

# Programming: Simple Control Structures

Alice



# Control Statements

- We have been using *Do in order* and *Do together* to control the way instructions are executed in your Alice program.
- Control statements can also be used for
  - ✎ conditional execution
  - ✎ repetition



# Conditional Execution

- **Conditional execution** is where some condition is checked and a decision is made about whether a block of the program will be executed.
- Conditional execution is extremely useful in
  - 🎮 games
  - 🎮 simulations
  - 🎮 real-time controls, e.g. robot systems



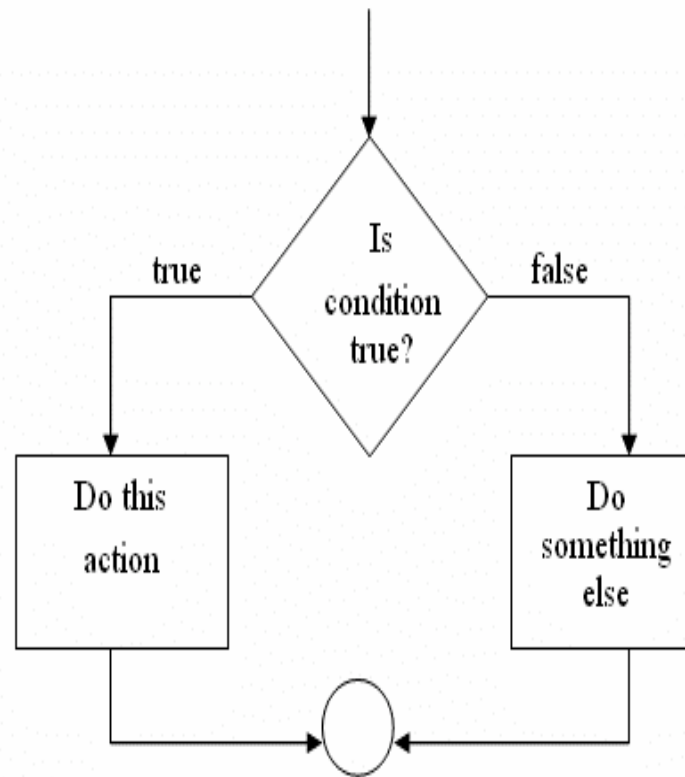
# Example

- As a simple example of conditional execution, let's revisit the Egyptian scene in the Hollywood movie set.
  - ▶ The camera angle can mislead our perception of the scene
  - ▶ We want to check a condition (is the mummy taller than the pharaoh?) and then perform an action based on the whether the condition is true.



# If/Else

- 🌐 In Alice, an **If/Else** control statement is used to check a condition and make a decision.



# Storyboard

- 🌐 In our example, we can demonstrate which object is the tallest by having that object turn around.
- 🌐 A storyboard design for this action is:

```
If mummy is taller than pharaoh  
    mummy turns 1 revolution  
Else  
    pharaoh turns 1 revolution
```

- 🌐 The condition in an *If* statement is a Boolean function returning a Boolean (*true* or *false*) value.



# Demo

## 🌐 Ch03Lec2mummyIfElse

🌐 Concepts illustrated in this example program:

- 🦊 The condition of an If/Else statement is constructed by using a function that returns a Boolean value (true or false).
- 🦊 *Do nothing* in the Else part of the statement means that if the condition is false, Alice will skip this part and go to the next instruction.



# A different scenario

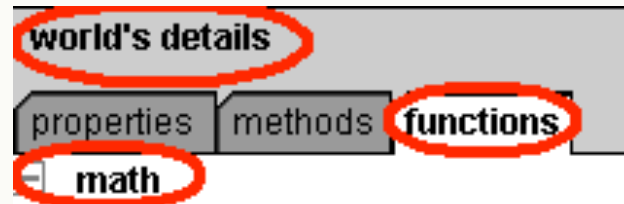
- 🌐 In some cases, the built-in functions are not sufficient for a condition that we want to check.
  - 👤 For example, the built-in function *is taller than* compares the heights of two objects. We used this function to compare the heights of the mummy and the pharoah.
  - 👤 Suppose, however, that the casting call for the mummy requires that the actor is more than 2 meters tall.
  - 👤 How can we compare the height of the mummy to a specific measurement (2 meters)?





