Python Tutorial – Part 2: Objects and Classes

Mark A. Austin

University of Maryland

austin@umd.edu
ENCE 688P, Spring Semester 2022

January 13, 2023

Overview

Working with Objects and Classes

Working with Objects and Classes

Working with Objects and Classes:

- Collections of objects share similar traits (e.g., data, structure, behavior).
- Collections of objects will form relationships with other collections of objects.

Definition of a Class

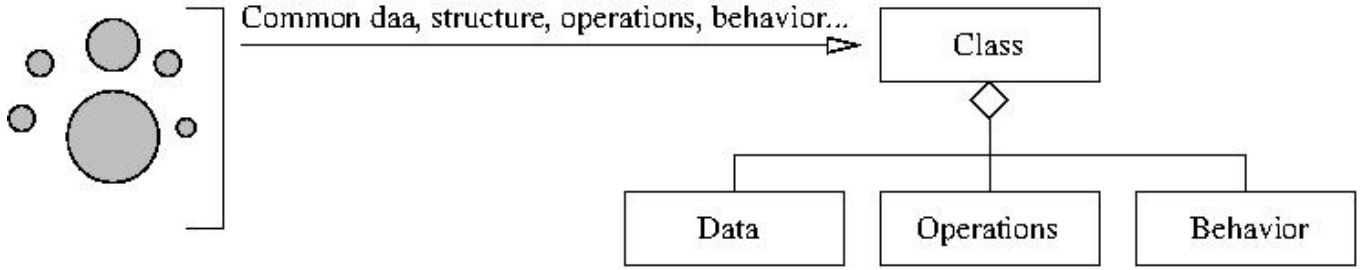
A class is a specification (or blueprint) of an object's structure and behavior.

Definition of an Object

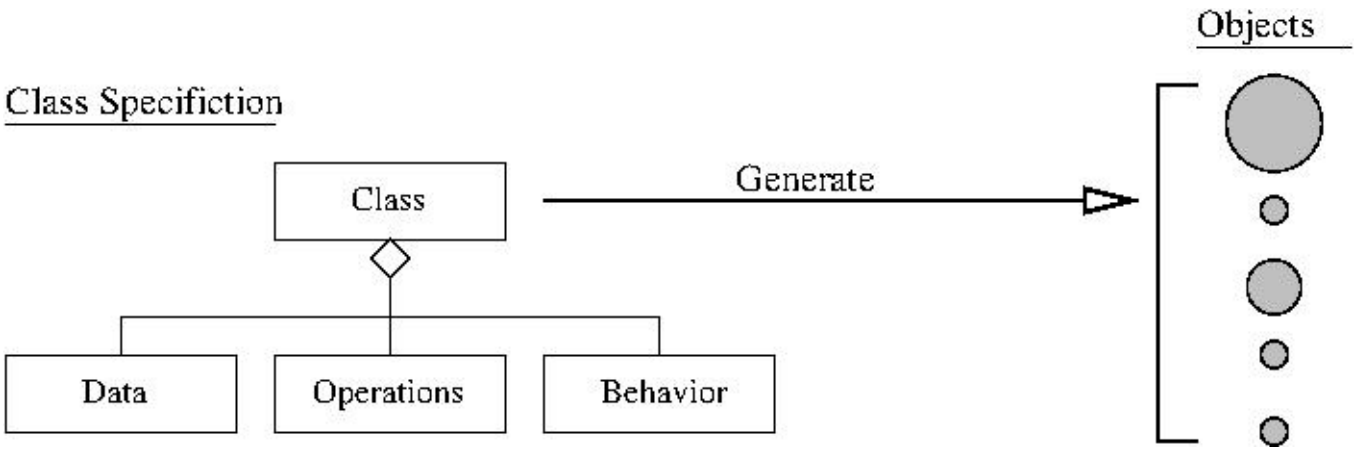
An object is an instance of a class.

Working with Objects and Classes

From Collections of Objects to Classes:



Generation of Objects from Class Specifications:



Working with Objects and Classes

Principles for Development of Reusable Code:

- **Inheritance:** Create new (specialized) classes from existing classes through mechanism of concept extension.
- **Encapsulation:** Hide some details of a class from other (external) classes.
- **Polymorphism:** Use common operation in different ways depending on details of data input.

Key Design Tasks

- Identify objects and their attributes and functions,
- Establish relationships among the objects,
- Implement and test the individual objects,
- Assemble and test the system.

Example 1. Working with Points

A Very Simple Class in Python

```
1   # ========================================================
2   # Point.py: Create point objects ...
3   #
4   # Modified by: Mark Austin                        October, 2020
5   # ========================================================
6
7   import math
8
9   class Point:
10
11      def __init__(self, xCoord=0, yCoord=0):
12          self.__xCoord = xCoord
13          self.__yCoord = yCoord
14
15      # compute distance between two points ...
16
17      def distance(self, second):
18          x_d = self.__xCoord - second.__xCoord
19          y_d = self.__yCoord - second.__yCoord
20          return (x_d**2 + y_d**2)**0.5
21
22      # return string represention of object ...
23
24      def __str__(self):
25          return "( %6.2f, %6.2f ) " % ( self.__xCoord, self.__yCoord )
```

Example 1. Working with Points

Create and Print two Point Objects

```
 8       pt1 = Point( 0.0, 0.0 )
 9       pt2 = Point( 3.0, 4.0 )
10
11       print("--- pt1 = %s ..." % (pt1) )
12       print("--- pt2 = %s ..." % (pt2) )
```

Output:

```
--- pt1 = (   0.00,   0.00 )  ...
--- pt2 = (   3.00,   4.00 )  ...
```

