Review

```
import datetime
class Person(object):
    def __init__(self, name):
        """create a person called name"""
        self.name = name
        self.birthday = None
        self.lastName = name.split(' ')[-1]

    def getLastName(self):
        """return self's last name"""
        return self.lastName

    def setBirthday(self,month,day,year):
        """Sets self's birthday to birthDate"""
        self.birthday = datetime.date(year,month,day)

    def getAge(self):
        """returns self's current age in days"""
        if self.birthday == None:
            raise ValueError
        return (datetime.date.today() - self.birthday).days

    def __lt__(self, other):
        """return True if self's name is lexicographically
           less than other's name, and False otherwise"""
        if self.lastName == other.lastName:
            return self.name < other.name
        return self.lastName < other.lastName

    def __str__(self):
        """return self's name"""
        return self.name
```

```
class MITPerson(Person):
    nextIdNum = 0   # next ID number to assign
    def __init__(self, name):
        # initialize Person attributes
        Person.__init__(self, name)
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1
    def getIdNum(self):
        return self.idNum
    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum

class Student(MITPerson):
    pass

class UG(Student):
    def __init__(self, name, classYear):
        MITPerson.__init__(self, name)
        self.year = classYear

    def getClass(self):
        return self.year()

class Grad(Student):
    pass
```

What new types (classes) are defined?

Review

```
import datetime
class Person(object):
    def __init__(self, name):
        """create a person called name"""
        self.name = name
        self.birthday = None
        self.lastName = name.split(' ')[-1]

    def getLastName(self):
        """return self's last name"""
        return self.lastName

    def setBirthday(self,month,day,year):
        """Sets self's birthday to birthDate"""
        self.birthday = datetime.date(year,month,day)

    def getAge(self):
        """returns self's current age in days"""
        if self.birthday == None:
            raise ValueError
        return (datetime.date.today() - self.birthday).days

    def __lt__(self, other):
        """return True if self's name is lexicographically
           less than other's name, and False otherwise"""
        if self.lastName == other.lastName:
            return self.name < other.name
        return self.lastName < other.lastName

    def __str__(self):
        """return self's name"""
        return self.name
```

```
class MITPerson(Person):
    nextIdNum = 0   # next ID number to assign
    def __init__(self, name):
        # initialize Person attributes
        Person.__init__(self, name)
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1
    def getIdNum(self):
        return self.idNum
    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum

class Student(MITPerson):
    pass

class UG(Student):
    def __init__(self, name, classYear):
        MITPerson.__init__(self, name)
        self.year = classYear

    def getClass(self):
        return self.year()

class Grad(Student):
    pass
```

Describe the results of: mascot = Student("Tim Beaver")

Review

```
import datetime
class Person(object):
    def __init__(self, name):
        """create a person called name"""
        self.name = name
        self.birthday = None
        self.lastName = name.split(' ')[-1]

    def getLastName(self):
        """return self's last name"""
        return self.lastName

    def setBirthday(self,month,day,year):
        """Sets self's birthday to birthDate"""
        self.birthday = datetime.date(year,month,day)

    def getAge(self):
        """returns self's current age in days"""
        if self.birthday == None:
            raise ValueError
        return (datetime.date.today() - self.birthday).days

    def __lt__(self, other):
        """return True if self's name is lexicographically
           less than other's name, and False otherwise"""
        if self.lastName == other.lastName:
            return self.name < other.name
        return self.lastName < other.lastName

    def __str__(self):
        """return self's name"""
        return self.name
```

```
class MITPerson(Person):
    nextIdNum = 0   # next ID number to assign
    def __init__(self, name):
        # initialize Person attributes
        Person.__init__(self, name)
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1
    def getIdNum(self):
        return self.idNum
    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum

class Student(MITPerson):
    pass

class UG(Student):
    def __init__(self, name, classYear):
        MITPerson.__init__(self, name)
        self.year = classYear

    def getClass(self):
        return self.year()

class Grad(Student):
    pass
```

What does this do: mascot.getLastName()