# Relational Operators

- 🌐 In situations where you need to write your own comparison, you can use a relational operator.
- 🌐 Relational operators are provided in the World's built-in functions.



The diagram shows a hierarchy of 'world's details' which includes 'properties', 'methods', and 'functions'. The 'math' category is also highlighted.



$a == b$	<i>is equal to</i>
$a != b$	<i>is not equal to</i>
$a > b$	<i>is greater than</i>
$a \geq b$	<i>is greater than or equal to</i>
$a < b$	<i>is less than</i>
$a \leq b$	<i>is less than or equal to</i>



# Demo

## Ch03Lec2mummyRelationalOps

### Concepts illustrated in this example program:

-  Relational operations are defined using the world's built-in functions.
-  Placeholder values are first selected for the "a" and "b" components of the relational expression. Then the placeholder values are each replaced by either a function or a number.



# Example

- 🌐 Let's write code to make the mummy "walk" – a somewhat stilted motion like you would see in an old Hollywood horror film.
- 🌐 The code will be more complex because we need to
  - ▶️👤 alternate left and right leg motions
  - ▶️👤 coordinate leg motions with the body moving forward



# Storyboard



*Do in order*

*Do together //move body and left leg*

mummy move forward 0.25 meters

*Do in order*

mummy's left leg move backward

mummy's left leg move forward

*Do together //move body and right leg*

mummy move forward 0.25 meters

*Do in order*

mummy's right leg move backward

mummy's right leg move forward



# Demo

 Ch03Lec2mummySimpleSteps



# Need for Repetition

- In this example the mummy takes 1 step forward on each leg.
- Suppose we want the mummy to take 20 steps forward?



# Loop

- 🌐 The **Loop** statement is a simple control structure that provides for repeating an instruction (or block of instructions) a counted number of times.



# Demo

 Ch03Lec2mummyLoop





# Classes, Objects, and World-level Methods

Alice



# Larger Programs

- As you become more skilled in writing programs, you will find that programs quickly increase to many, many lines of code.
- Games and other "real world" software applications can have thousands, even millions of lines of code.



# Classes, Objects, & Methods

- Object-oriented programming uses classes, objects, and methods as basic programming components.
- These components help to
  - ▶ organize a large program into small modules
  - ▶ design and think about an intricate program
  - ▶ find and remove errors (bugs)



# In our programs, we have been using...

## Classes

 In Alice, classes are predefined as 3D models



## Objects

 An object is an **instance** of a class.

Class: Frog (Uppercase name)

Objects: frog, frog1, frog2, frog3  
(lowercase names)



# We have also used...

- 🌐 built-in (predefined) methods

  - ▶️ Examples: *move, turn to face, say*

- 🌐 *World.my first method*

  - ▶️ Example:

In the Snowpeople world, we wrote program code where the snowman tried to get the attention of a snowwoman.

All the program code was written in this one method, see next slide...



## World.my first method

World.my first method *No parameters*

create

No variables

create

### Do in order

// The snowman tries to catch the attention of the snowwoman by talking and blinking his eye

snowman turn to face snowwoman more...

snowman say Ahem... more...

### Do together

snowwoman.head turn to face snowman more...

#### Do in order

snowman.head.leftEye move up 0.04 meters duration = 0.5 seconds more...

snowman.head.leftEye move down 0.04 meters duration = 0.5 seconds more...

#### Do in order

snowman.head.rightEye move up 0.04 meters duration = 0.5 seconds more...

snowman.head.rightEye move down 0.04 meters duration = 0.5 seconds more...

// The snowwoman blushes and turns away

### Do together

snowwoman.head set color to red more...

snowwoman.head turn to face snowwoman2 more...

snowwoman.head set color to white more...

snowman.head turn forward 0.15 revolutions more...

snowman turn right 0.5 revolutions more...



# Potential Problem

- The program code just seemed to grow and grow.
- If we continue to write programs this way the programs will become longer and more difficult to read and think about.



