Object-Oriented Programming

Section 1

Chapter 7

Quiz 10

Classes and Objects

- Programs grow in size, become more complex,
  - Number of programmers working on same project increases

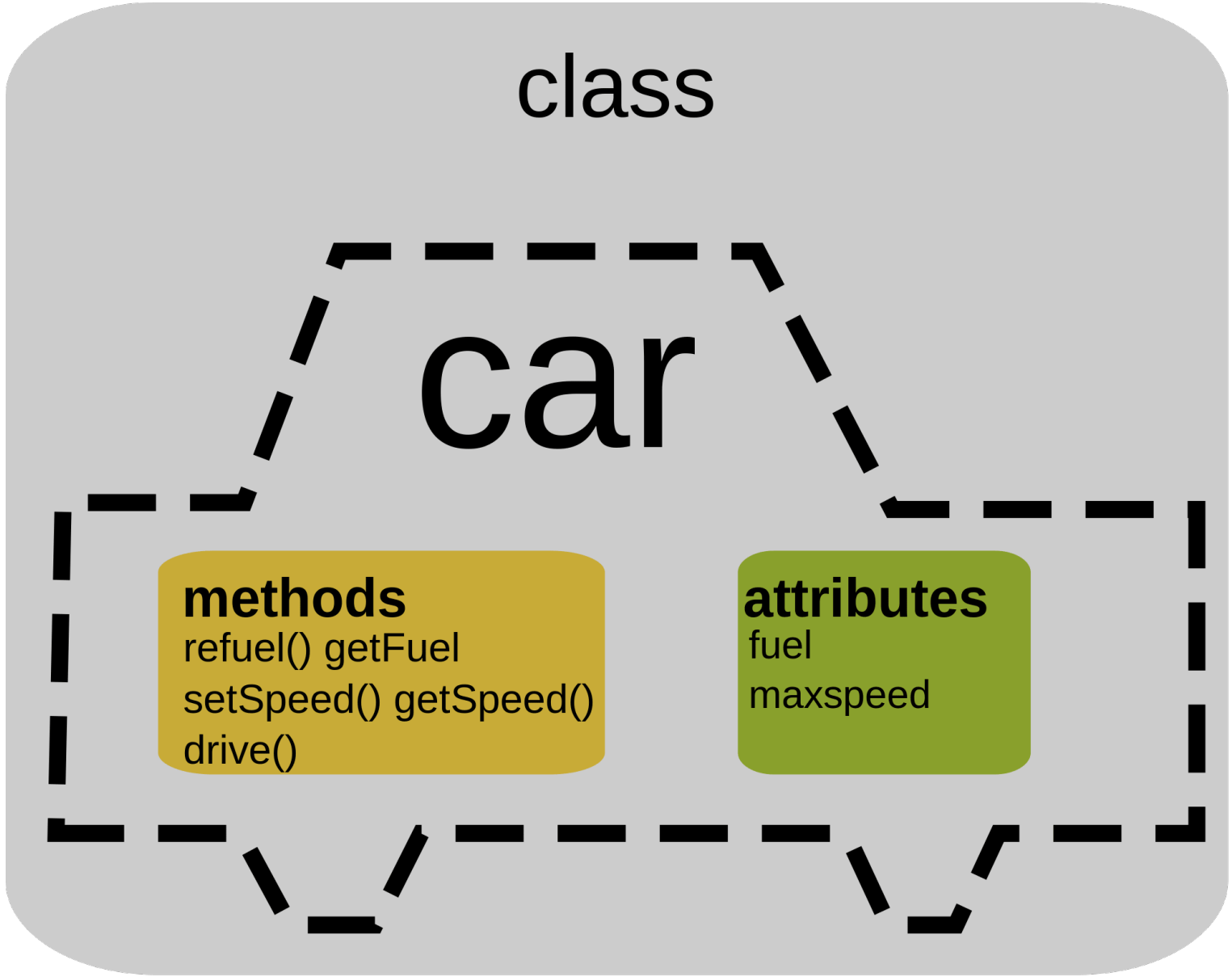
- Dependencies and interrelationships throughout code increases

In spaghetti code, the relations between the pieces of code are so tangled that it is nearly impossible to add or change something without unpredictably breaking something somewhere else.