Compute Distance between Two Points

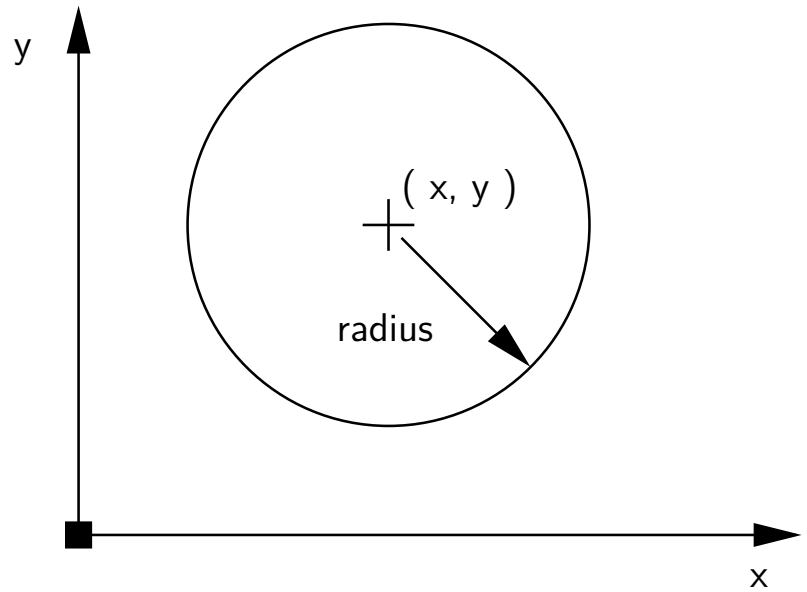
```
10       distance = pt1.distance(pt2)
11       print("--- Distance between pt1 and pt2 --> %.2f ..." % (distance) )
```

Output:

```
--- Distance between pt1 and pt2 --> 5.00 ...
```

Example 2. Working with Circles

A circle can be described by the `(x,y)` position of its center and by its `radius`.



There are numerous things we can do with circles:

- Compute their circumference, perimeter or area,
- Check if a point is inside a circle.

Example 2. Working with Circles

```
1   # ========================================================
2   # Circle.py: Simplified modeling of a circle ...
3   #
4   # Written by: Mark Austin                        October, 2020
5   # ========================================================
6
7   import math
8
9   class Circle:
10    radius = 0
11    area   = 0
12    perimeter = 0
13
14    def __init__(self, x, y, radius):
15      self.radius    = radius
16      self.area      = math.pi*radius*radius
17      self.perimeter = 2.0*math.pi*radius
18      self.x = x
19      self.y = y
20
21    # Set circle radius, recompute area and perimeter ...
22
23    def setRadius(self, radius):
24      self.radius = radius
25      self.area   = math.pi*radius*radius
26      self.perimeter = 2.0*math.pi*radius
```

Example 2. Working with Circles

```
27
28      # Print details of circle ...
29
30      def printCircle(self):
31        print("--- Circle: (x,y) = (%.2f, %.2f): radius = %.2f: area = %.2f: perimeter = %.2
32              % ( self.x, self.y, self.radius, self.area, self.perimeter ) )
```

Create and Print two Circle Objects

```
1       x = Circle( 0.0, 0.0, 3.0 )
2       y = Circle( 1.0, 2.0, 4.0 )
3       x.printCircle()
4       y.printCircle()
```

Output:

```
--- Circle: (x,y) = (0.00, 0.00): radius = 3.00: area = 28.27
--- Circle: (x,y) = (1.00, 2.00): radius = 4.00: area = 50.27
```

Example 3. Object Model of a Person

Part I: Person Object Model:

```
1   # ========================================================
2   # Person.py: Simplified model of a person ...
3   #
4   # Written by: Mark Austin                        October, 2022
5   # ========================================================
6
7   class Person:
8     age = 0
9     ssn = 0
10
11    def __init__(self, fname, lname):
12      self.firstname = fname
13      self.lastname  = lname
14
15    def printname(self):
16      print("--- Name: %s, %s" % ( self.firstname, self.lastname) )
17
18    # Get first and last names ...
19
20    def getFirstName(self):
21      return self.firstname
22
23    def getLastName(self):
24      return self.lastname
25
26    # Set/print age ...
```

Example 3. Object Model of a Person

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

```
27
28      def setAge(self, age):
29        self.age = age
30
31      def printAge(self):
32        print("--- Age = %d " % (self.age) )
33
34      # Set/print social security number ...
35
36      def setSSN(self, ssn ):
37        self.ssn = ssn
```

Part II: Person Test Program:

```
1   # ====================================================
2   # TestPerson.py: Test program for person objects ...
3   # ====================================================
4
5   from Person import Person
6
7   # main method ...
8
9   def main():
10      print("--- Enter TestPerson.main()          ... ");
11      print("--- ============================== ... ");
```

Example 3. Test Program for Person Object Model

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

```
13      # Exercise methods in class Person ...
14
15      x = Person( "Angela", "Austin" )
16      x.printname()
17
18      print("--- First name: %s" % ( x.getFirstName() ) )
19      print("--- Family name: %s" % ( x.getLastName() ) )
20
21      # Initialize attribute values ..
22
23      x.setAge(29)
24      x.setSSN(123456789)
25
26      # Print attribute values ..
27
28      x.printAge()
29      x.printSSN()
30
31      print("--- ============================== ... ");
32      print("--- Finished TestPerson.main()     ... ");
33
34  # call the main method ...
35
36  main()
```

Example 3. Object Model of a Person

Output:

```
--- Enter TestPerson.main()          ...
--- ============================== ...
--- Name: Angela, Austin
--- First name: Angela
--- Family name: Austin
--- Age = 29
--- Social Security No: 123456789
--- ============================== ...
--- Finished TestPerson.main()       ...
```

Data Hiding and Encapsulation

Hiding Information

Data Hiding

Data Hiding is isolation of the client from a part of program implementation. Some objects in the module are kept internal, invisible, and inaccessible to the user.

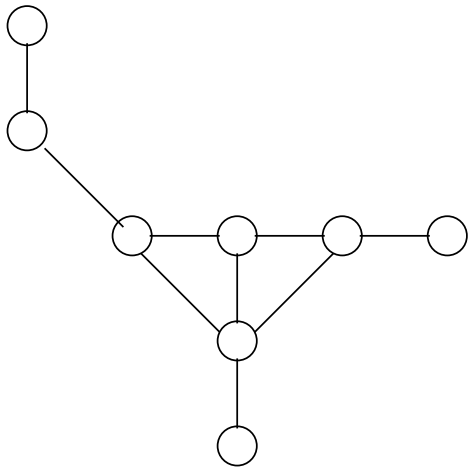
Principle of Information Hiding

The principle of information hiding states that information which is likely to change (e.g., over the lifetime of a software/systems package) should be hidden inside a module.

