Python Tutorial II – Objects and Classes

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 201, Spring Semester 2025

August 29, 2025

Overview

Relationships Among Classes

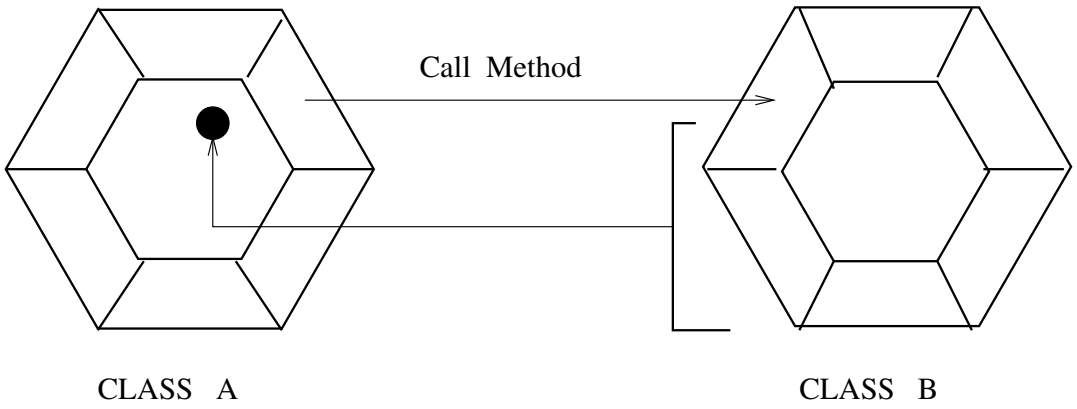
Relationships Among Classes

Motivation

- Classes and objects by themselves are not enough to describe the structure of a system.
- We also need to express relationships among classes.
- Object-oriented software packages are assembled from collections of classes and class-hierarchies that are related in three fundamental ways.

Relationships Among Classes

1. **Use:** Class A uses Class B (method call).



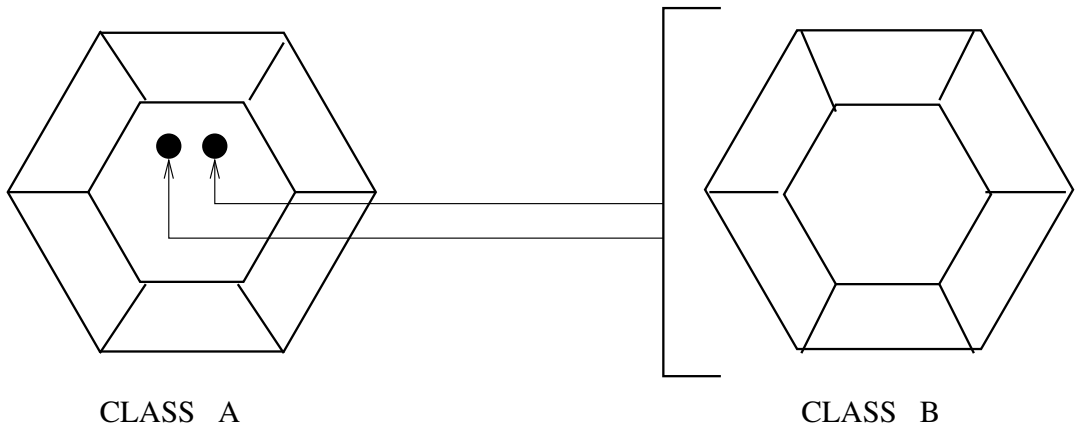
Class `A` uses Class `B` if a method in `A` calls a method in an object of type B.

