Review 2

Classes and Subclasses
**Class Definition**

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

- Every class must extend *something*
- Most classes will extend *object*
Attribute Invariants

• What are the attribute invariants below?
• Why are they there?

class Time(object):
    """An instance is a time of day
    hr: hour of the day [int in range 0..23]
    min: minute of the hour [int in range 0..59]
    """
...
Attribute Invariants

- Attribute invariants are important for programmer
  - Can look at them when writing methods
  - Any reader of the code will benefit as well

```python
class Time(object):
    """An instance is a time of day
    hr: hour of the day [int in range 0..23]
    min: minute of the hour [int in range 0..59]
    """
...
Enforcing Invariants

- Attribute invariants are the purpose of constructors
- They initialize the attributes to satisfy invariants

```python
class Time(object):
    ...
    def __init__(self, t):
        """Initializer: makes an instance with time t, in minutes, in range 0..24*60-1""
        self.hr = t / 60
        self.min = t % 60
```

- Without seeing the invariants, might write `self.min = t`
Enforcing Invariants

- Restrict attribute access
  - Make attributes hidden
  - Force access through methods: getter & setter
- **Getter**: Read attribute
  - Just return attribute
- **Setter**: Change attribute
  - Checks that new value satisfies the invariant
  - If so, changes attribute

```python
class Time(object):
    """Instance Attributes:
    _hr [int in range 0..23]
    _min [int in range 0..59]""

    def getHour(self):
        """Returns: hour of the day""
        return self._hr

    def setHour(self, value):
        """Sets hour to value""
        assert type(value) == int
        assert value >= 0 and value <= 23
        self._hr = value
```
Special Methods

• Start/end with underscores
  ▪ `__init__` for initializer
  ▪ `__str__` for `str()`
  ▪ `__repr__` for backquotes

• Actually defined in `object`
  ▪ You are overriding them
  ▪ Many more of them

• For a complete list, see
  http://docs.python.org/reference/datamodel.html

```python
class Point(object):
    """Instances are points in 3D space""
    ...

    def __init__(self, x=0, y=0, z=0):
        """Initializer: makes new Point""
        ...

    def __str__(self):
        """Returns: string with contents""
        ...

    def __repr__(self):
        """Returns: unambiguous string""
        ...
```

12/8/13  Review 2
An object of class `Course` (next slide) maintains a course name, the instructors involved, and the list of registered students, sometimes called the roster.

1. State the purpose of an initializer. Then complete the body of the initializer of `Course`, fulfilling this purpose.
2. Complete the body of method `add` of `Course`
3. Complete the body of method `__eq__` of `Course`. If you write a loop, you do not need to give a loop invariant.
4. Complete the body of method `__ne__` of `Course`. Your implementation should be a single line.
class Course(object):
    """An instance is a course at Cornell. Maintains the name of the course, the roster (list of netIDs of students registered for it), and a list of netIDs of instructors.
    name: Course name [str]
    instructors: instructor net-ids [nonempty list of string]
    roster: student net-ids [list of string, can be empty]"

    def __init__(self, name, b):
        """Instance w/ name, instructors b, no students. It must COPY b. Do not assign b to instructors. Pre: name is a string, b is a nonempty list""
        # IMPLEMENT ME

    def add(self, n):
        """If student with netID n is not in roster, add student. Do nothing if student is already there. Precondition: n is a valid netID.""
        # IMPLEMENT ME

    def __eq__(self, ob):
        """Return True if ob is a Course with the same name and same set of instructors as this; otherwise return False"
        # IMPLEMENT ME

    def __ne__(self, ob):
        """Return False if ob is a Course with the same name and same set of instructors as this; otherwise return True"
        # IMPLEMENT ME IN ONE LINE
1. State the purpose of a initializer. Complete the body of the constructor of Course, fulfilling this purpose.
   - The purpose is to initialize instance attributes so that the invariants in the class are all satisfied.

```python
def __init__(self, name, b):
    """Instance w/ name, instructors b, no students.
    Pre: name is a string, b is a nonempty list"
    self.name = name
    self.instructors = b[:]  # Copies b
    self.roster = []  # Satisfy the invariant!
```

2. Complete the body of method `add` of `Course`

```python
def add(self, n):
    '''If student with netID n is not in roster, add
    student. Do nothing if student is already there.
    Precondition: n is a valid netID.'''
    if not n in self.roster:
        self.roster.append(n)
```
3. Complete body of method `__eq__` of Course.

