Procedures

You have been using procedures all along. You have added code to the procedure myFirstMethod, and you have used the say, move, turn and roll procedures. In this lesson you will learn to create your own procedures.

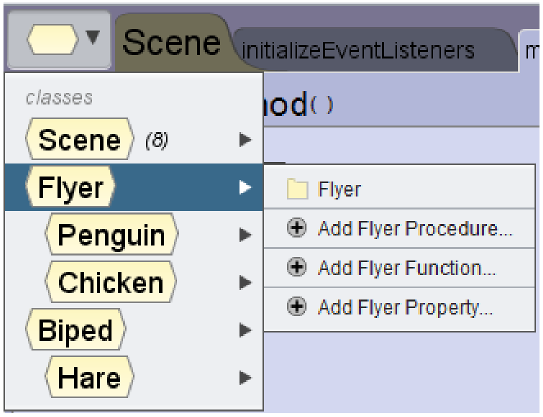
Procedures allow you to break your program into subparts, thus making it easier to create large programs. A procedure will let you write the code for an object to do something like wave. Then, whenever you want the character to wave, you just call the procedure.

Instead of writing a procedure for just one of the actors, for instance to make the bunny wave, we can write the procedure for any biped to wave. We will also use parameters so that we can tell the procedure how fast to wave, or how many times to wave. We can also use parameters to tell who the bunny faces when they wave.

Procedures are a very powerful tool and will let you greatly improve the efficiency of your code.

Creating a Procedure

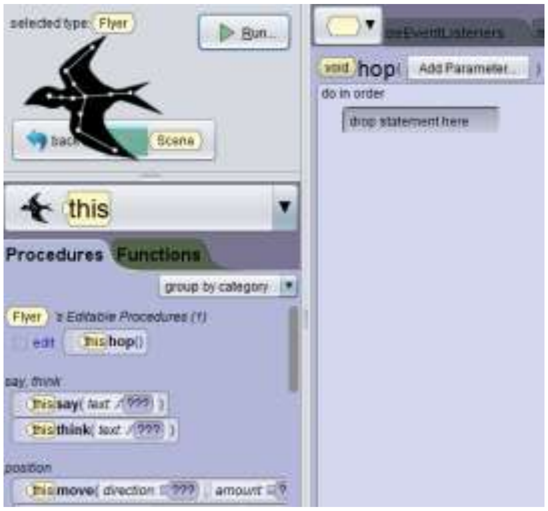
When you click on the procedure button, you can select the scene or class that you want to create a procedure for. In the illustration, we have a project with a penguin and a chicken. If we select the Flyer class, then select Add Flyer Procedure we can create a **hop** procedure. Both the penguin and the chicken will then have a hop procedure that we can call, just like we can call **say**, turn, roll or other procedures. If we select Penguin and add Penguin procedure, then only the penguin can hop. It is a good idea to select the highest class that has the necessary subparts for the procedure.



After selecting Add Flyer procedure, you give it a name. The name of a procedure is usually a verb or a verb and noun: hop, hopOver, waveWand. Notice that there are no spaces, camel case is used instead.

You will now see the graphic for the class.

The procedures panel will show the methods that are available for the class. Notice that is says **this** instead of the name of an object. **This** refers to whichever object calls the procedure.



You can drag statements into the code window for the procedure exactly the way you do in MyFirstMethod. You can also copy code from MyFirstMethod and change the object to **this**.

Top Down Design

Let's suppose you want to create a movie about a girl finding a balloon. You may have gone through the design of the program and have the following description of the movie:

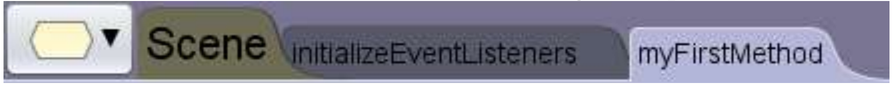
1. A girl (Alice) is playing in the park.
2. A balloon drifts in.
3. The girl chases the balloon.
4. The girl catches the balloon.
5. The girl claps her hands and the balloon escapes.

There are no details here. We haven't decided how we are going to have the balloon drift. We haven't decided what the girl is playing with. We haven't figured out how to make the girl run after the balloon and catch it, or how the balloon escapes.

