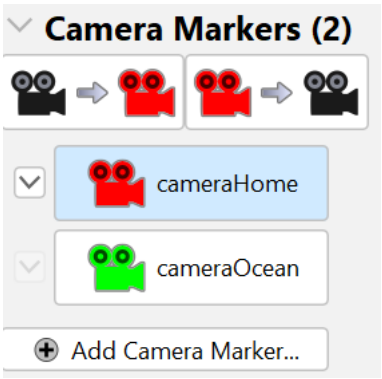
In this chapter, you will learn how to create and use **scene procedures** in Alice to organize a program using **top-down design**. You will learn how to begin with the overall task a program must accomplish and then break that task into smaller, more manageable procedures. By using scene procedures to organize related actions, you can create programs that are easier to understand, develop, test, debug, and modify.

Objectives

- Be able to create a Scene Procedure
- Be able to call a Scene Procedure
- Use Scene Procedures for Top-Down Design
- Disable a call to a Scene Procedure to facilitate testing and debugging

Locations, Camera Markers

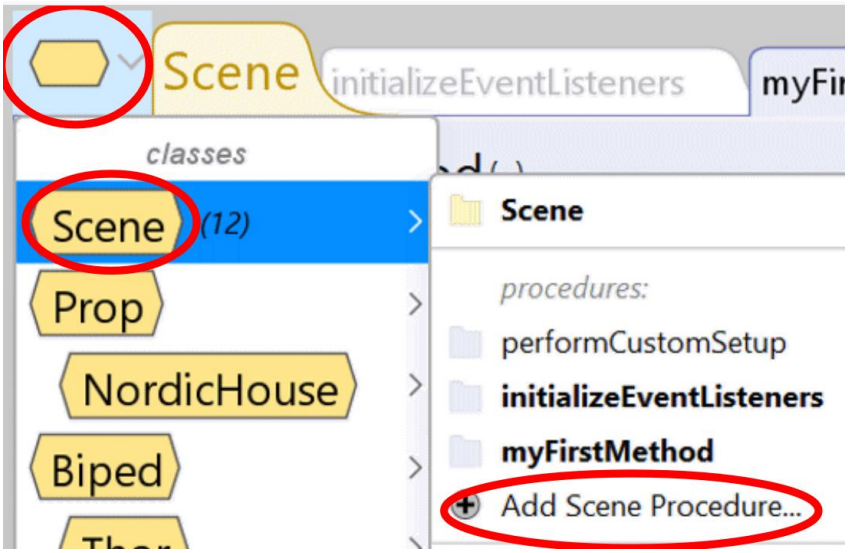
Camera markers are added the same way that object markers are added. There is a location on the right with a house and Thor and Freya. There is a camera marker (red) that shows just the house and two people. There is another location on the left with a second Thor in a boat. The green camera marker shows just Thor and the boat.



Scene Procedures

The movie will start with a scene at home. Thor and Freya say goodbye and Thor leaves. In the second scene Thor is out on the boat fishing. The sky gets dark and he says it's time to go home.

In code view, click the yellow 6-sided image and then select Scene, +Add Scene Procedure. Name the new scene procedure **morning**.

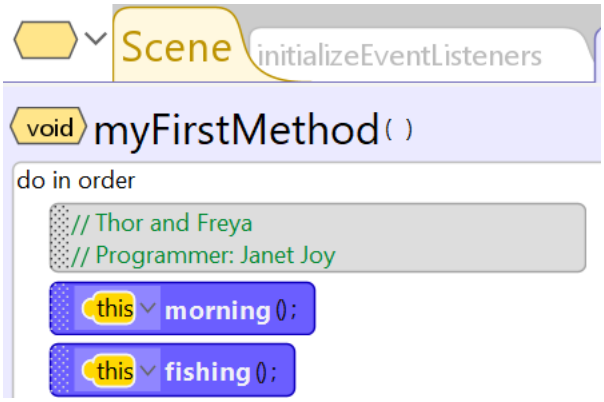


Write the code as shown below, notice that we set the **duration** for setting up the camera and the ground to 0 so that it happens instantly at the very start of the scene.

