Accelerating Information Technology Innovation

http://aiti.mit.edu

Ghana Summer 2011
Lecture 7 – Inheritance

MISTI

Agenda

- Goal
  - Use objects to represent everyone in this course
  - Include:
    - Instructors
    - Students
    - Teaching assistants
- Learn about
  - Inheritance
  - Multiple inheritance
  - Private Variables

```
class Participant:
    def __init__(self, name, university,
                 status, year, role):
        self.name = name
        self.university = university
        self.status = status
        self.year = year
        self.role = role
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.s tatus
        print 'Year:' + self.year
        print 'Role:' + self.role
```

```
>>> Student1 = Participant("John Thomas", "Legon",
                           "Student", 300, "Student")
>>> Student1.identify()
Name:John Thomas
Uni:Legon
Status:Student
Year:300
Role:Student
```

```
>>> TechLead = Participant("Michael Yu", "MIT",
   "Instructor", 0, "Tech Lead")
>>> TechLead.identify()
Name:Michael Yu
Uni:MIT
Status:Instructor
Year:0
Role:Tech Lead
```

But we have extra fields we do not always need

```
class Student:
    def __init__(self, name, university,
                 status, year):
        self.name = name
        self.university = university
        self.status = status
        self.year = year
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.status
        print 'Year:' + self.year
```

```
class Instructor:
    def __init__(self, name, university, status,
                 role):
        self.name = name
        self.university = university
        self.status = status
        self.role = role
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.status
        print 'Role:' + self.role
```

But wait, Students and Instructors share similar attributes!

What's wrong with both approaches?