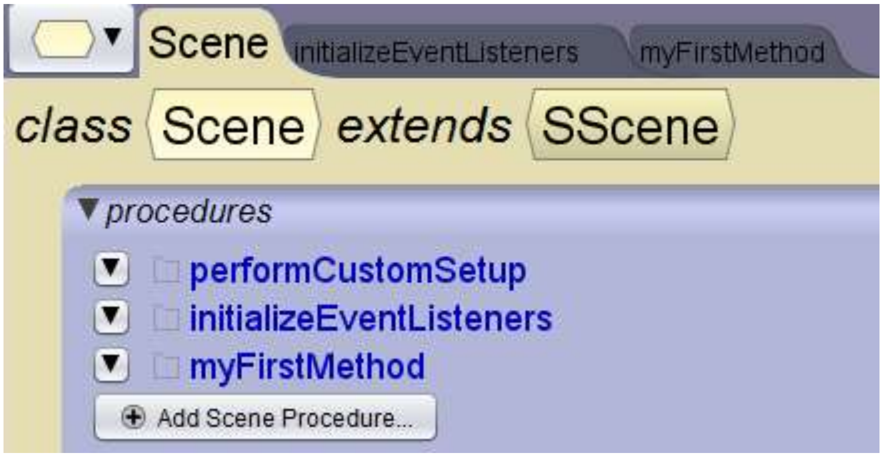
This is called top-down design. I've outlined what I want my movie to do, giving the main parts of the movie, but no details. (In order to follow along with the lesson, start by creating a new movie and adding a girl.

Procedures make it easy to create the program by following the top-down design and creating a procedure for each of these five main ideas in the story.

In the Code view window there are tabs along the top:



Click on the Scene tab: You will now see the window as shown below:



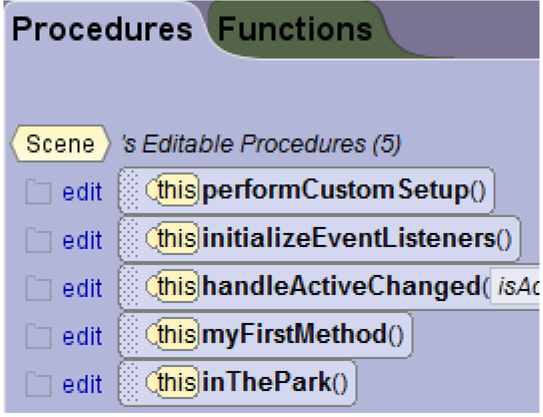
```
class Scene extends SScene
```

Click on Add Scene Procedure. You will be prompted for a name for the new procedure.

Author: Janet E. Joy; Publisher: Zebra0.com

Call it inThePark and add a comment. Every procedure should really have a comment that tells what it does, such as //girl is playing in the park.
In order to know that the code in the scene is being executed, have the girl say something.
I'm going to have the girl say something: "I'm bored. I wish something would happen."

Go back to my first method by click the tab. Select this by clicking on the ground.

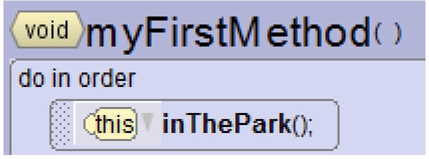


Scene 's Editable Procedures (5)

□ edit this performCustomSetup()
□ edit this initializeEventListeners()
□ edit this handleActiveChanged(isAc
□ edit this myFirstMethod()
□ edit this inThePark()

You will now see that inThePark is listed as procedure. You can click the edit button next to it when you are ready to add details.

Drag the inThePark procedure into myFirstMethod:

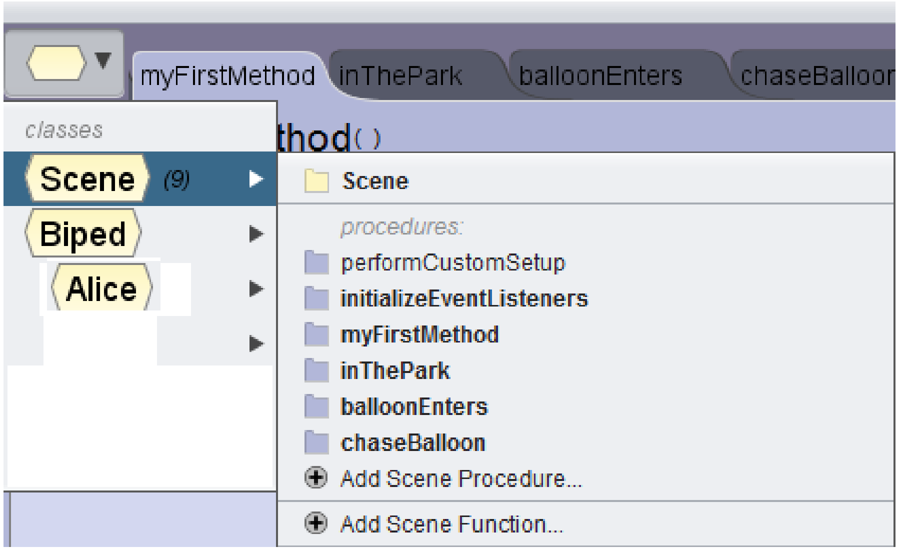


```
void myFirstMethod( )
do in order
    this inThePark();
```

Run the program and the girl will say "I'm bored. I wish something would happen." You know that the code in inThePark executed, because that is the only place that the girl says something.