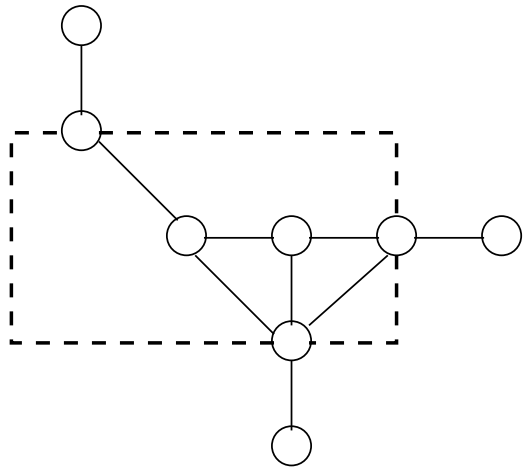
Key Advantages

- Prevents accidental linkage to incorrect data.
- It heightens the security against hackers that are unable to access confidential data.

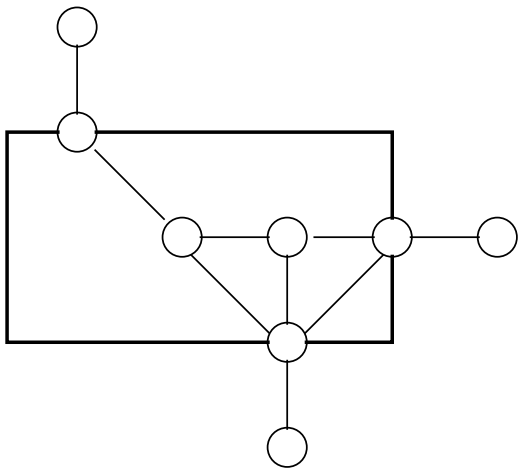
Data Hiding and Encapsulation



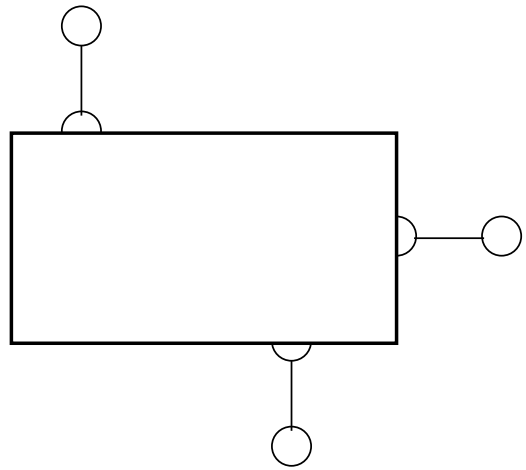
Unstructured Components



Aggregation



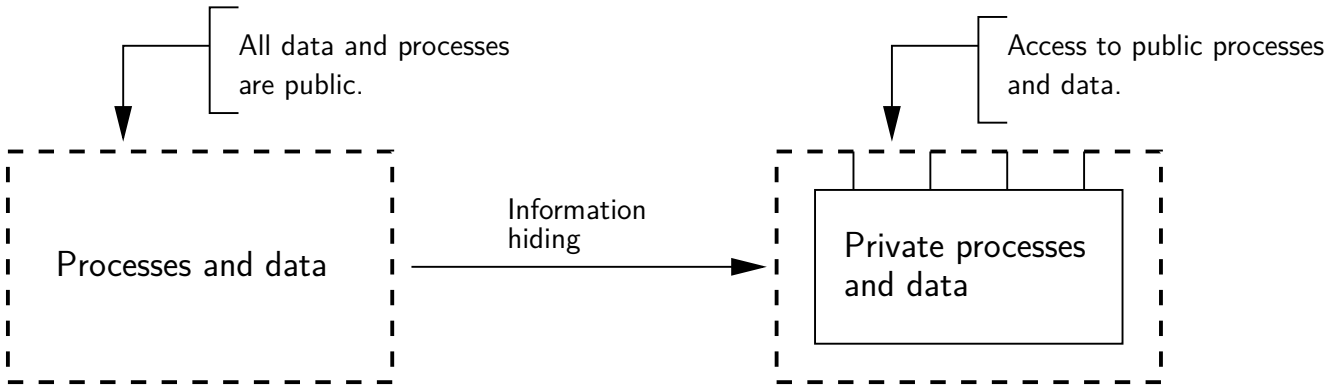
Designer's view of Aggregation



Encapsulation – User's view of Abstraction

Data Hiding and Encapsulation

Application. Process for Implementation of Information Hiding.



Data Hiding in Python (Private and Protected) ...

- Data hiding is implemented by using a double underscore before (prefix) the attribute name. Making an attribute private hides it from users.
- Use of a single underscore makes the variable/method protected. The variables/methods will be available to the class, and all of its subclasses.

Example 4. Revised Circle Object Model

Part I: Revised Circle Object Model

```
1   # ============================================================
2   # Circle.py: Implementation of circle model with encapsulation
3   # (hiding) of circle parameters and properties.
4   #
5   # Written by: Mark Austin                              October, 2020
6   # ============================================================
7
8   import math
9
10  class Circle:
11    __radius = 0            # <-- private parameters ....
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
```

Example 4. Revised Circle Object Model

Part I: Revised Circle Object Model (Continued) ...

```
27      def setY(self, y):
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 4. Revised Circle Object Model

Part I: Revised 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:
58                  perimeter = %.2f" % ( self.__x, self.__y, self.__radius,
59                  self.__area, self.__perimeter )
```

Part II: Test Program for Circle Object Model

```
1   # ========================================================
2   # TestCircles.py: Exercise circle objects.
3   #
4   # Written by: Mark Austin                          December 2022
5   # ========================================================
6
7   from Circle import Circle
8
9   # main method ...
10
11  def main():
12      print("--- Enter TestCircles.main()          ... ");
13      print("--- ================================ ... ");
14
15      print("--- Part 1: Create and print circle ... ");
16
17      x = Circle( 0.0, 0.0, 3.0 )
18      print(x)
```