Example

```
import math

dAngle = math.sin ( math.PI / 3.0 );
```

Relationships Among Classes

2. Containment (Has a): Class A contains a reference to Class B.



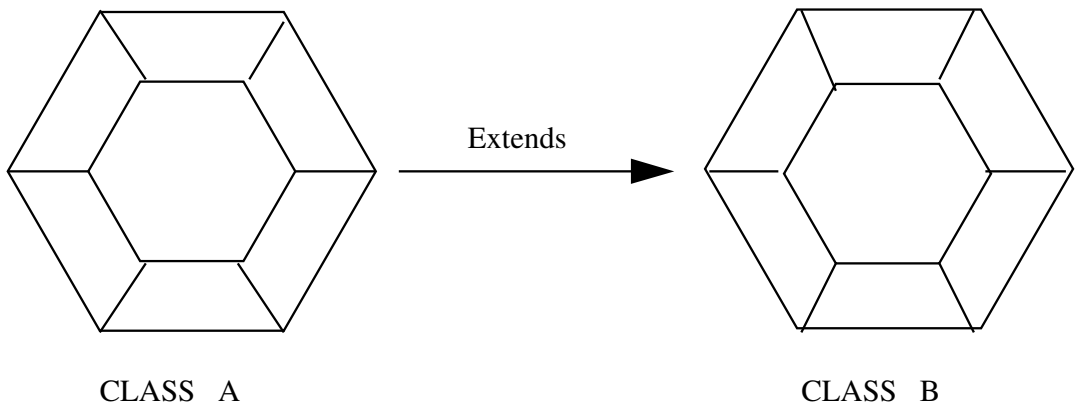
Clearly, containment is a special case of use (i.e., see Item 1.).

Example

```
class LineSegment
   self.start = Point() ...
   self.end   = Point() ...
```

Relationships Among Classes

3. Inheritance (Is a): In everyday life, we think of inheritance as something that is received from a predecessor or past generation. Here, Class B inherits the data and methods (extends) from Class A.



Two Examples from Python

```
class ColoredCircle (Circle) ....
class Student (Person) ....
```

Inheritance Mechanisms

Inheritance Mechanisms

Inheritance Structures

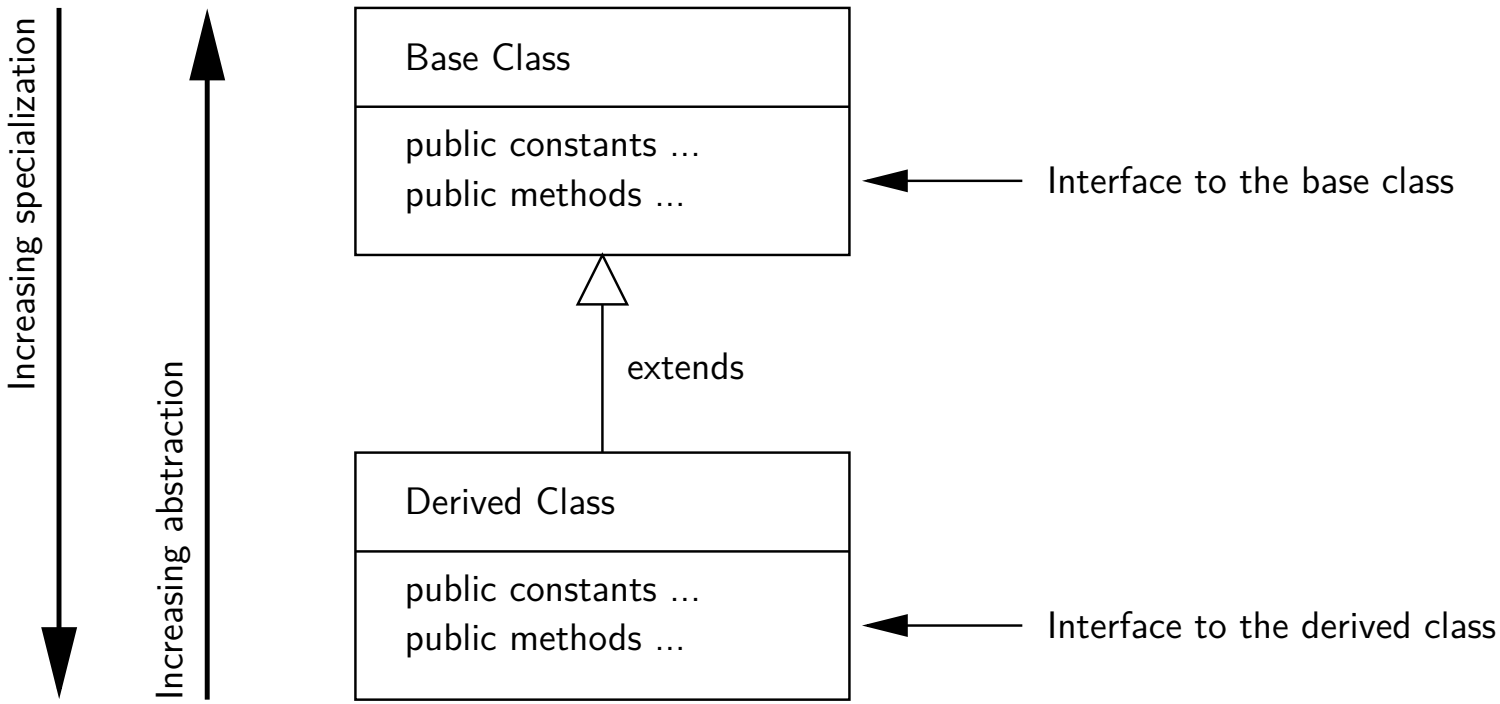
Inheritance structures allow you to capture common characteristics in one model artifact and permit other artifacts to inherit and possibly specialize them. Class hierarchies are explicitly designed for customization through extension.