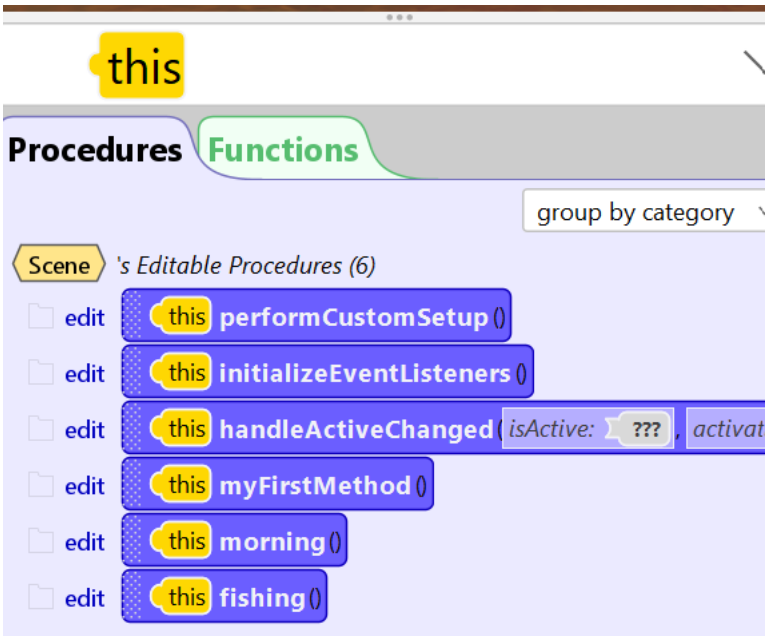
```
public void morning() {
    // Morning at the home of Thor and Freya
    // Setup scene
  this.camera.moveAndOrientTo(this.cameraHome, MoveAndOrientTo.duration(0.0);
  this.ground.setPaint( SurfaceAppearance.FOREST_FLOOR, SetPaint.duration(0.0
  );
    // Thor and Freya say goodbye
  this.thor.say( "I better be going." );
   this.freya.say( "Come home safe." );
   // Thor turns and leaves
   this.thor.turn( TurnDirection.LEFT, 0.25 );
   this.thor.move( MoveDirection.FORWARD, 10.0 );
 }
```

Call the Procedure from myFirstMethod

When you run the program, nothing happens. For this new code to **execute**, we must **call** the procedure from **myFirstMethod**:



The fishing scene procedure is shown below. You will find the scene procedures after you select **this** in code view:



Fishing

The code for **fishing** is like the code for **morning**, and they will execute one right after the other.

```
public void fishing() {
    // Thor is in the boat fishing
    // Setup scene
    this.camera.moveAndOrientTo( this.cameraOcean, MoveAndOrientTo.duration(
0.0 ) );
    this.ground.setPaint( SurfaceAppearance.OCEAN, SetPaint.duration( 0.0 ) )
;
    this.thorBoat.think( "I hope Freya is OK." );
    // Pause for a bit
    this.delay( 2.0 );
    // Sky gets dark, fog appears
    this.setFromAboveLightColor( Color.DARK_BLUE );
    this.setFogDensity( 0.5 );
    this.thorBoat.think( "It's getting dark, I'm going home." );
    // Make sure Thor moves with boat, then go home.
    this.thorBoat.setVehicle( this.boat );
    this.boat.move( MoveDirection.FORWARD, 10.0 );
 }
```

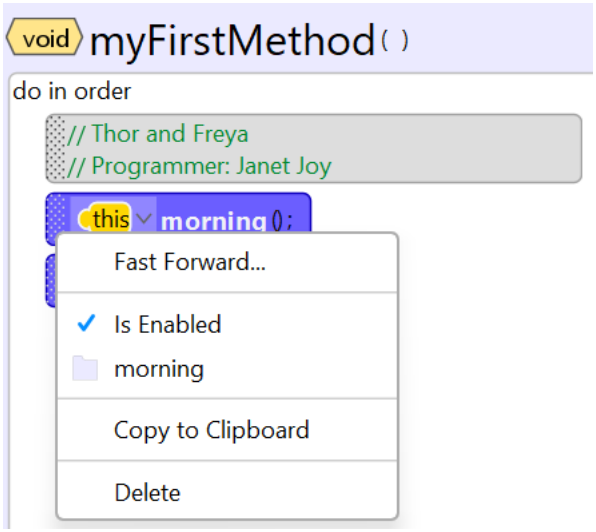
Advantages of Scene Procedures and Top-Down Design

