Class-level Methods and Inheritance

Alice
Class-level Methods

Some actions are naturally associated with a specific class of objects.

Examples

- A person walking
- A wheel rolling

We can write our own methods to define an action for a specific class of objects -- a **class-level** method.
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.
Demo: The solution

First, to associate the animation with the ice skater

- select the iceSkater tile in the Object Tree
- select the methods tab in the details panel
- click on the *create new method* button
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

Skate:
Do together
1) move forward 2 meters
2) Do in order
   slideLeft
   slideRight

Refinement of slideLeft
Do in order
   Lift right leg and turn upper body forward
   Lower right leg and return body upright

Refinement of slideRight
Do in order
   Lift left leg and turn upper body forward
   Lower left leg and return body upright
Concepts illustrated in this example world

- A method defined for a specific type of object defines an action for that object.
- A method can call other methods.

In this example, the \textit{skate} method calls \textit{slideRight} and \textit{slideLeft}. 
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`. 
A new class

1) Rename `iceSkater` as `cleverSkater`.
2) Save out as a new class. Alice saves the new class as `CleverSkater.a2c`
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.
Importing CleverSkater

An instance of the CleverSkater class can be added to a new world – use File|Import.
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.
What if there is no penguin in the new world where a cleverSkater object is imported?
Problem

Suppose you really want to write a class-level method where another object is involved? For example, a method to make the skater skate around another object-- in this scene, the penguin.
A solution is to write a class-level method with an object parameter that allows you to pass in the specific object.

```python
cleverSkater.skateAround
Parameter: whichObject

Do in order

Do together
  cleverSkater turn to face whichObject
  cleverSkater lift right leg
  cleverSkater move to whichObject
  cleverSkater turn around whichObject
```
Most of the *skateAround* storyboard design is straightforward and easy to code.

One step, however, requires some thought:

`cleverSkater` move to *whichObject* --
what distance should the `cleverSkater` move?
Calling a built-in function

The instruction to move the skater to `whichObject` (penguin, in this example) would look like this:

Unfortunately, the skater will collide with the penguin because the distance between two objects is measured center-to-center.
Expression

To avoid a collision, use a math operator to create an expression that adjusts the distance.

Math operators in Alice:

- addition +
- subtraction −
- multiplication *
- division /

Example:

cleverSkater move forward cleverSkater distance to whichObject - 1
Demo

Ch04Lec3SkateAround

Concepts illustrated:

- A parameter acts as a placeholder for the object that will be passed in
- A call to the *distance to* function returns a number value
- A math expression can be created as part of an instruction
Interactive Programming

Alice
Control of flow

*Control of flow* -- how the sequence of actions in a program is controlled.

- What action happens first, what happens next, and then what happens… and so on.

In movie-style programs (Chapters 1-4) the sequence of actions is determined by the programmer

- Creating a storyboard design
- Writing program methods to carry out the designed sequence
Interactive Animations

In interactive programs, the sequence of actions is determined at runtime when the user provides input
- clicks the mouse
- presses a key on the keyboard
- some other source of input

In essence, control of flow is now “in the hands of the user!”
Events

Each time the user provides some sort of input, we say an event is generated.

An event is “something that happens”
Event Handling methods

An event may
- Trigger a response, or
- Move objects into positions that create some condition (e.g., a collision) that triggers a response.

A method is called to carry out the response. We call this kind of method an **event handling method**.

When an event is linked to a method that performs an action, a **behavior** is created.
Example

Build an air show flight simulator. In an air show, the pilot uses biplane controls to perform acrobatic stunts.
Problem

The whole idea in a flight simulator is to allow the user to control the flight path. The problem is: how do we write our program code to provide a guidance system that allows the user to be the pilot?
Solution

Use keyboard input

- Up-arrow key to move the biplane forward
- Spacebar to make the biplane do a barrel turn

(Note: other sets of keys could be used, we just arbitrarily picked a couple of keys on the keyboard.)