Classes and Objects

- Partial solution to this problem is data hiding
  - Within a unit, as much implementation detail as possible is hidden

- Programmer using an object is concerned only with
  - Tasks that the object can perform
  - Parameters used by these tasks


class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed

Built-in Classes

- Note the use of word “class” instead of “type”.
- We refer to a specific literal from one of these classes as an ***instance*** (object) of the class.

```
>>> type(5)
<class 'int'>
>>> type(5.0)
<class 'float'>
>>> type("huh")
<class 'str'>
>>> type([1,2,'ab'])
<class 'list'>
>>> type((2,3))
<class 'tuple'>
>>> type({2,3,4})
<class 'set'>
>>> type(True)
<class 'bool'>
```

Why classes?

```
#Traditional method to define three rectangles and their areas
#Very repetitive and prone to errors.

rec1_width = 4
rec1_height = 3
rec1_area = rec1_width * rec1_height

rec2_width = 5
rec2_height = 4
rec2_area = rec2_width * rec2_height

rec3_width = 6
rec3_height = 5
rec3_area = rec3_width * rec3_height
```

User-defined Classes

- Classes are templates from which objects are created
  - Specifies properties, methods that will be common to all objects, instances of that class

- Classes can be
  - Typed directly into programs
  - Stored in modules and brought into programs with an import statement

User-defined Classes

- Each class defined by users will have a specified set of methods
- Each instance(object) of the class will have its own value(s)

```
class ClassName:
    indented list of methods for the class
```

User-defined Classes

- The **__init__** method (also known as the **constructor**) is automatically called when an object is created. It creates and assigns values to the instance variables that store the values for the object.

- **Instance variables** are also called the **properties** of the class, and the collections of values of the instance variables are called the state of the object.

- **Methods** are defined much like ordinary functions
  - Methods have *self* as their first parameter.
  - When an object (instance of the class) is created, each method's self parameter references the object so that the method knows which object to operate on.

Example 1: Class of rectangles

The *Rectangle* class can set and get measurements of rectangle

```
class Rectangle:

    def __init__(self, width, height): #__init__ creates objects
        self.width = width
        self.height = height

    def area(self): #method
        return self.width * self.height
```

```
>>> rec = Rectangle(4,3)
>>> rec.width
4
>>> rec.height
3
>>> rec.area()
12
```

```
>>> "CityU".upper()
'CITYU'
```

Traditional way vs class

```
#Things are a lot more elegant when using the user-defined class Rectangle!
rec1=Rectangle(4,3)
rec2=Rectangle(5,4)
rec3=Rectangle(6,5)

#Note that area is also defined this way
```

```
#Traditional method to define three rectangles and their areas
#Very repetitive and prone to erros.

rec1_width = 4
rec1_height = 3
rec1_area = rec1_width * rec1_height

rec2_width = 5
rec2_height = 4
rec2_area = rec2_width * rec2_height

rec3_width = 6
rec3_height = 5
rec3_area = rec3_width * rec3_height
```

Calling with the class name

```
>>> rec = Rectangle(10, 9)
>>>
>>> rec.area()
90
>>>
>>> #using the class name
>>> Rectangle.area(rec)    #note that the object must be passed into method
90
```

Example 2: Class of students

```
class Student:

    def __init__(self, first, last):
        self.first = first
        self.last = last

    def email(self):
        return f"{self.first}.{self.last}@cityu.edu.hk"
```

```
>>> student1 = Student("John", "Smith")
>>> student1.email()
'John.Smith@cityu.edu.hk'
```

Class variable

- The first and last variables in the previous examples are called instance variables – the value varies from instance(object) to instance (object).

- A class variable is a variable that is shared among all instances in the class.

