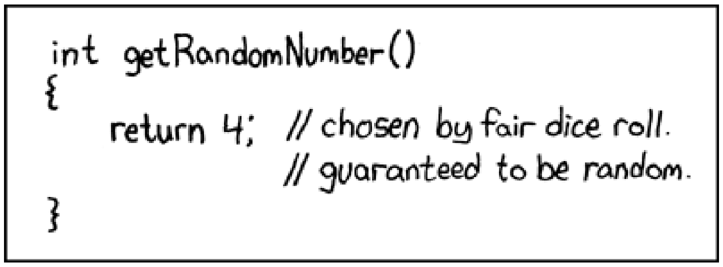
Example: Class MSDie
Introduction to Graphs and NetworkX

Monday, March 30, 2009

1

Be careful whose random number generator you trust …


```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

http://xkcd.com/221/

2

Reminders …

- Sample exam questions have been posted on Blackboard
- Exam 2 in Thursday, April 2, 6:30 in Lab.
- Questions on Project 3?
  - 4 Plots generated, 3 inside the simulation are optimal
  - Plots will be similar for the two initial states
  - What values are interesting change?
    steps (larger than 3000), the range of Dv (larger)

3

Class example: Multi-Sided Dice

- **Design** a generic class MSDie to model multi-sided dice.
- When a new object is created, we specify *n*, the number of sides it will have.
- Three methods will operate on the die:
  - **roll** – set the die to a random value between 1 and *n*, inclusive
  - **setValue** – set the die to a specific value (i.e. cheat)
  - **getValue** – see what the current value is.

4

Example: Multi-Sided Dice

```
>>> die1 = MSDie(6)
>>> die1.getValue()
1
>>> die1.roll()
>>> die1.getValue()
5
>>> die2 = MSDie(13)
>>> die2.getValue()
1
>>> die2.roll()
>>> die2.getValue()
9
>>> die2.setValue(8)
>>> die2.getValue()
8
```

5

Example: Multi-Sided Dice

- Using object-oriented vocabulary, we create a die by invoking the MSDie *constructor* and providing the number of sides as a *parameter*.
- Die objects will keep track of this number internally as an *instance variable*.
- Another *instance variable* is used to keep the current value of the die.
- We initially set the value of the die to be 1 because that value is valid for any die.
- That value can be changed by the roll and setRoll methods, and returned by the getValue method.

6

Example: Multi-Sided Dice

```
# msdie.py
#     Class definition for an n-sided die.

from random import randrange

class MSDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value
```

7

Example: Multi-Sided Dice

- Class definitions have the form
  ```
  class <class-name>:
      <method-definitions>
  ```
- Methods look a lot like functions!
  Placing a function inside a class makes it a method of the class, rather than a stand-alone function.
- The first parameter of a method is *always* named `self`, which is a reference to the object on which the method is acting.

8

Example: Multi-Sided Dice

- Suppose we have a main function that executes die1.setValue(8).
- Just as in function calls, Python executes the following sequence:
  1. main suspends at the point of the method application.
  2. Python locates the appropriate method definition inside the class of the object to which the method is being applied.
  3. Control is transferred to the setValue method in the MSDie class.

9

Example: Multi-Sided Dice

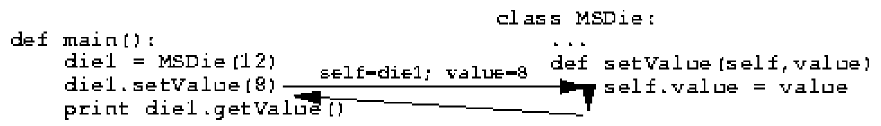
4. The formal parameters of the method get assigned the values supplied by the actual parameters of the call. In the case of a method call, the first formal parameter refers to the object:

   self = die1

   value = 8
5. The body of the method is executed.

10

Example: Multi-Sided Dice



```
                                              class MSDie:
def main():                                       ...
    die1 = MSDie(12)                          def setValue(self,value)
    die1.setValue(8)                              self.value = value
    print die1.getValue()
```

Example: Multi-Sided Dice

- Objects contain their own data.
  Instance variables provide storage locations inside of an object.
- Instance variables are accessed by name using our dot notation: `<object>.<instance-var>`
- `self.value` refers to the instance variable `value` inside the object.
  Each `MSDie` object has its own `value`.

Example: Multi-Sided Dice

- Certain methods have special meaning.
  Thet have names that start and end with two _'s.