In this approach to development:

- Forces us to identify and separate the common elements of a system from those aspects that are different/distinct.
- Commonalities are captured in a super-class and inherited and specialized by the sub-classes.
- Inherited features may be overridden with extra features designed to deal with exceptions.

Base and Derived Classes

Goal: Avoid duplication and redundancy of data in a problem specification.



Base and Derived Classes

Points to note:

- A class in the upper hierarchy is called a superclass (or base, parent class).
- A class in the lower hierarchy is called a subclass (or derived, child, extended class).
- The classes in the lower hierarchy inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher-level classes.

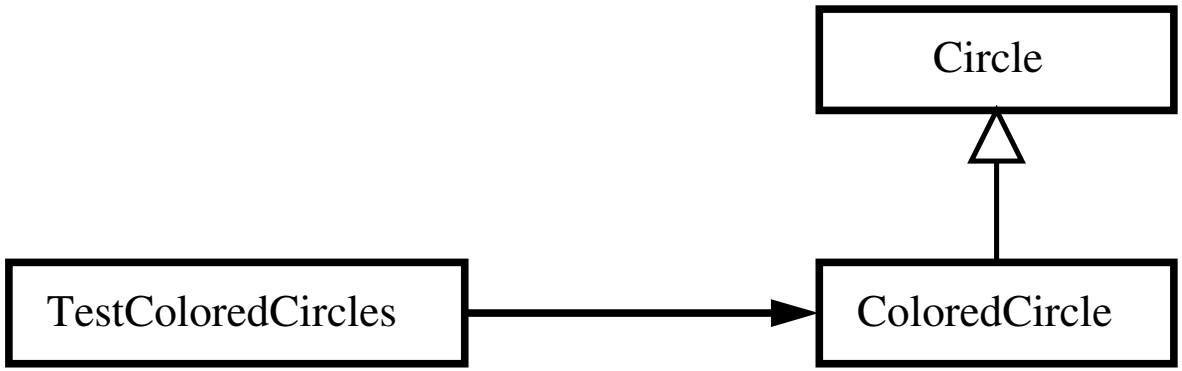
Base and Derived Classes

Python Syntax:

```
# ------------------------------------
# Base Class ...
# ------------------------------------

class BaseClass:

      # Constructor of Base Class

      # Base class variables and methods ...

# ------------------------------------
# Derived class extends Base Class ...
# ------------------------------------

class DerivedClass( BaseClass ):

      # Constructor of Derived Class

      # Derived class variables and methods ...
```

Example 5. Model Colored Circles by Extending Circle

Part I: Program Architecture. The TestCircle program will create objects of type ColoredCircle.



Circle Attributes:

- `_x`, `_y`, `_radius`, `_area`, `_perimeter`.

ColoredCircle Attributes:

- `_color`.

Example 5. Model Colored Circles by Extending Circle

Part IIa: Circle Object Model (with Protected Variables)

```
1   # ============================================================
2   # Circle.py: Implementation of circle model with protection of
3   # circle parameters and methods.
4   #
5   # Written by: Mark Austin                                October, 2020
6   # ============================================================
7
8   import math
9
10  class Circle:
11    _radius = 0
12    _area   = 0
13    _perimeter = 0
14
15    def __init__(self, x, y, radius):
16      self._radius    = radius
17      self._area      = math.pi*radius*radius
18      self._perimeter = 2.0*math.pi*radius
19      self._x = x
20      self._y = y
21
22    # Set circle coordinates ...
23
24    def setX(self, x):
25      self._x = x
26
27    def setY(self, y):
```

