

# Inheritance and Miscellaneous Things

## 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)

## 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.
- ✦ AND...
  - With the current version of Alice we're working with, there is greater chance that it'll crash!

## Solution

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

```
Do in order
surprise - spiderRobot and alienOnWheels surprise each other
investigate - spiderRobot gets a closer look at alienOnWheels
react - alienOnWheels hides and spiderRobot sends message
```

## Writing Custom Class-Level Methods

- ✦ If a class does not provide a method that is needed, a custom method can be written for an object of that class
- ✦ Primitive methods are available for various classes
  - Move, Turn, Roll, Say, Think, etc.
- ✦ Some classes have customized methods written just for them
  - wing\_flap, hop, walk, jump

## Next Step

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

– Example:

```
surprise
Do in order
alienOnWheels moves up
alienOnWheels says "Slithy toves?"
spiderRobot's head turns around
```

## 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.

## An example (building technique)

✦ How can we create a *skate* method for ice skater objects?



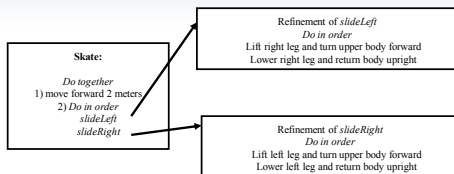
- We need to:
- (1) tell Alice to associate the new method (the one we are about to write) with an ice skater, and
  - (2) write a new method to animate the ice skater skating.

## Storyboard for *skate*

```
Skate:
Do together
move skater forward 2 meters
Do in order
slide on left leg
slide on right leg
```

- ✦ The **slide** actions each require several motion instructions, so we will break down these two actions into smaller steps

## Stepwise Refinement



## Inheritance

- ✦ The *CleverSkater* class
  - **inherits** all the properties and methods from the original *IceSkater* class, and also
  - has the newly defined methods (*skate*, *slideLeft*, *slideRight*)
- ✦ In other programming languages, the concept of creating a new class based on a previously defined class is called **inheritance**.

## Inheritance

- ✦ New methods were added and a new class was created.
- ✦ The new class still contained the methods that had previously been associated with the original object and class (such as Move, Turn, Say)
- ✦ The new class also contains the properties that had previously been associated with the original object and class (such as Color, Opacity, Vehicle)
- ✦ The new class *inherited* all the original methods and properties from the original class

5-13

## Stepwise Refinement

- ✦ Sometimes the algorithm does NOT have enough detail

- ✦ The algorithm must be *refined* and have more detail added to it

- ✦ This may mean that the algorithm becomes several methods

5-14

## Stepwise Refine Example



5-3

Original Algorithm:  
Raise the right leg  
Loop 10 times:  
Do together:  
    Lower the right leg  
    Raise the left leg  
End Do together  
Do together:  
    Lower the left leg  
    Raise the right leg  
End Do together  
End Loop  
Lower the right leg

*So, where can the changes be made?*

*What needs more detail?*

5-15

## Stepwise Refine Example



5-3

Original Algorithm:  
Raise the right leg  
Loop 10 times:  
Do together:  
    Lower the right leg  
    Raise the left leg  
End Do together  
Do together:  
    Lower the left leg  
    Raise the right leg  
End Do together  
End Loop  
Lower the right leg

Modified Algorithm:  
Call the `rightLegUp` method  
Loop 10 times:  
Do together:  
    call the `rightLegDown` method  
    call the `leftLegUp` method  
End Do together  
Do together:  
    call the `leftLegDown` method  
    call the `rightLegUp` method  
End Do together  
End Loop  
call the `rightLegDown` method

Example:  
tutorial 5\_3

5-16

## Stepwise Refinement

- ✦ The algorithm is not necessarily incorrect, it just needs more detail

- ✦ Development and use of new methods means that code can sometimes be reused

- ✦ Development and use of new methods reduces the amount of code written

5-17

## Passing Arguments

- ✦ Methods are written to accept *arguments*
- ✦ Arguments are values that are passed

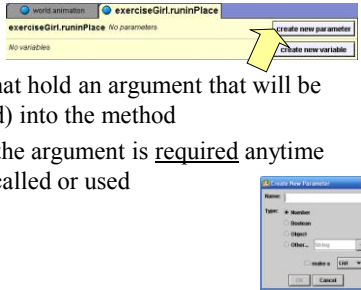


- ✦ Example of arguments:  
– Direction, distance traveled  
*What are the arguments here?*

5-18

### Passing Arguments

- ✦ Passing arguments requires the use of parameters
- ✦ Parameters are variables that hold an argument that will be passed (or used) into the method
- ✦ Once created, the argument is required anytime the method is called or used



### Passing Arguments Example



- ✦ Rather than have ExerciseGirl runInPlace X times, ...
  - a parameter could be created called *repetitions*
  - would allow the viewer to enter an amount ExerciseGirl should run in place

### Object Parameters

- ✦ Some parameters can be an object since some methods use objects as an argument
  - turn to face is a method with an object as an argument
  - move to is another example
- ✦ When creating an object parameter, identify the new parameter as an *object*



### Guidelines

- ✦ To avoid potential misuse of class-level methods, follow these guidelines:
  - Avoid references to other objects
  - Avoid calls to world-level methods
  - Play a sound only if the sound has been imported and saved out as part of the new class
- ✦ If these guidelines are not followed and an instance of the new class is added to another world, Alice will open an Error dialog box to tell you something is wrong.

### Calling a built-in function

- ✦ Custom Procedures can call (use) existing procedures from the object!

### Reuse

- ✦ Writing methods to make an ice skater perform a skating motion is an intricate task.
- ✦ We would like to have the iceSkater skate in other worlds without having to write the methods again.
- ✦ The idea of being able to use previously written program code in another program is known as reuse.

# Creating a new method



- When creating new procedure/method:
- remember that **ONLY** the object that is getting the new procedure can be in it
  - Why?

# Passing Arguments

- Methods are written to accept *arguments*
- What are arguments?



# Passing Arguments

- So, what is a parameter?



- Parameters are variables that hold an argument that will be passed (or used) into the method
- Once created, the argument is required anytime the method is called or used

# 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 *surprise* instead of "The alien moves up and says 'Slithy toves?' and then the robot's head turns around."
    - the technical term for "think at a higher level" is "**abstraction**"

# Tips for Visual Effects and Animation

- Programming the Camera
  - Camera can be programmed just like other objects
  - All the same primitive methods and functions are available for it:
    - Point at, Move, Turn, etc.
  - Camera can also be a vehicle for other objects!



# Tips for Visual Effects and Animation

- Creating Dummy Objects
  - Dummy objects are invisible objects that are placed in the world
  - Camera then moves to the invisible object to get different perspectives

