Mini-Lecture 19

Designing Classes
Invariants

• Properties of an attribute that must be true
• Works like a precondition:
  ▪ If invariant satisfied, object works properly
  ▪ If not satisfied, object is “corrupted”
• Examples:
  ▪ Point3 class: all attributes must be floats
  ▪ RGB class: all attributes must be ints in 0..255
• Purpose of the class specification
The Class Specification

class Worker(object):

    """An instance is a worker in an organization.
    Instance has basic worker info, but no salary information.

    ATTRIBUTES:
    lname:   Worker’s last name.   [str]
    ssn:     Social security no.   [int in 0..999999999]
    boss:    Worker's boss.        [Worker, or None if no boss]
class Worker(object):

    """An instance is a worker in an organization. Instance has basic worker info, but no salary information.

    ATTRIBUTES:
    
    lname: Worker’s last name. [str]
    ssn: Social security no. [int in 0..999999999]
    boss: Worker's boss. [Worker, or None if no boss]"""
Creating a new Worker is a multi-step process:

- `w = Worker()`  
- `w.lname = 'White'`
- `...`

Want to use something like

```python
w = Worker('White', 1234, None)
```

- Create a new Worker **and** assign attributes
  - `lname` to `'White'`, `ssn` to 1234, and `boss` to None

Need a **custom constructor**
Special Method: \texttt{\_\_init\_}\n
\begin{verbatim}

def \_\_init\_(self, n, s, b):
    """Initializer: creates a Worker
    Has last name n, SSN s, and boss b
    Precondition: n a string, s an int in range 0..999999999, and b either
    a Worker or None.
    self.lname = n
    self.ssn = s
    self.boss = b"

w = Worker('Obama', 1234, None)
\end{verbatim}

Called by the constructor

\begin{itemize}
\item \texttt{lname} = 'White'
\item \texttt{ssn} = 1234
\item \texttt{boss} = None
\end{itemize}
Special Method: \texttt{\_\_init\_\_}

```python
def \_\_init\_(self, n, s, b):
    
    """Initializer: creates a Worker
    
    Has last name \texttt{n}, SSN \texttt{s}, and boss \texttt{b}
    
    Precondition: \texttt{n} a string, \texttt{s} an int in range 0..999999999, and \texttt{b} either a Worker or None.
    \texttt{self.lname} = \texttt{n}
    \texttt{self.ssn} = \texttt{s}
    \texttt{self.boss} = \texttt{b}
    
    """
```

Called by the constructor

```
Worker

<table>
<thead>
<tr>
<th>lname</th>
<th>'White'</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn</td>
<td>1234</td>
</tr>
<tr>
<td>boss</td>
<td>None</td>
</tr>
</tbody>
</table>
```

10/12/18

use \texttt{self} to assign attributes
Evaluating a Constructor Expression

Worker('Obama', 1234, None)

1. Creates a new object (folder) of the class Worker
   - Instance is initially empty
2. Puts the folder into heap space
3. Executes the method `__init__`
   - Passes folder name to self
   - Passes other arguments in order
   - Executes the (assignment) commands in initializer body
4. Returns the object (folder) name
Designing a Class

1. Think about what values you want in the set
   - What are the attributes? What values can they have?

2. Think about what operations you want
   - This often influences the previous question
     - To make (1) precise: write a *class invariant*
       - Statement we promise to keep true *after every method call*
     - To make (2) precise: write *method specifications*
       - Statement of what method does/what it expects (preconditions)
     - Write your code to make these statements true!
Planning out a Class

class Time(object):
    """Instances represent times of day.
    Instance Attributes:
    hour: hour of day [int in 0..23]
    min: minute of hour [int in 0..59]"

def __init__(self, hour, min):
    """The time hour:min.
    Pre: hour in 0..23; min in 0..59"

def increment(self, hours, mins):
    """Move this time <hours> hours
    and <mins> minutes into the future.
    Pre: hours is int >= 0; mins in 0..59"

def isPM(self):
    """Returns: this time is noon or later."""
Implementing a Class

• All that remains is to fill in the methods. (All?!)  
• When implementing methods:  
  1. Assume preconditions are true  
  2. Assume class invariant is true to start  
  3. Ensure method specification is fulfilled  
  4. Ensure class invariant is true when done  
• Later, when using the class:  
  ▪ When calling methods, ensure preconditions are true  
  ▪ If attributes are altered, ensure class invariant is true
Implementing an Initializer

```python
def __init__(self, hour, min):
    """The time hour:min.
    Pre: hour in 0..23; min in 0..59""
    self.hour = hour
    self.min = min
```

Instance variables:
- hour: hour of day [int in 0..23]
- min: minute of hour [int in 0..59]
Implementing a Method

Instance variables:
- hour: hour of day [int in 0..23]
- min: minute of hour [int in 0..59]

```python
def increment(self, hours, mins):
    """Move this time <hours> hours and <mins> minutes into the future. Pre: hours [int] >= 0; mins in 0..59""
    self.min = self.min + mins
    self.hour = self.hour + hours
```

This is true to start
What we are supposed to accomplish
This is also true to start
You put code here
This should be true at the end
Implementing a Method

Instance variables:
- hour: hour of day [int in 0..23]
- min: minute of hour [int in 0..59]

def increment(self, hours, mins):
    """Move this time <hours> hours and <mins> minutes into the future.
    Pre: hours [int] >= 0; mins in 0..59"
    self.min = self.min + mins
    self.hour = (self.hour + hours +
                    self.min // 60)
    self.min = self.min % 60
    self.hour = self.hour % 24

This is true to start
What we are supposed to accomplish
This is also true to start
You put code here
This should be true at the end
Role of Invariants and Preconditions

- They both serve two purposes
  - Help you think through your plans in a disciplined way
  - Communicate to the user* how they are allowed to use the class
- Provide the *interface* of the class
  - interface btw two programmers
  - interface btw parts of an app
- Important concept for making large software systems
  - Will return to this idea later

* ...who might well be you!

---

**interface** lˈɪntərˌfāsl noun
1. a point where two systems, subjects, organizations, etc., meet and interact : the interface between accountancy & the law.
   - *chiefly Physics* a surface forming a common boundary between two portions of matter or space, e.g., between two immiscible liquids: the surface tension of a liquid at its air/liquid interface.

2. *Computing* a device or program enabling a user to communicate with a computer.
   - a device or program for connecting two items of hardware or software so that they can be operated jointly or communicate with each other.

—The Oxford American Dictionary