Example 5. Model Colored Circles by Extending Circle

Part IIa: Circle Object Model (Continued) ...

```
28        self._y = y
29
30      # Set circle radius, recompute area and perimeter ...
31
32      def setRadius(self, radius):
33        self._radius = radius
34        self._area   = math.pi*radius*radius
35        self._perimeter = 2.0*math.pi*radius
36
37      # Get circle parameters ...
38
39      def getX(self):
40        return self._x
41
42      def getY(self):
43        return self._y
44
45      def getRadius(self):
46        return self._radius
47
48      def getArea(self):
49        return self._area
50
51      def getPerimeter(self):
52        return self._perimeter
```

Example 5. Model Colored Circles by Extending Circle

Part IIa: Circle Object Model (Continued) ...

```
54      # String represention of circle ...
55
56      def __str__(self):
57          return "--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f: perimeter = %
58                  self._x, self._y, self._radius, self._area, self._perimeter )
```

Example 5. Model Colored Circles by Extending Circle

Part IIb: Colored Circle Object Model

```
1   # ============================================================
2   # ColoredCircle.py: Extend circle to create coloredcircles.
3   #
4   # Written by: Mark Austin                          October, 2022
5   # ============================================================
6
7   from Circle import Circle
8
9   class ColoredCircle(Circle):
10    def __init__(self, x, y, radius, color):
11      Circle.__init__(self, x, y, radius)
12      self._color = color
13
14    # Set/get color ...
15
16    def setColor(self, color):
17      self._color = color
18
19    def getColor(self):
20      return self._color
21
22    # String representation of colored circle ...
23
24    def __str__(self):
25      return "--- ColoredCircle: (x,y) = (%4.1f, %4.1f): radius = %5.2f: area = %6.2f: col
26              self._x, self._y, self._radius, self._area, self._color )
```

Example 5. Model Colored Circles by Extending Circle

Part II: Colored Circle Test Program

```
1   # ========================================================
2   # TestColoredCircles.py: Exercise colored circle objects.
3   #
4   # Written by: Mark Austin                        December 2022
5   # ========================================================
6
7   from Circle import Circle
8   from ColoredCircle import ColoredCircle
9
10  # main method ...
11
12  def main():
13      print("--- Enter TestCircles.main()         ... ");
14      print("--- ================================ ... ");
15
16      print("--- Part 1: Create and print circle ... ");
17
18      x = Circle( 0.0, 0.0, 3.0 )
19      print(x)
20
21      print("--- Part 2: Create and print colored circle ... ");
22
23      y = ColoredCircle( 0.0, 0.0, 0.0, "blue" )
24      print(y)
25      y.setRadius(1.0)
26      print(y)
27      y.setRadius(2.0)
```

Example 5. Model Colored Circles by Extending Circle

Part II: Colored Circle Test Program (Continued) ...

```
28      print(y)
29
30      print("--- Part 3: Change coordinates and color ... ");
31
32      y.setX( 1.0 )
33      y.setY( 1.0 )
34      y.setColor("red" )
35      y.setRadius(3.0)
36
37      print(y)
38
39      print("--- ============================== ... ");
40      print("--- Finished TestCircles.main()    ... ");
41
42  # call the main method ...
43
44  main()
```

Example 5. Model Colored Circles by Extending Circle

Part III: Abbreviated Output:

```
--- Enter TestCircles.main()          ...
--- ============================== ...
--- Part 1: Create and print circle ...
--- Circle: (x,y) = (0.00, 0.00): radius = 3.00: area = 28.27: perimeter = 18.85
--- Part 2: Create and print colored circle ...
--- ColoredCircle: (x,y) = ( 0.0,  0.0): radius =  0.00: area =  0.00: color = blue
--- ColoredCircle: (x,y) = ( 0.0,  0.0): radius =  1.00: area =  3.14: color = blue
--- ColoredCircle: (x,y) = ( 0.0,  0.0): radius =  2.00: area = 12.57: color = blue
--- Part 3: Change coordinates and color ...
--- ColoredCircle: (x,y) = ( 1.0,  1.0): radius =  3.00: area = 28.27: color = red
--- ============================== ...
--- Finished TestCircles.main()       ...
```

