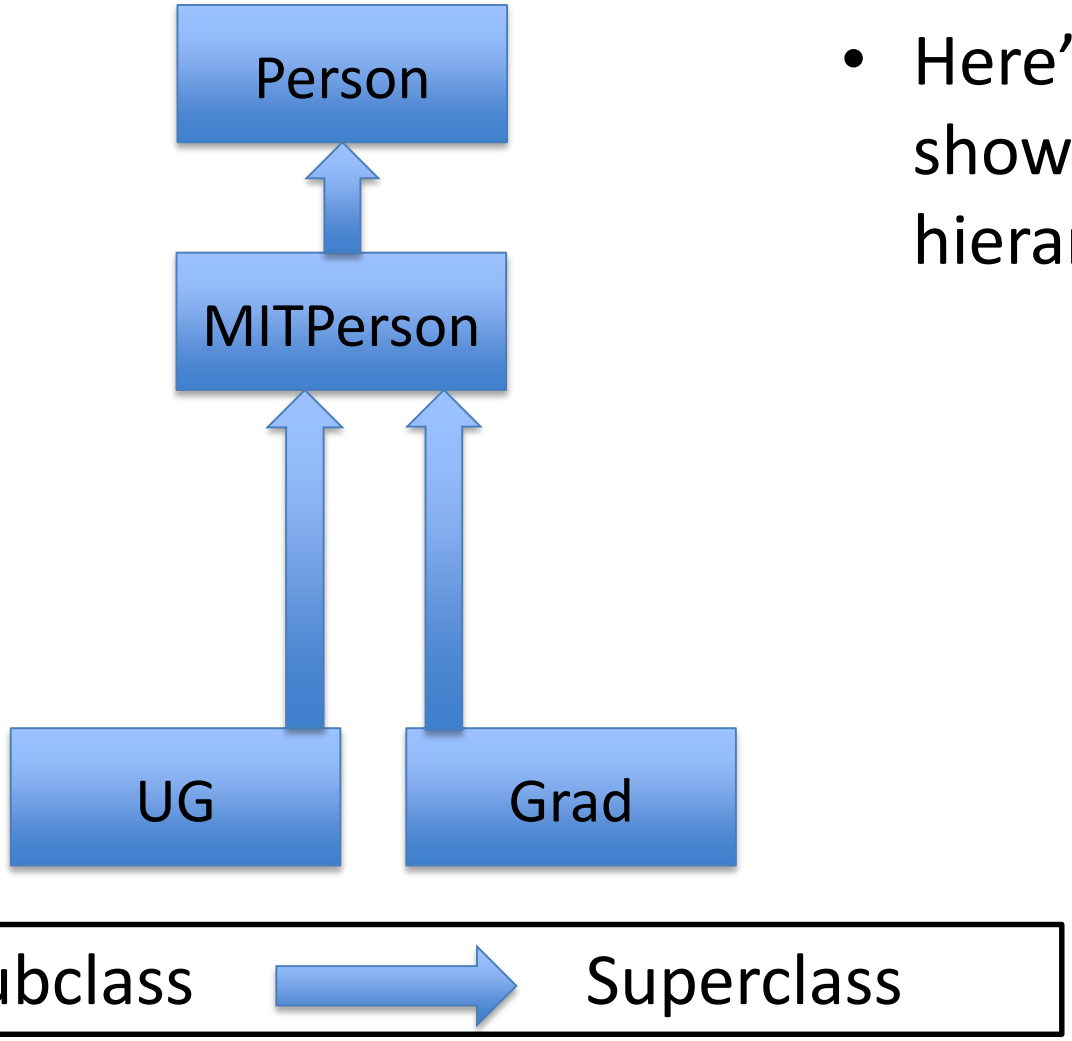
Using Inheritance

- Let's build an application that organizes info about people!
  - Person: name, birthday
    - Get last name
    - Sort by last name
    - Get age
  - MITPerson: Person + ID Number
    - Assign ID numbers in sequence
    - Get ID number
    - Sort by ID number
  - Students: several types, all MITPerson
    - Undergraduate student: has class year
    - Graduate student