# Solution

- 🌐 A solution is to organize the instructions into smaller methods.
- 🌐 A possible storyboard

*Do in order*

*catchAttention* – snowman tries to get the attention of the snowwoman

*blink eyes* – snowwoman turns to look and the snowman blinks his eyes

*react* – snowwoman blushes and turns away and the snowman  
is disappointed





# Next Step

🌐 The next step is to break down each major task into simpler steps.

🎥 Example:

*catchAttention*

*Do in order*

snowman's head turns to face camera

snowman says "Ahem"

snowman's head turns to face snowwoman

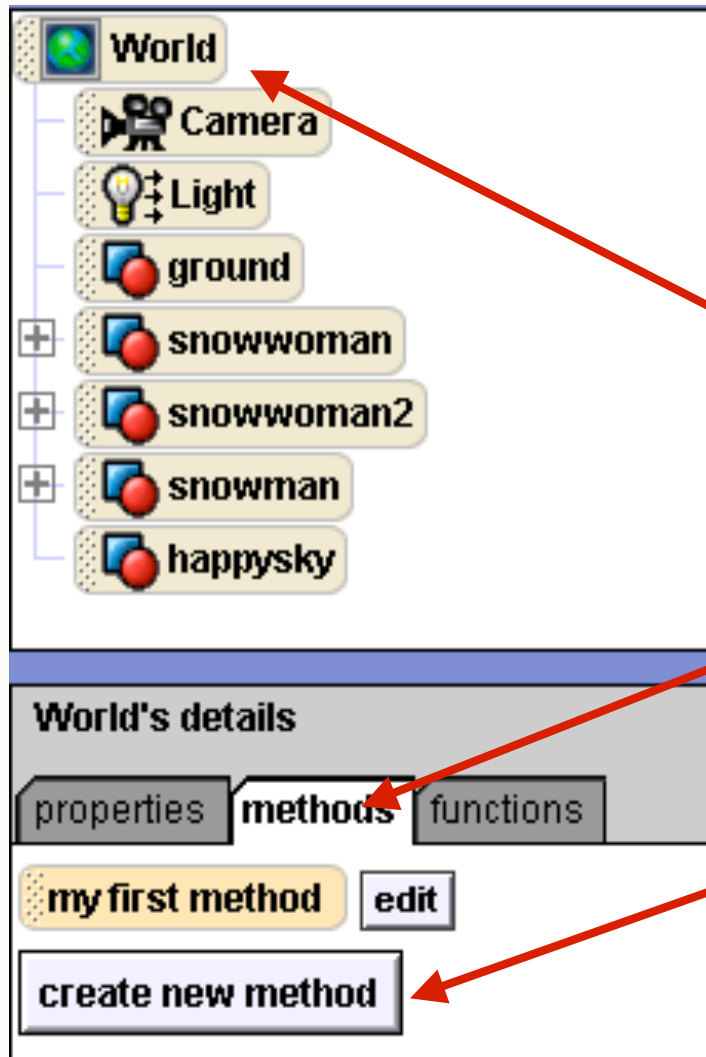


# Stepwise Refinement

- The process of breaking a problem down into large tasks and then breaking each task down into simpler steps is called **stepwise refinement**.
- Once the storyboard is completed, we write a method for each task.



# Demo: Starting a new method



First, to associate the new method with the World



- select the World tile in the Object Tree
- select the methods tab in the details area
- click on the "create new method" button



# Demo

## Ch04Lec1 Snowpeople

### Concepts illustrated in this example world:

-  *catchAttention* is a **world-level method** because it is defined as a method for World and has instructions that involve more than one object (snowman, snowwoman, camera)
-  The *catchAttention* method is executed by **calling** (invoking) the method .



# Why?

## 🌐 Why do we want to write our own methods?

- 👤 saves time -- we can call the method again and again without reconstructing code
- 👤 reduces code size – we call the method rather than writing the instructions again and again
- 👤 allows us to "think at a higher level"
  - 💡 can think *catchAttention* instead of  
"turn head to face the camera, then say 'Ahem' while moving eyes up and down"
  - 💡 the technical term for "think at a higher level" is "**abstraction**"



# Parameters

Alice



# A beetle band



- Our task is to create an animation for a bug band as an advertisement for their next concert.