Source Code: See: python-code.d/inheritance/

Example 5. Model Colored Circles by Extending Circle

Part IV: Files before Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Circle.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 ColoredCircle.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestColoredCircles.py
```

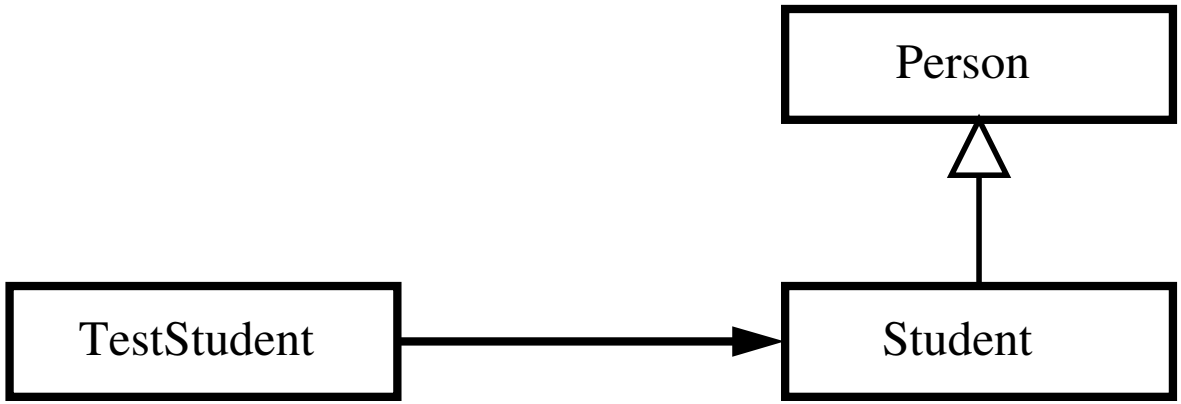
Part IV: Files after Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Circle.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 ColoredCircle.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestColoredCircles.py
drwxr-xr-x  4 austin  staff   128 Feb 18 13:27 __pycache__

./__pycache__:
total 16
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Circle.cpython-37.pyc
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 ColoredCircle.cpython-37.pyc
```

Note: Python builds compiled bytecodes for Circle and ColoredCircle (with .pyc extension).

Example 6. Student is an Extension of Person

Part I: Program Architecture. The TestStudent program will create objects of type Student.



Person Attributes:

- `_firstname`, `_lastname`, `_age` (age), `_ssn` (social security), `_dob` (date of birth).

Student Attributes:

- `_gpa` (grade point average).

Example 6. Student is an Extension of Person

Part IIa: Person Object Model (with Protected Variables)

```
1   # ================================================================
2   # Person.py: Simple model of a Person. The scope of variables
3   # _age, _ssn, and _dob are protected to Person and all subclasses.
4   #
5   # Written by: Mark Austin                                November 2022
6   # ================================================================
7
8   from datetime import date
9
10  class Person:
11      _age = 0    # <-- age ...
12      _ssn = 0    # <-- social security number ...
13      _dob = 0    # <-- date of birth ...
14
15      # Constructor method ...
16
17      def __init__(self, fname, lname, dob ):
18        self._firstname = fname
19        self._lastname  = lname
20        self._dob       = dob
21        self._age       = self.calculateAge()
22
23      # Get first and last names ...
24
25      def getFirstName(self):
26        return self._firstname
```

Example 6. Student is an Extension of Person

Part IIa: Person Object Model (Continued) ...

```
27
28      def getLastName(self):
29        return self._lastname
30
31      # Set/get date of birth ...
32
33      def setDob(self, dob):
34        self._dob = dob
35
36      def getDob(self, dob):
37        return self._dob
38
39      # Calculate age ...
40
41      def calculateAge(self):
42         today = date.today()
43         age   = today.year - self._dob.year - ((today.month, today.day) < (self._dob.month
44         return age
45
46      # Set/get/print age ...
47
48      def setAge(self, age):
49        self._age = age
50
51      def getAge(self):
52        return self._age
```

Example 6. Student is an Extension of Person

Part IIa: Person Object Model (Continued) ...

```
53
54      # Set/get/print social security number ...
55
56      def setSSN(self, ssn ):
57        self._ssn = ssn
58
59      def getSSN(self):
60        return self._ssn
61
62      # return string represention of object ...
63
64      def __str__(self):
65          return "Person: {:6.2f} {:6.2f}: age = {:f} ".format( self._firstname,
66                                                               self._lastname,
67                                                               self._age )
```