Review

```
import datetime
class Person(object):
    def __init__(self, name):
        """create a person called name"""
        self.name = name
        self.birthday = None
        self.lastName = name.split(' ')[-1]

    def getLastName(self):
        """return self's last name"""
        return self.lastName

    def setBirthday(self,month,day,year):
        """Sets self's birthday to birthDate"""
        self.birthday = datetime.date(year,month,day)

    def getAge(self):
        """returns self's current age in days"""
        if self.birthday == None:
            raise ValueError
        return (datetime.date.today() - self.birthday).days

    def __lt__(self, other):
        """return True if self's name is lexicographically
           less than other's name, and False otherwise"""
        if self.lastName == other.lastName:
            return self.name < other.name
        return self.lastName < other.lastName

    def __str__(self):
        """return self's name"""
        return self.name
```

```
class MITPerson(Person):
    nextIdNum = 0   # next ID number to assign
    def __init__(self, name):
        # initialize Person attributes
        Person.__init__(self, name)
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1
    def getIdNum(self):
        return self.idNum
    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum

class Student(MITPerson):
    pass

class UG(Student):
    def __init__(self, name, classYear):
        MITPerson.__init__(self, name)
        self.year = classYear

    def getClass(self):
        return self.year()

class Grad(Student):
    pass
```

Sort order for Person? Student?

Review

```
import datetime
class Person(object):
    def __init__(self, name):
        """create a person called name"""
        self.name = name
        self.birthday = None
        self.lastName = name.split(' ')[-1]

    def getLastName(self):
        """return self's last name"""
        return self.lastName

    def setBirthday(self,month,day,year):
        """Sets self's birthday to birthDate"""
        self.birthday = datetime.date(year,month,day)

    def getAge(self):
        """returns self's current age in days"""
        if self.birthday == None:
            raise ValueError
        return (datetime.date.today() - self.birthday).days

    def __lt__(self, other):
        """return True if self's name is lexicographically
           less than other's name, and False otherwise"""
        if self.lastName == other.lastName:
            return self.name < other.name
        return self.lastName < other.lastName

    def __str__(self):
        """return self's name"""
        return self.name
```

```
class MITPerson(Person):
    nextIdNum = 0   # next ID number to assign
    def __init__(self, name):
        # initialize Person attributes
        Person.__init__(self, name)
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1
    def getIdNum(self):
        return self.idNum
    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum

class Student(MITPerson):
    pass

class UG(Student):
    def __init__(self, name, classYear):
        MITPerson.__init__(self, name)
        self.year = classYear

    def getClass(self):
        return self.year()

class Grad(Student):
    pass
```

Why have an “empty” class, e.g., Student?

Review

```
import datetime
class Person(object):
    def __init__(self, name):
        """create a person called name"""
        self.name = name
        self.birthday = None
        self.lastName = name.split(' ')[-1]

    def getLastName(self):
        """return self's last name"""
        return self.lastName

    def setBirthday(self,month,day,year):
        """Sets self's birthday to birthDate"""
        self.birthday = datetime.date(year,month,day)

    def getAge(self):
        """returns self's current age in days"""
        if self.birthday == None:
            raise ValueError
        return (datetime.date.today() - self.birthday).days

    def __lt__(self, other):
        """return True if self's name is lexicographically
           less than other's name, and False otherwise"""
        if self.lastName == other.lastName:
            return self.name < other.name
        return self.lastName < other.lastName

    def __str__(self):
        """return self's name"""
        return self.name
```

```
class MITPerson(Person):
    nextIdNum = 0   # next ID number to assign
    def __init__(self, name):
        # initialize Person attributes
        Person.__init__(self, name)
        # new MITPerson attribute: a unique ID number
        self.idNum = MITPerson.nextIdNum
        MITPerson.nextIdNum += 1
    def getIdNum(self):
        return self.idNum
    # sorting MIT people uses their ID number, not name!
    def __lt__(self, other):
        return self.idNum < other.idNum

class Student(MITPerson):
    pass

class UG(Student):
    def __init__(self, name, classYear):
        MITPerson.__init__(self, name)
        self.year = classYear

    def getClass(self):
        return self.year()

class Grad(Student):
    pass
```

What is the substitution principle?