```
def __init__(self, sides):
      self.sides = sides
      self.value = 1
```

- __init__ is the object contructor.
  - Python calls this method to initialize a new MSDie.
  - __init__ provides initial values for the instance variables of an object.

13

Example: Multi-Sided Dice

- Outside the class, the constructor is referred to by the class name:
  die1 = MSDie(6)
  - When this statement is executed, a new MSDie object is created and __init__ is executed on that object.
  - The net result is that die1.sides is set to 6 and die1.value is set to 1.
- Instance variables can remember the state of a particular object
  - This is different than local function variables, whose values disappear when the function terminates.

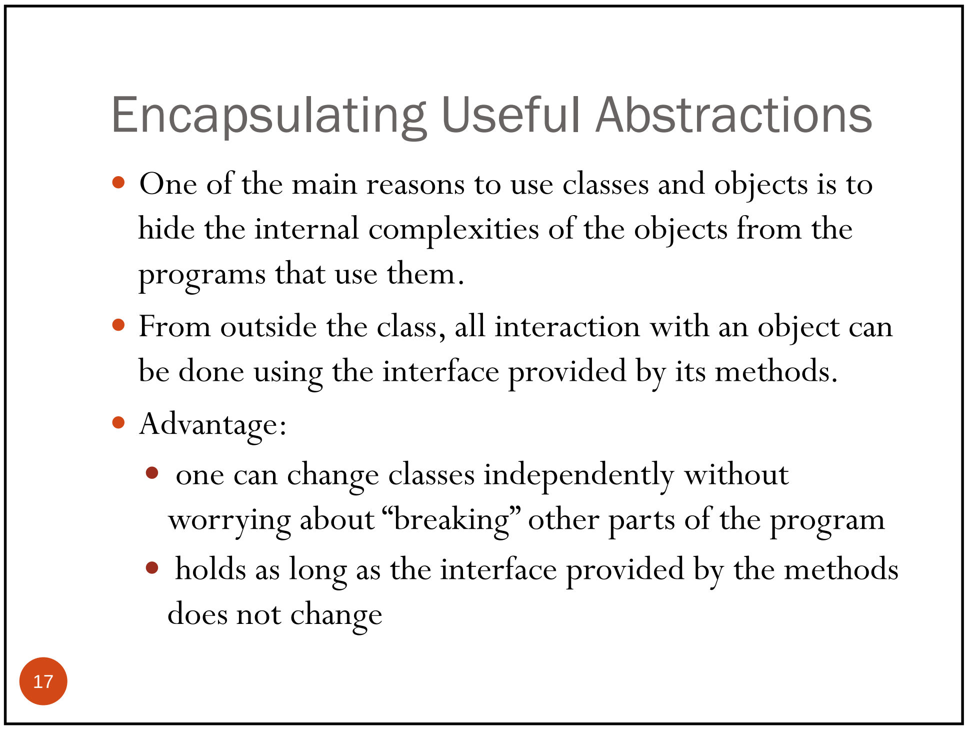
14

Object Oriented (OO) Design

- Any program using classes can be written as one without classes
- Defining new classes is a good way to modularize a program.
- Once some useful objects are identified, the implementation details of the algorithm can be moved into a suitable class definition.
- OO allows: encapsulation, polymorphism, inheritence, information hiding

Encapsulating Useful Abstractions

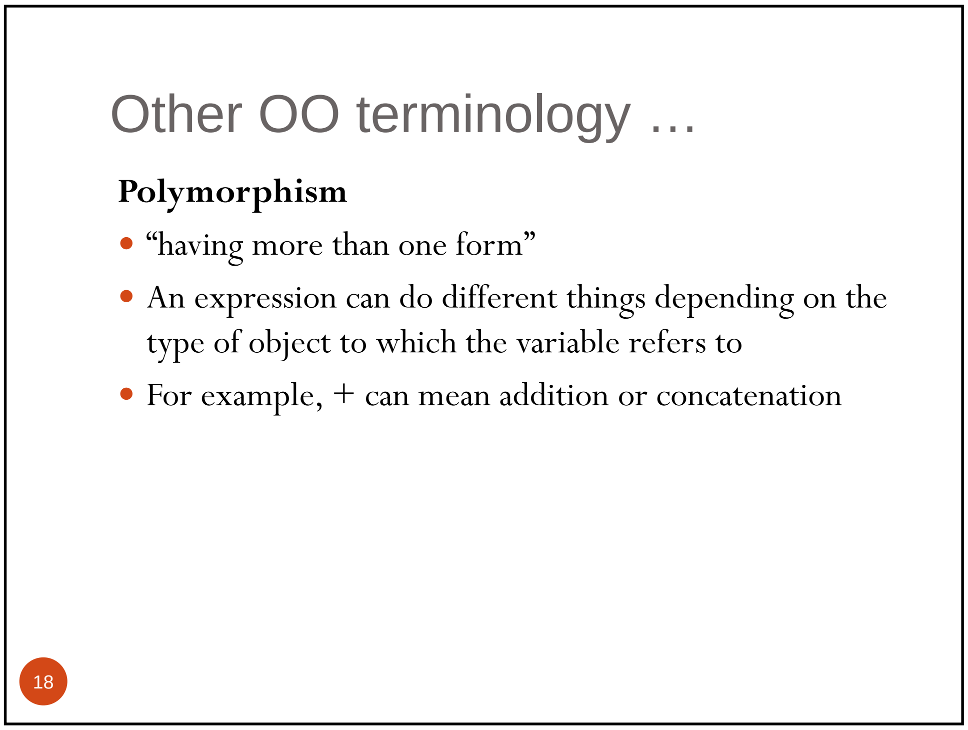
- The main program only has to worry about what objects can do, not about how they are implemented.