Example 4. Revised Circle Object Model

Part II: Test Program for Circle Object Model (Continued) ...

```
20      print("--- ============================== ... ");
21      print("--- Finished TestCircles.main()     ... ");
22
23  # call the main method ...
24
25  main()
```

Part III: Program Output

```
--- Enter TestCircles.main()         ...
--- ============================== ...
--- Circle: (x,y) = (0.00, 0.00): radius = 3.00: area = 28.27
--- ============================== ...
--- Finished TestCircles.main()      ...
```

Relationships Among Classes

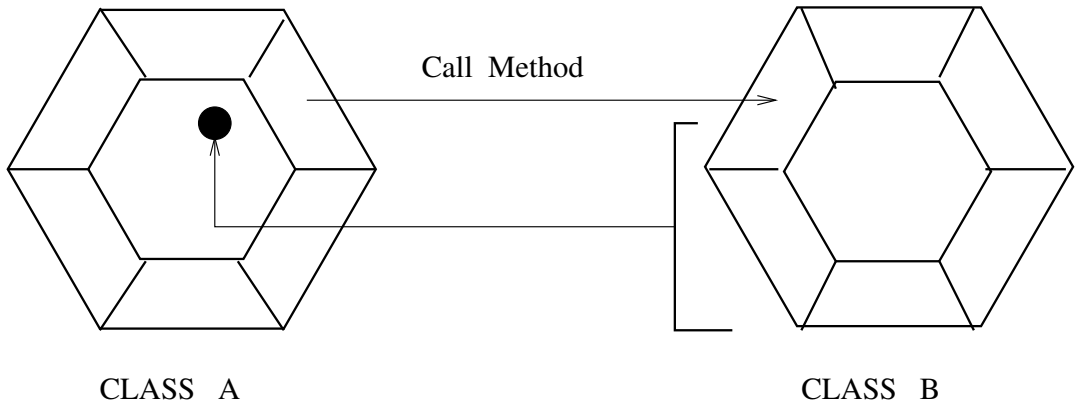
Relationships Among Classes

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.

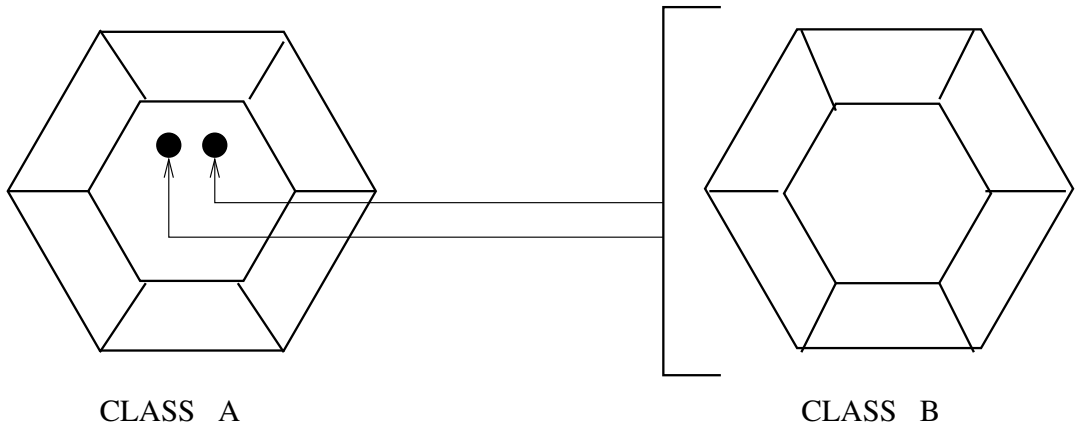
Example

```
import math

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

Relationships Among Classes

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



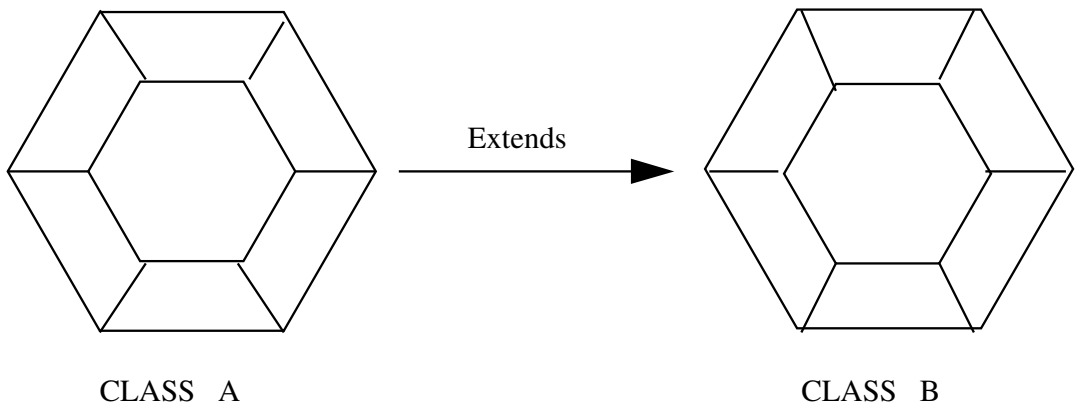
Clearly, containment is a special case of use (i.e., see Item 1.).

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

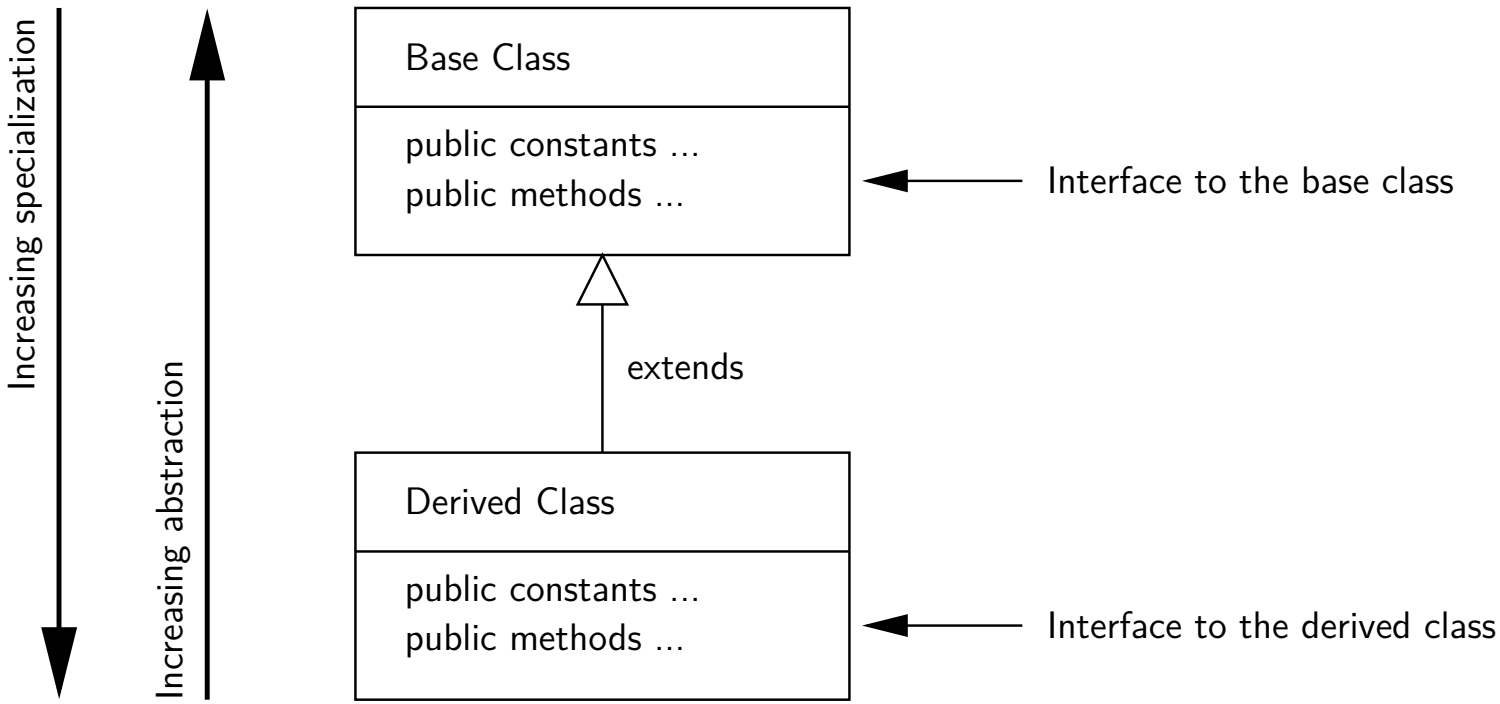
Inheritance structures allow you to capture common characteristics in one model artifact and permit other artifacts to inherit and possibly specialize them. Class hierarchies are explicitly designed for customization through extension.

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 Ia: 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 Ia: 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 Ia: 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 )
```