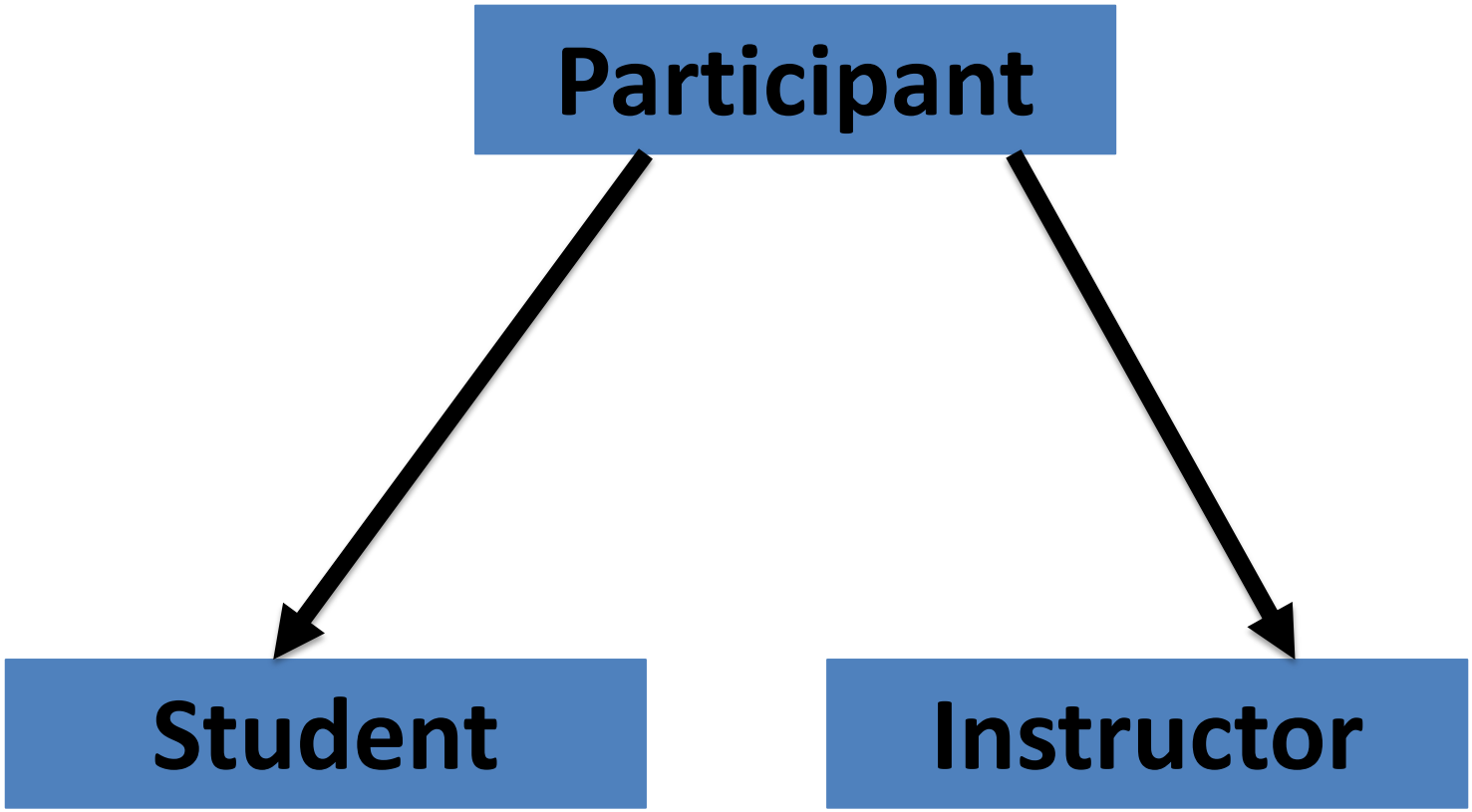
One class

- Sometimes variables are irrelevant
- A very large, complex class

Two classes

- Repeated behavior
- Increases our work
- We might not implement shared features the same way in the two classes

Inheritance



```
class Participant:
    def __init__(self, name, university,
                 status):
        self.name = name
        self.university = university
        self.status = status
    def identify(self):
        print 'Name:' + self.name
        print 'Uni:' + self.university
        print 'Status:' + self.status
```

```
class Student(Participant):
    def __init__(self, name, university, year):
        Participant.__init__(self, name, university,
                             "Student")
        self.year = year
    def identify(self):
        Participant.identify(self)
        print 'Year:' + self.year

class Instructor(Participant):
    def __init__(self, name, university, role):
        Participant.__init__(self, name, university,
                              "Instructor")
        self.role = role
    def identify(self):
        Participant.identify(self)
       print 'Role:' + self.role
```

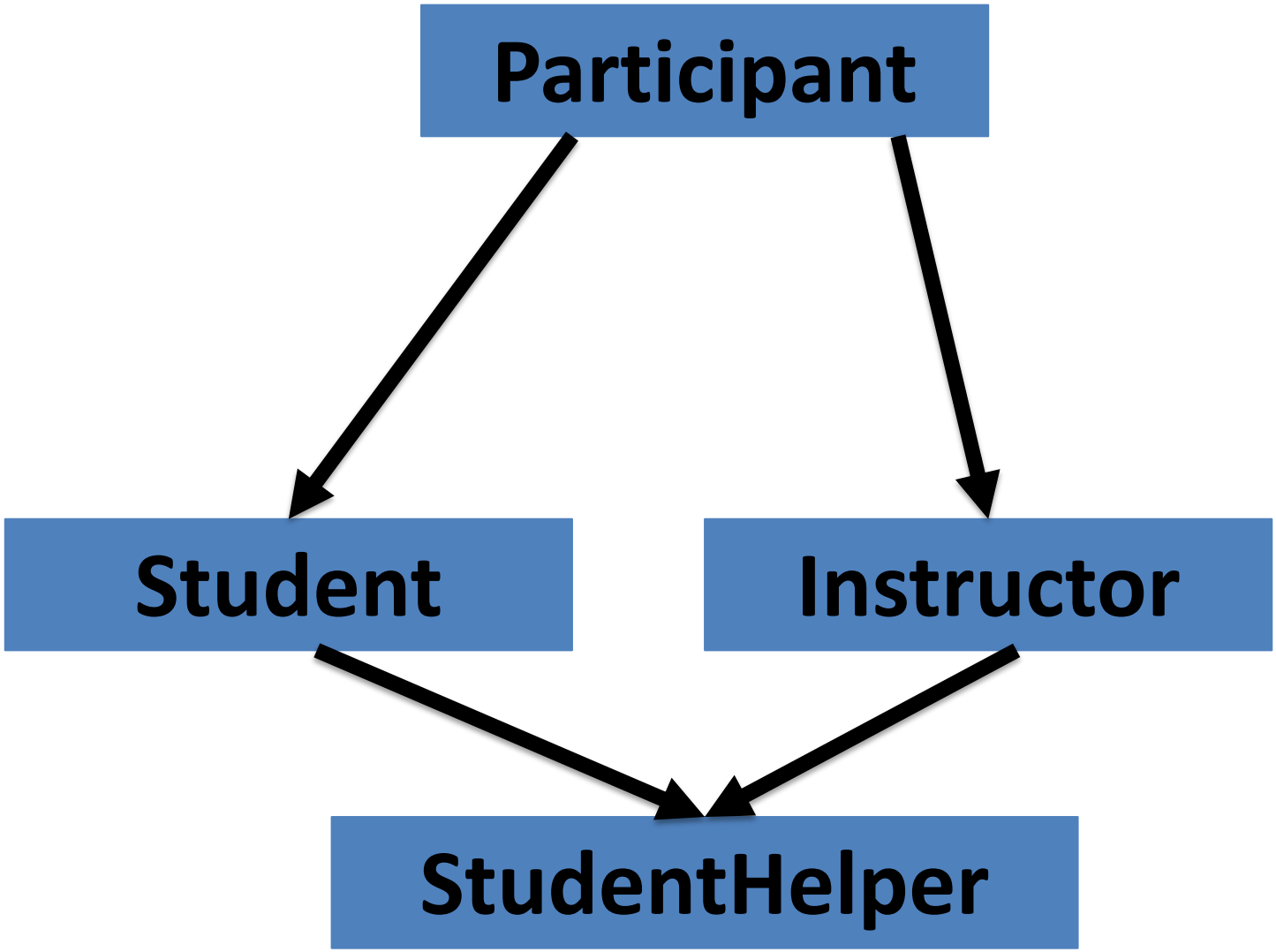
```
>>> Student1 = Student("John Thomas",“Legon“,300)
>>> Student1.identify()
Name:John Thomas
Uni:Legon
Status:Student
Year:300

>>> TechLead = Instructor(“Michael", "MIT", "Tech
  Lead")
>>> TechLead.identify()
Name:Michael
Uni:MIT
Status:Instructor
Role:Tech Lead
```