When you add a second procedure the screen looks a bit different:

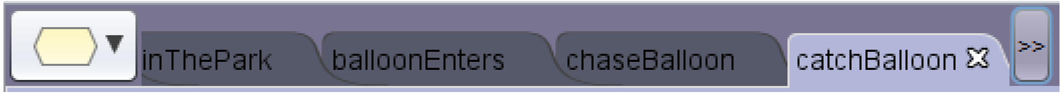
Author: Janet E. Joy; Publisher: Zebra0.com



Continue to add a new procedure for each part of the top-down design:

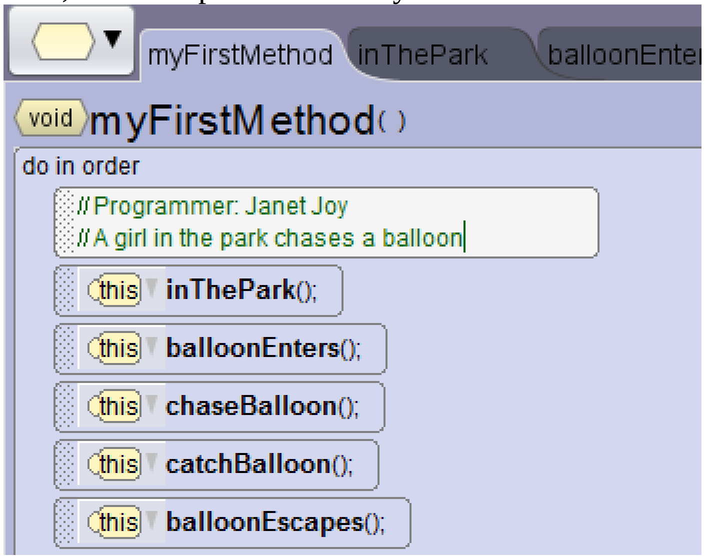
Procedure:	**Girl Says:**
inThePark	I'm bored. I wish something would happen.
balloonEnters	Oh look, a balloon.
chaseBalloon	I'm going to catch it!
catchBalloon	Yeah! I caught it!
balloonEscapes	Oh no! I lost it!

As you add procedures, the tabs will fill up:



You may need to use the >> tab to return to myFirstMethod.

Next, add each procedure to myFirstMethod:



```
void myFirstMethod( )
do in order
    // Programmer: Janet Joy
    // A girl in the park chases a balloon
    this inThePark();
    this balloonEnters();
    this chaseBalloon();
    this catchBalloon();
    this balloonEscapes();
```

Author: Janet E. Joy; Publisher: Zebra0.com

We're building our program from the top down. Ideally, myFirstMethod is nothing but comments and calls to procedures.

Now we are ready to start adding details to each scene. One of the nice things about top-down design and working this way with procedures is that you can add the details in almost any order. Maybe you have an idea about how the balloon will drift in and move across the scene. If so, start with that. Add the parts that you feel most comfortable with and give yourself time to think about how you will implement the more complicated details.

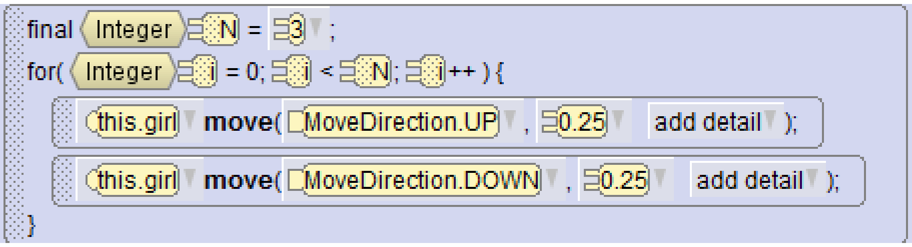
If you are working on one of the procedures, you may want to disable some of the other procedures so that the movie starts with the one you are working on. It will be easier to test and debug one piece at a time.

Some of these procedures may call other procedures. For instance, you may want the girl to jump up and down when she sees the balloon, and again when she catches it. A jump procedure will be very useful.