Part Ib: 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    def printColoredCircle(self):
15      print("--- ColoredCircle:", 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()
```

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

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()       ...
```

Example 6. Student is an Extension of Person

Part Ia: Person Object Model (with Protected Variables)

```
1   # ============================================================
2   # Person.py: Simple model of a Person. The scope of variables
3   # _age and _ssn are protected to Person and all subclasses ..
4   #
5   # Written by: Mark Austin                                November 2022
6   # ============================================================
7
8   class Person:
9       _age = 0      # <-- protected variable !! ....
10      _ssn = 0
11
12      # Constructor method ...
13
14      def __init__(self, fname, lname):
15        self._firstname = fname
16        self._lastname  = lname
17
18      def printname(self):
19        print("--- Name: %s, %s" % ( self._firstname, self._lastname) )
20
21      # Get first and last names ...
22
23      def getFirstName(self):
24        return self._firstname
25
26      def getLastName(self):
27        return self._lastname
```

Example 6. Student is an Extension of Person

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

```
28
29      # Set/get/print age ...
30
31      def setAge(self, age):
32        self._age = age
33
34      def getAge(self):
35        return self._age
36
37      def printAge(self):
38        print("--- Age = %d " % (self._age) )
39
40      # Set/get/print social security number ...
41
42      def setSSN(self, ssn ):
43        self._ssn = ssn
44
45      def getSSN(self):
46        return self._ssn
47
48      def printSSN(self):
49        print("--- Social Security No: %d " % (self._ssn) )
50
51      # return string represention of object ...
52
53      def __str__(self):
54          return "Person: %6.2f %6.2f: age = %f " % ( self._firstname, self._lastname, self
```

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
9       # Example of a parameterized constructor ...
10
11      def __init__(self, fname, lname, year):
12        Person.__init__(self, fname, lname)
13        self._graduationyear = year
14
15      # Boolean to confirm person is a student ...
16
17      def isStudent(self):
18         return True
19
20      # String represention of student ...
21
22      def __str__(self):
23          return "--- Student: %s %s, age = %d, graduation year = %d " % (
```

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
9   # main method ...
10
11  def main():
12      print("--- Enter TestStudents.main()              ... ");
13      print("--- ===================================== ... ");
14
15      print("--- Part 1: Create student Angela Austin ...")
16
17      y = Student( "Angela", "Austin", 2023)
18      y.setAge(20)
19      y.setSSN(1234)
20
21      print("--- Part 2: Retrieve student parameters ...")
22
23      print("---  First Name: %s" % ( y.getFirstName() ) )
24      print("---  Last Name: %s" % ( y.getLastName() ) )
25      print("---  Age = %d" % ( y.getAge() ) )
26      print("---  Social Security Number = %d" % ( y.getSSN() ) )
27      print("---  Is student: %s" % ( y.isStudent() ) )
```

Example 6. Student is an Extension of Person

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

```
28
29      print("--- Part 3: String representation of student ...")
30
31      print( y.__str__() )
32
33      print("--- ===================================== ... ");
34      print("--- Finished TestStudents.main()          ... ");
35
36  # call the main method ...
37
38  main()
```