Top-down design is a problem-solving approach in which a large programming task is divided into smaller, more manageable tasks. In Alice 3, a programmer can begin by thinking about the overall animation or story and then break it into individual actions or scenes. For example, an animation about a character entering a room, greeting a friend, and sitting down could be divided into three tasks: enterRoom, greetFriend, and sitDown. Each task can then be developed and tested separately. This approach makes a complex Alice project easier to plan, create, understand, and debug.

In Alice 3, procedures are especially useful for implementing top-down design. A procedure contains a group of instructions that work together to perform a particular task. Instead of placing every instruction in one long program, programmers can create procedures with meaningful names and call them when they are needed. Procedures can also be reused, reducing the need to repeat the same instructions. For example, a wave procedure could contain all the movements needed to make a character wave and could be called several times during an animation. Using procedures helps keep Alice programs organized and demonstrates an important programming principle: breaking a large problem into smaller, reusable parts.

Enable/Disable Procedures

If you right-click on the area at the left of a procedure, you can disable a procedure. For instance, if you want to work on some code in the fishing procedure, you can disable the morning procedure, so that when you run it, it goes straight to the code you are working on. Once you have that working, you can enable the morning procedure and work on something else.

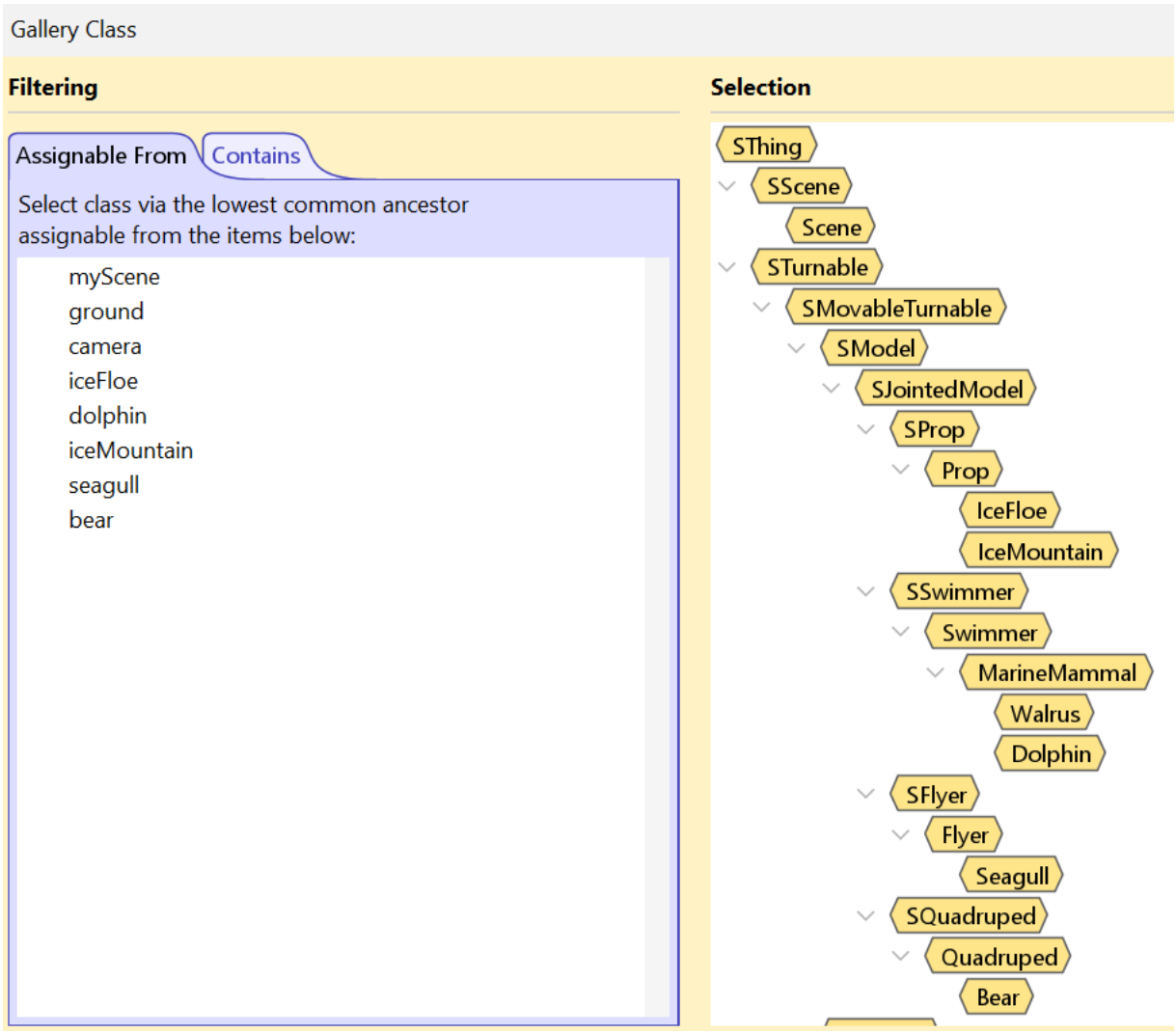


Parameters

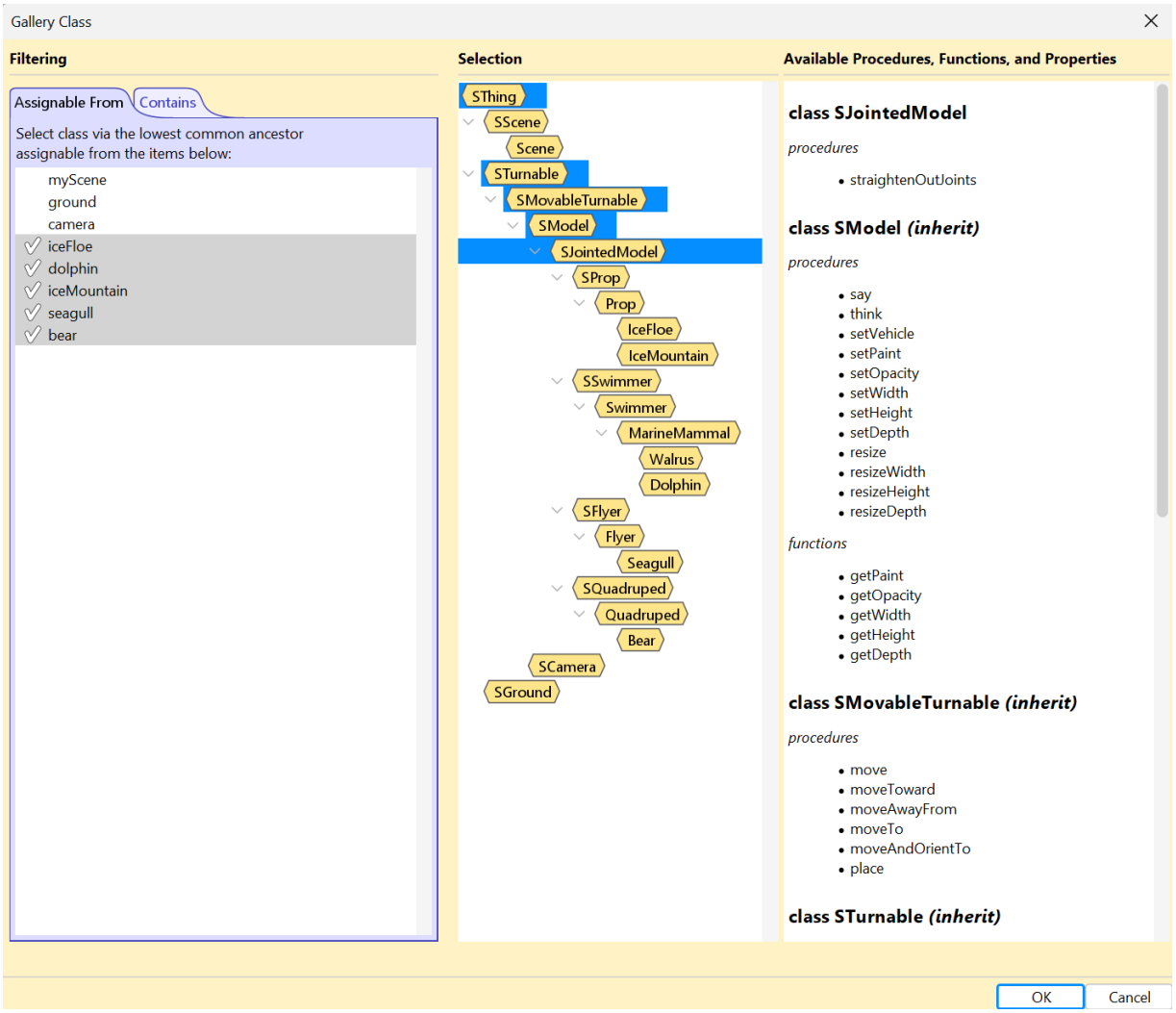
In the scene below, there is a dolphin (swimmer class), a polar bear (quadruped) and a seagull (flyer class). Even though they are different classes, they can all **say**, **think**, **move**, **turn**, and **roll** to put on a little show. We will create a scene procedure called **perform**, when we call the procedure **perform** we will pass an **argument** telling which creature we want to perform.