# Storyboards

 Each bug band member will perform a solo.

*Do together*  
*Do in order*  
**georgeBeetle** move up  
**georgeBeetle** move down  
*play sound*

*Do together*  
*Do in order*  
**ringoBeetle** move up  
**ringoBeetle** move down  
*play sound*

*Do together*  
*Do in order*  
**paulBeetle** move up  
**paulBeetle** move down  
*play sound*

*Do together*  
*Do in order*  
**lennonBeetle** move up  
**lennonBeetle** move down  
*play sound*








# Demo

 Ch04Lec2BeetleBand-v1

 Concepts illustrated

-  To play a sound, a sound file must first be imported into Alice. (Alice is not a sound editor.)
-  This code is only for `georgeBeetle`.
-  Three more methods (one for each band member) will be needed!



# A Better Solution

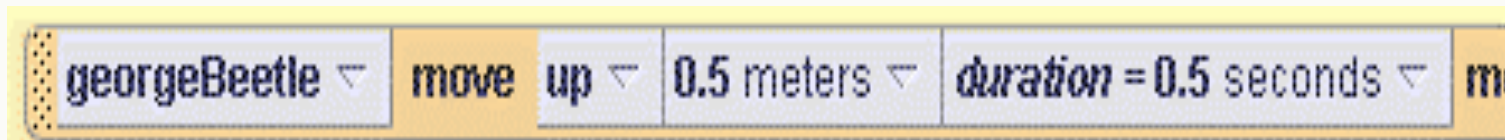
- Four versions of very similar code seems a bit tedious. The only things that change are the beetle and the music that plays.
- A better solution is to write a more flexible method.



# Parameters

- Built-in methods provide **flexibility** by providing parameters such as distance and direction.
- Parameters allow you to pass in values (**arguments**).