More classes for the hierarchy

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

    def getClass(self):
        return self.year

class Grad(MITPerson):
    pass

def isStudent(obj):
    return isinstance(obj,UG) or
isinstance(obj,Grad)
```

Class Hierarchy & Substitution Principle



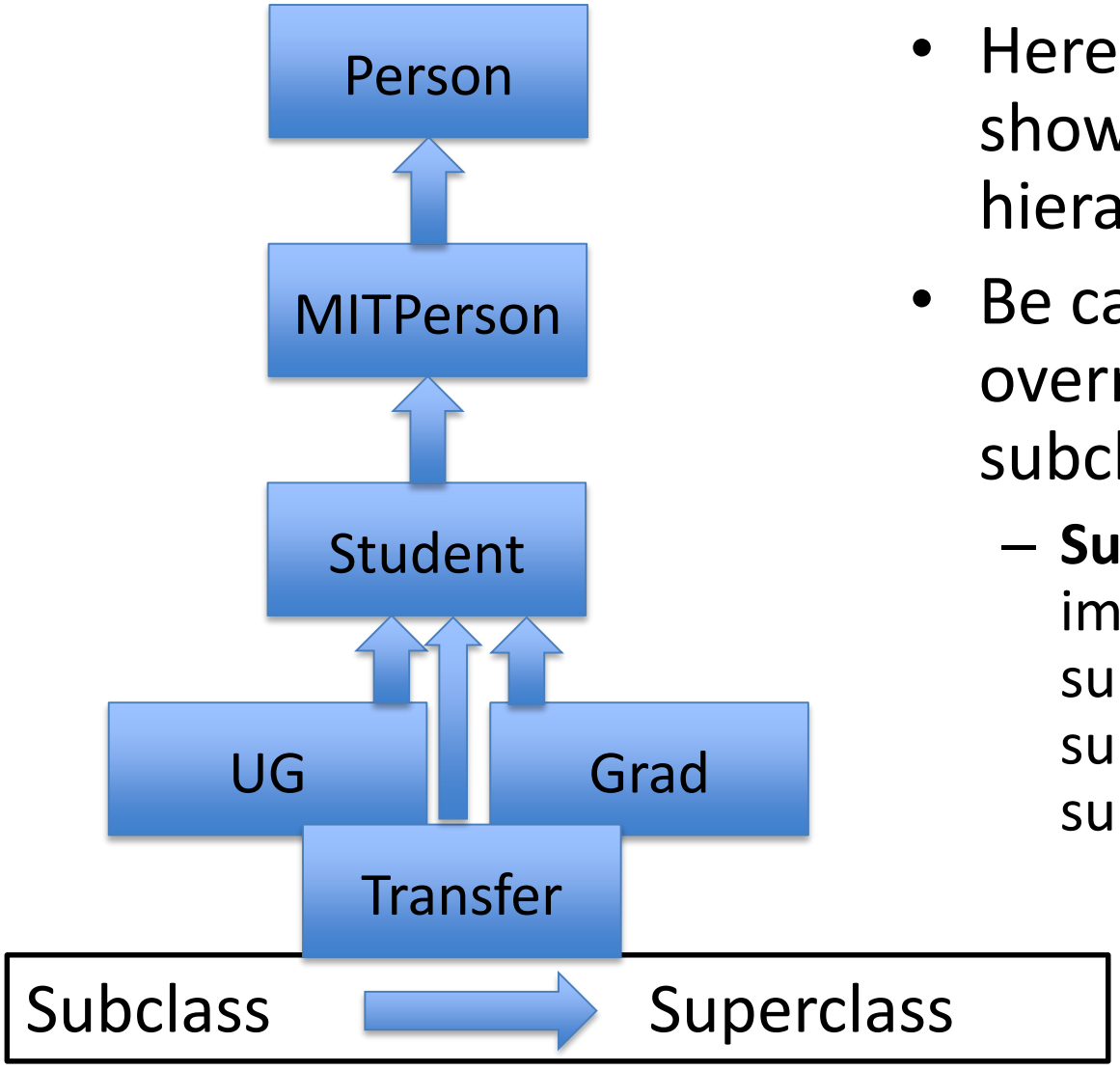
- Here's a diagram showing our class hierarchy

Cleaning up the hierarchy

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

    def getClass(self):
        return self.year

class Grad(MITPerson):
    pass

class TransferStudent(MITPerson):
    pass

def isStudent(obj):
    return isinstance(obj,UG) or isinstance(obj,Grad)
```

Now I have to rethink isStudent

Class Hierarchy & Substitution Principle



- Here's a diagram showing our class hierarchy
- Be careful when overriding methods in a subclass!
  - **Substitution principle:** important behaviors of superclass should be supported by all subclasses

Cleaning up the hierarchy

```
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

class TransferStudent(Student):
    pass

def isStudent(obj):
    return isinstance(obj,Student)
```

Better is to create a superclass that covers all students

In general, creating a class in the hierarchy that captures common behaviors of subclasses allows us to concentrate methods in a single place, and lets us think about subclasses as a coherent whole