The way we can tell the procedure which one is going to perform is by adding a **parameter**.

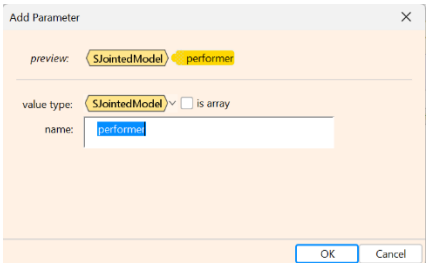
Add a scene procedure, then click on Add Parameter. Select Gallery Class. From Gallery Class we need to pick a type that will include all of the animals. ON the right you can see the hierarchy of the classes. The lower the classification, the more things it can do. The **SJointedModel** is the lowest one that has all of the animals in our scene as subclasses.



When we click on **SJointedModel** we see that all our animals are checked off on the right, and the available procedures and functions are shown on the right. In fact, iceFlow and iceMountain are also checked off, so we could make them perform too.

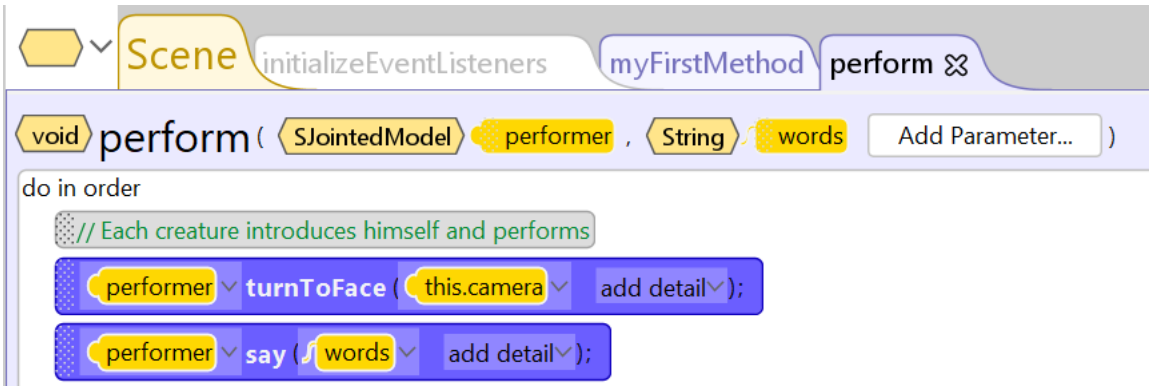


After you select the type, you need to give it a name. Name it **performer**. The drop-down box next to **this** now has **performer** at the bottom. Select performer, then select turnToFace and drag it into the code. Have the performer face the camera:




```
public void perform( SJointedModel performer ) {
    // Each creature introduces himself and performs
    performer.turnToFace( this.camera );
}
```

Next, we would like the performer to introduce himself, so we will add a **parameter** for what it is going to say. That will be a **string**.