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: String representation of student ...
--- Student: Angela Austin, age = 20, graduation year = 2023
```

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

Mutiple Inheritance Mechanisms

Multiple Inheritance Structures

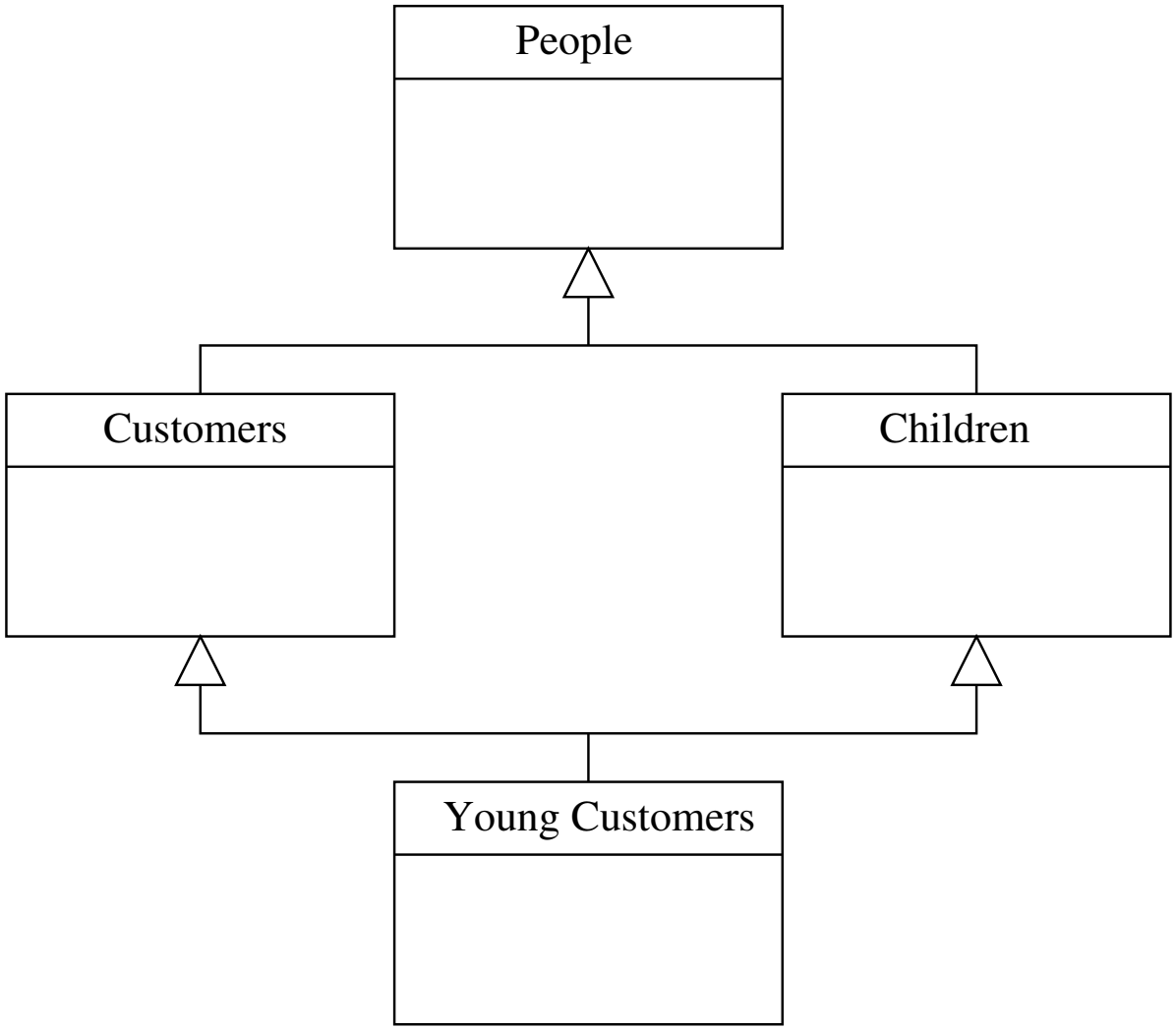
- In a multiple inheritance structure, a class can inherit properties from multiple parents.
- The downside is that properties and/or operations may be partially or fully contradictory.

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 ...
```

Composition of Object Models

Composition of Object Models

Definition

Composition is known as is a part of or is a relationship.

The member object is a part of the containing class and the member object cannot survive or exist outside the enclosing or containing class or doesn't have a meaning after the lifetime of the enclosing object.

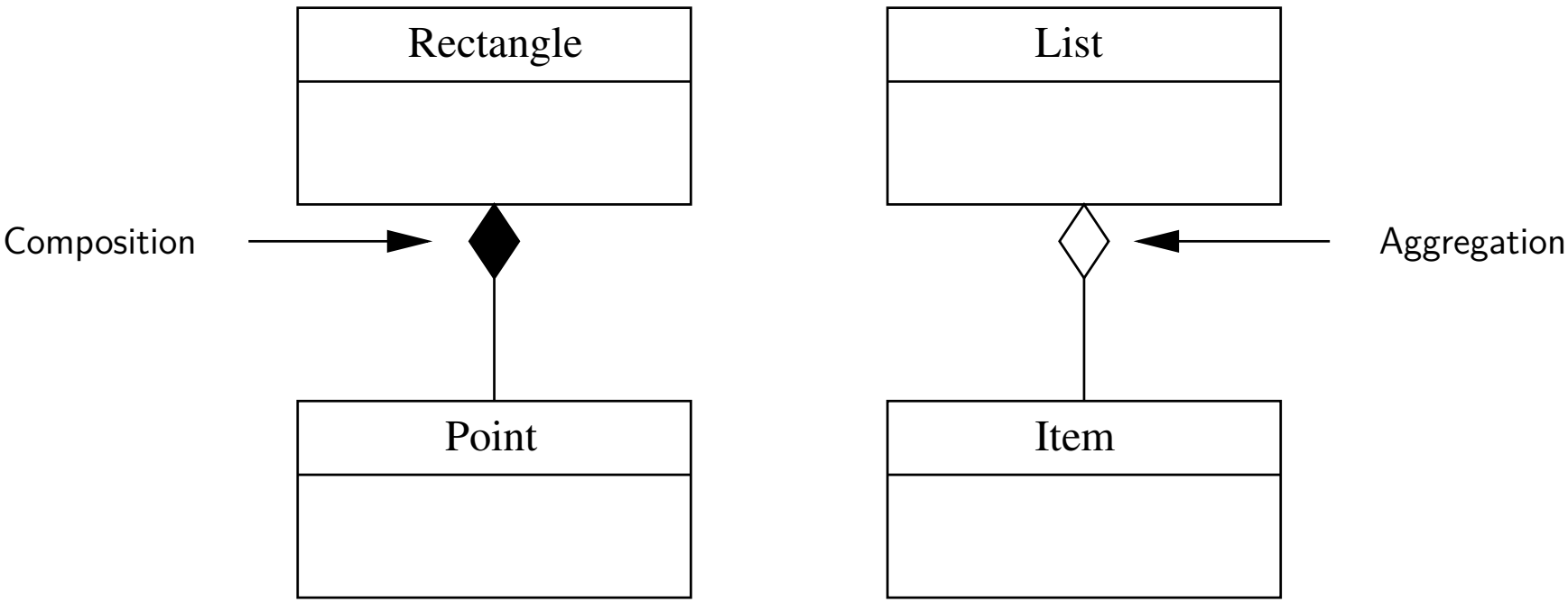
Is it Aggregation or Composition?