Procedures for Classes

We want the girl to jump up and down in two different places. Instead of writing this code in two places, we write in once to get it working, then change it to a procedure for the class. We will have the code in just one place, but we can call the procedure in as many places as we want: alice.jump() A nice thing about a procedure for a class is that we can make it a procedure for the entire biped class. We can have the girl jump: girl.jump() or her stuffed tiger: stuffedTiger.jump() or any other biped.

This is a very simple jump:

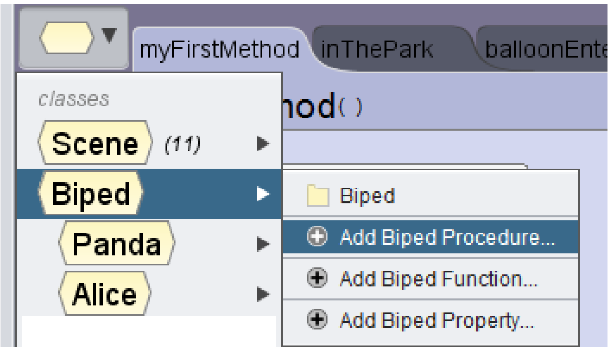


```
final Integer N = 3 ;
for( Integer i = 0; i < N; i++ ) {
    this.girl move( MoveDirection.UP , 0.25 add detail );
    this.girl move( MoveDirection.DOWN , 0.25 add detail );
}
```

We may want to make this a bit better by using a random number for amountToJump instead of 0.25, we may want to have her jump a random number of times instead of 3. We may want to make her bend her knees to jump, and then straighten them out. It will be much easier if we have the code to jump in just one place.

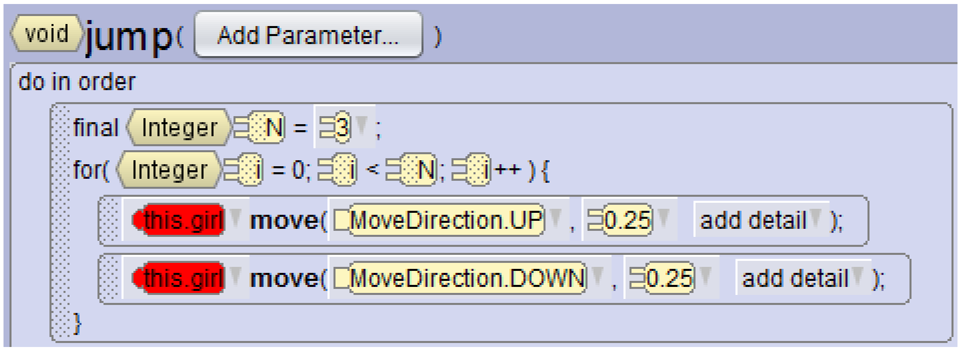
Right-click on that whole count block and copy it to the clipboard.

Click the ⬭ ▼ procedure button and select Add Biped Procedure as shown below:



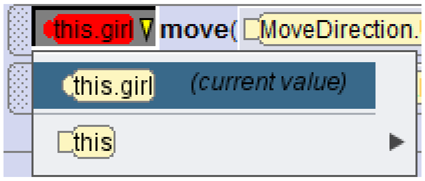
Name the procedure jump.

Now drag the clipboard into the code window:



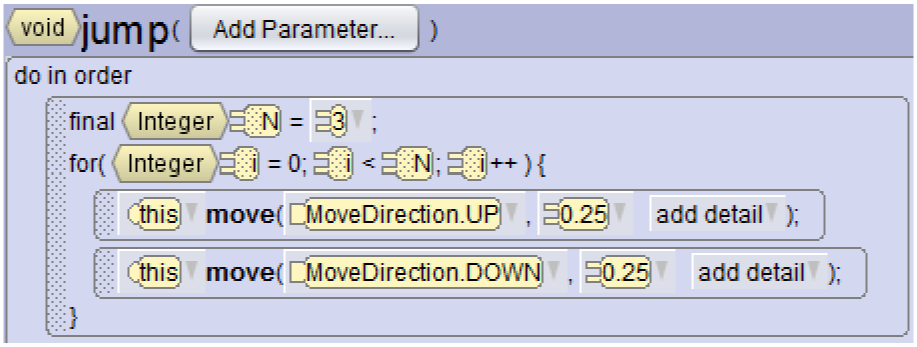
```
void jump( Add Parameter... )
do in order
    final Integer N = 3 ;
    for( Integer i = 0; i < N; i++ ) {
        this.girl move( MoveDirection.UP , 0.25 add detail );
        this.girl move( MoveDirection.DOWN , 0.25 add detail );
    }
```

Notice that this.girl is highlighted in red. We can not refer to this.girl in the Biped procedure. We must change this.girl to this. If we click on the red highlighted this.girl we can select this from the drop down list:



Do this for each instance.