Example 6. Student is an Extension of Person

Part Ib: Student Object Model

```
1   # ============================================================
2   # Student.py: A Student is a specialization of Person ...
3   # ============================================================
4
5   from Person import Person
6
7   class Student(Person):
8       _gpa = 0
9
10      # Parameterized constructor ...
11
12      def __init__(self, fname, lname, dob, year):
13        Person.__init__(self, fname, lname, dob)
14        self._graduationyear = year
15
16      # Set/get gpa ...
17
18      def setGpa(self, gpa):
19        self._gpa = gpa
20
21      def getGpa(self):
22        return self._gpa
```

Example 6. Student is an Extension of Person

Part Ib: Student Object Model

```
24      # Boolean to confirm person is a student ...
25
26      def isStudent(self):
27          return True
28
29      # Assemble string represention of student ...
30
31      def __str__(self):
32          studentinfo = [];
33          studentinfo.append("\n");
34          studentinfo.append("--- Student: {:s} {:s} ...  \n".format( self._firstname,
35                                                                      self._lastname));
36          studentinfo.append("--- ------------------------------------------------ \n");
37          studentinfo.append("---   Gpa = {:6.2f} ... \n".format( self._gpa));
38          studentinfo.append("---   Age = {:6d} ... \n".format( self._age));
39          studentinfo.append("---   Graduation year = {:d} ... \n".format(
40                                                                  self._graduationyear ));
41          studentinfo.append("--- ------------------------------------------------ ");
42          return "".join(studentinfo);
```

Example 6. Student is an Extension of Person

Part II: Student Test Program

```
1   # ======================================================
2   # TestStudent.py: Exercise methods in Student class ...
3   #
4   # Written by: Mark Austin                    November 2022
5   # ======================================================
6
7   from Student import Student
8   from datetime import date
9
10  # main method ...
11
12  def main():
13      print("--- Enter TestStudents.main()              ... ");
14      print("--- ====================================== ... ");
15
16      print("--- Part 1: Create student Angela Austin ...")
17
18      y = Student( "Angela", "Austin", date(2002,3,2) ,2023)
19      y.setGpa(3.5)
20      y.setSSN(1234)
21
22      print("--- Part 2: Retrieve student parameters ...")
23
24      print("---  First Name: {:s}".format( y.getFirstName() ) )
25      print("---  Last Name: {:s}".format( y.getLastName() ) )
26      print("---  Age = {:d}".format( y.getAge() ) )
27      print("---  Social Security Number = {:d}".format( y.getSSN() ) )
```

Example 6. Student is an Extension of Person

Part II: Student Test Program (Continued) ...

```
28      print("---  Is student: {:s}".format( str( y.isStudent()) ) )
29
30      print("--- Part 3: Assemble string representation of student ...")
31
32      print( y.__str__() )
33
34      print("--- ==================================== ... ");
35      print("--- Finished TestStudents.main()          ... ");
36
37  # call the main method ...
38
39  main()
```

Example 6. Student is an Extension of Person

Part III: Abbreviated Output:

```
--- Part 1: Create student Angela Austin ...
--- Part 2: Retrieve student parameters ...
---
---  First Name: Angela
---  Last Name: Austin
---  Age = 20
---  Social Security Number = 1234
---  Is student: True
---
--- Part 3: Assemble string representation of student ...
---
--- Student: Angela Austin ...
--- -----------------------------------------------
---   Gpa =   3.50 ...
---   Age =     20 ...
---   Graduation year = 2023 ...
--- -----------------------------------------------
```

Source Code: See: python-code.d/inheritance/

Example 6. Student is an Extension of Person

Part IV: Files before Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Person.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Student.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestStudents.py
```

Part IV: Files after Program Execution:

```
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Person.py
-rw-r--r--  1 austin  staff   903 Feb 18 13:21 Student.py
-rw-r--r--  1 austin  staff   847 Feb 18 13:26 TestStudents.py
drwxr-xr-x  4 austin  staff   128 Feb 18 13:27 __pycache__

