Special Topics in Computer Science- CSC 4992

Introduction to Object-Based Programming

Organizing Programs

- As problems get more interesting and difficult, solutions (programs) become more complex
- How do we control this complexity, so as to keep it manageable and not be overwhelmed by it?

One Answer: Abstraction

Find a way of hiding complex details in a new unit of code, thereby treating many things as one thing

Examples:

A function - hides an algorithm in a single entity (**`math.sqrt`**, **`reply`**)

A module - hides a set of functions in a single entity (**`random`**, **`math`**)

A data structure - hides a collection of data in a single entity (list, dictionary)

Using Abstractions

- A program becomes a set of cooperating modules, functions, and data structures
- Each program component is simple enough to be understood immediately and tested independently
- When interactions among components become complex, package these in a new abstraction (another function, data structure, or module)

Another Abstraction: Objects

- An *object* combines data and operations into a single unit
- An object is like an intelligent agent that knows what to do with its own data
- Examples:
  - A GUI window
  - A command button, a data entry field, a drop-down menu
  - A point, circle, rectangle, or polygon
  - An image, a pixel, a sound clip, or a network connection
  - Actually, any Python data value (lists, strings, integers, etc.)

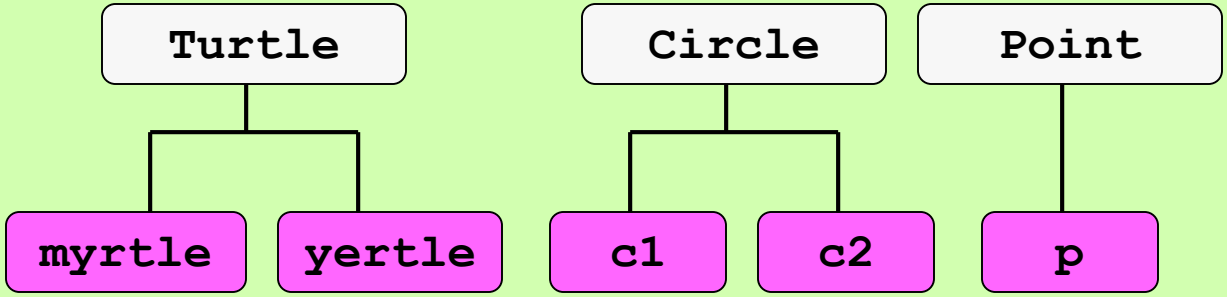
Creating Objects from Classes

```
myrtle = Turtle()
yertle = Turtle()

p = Point(50, 50)

c1 = Circle(p, 50)

c2 = Circle(p, 30)
```



`Turtle`, **`Point`**, and **`Circle`** are *classes*

`myrtle`, **`yertle`**, **`p`**, **`c1`**, and **`c2`** refer to objects or *instances* of these classes

A *class* defines the behavior of a set of *objects*

Getting Objects to Do Things

```
myrtle = Turtle()

p = Point(50, 50)

c1 = Circle(p, 50)

c2 = Circle(p, 30)

c1.draw(myrtle), c2.draw(myrtle)

print(c1.getRadius(), c2.getRadius())
```

We get an object to do things by running *methods*

A method is like a function: it hides an algorithm for performing a task

`draw` and **`getRadius`** are *methods*

Objects: State and Behavior

An object has two essential features:

1. A *behavior*, as defined by the set of methods it recognizes

2. A *state*, as defined by the data that it contains

Object-based programming uses computational objects to model the state and behavior of real-world objects

Modeling

- Closely study the relevant attributes and behavior of the objects in the system being modeled

- Design and code computational objects that reflect these attributes and behavior

Example: Dice

- Dice are used in many games of chance (backgammon, monopoly, etc.)
- A single die is a cube whose sides represent the numbers 1-6
- When a die is rolled, the number selected is the one on the side that happens to be facing up

State and Behavior of a Die

- The state of a die is the number currently visible on its top face; its initial state is a randomly chosen number from 1 through 6

- Behavior:
  - *Roll* the die to reset its state to a randomly chosen number from 1 through 6
  - *Get* the die's current number

- Verbs in the description indicate behavior; nouns indicate attributes (state)

The `Die` Class: External View

```
Die()                   # Returns a new Die object

roll()                  # Resets the die's value

getValue()              # Returns the die's value
```

The set of a class's methods is also called its *interface*

The user of class only needs to know its interface

Using Some Dice

```
from die import Die

# Instantiate a pair of dice
d1 = Die()
d2 = Die()

# Roll them and view them 100 times
for x in range(100):
    d1.roll()
    d2.roll()
    print(d1.getValue(), d2.getValue())
```

Object Instantiation

```
from die import Die

# Instantiate a pair of dice
d1 = Die()
d2 = Die()

# Roll them and view them 100 times
for x in range(100):
    d1.roll()
    d2.roll()
    print(d1.getValue(), d2.getValue())
```

Syntax of object instantiation:

```
<variable> = <class name>(<arguments>)
```

Calling a Method

```
from die import Die

# Instantiate a pair of dice
d1 = Die()
d2 = Die()

# Roll them and view them 100 times
for x in range(100):
    d1.roll()
    d2.roll()
    print(d1.getValue(), d2.getValue())
```

Syntax of a method call:

```
<object>.<method name>(<arguments>)
```

Using Dice: The Game of Craps

- Played with a pair of dice
- An initial roll of 7 or 11 wins
- An initial roll of 2, 3, or 12 loses
- Any subsequent roll of 7 loses
- Any subsequent roll that equals the initial roll wins

Example: Checking Accounts

- A checking account holds a balance
- We can make deposits or withdrawals
- Interest checking allows for a periodic computation of interest, which is added to the balance

State and Behavior of a Checking Account

- The state of an account is
  - The owner's name
  - The owner's PIN
  - The current balance

- The interest rate is common to all accounts

- Behavior:
  - Make a deposit of a given amount
  - Make a withdrawal of a given amout
  - Compute the interest
  - Get the current balance

The Interface of the **`CheckingAccount`** Class

```
CheckingAccount(name, pin, bal)  # Returns a new object

getBalance()                     # Returns the current balance

deposit(amount)                  # Makes a deposit

withdraw(amount)                 # Makes a withdrawal

computeInterest()                # Computes the interest and
                                 # deposits it
```

Using a Checking Account

```
from bank import CheckingAccount

# Instantiate an account
account = CheckingAccount('Ken', '3322', 1000.00)

# Do some things with it
print(account.getBalance())
account.deposit(500.00)
print(account.getBalance())
account.withdraw(1200.00)
print(account.getBalance())
account.computeInterest()
print(account.getBalance())
```

Each Account Has Its Own State

```
from bank import CheckingAccount

# Instantiate two acounts
jackAccount = CheckingAccount('Jack', '3322', 1000.00)
jillAccount = CheckingAccount('Jill', '3323', 1000.00)

jackAccount.deposit(50.00)

jillAccount.withdraw(100)
```

The states of distinct objects can vary, but
the methods that apply to them are the same

Object-Based Programming: Summary

- Study the system of objects that you're modeling to discover the relevant attributes and behavior
- For each type of object, choose a class whose methods realize its behavior
- Write a short tester program that creates some objects and runs their methods
- Use the objects in your application