Write event handler methods that respond to each key press
Since two keys are used, two events are possible – so two storyboards are needed:

**Event:** Spacebar press
**Response:**
*Do together*
- roll biplane a full revolution
- play biplane engine sound

**Event:** Up Arrow key press
**Response:**
*Do together*
- move biplane forward
- play biplane engine sound

Each storyboard outlines an event handler that responds to a particular event.
Demo

Ch05Lec1BiplaneAcrobat

Concepts illustrated:

- Events are created in the event editor
- A method is called to handle each event
- Synchronize the duration of the animation with the length of a sound. To change the length of a sound, use audio editing software.
Mouse input

Interactive programs often allow the user to use a mouse to click

- buttons in a windows-based interface
- targets in a game
- a checklist of items on a business form

In this session, we look at how to use mouse input.
Example

People are trapped in a burning building and the user will click the mouse cursor on the person to be rescued next.

In setting up this demo world, we positioned the fire truck so the distance of the ladder from the 1st floor is 1 meter, 2nd floor is 2 meters, and 3rd floor is 3 meters.
Storyboard

Three people are to be rescued. So, we could write three different methods.

- **Event:** Click on guy1
  **Responding Method:** Save guy on the first floor

- **Event:** Click on girl2
  **Responding Method:** Save girl on the second floor

- **Event:** Click on girl3
  **Responding Method:** Save girl on the third floor
A Better Solution

A better solution is to write one event handler method and send in the information needed to perform the action.

```plaintext
firetruck.savePerson:

parameters: whichFloor, whichPerson, howFar

Do in order
  point ladder at whichFloor
  extend the ladder howFar meters
  whichPerson slide down the ladder to the fire truck
  pull the ladder back howFar meters
```

*whichFloor and whichPerson* are Object parameters

*howFar* is a Number parameter
Concepts illustrated:

- Parameters allow one event handling method to handle three different events.
- The fireAnim object has a built-in event and event handling method.
Example 2

Zeus was a powerful god in Greek mythology. When Zeus was angry, he would shoot a thunderbolt out of the heavens to strike anyone who got in the way.

The user will choose the philosopher who will be the next target of Zeus’s anger.
A possible design is a method with an Object parameter, named *who*, for the object that was clicked.

| Event: | An object is mouse-clicked |
| Event handler: | shootBolt |
| Parameter: | *who* — the object that was clicked |

**Do in order**
- prepare to strike the object that was clicked
- thunder plays and lightning strikes the object that was clicked
- lightning is repositioned for the next strike

The actions in this storyboard are complex.
We can break the actions down into simpler steps using stepwise refinement.
Event: An object is mouse-clicked

Event handler: shootBolt

Parameter: who — the object that was clicked

Do in order
  call prepareToShoot method — send who as the target
  call lightningAndThunder method — send who as the target
  lightning move to cloud’s position

prepareToShoot:

Parameter: target

Do together
  turn Zeus to face the target
  make the lightning bolt visible

lightningAndThunder:

Parameter: target

Do together
  play sound
  call specialEffects method — send target

specialEffects:

Parameter: target

Do in order
  Do together
    lightning bolt move to target
    smoke move to target
  Do together
    set smoke to visible
    set lightning to invisible
    call smoke cycle — built-in method
    set target color to black
    move target up and down
Demo

Ch05Lec2Zeus

Concepts illustrated:

- The *shootBolt* method is at the top of the storyboard design. This method calls two other methods – a *driver* method.

- *object under the mouse cursor* is used in the *When mouse is clicked* event to pass the clicked object as the target.
Opacity

In this example, opacity is used to make the lightning bolt visible and invisible.

- In setting up the initial scene, the lightning bolt is made invisible by setting its opacity to 0 (0%).
- In preparing to shoot the lightning bolt, it was made visible by setting its opacity to 1 (100%).

(Review Tips & Techniques 4 for more details about opacity.)
Built-in methods

*cycleSmoke* is a special built-in method for the smoke object. The duration of *cycleSmoke* is about 2 1/2 seconds.

Several statements in the *shootBolt* and *specialEffects* methods use a *move to* instruction.

The *move to* instruction moves an object to a particular position in the world. In the example above, the lightening bolt is moved to the position of the cloud.

(The *move to* instruction is described in detail in **Tips & Techniques 2**.)
Testing

When parameters are used in interactive programming, it is especially important to test that all possible parameter values work as expected.

What happens if you click on each philosopher, one at a time?

Also try things that shouldn’t work.

What happens if you click on a column?
What happens if you click on a philosopher twice?