🐞 Example



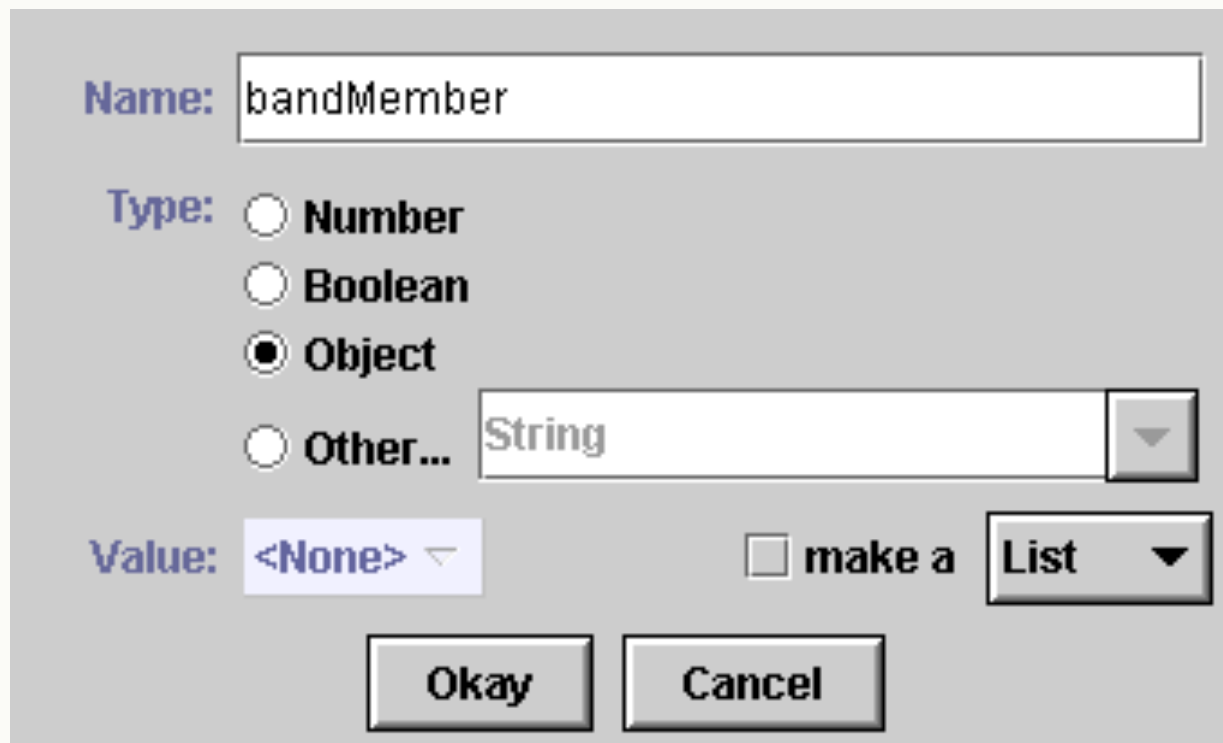
Parameters: distance, direction

Arguments: 0.5 meters, 0.5 seconds



# Kinds of Parameters

- 🌐 Alice provides several kinds of parameters that can be used in your own methods.



A screenshot of a parameter configuration dialog box. The dialog has a light gray background and contains the following elements:

- Name:** A text field containing the text "bandMember".
- Type:** A group of four radio buttons:
  - ☐ Number
  - ☐ Boolean
  - ☒ Object
  - ☐ Other...
- String Selection:** A text field containing the text "String" with a dropdown arrow on the right.
- Value:** A dropdown menu currently showing "<None>".
- make a List:** A checkbox that is currently unchecked, followed by a dropdown menu showing "List".
- Buttons:** Two buttons at the bottom: "Okay" and "Cancel".



# The storyboard

In this example, we can write just one method and use parameters to specify:  
which band member is to perform and  
which music should be played.

*solo*

Parameters: ***bandMember, music***

*Do together*

*Do in order*

***bandMember*** move up

***bandMember*** move down

play ***music***



# Demo

## Ch04Lec2BeetleBand-v2

## Concepts illustrated

 Enter name and select the type of each parameter

 bandMember is an **Object parameter**

 music is a **Sound parameter**

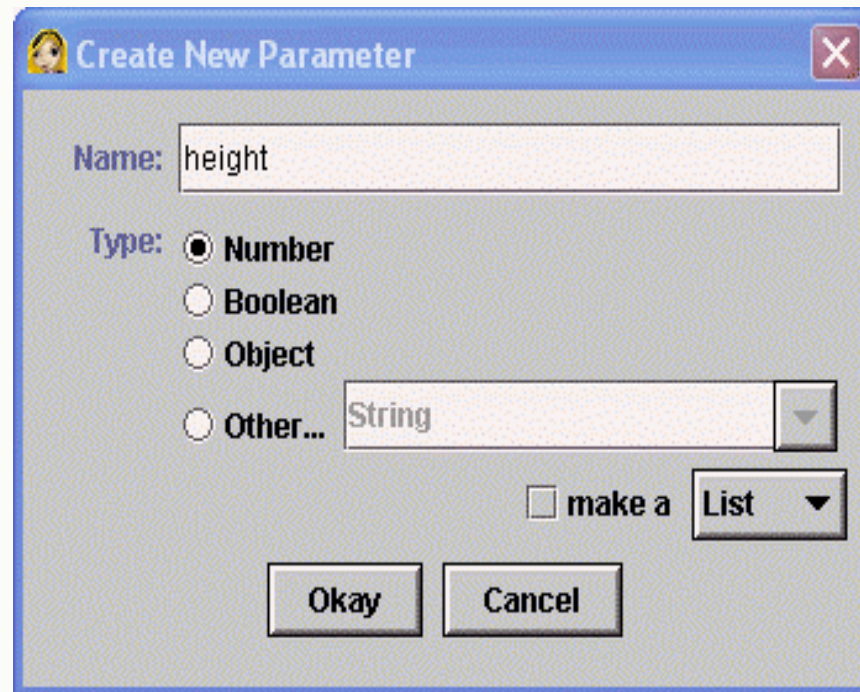
 A parameter acts as a placeholder in the instruction

 Arguments are passed to the parameter in the call to the method



# A Number parameter

Add a Number parameter to specify the *height* the *bandMember* jumps up and down.



Note that the call to the method must now include an argument for the height.