- Ask the question: if the part moves, can one deduce that the whole moves with it in normal circumstances?

Example: A car is composition of wheels and an engine. If you drive the car to work, hopefully the wheels go too!

Composition of Object Models

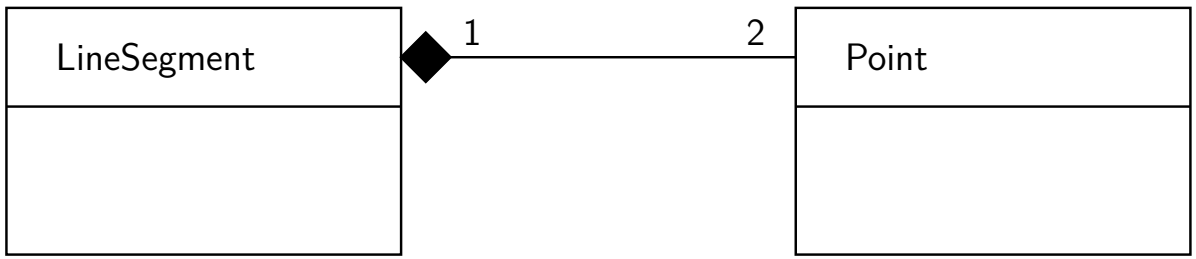
Notation for Aggregation and Composition



Recall: Aggregation is all about grouping of things ...

Example 7. Modeling Line Segments

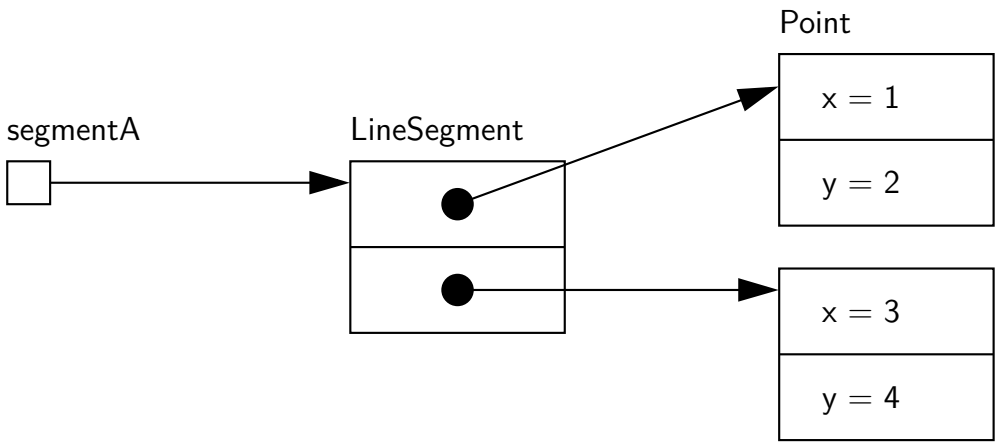
Model Composition



Creating a line segment object with:

```
segmentA = LineSegment( 1, 2, 3, 4 );
```

should give a layout of memory:



Example 7. Modeling Line Segments

Part I: Line Segment Object Model

```
1   # ================================================================
2   # LineSegment.py: Line segments are defined by end points (x1, y1) and
3   # (x2, y2). Compute length and angle of the line segment in radians.
4   #
5   # Written by: Mark Austin                                October, 2022
6   # ================================================================
7
8   import math
9
10  from Point import Point
11
12  class LineSegment:
13    __length = 0
14    __angle  = 0
15
16    def __init__(self, x1, y1, x2, y2 ):
17      self.__pt1 = Point(x1,y1)                  # <-- Object composition ...
18      self.__pt2 = Point(x2,y2)                  # <-- Object composition ...
19      self.__length = self.__pt1.distance(self.__pt2)
20      self.__angle  = self.getAngle()
21
22    # Compute angle (radians) for coordinates in four quadrants ....
23
24    def getAngle(self):
25       dX = self.__pt2.get_xCoord() - self.__pt1.get_xCoord();
26       dY = self.__pt2.get_yCoord() - self.__pt1.get_yCoord();
```

Example 7. Modeling Line Segments

Part I: Line Segment Object Model (Continued) ...

```
27
28          if dY >  0.0 and dX == 0.0:
29              angle = math.pi/2.0
30          if dY >= 0.0 and dX > 0.0:
31              angle = math.atan( dY/dX )
32          if dY >= 0.0 and dX < 0.0:
33              angle = math.pi + math.atan( dY/dX )
34          if dY < 0.0 and dX < 0.0:
35              angle = math.pi + math.atan( dY/dX )
36          if dY < 0.0 and dX >= 0.0:
37              angle = 2*math.pi + math.atan( dY/dX )
38
39          return angle
40
41      # String represention of line segment ...
42
43      def __str__(self):
44          x1 = self.__pt1.get_xCoord();
45          y1 = self.__pt1.get_yCoord();
46          x2 = self.__pt2.get_xCoord();
47          y2 = self.__pt2.get_yCoord();
48          return "--- LineSegment: (x1,y1) = (%5.2f, %5.2f), (x2,y2) = (%5.2f, %5.2f),
49                      angle = %.2f, length = %.2f" % ( x1, y1, x2, y2, self.__angle, self.__l
```

