Example class: A Gradebook

- Create class that includes instances of other classes within it
- Concept:
  - Build a data structure that can hold grades for students
  - Gather together data and procedures for dealing with them in a single structure, so that users can manipulate without having to know internal details

Example: A Gradebook

```
class Grades(object):
    """A mapping from students to a list of grades"""
    def __init__(self):
        """Create empty grade book"""
        self.students = []  # list of Student objects
        self.grades = {}    # maps idNum -> list of grades
        self.isSorted = True # true if self.students is
sorted

    def addStudent(self, student):
        """Assumes: student is of type Student
           Add student to the grade book"""
        if student in self.students:
            raise ValueError('Duplicate student')
        self.students.append(student)
        self.grades[student.getIdNum()] = []
        self.isSorted = False
```

Example: A Gradebook

```
class Grades(object):

    def addGrade(self, student, grade):
        """Assumes: grade is a float
           Add grade to the list of grades for student"""
        try:
            self.grades[student.getIdNum()].append(grade)
        except KeyError:
            raise ValueError('Student not in grade book')

    def getGrades(self, student):
        """Return a list of grades for student"""
        try:    # return copy of student's grades
            return self.grades[student.getIdNum()][:]
        except KeyError:
            raise ValueError('Student not in grade book’)
```

Example: A Gradebook

```
class Grades(object):


    def allStudents(self):
        """Return a list of the students in the grade book"""
        if not self.isSorted:
            self.students.sort()
            self.isSorted = True
        return self.students[:]
        #return copy of list of students
```

Using a gradebook without knowing internal details

```
def gradeReport(course):
    """Assumes: course if of type grades"""
    report = []
    for s in course.allStudents():
        tot = 0.0
        numGrades = 0
        for g in course.getGrades(s):
            tot += g
            numGrades += 1
        try:
            average = tot/numGrades
            report.append(str(s) + '\'s mean grade is '
                          + str(average))
        except ZeroDivisionError:
            report.append(str(s) + ' has no grades')
    return '\n'.join(report)
```

Setting up an example

```
ug1 = UG('Jane Doe', 2014)
ug2 = UG('John Doe', 2015)
ug3 = UG('David Henry', 2003)
g1 = Grad('John Henry')
g2 = Grad('George Steinbrenner')

six00 = Grades()
six00.addStudent(g1)
six00.addStudent(ug2)
six00.addStudent(ug1)
six00.addStudent(g2)

for s in six00.allStudents():
    six00.addGrade(s, 75)
six00.addGrade(g1, 100)
six00.addGrade(g2, 25)

six00.addStudent(ug3)
```

Using this example

- I could list all students using

```
for s in six00.allStudents():
    print s
```

- This prints out the list of student names sorted by idNum
- Why not just do

```
for s in six00.students:
    print s
```

- This violates the data hiding aspect of an object, and exposes internal representation
  - If I were to change how I want to represent a grade book, I should only need to change the methods within that object, not external procedures that use it

Comments on the example

- Nicely separates collection of data from use of data
- Access is through methods associated with the gradebook object
- But current version is inefficient – to get a list of all students, I create a copy of the internal list
  - Let's me manipulate without change the internal structure
  - But expensive in a MOOC with 100,000 students