An Application

- **Goal**: Presentation program (e.g. PowerPoint)
- **Problem**: There are many types of content
  - **Examples**: text box, rectangle, image, etc.
  - Have to write code to display each one
- **Solution**: Use object oriented features
  - Define class for every type of content
  - Make sure each has a draw method:
    ```python
    for x in slide[i].contents:
        x.draw(window)
    ```

Defining a Subclass

```python
class SlideContent(object):
    """Any object on a slide."""
    def __init__(self, x, y, w, h):
        pass
    def draw_frame(self):
        pass
    def select(self):
        pass

class TextBox(SlideContent):
    """An object containing text."""
    def __init__(self, x, y, text):
        pass
    def draw(self):
        pass

class Image(SlideContent):
    """An image."""
    def __init__(self, x, y, image_file):
        pass
    def draw(self):
        pass
```

Class Definition: Revisited

```python
class <name>(<superclass>):
    """Class specification"""
    getters and setters
    initializer (__init__)
    definition of operators
    definition of methods
    anything else
```

Object and the Subclass Hierarchy

- **Subclassing creates a hierarchy** of classes
  - Each class has its own super class or parent
  - Until object at the "top"
  - object has many features
    - Special built-in fields: __class__, __dict__
    - Default operators: __str__, __eq__

A Simpler Example

```python
class Employee(object):
    """Instance is a salaried worker
    INSTANCE ATTRIBUTES:
    name: full name [string]
    start: first year hired
    salary: yearly wage [float]
    """

class Executive(Employee):
    """An Employee with a bonus
    INSTANCE ATTRIBUTES:
    bonus: annual bonus [float]
    """
```
Method Overriding

- Which __str__ do we use?
  - Start at bottom class folder
  - Find first method with name
  - Use that definition
- New method definitions override those of parent
- Also applies to
  - Initializers
  - Operators
  - Properties
    - all “methods”

New method definitions override those of parent

- Also applies to
  - Initializers
  - Operators
  - Properties
    - all “methods”

Accessing the “Previous” Method

- What if you want to use the original version method?
  - New method = original + more
  - Do not want to repeat code from the original version
- Call old method explicitly
  - Use method as a function
  - Pass object as first argument
- Example:
  ```python
  Employee.__str__(self)
  ```

Primary Application: Initializers

```python
class Executive(Employee):
    def __init__(self, n, d, b=0.0):
        Employee.__init__(self, n, d)
        self._bonus = b
```

Instance Attributes are (Often) Inherited

```python
class Executive(Employee):
    __init__ = Employee.__init__
    __str__ = Employee.__str__
    __eq__ = Employee.__eq__
```

Also Works With Class Attributes

**Class Attribute**: Assigned outside of any method definition

```python
class Employee(object):
    STD_SALARY = 50000.0
```

```python
class Executive(Employee):
    STD_BONUS = 10000.0
```

Properties and Inheritance

- Properties: all or nothing
  - Typically inherited
  - Or fully overridden (both getter and setter)
  - When override property, completely replace it
    - Cannot use super()
  - Very rarely overridden
    - Exception: making a property read-only
- See employee2.py