- In computer science, this separation of concerns is known as **encapsulation.**
- The implementation details of an object are encapsulated in the class definition, which insulates the rest of the program from having to deal with them.



Encapsulating Useful Abstractions

- One of the main reasons to use classes and objects is to hide the internal complexities of the objects from the programs that use them.
- From outside the class, all interaction with an object can be done using the interface provided by its methods.
- Advantage:
  - one can change classes independently without worrying about "breaking" other parts of the program
  - holds as long as the interface provided by the methods does not change

17

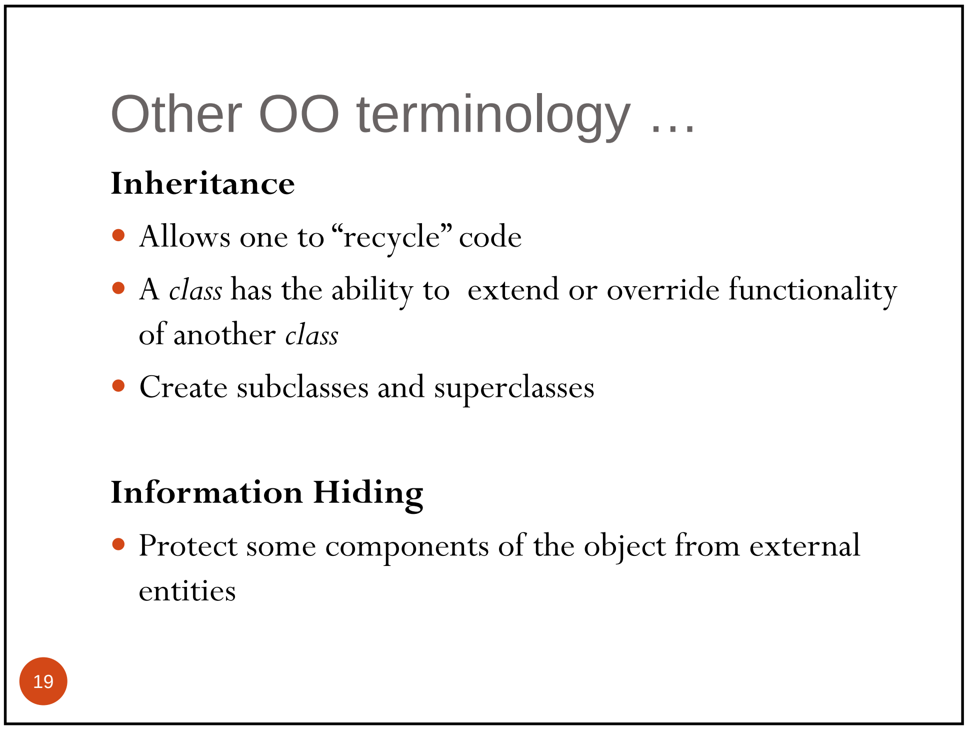


Other OO terminology …

Polymorphism

- "having more than one form"
- An expression can do different things depending on the type of object to which the variable refers to
- For example, + can mean addition or concatenation

18



Other OO terminology …

Inheritance

- Allows one to “recycle” code
- A *class* has the ability to extend or override functionality of another *class*
- Create subclasses and superclasses

Information Hiding

- Protect some components of the object from external entities

19