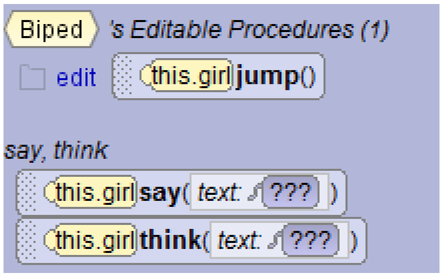
Author: Janet E. Joy; Publisher: Zebra0.com



```
void jump( Add Parameter... )
do in order
    final Integer N = 3 ;
    for( Integer i = 0; i < N; i++ ) {
        this move( MoveDirection.UP , 0.25 add detail );
        this move( MoveDirection.DOWN , 0.25 add detail );
    }
```

Go to the balloonEnters procedure and click on the girl.

You will now see girl.jump() as an Editable Procedure just above the say and think procedures.



Biped 's Editable Procedures (1)

edit this.girl jump()

say, think

this.girl say(text: ???)

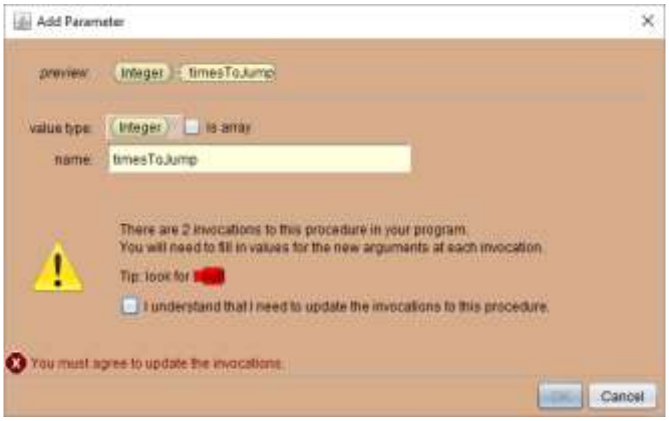
this.girl think(text: ???)

Drag girl.jump() into the code for both the balloonEnters and the catchesBalloon procedures and run the program. The girl jumps in two different places. Any change to the jump procedure will affect any call to that procedure.

Parameters

Suppose you want the girl to jump 3 times when the balloon enters, but just 2 times when she catches it. We can make the number of times she jumps a parameter. If we add timesToJump as an integer parameter, we will need to change the call in balloonEnters to girl.jump(3); and the call in catchesBalloon to girl.jump(2); The parameter must be matched by a corresponding argument in the calling statement.

In the jump procedure click on Add Parameter. Name the parameter timesToJump and make it an integer. Alice displays a big warning and you have to click to agree that you will make the necessary changes.



Add Parameter

preview: Integer timesToJump

value type: Integer □ is array

name: timesToJump

There are 2 invocations to this procedure in your program.
You will need to fill in values for the new arguments at each invocation.

Tip: look for

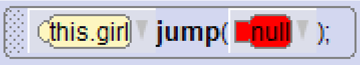
□ I understand that I need to update the invocations to this procedure.

You must agree to update the invocations.

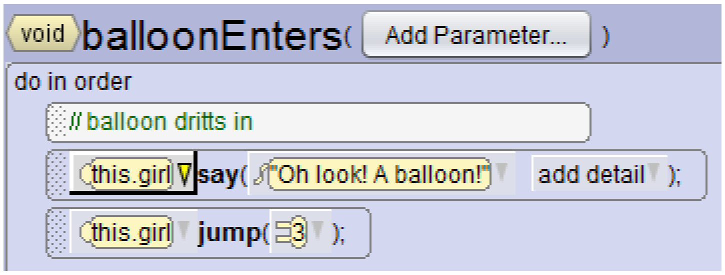
Author: Janet E. Joy; Publisher: Zebra0.com

After agreeing and clicking OK, go to the two procedures that call jump. Add the number as an argument.

Change this: by adding the integer argument.



```
this.girl jump( null );
```



```
void balloonEnters( Add Parameter... )
do in order
    // balloon drifts in
    this.girl say( "Oh look! A balloon!"  add detail );
    this.girl jump( 3 );
```

Author: Janet E. Joy; Publisher: Zebra0.com