Agenda

- Goal
  - Use objects to represent everyone in this course
  - Include:
    - Instructors
    - Students
    - **Students who assist instructors**
- Learn about:
  - Inheritance
  - Multiple inheritance
  - Private Variables

Multiple Inheritance



```
class Participant:
    def __init__(self, name, university, status):
        self.name = name
        self.university = university
        self.status = status
    def identify(self):
        # ...

class Student(Participant):
    def __init__(self, name, university, year):
        Participant.__init__(self, name, university, "Student")
        self.year = year
    def identify(self):
        Participant.identify(self)
        print 'Year:' + self.year

class Instructor(Participant):
    def __init__(self, name, university, role):
        Participant.__init__(self, name, university, "Instructor")
        self.role = role
    def identify(self):
        Participant.identify(self)
        print 'Role:' + self.role
```

```
class TeachingAssistant(Student, Instructor):
    def __init__(self, name, university, year):
        Instructor.__init__(self, name, university,
                            "Teaching Assistant")
        Student.__init__(self, name, university, year)

>>> TA = TeachingAssistant("Anthony Jones",
                           "Legon", 400)
>>> TA.identify()
Name:Anthony Jones
Uni:Legon
Status:Student
>>> print TA.year
400
>>> print TA.role
Teaching Assistant
```

Private Variables

- In Python, our class variables are “exposed”
- How do we keep variables hidden from outsiders

```
Class TeachingAssistant(Student, Instructor):
    def __init__(self, name, university, group, email):
        Instructor.__init__(self, name, university, "Helper")
        Student.__init__(self, name, university, group)
        self.__email = email
    def getEmail(self):
        return self.__email

>>> TA = TeachingAssistant("Anthony Jones", "Unilag", 2,
   "anthonyj@yahoo.com")

>>> print TA.__email
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print TA.__email
AttributeError: TeachingAssistant instance has no attribute '__email'

>>> print TA.getEmail()
anthonyj@yahoo.com
```

Private Variables

- Accessible only by variables within the same class
- In Python, variables are made private by starting the name with __
  - E.g. __email
  - Different from Java, where you use special keywords: *public* and *private*
- There are only private and public variables in Python
  - No such thing as a protected variable

Agenda

- Goal
  - Use objects to represent everyone in this course
  - Include:
    - Instructors
    - Students
    - Students who assist instructors
- Learn about
  - Inheritance
  - Multiple inheritance
  - Private Variables