Example 7. Modeling Line Segments

Part II: Line Segment Test Program

```
1   # ====================================================
2   # TestLineSegment.py: Exercise line segment class ...
3   # ====================================================
4
5   from LineSegment import LineSegment
6
7   # main method ...
8
9   def main():
10      print("--- Enter TestLineSegment.main()    ... ");
11      print("--- ============================== ... ");
12
13      print("--- Part 1: Create test line segment ... ");
14
15      segmentA = LineSegment( 1.0, 2.0,  3.0,  4.0 )
16      print(segmentA)
17
18      print("--- Part 2: Sequence of line segments ... ");
19
20      a = LineSegment( 0.0, 0.0,  3.0,  0.0 )
21      print(a)
22      b = LineSegment( 0.0, 0.0,  3.0,  3.0 )
23      print(b)
24      c = LineSegment( 0.0, 0.0,  0.0,  3.0 )
25      print(c)
26      d = LineSegment( 0.0, 0.0, -3.0,  3.0 )
27      print(d)
```

Example 7. Modeling Line Segments

Part II: Line Segment Test Program (Continued) ...

```
28      e = LineSegment( 0.0, 0.0, -3.0,  0.0 )
29      print(e)
30
31      print("--- ============================== ... ");
32      print("--- Finished TestLineSegment.main() ... ");
33
34  # call the main method ...
35
36  main()
```

Part III: Abbreviated Program Output:

```
--- Part 1: Create test line segment ...
--- LineSegment: (x1,y1) = ( 1.00,  2.00), (x2,y2) = ( 3.00,  4.00), angle = 0.79, length = 2.83
--- Part 2: Sequence of line segments ...
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = ( 3.00,  0.00), angle = 0.00, length = 3.00
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = ( 3.00,  3.00), angle = 0.79, length = 4.24
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = ( 0.00,  3.00), angle = 1.57, length = 3.00
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = (-3.00,  3.00), angle = 2.36, length = 4.24
--- LineSegment: (x1,y1) = ( 0.00,  0.00), (x2,y2) = (-3.00,  0.00), angle = 3.14, length = 3.00
```

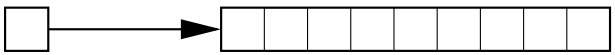
Source Code: See: python-code.d/classes/

Working with Groups of Objects

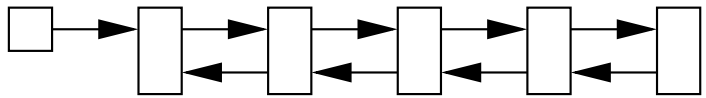
Pathway From Objects to Groups of Objects

Memory Layout: Arrays, Lists, Queues, Trees, and Graphs

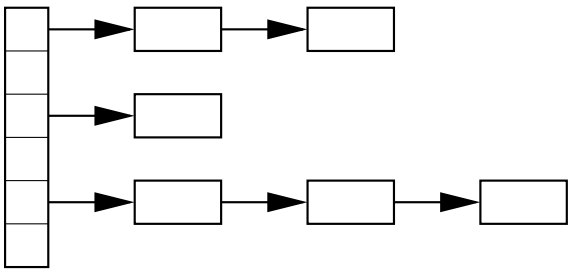
Arrays



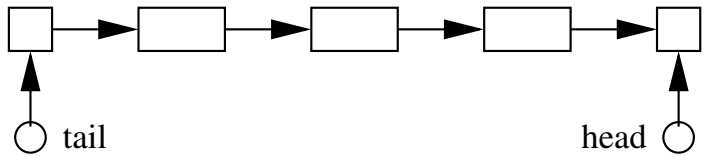
Linked List



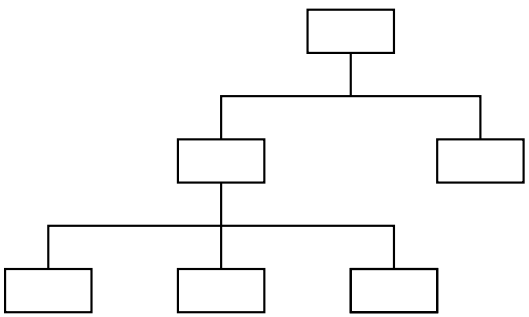
Hash Map



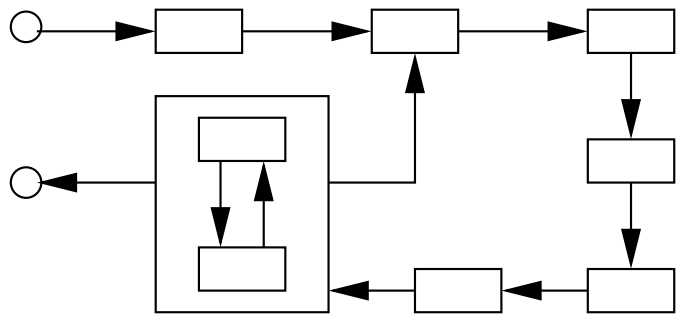
Queues



Trees



Graphs



Example 8: Create List of Objects

List of Student Objects ...

....

Output:

....

Example 9: Create Dictionary of Objects

Dictionary of Student Objects:

....

Output:

....

Case Study

(GeoModeling the World's Cities)

Case Study: GeoModeling the World's Cities

Parameters of City Data Model

- ...
- ...
- ...
- ...

Case Study: GeoModeling the World's Cities

Abbreviated Header for City Data File

Loading CSV Data into Pandas

Case Study: GeoModeling the World's Cities

City Object Model

Case Study: GeoModeling the World's Cities

Collection of City Object Models

Case Study: Visualize Cities in GeoPandas

Case Study: Visualize Cities in GeoPandas

Filter Collection of City Objects

Case Study: Visualize Filtered Collection of Cities

Case Study: GeoModeling the World's Cities

Haversine Formula

Case Study: Modeling the World's Cities

Compute Distance between Baltimore and NYC

References

- ....
- ....