```python
def __eq__(self, ob):
    """Return True if ob is a Course with the same name and same
    set of instructors as this; otherwise return False""
    if not (isinstance(ob, Course)):
        return False
    # Check if instructors in ob are in this
    for inst in ob.instructors:
        if not inst in self.instructors:
            return False
    # If instructors of ob are those in self, same if length is same
    return self.name==ob.name and len(self.instructors)==len(ob.instructors)
```

12/8/13
4. Complete body of method \_
\_
\_ne\_
\_ of Course. Your implementation should be a single line.

```python
def __ne__(self, ob):
    '''Return False if ob is a Course with the same name and same set of instructors as this; otherwise return True'''
    # IMPLEMENT ME IN ONE LINE
    return not self == ob  # Calls __eq__
```
Modified Question from Fall 2010

• An instance of Course always has a lecture, and it may have a set of recitation or lab sections, as does CS 1110. Students register in the lecture and in a section (if there are sections). For this we have two other classes: Lecture and Section. We show only components that are of interest for this question

• Do the following:
  ▪ Complete the constructor in class Section
  ▪ Complete the method add in Section

• Make sure invariants are enforced at all times
**Modified Question from Fall 2010**

```python
class Lecture(Course):
    """Instance is a lecture, with list of sections
    seclist: sections associated with lecture.
        [list of Section; can be empty]
    """

def __init__(self, n, ls):
    """Instance w/ name, instructors ls, no students.
    It must COPY ls. Do not assign ls to instructors.
    Pre: name is a string, ls is a nonempty list"
    Course.__init__(self, n, ls)
    self.seclist = []

class Section(Course):
    """Instance is a section associated w/ a lecture"
    mainlecture: lecture this section is associated.
        [Lecture; should not be None]"

def __init__(self, n, ls, lec):
    """Instance w/ name, instructors ls, no
    students AND primary lecture lec.
    Pre: name a string, ls list, lec a Lecture"
    # IMPLEMENT ME

def add(self,n):
    """If student with netID n is not in roster of
    section, add student to this section AND the
    main lecture. Do nothing if already there.
    Precondition: n is a valid netID.""
    # IMPLEMENT ME
```

12/8/13

Review 2
def __init__(self, n, ls, lec):
    """Instance w/ name, instructors ls no students AND main lecture lec.
    Pre: name a string, ls list, lec a Lecture"
    Course.__init__(self, n, ls)
    self.mainlecture = lec

def add(self, n):
    """If student with netID n is not in roster of section, add student to this section AND the main lecture.
    Do nothing if already there.
    Precondition: n is a valid netID."
    # Calls old version of add to
    # add to roster
    Course.add(self, n)
    # Add to lecture roster
    self.mainlecture.add(n)
Diagramming Subclasses

Important Details:

- Draw a line from subclass to the parent class
- Do not duplicate inherited methods and attributes
- Include initializer and operators with methods
- Method parameters are always optional
- Class attributes are a box with (current) value
Example: Class **Point**

**Class Folders**

- **__init__()**
- **__str__()**
- ....

**Point(object)**

- **__init__(x=0.0,y=0.0,z=0.0)**
- **__str__()**
- distanceTo()

**id1**

**Point**

- x: 0.0
- y: 0.0
- z: 0.0

**Object Folder**

- Supports the default constructor
- Default str() (and `)` behavior
- Override original methods in object
Example: Class `Point`
Two Example Classes

```python
class A(object):
    x = 3
    y = 5
    def __init__(self, y):
        self.y = y
    def f(self):
        return self.g()
    def g(self):
        return self.x + self.y

class B(A):
    y = 4
    z = 10
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def g(self):
        return self.x + self.z
    def h(self):
        return 42
```

Execute:

```python
>>> a = A(1)
>>> b = B(7,3)
```
Example from Fall 2013

Execute:

```python
>>> a = A(1)
>>> b = B(7,3)
```
What is…
(1) a.y
(2) a.z
(3) b.x
(4) B.x
What is…

(1) a.y 1  (2) a.z ERROR
(3) b.x 7  (4) B.x 3
Example from Fall 2013

What is...

(1) a.f()
(2) a.h()
(3) b.f()
(4) A.g(b)
What is…

(1) a.f()  4
(2) a.h()  ERROR
(3) b.f()  17
(4) A.g(b) 10