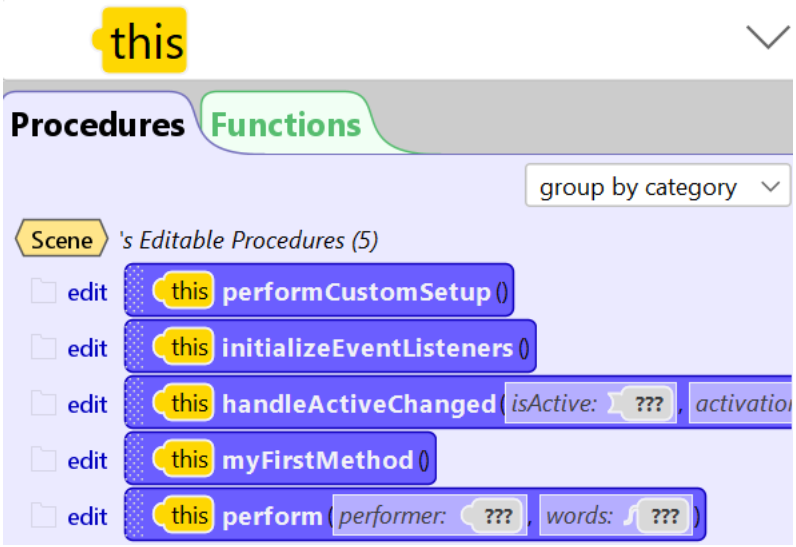


At this point, the **performer** turns to face the camera and says whatever the value of the parameter **words** is. Note the parameter list after perform: SJointedModel performer and String words.

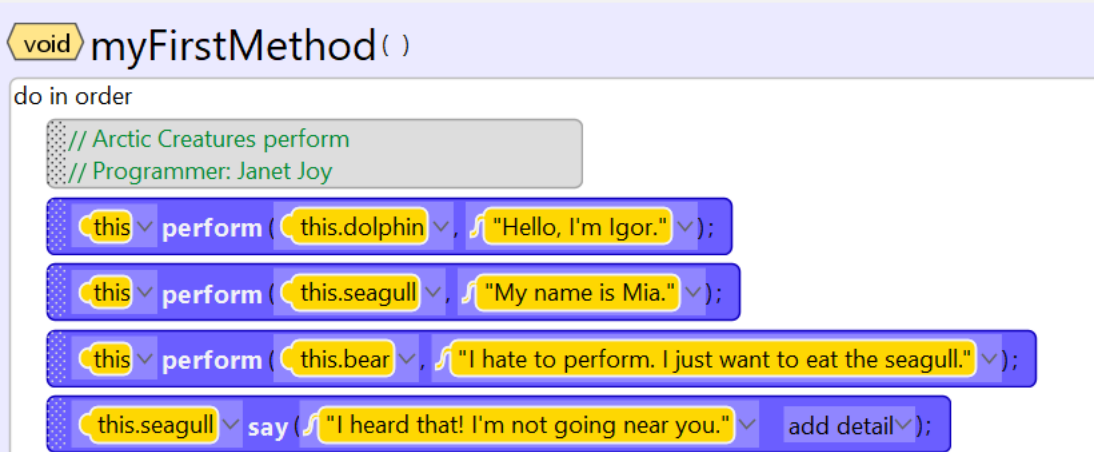
Test So Far

Let's go to **myFirstMewhatever** and add the **calls** to **perform**, with the appropriate **arguments** to match the **parameters**.

You will see that **perform** is now one of the procedures available for **this**.



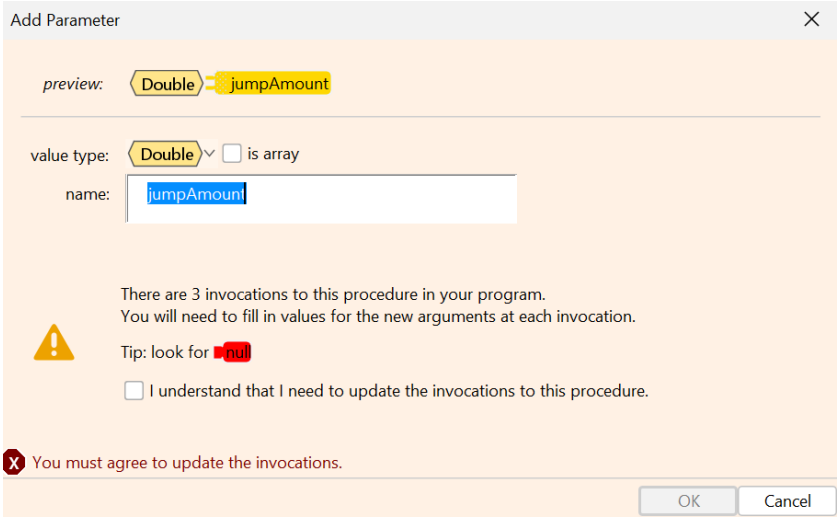
Drag perform into the code and add the arguments:

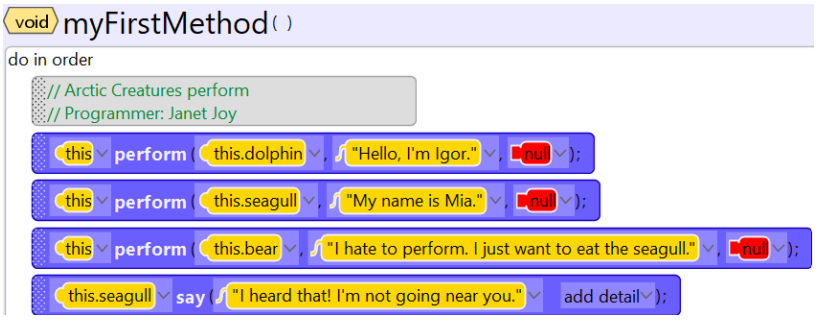


Notice that in addition to **perform**, we can still have each of the animals do all the things they can normally do, such as the comment by the seagull using **say**.

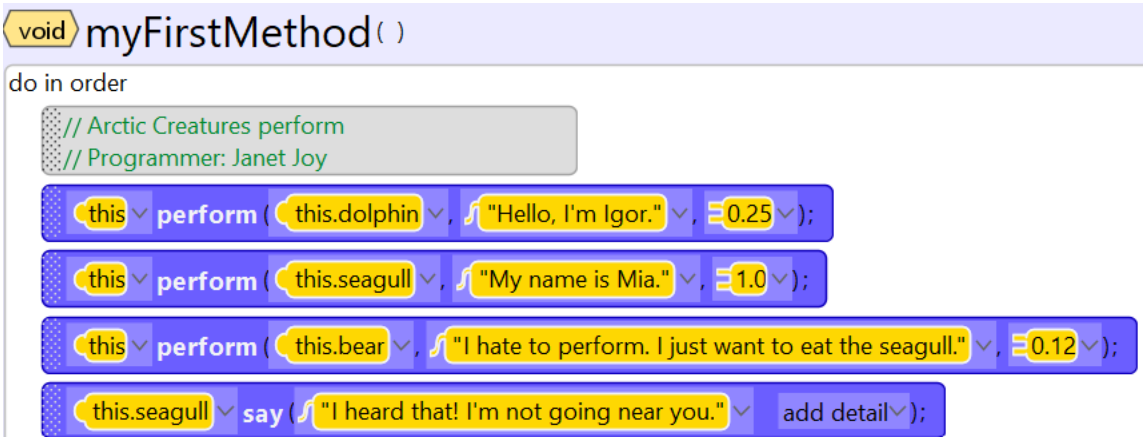
Warnings

If we want the performer to jump, turn or roll, we would add a parameter that is a **double**. When we add a new **parameter,** we will get a warning that we already have three calls (invocations) to the procedure. We must check the box to agree, then go back to the program and fix all the calls to **perform** by adding a double argument.





This is going to be the amount to jump. Let's add some appropriate values, then go to the perform procedure and add the code to jump.



Jump

To jump, we will move up the amount, then down the same amount.

```
public void perform(SJointedModel performer, String words, Double jumpAmount
) {
    // Each creature introduces himself and performs
    performer.turnToFace( this.camera );
    performer.say( words );
    //
    performer.move( MoveDirection.UP, jumpAmount );
    performer.move( MoveDirection.DOWN, jumpAmount );
 }
```

Adding parameters make the movie more realistic by having some of the characters move faster, jump higher, or move farther,.

Experiment! Add to this movie or create your own.