./__pycache__:
total 16
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Person.cpython-37.pyc
-rw-r--r--  1 austin  staff  1476 Feb 18 13:27 Student.cpython-37.pyc
```

Note: Python builds compiled bytecodes for Student and Person (with .pyc extension).

Mutiple Inheritance Mechanisms

Multiple Inheritance Structures

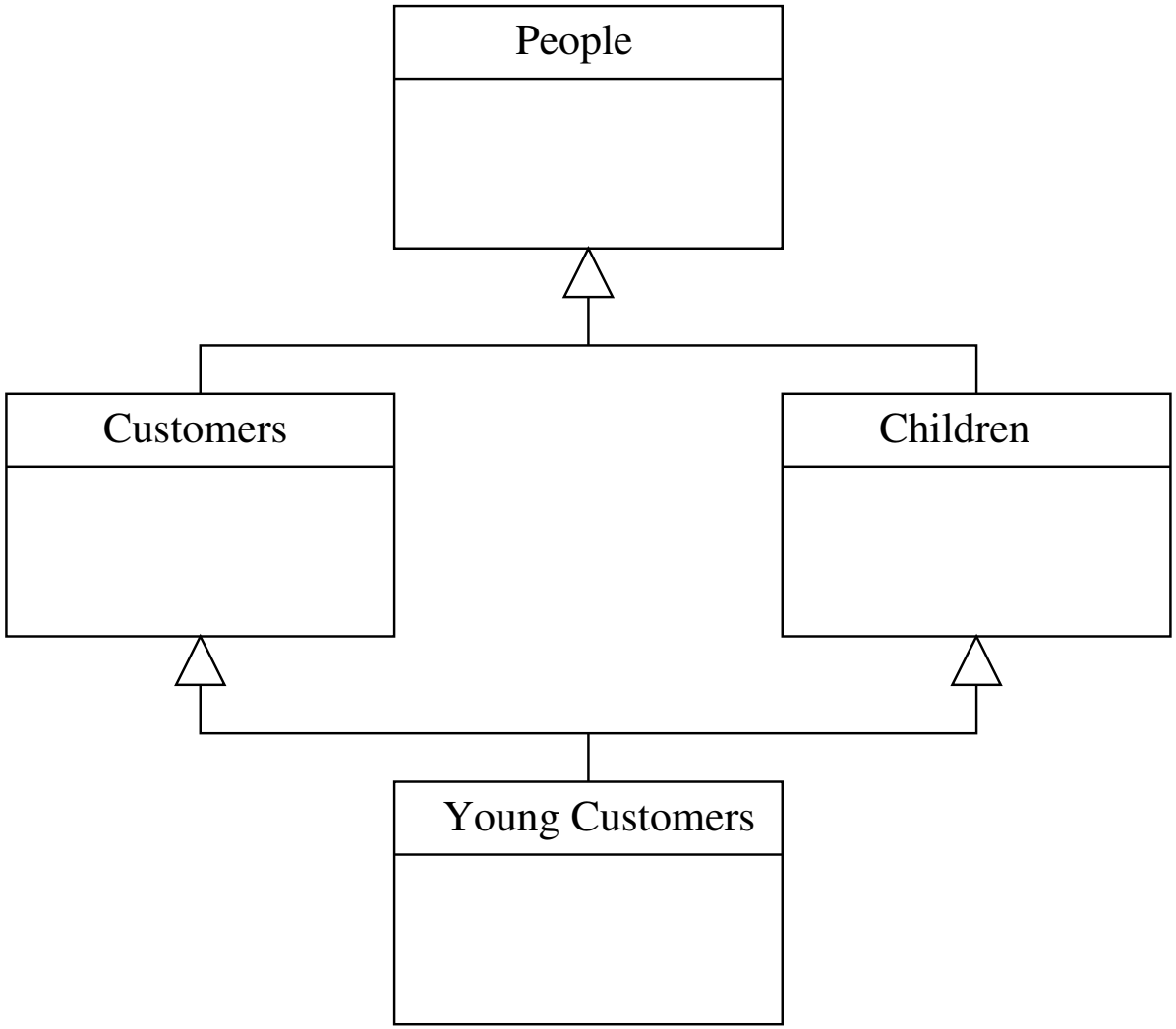
- In a multiple inheritance structure, a class can inherit properties from multiple parents.
- The downside is that properties and/or operations may be partially or fully contradictory.

Example

- People is a generalization of Children and Customers.
- Young customers inherits properties from Customers and Children.

Note. Python supports use of multiple inheritance. Java explicitly prevents multiple inheritance – instead, it allows classes to have multiple interfaces.

Mutiple Inheritance Mechanisms



Mutiple Inheritance Mechanisms

Python Syntax:

```
class People:

      # People constructor ...
      # People variables, and methods ...

class Customers (People):

      # Customers constructor ...
      # Customers variables, and methods ...

class Children (People):

      # Children constructor ...
      # Children variables, and methods ...

class YoungCustomers( Customers, Children ):

      # YoungCustomer constructor ...
      # YoungCustomer variables, and methods ...
```