Example 3: class variable

```
class Rectangle:
    count = 0
    def __init__(self, width, height):
        self.width = width
        self.height = height
        Rectangle.count += 1    #count is a class variable

    def area(self):
        return self.width * self.height
```

```
>>> rec1= Rectangle(4,3)
>>> rec2= Rectangle(5,4)
>>>
>>> rec1.count
2
>>> rec2.count
2
>>> Rectangle.count
2
```

Inheritance

Section 2

Chapter 7

Inheritance

- Inheritance allows us to define a new class
  - Called the subclass, child class, or derived class
  - A modified version of an existing class

- Subclass inherits properties and methods of the superclass
  - Adding some of its own properties and methods
  - Overriding some of the superclass' methods

Subclass

```
class Rectangle:
    def __init__(self, width, height): #__init__ creates objects
        self.width = width
        self.height = height
    def area(self): #method
        return self.width * self.height


#Creating a subclass

class Color_Rectangle(Rectangle):
    pass
```

```
>>> rec1=Color_Rectangle(4,3)
>>> rec1.area()
12
>>> #Note that rec1 was defined via the subclass.
```

issubclass

```
class Shape:
    def __init__(self, color):
        self.color = color
    def getColor(self):
        return self.color
    def __str__(self):
        return "A formless shape"
    def area(self):
        return None

class Circle(Shape):
    def __init__(self, radius, color):
        super().__init__(color)
        self.radius = radius
    def __str__(self):
        return "A " + self.color + " circle"
    def area(self):
        return 3.14159 * (self.radius**2)

class Rectangle(Shape):
    def __init__(self, length, width, color):
        super().__init__(color)
        self.length = length
        self.width = width
    def __str__(self):
        return "A " + self.color + " Rectangle"
    def area(self):
        return self.length * self.width

print(issubclass(Circle, Shape))
print(issubclass(Shape, Circle))
print(issubclass(Circle, Rectangle))
print(issubclass(Rectangle, Circle))
```

```
True
False
False
False
```

Example 4

```
class Rectangle:
    def __init__(self, width, height): #__init__ creates objects
        self.width = width
        self.height = height
    def area(self): #method
        return self.width * self.height


#Creating a subclass
class Color_Rectangle(Rectangle):
    def __init__(self, width, height, color):
        super().__init__(width, height)  #super() passes width & height to the super class __init__ method
        self.color = color
```

```
>>> rec1 = Color_Rectangle(4, 3, "Black")
>>> rec1.color
'Black'
>>> rec1.area()
12
```

The "is-a" Relationship

- Child classes are specializations of their parent's class
  - Have all the characteristics of their parents
  - But more functionality
  - Each child satisfies the "is-a" relationship with the parent

```
isinstance(object, className)
```

The *isinstance* Function

```
>>> isinstance("CityU", str)
True
>>> isinstance("CityU", list)
False

>>> rec1 = Color_Rectangle(4, 3, "Red")
>>> isinstance(rec1, Color_Rectangle)
True
>>> isinstance(rec1, Rectangle)
True
>>>
>>> rec2 = Rectangle(4, 3)
>>> isinstance(rec2, Rectangle)
True
>>> isinstance(rec2, Color_Rectangle)
False
>>>
>>>
>>> #rec1 defined via the subclass is also an instance(object) of the superclass
>>> #rec2 defined via the superclass is not an instance(object) of the subclass
```

isinstance

```
class Shape:
    def __init__(self, color):
        self.color = color
    def getColor(self):
        return self.color
    def __str__(self):
        return "A formless shape"
    def area(self):
        return None

class Circle(Shape):
    def __init__(self, radius, color):
        super().__init__(color)
        self.radius = radius
    def __str__(self):
        return "A " + self.color + " circle"
    def area(self):
        return 3.14159 * (self.radius**2)

class Rectangle(Shape):
    def __init__(self, length, width, color):
        super().__init__(color)
        self.length = length
        self.width = width
    def __str__(self):
        return "A " + self.color + " Rectangle"
    def area(self):
        return self.length * self.width

shape1 = Circle(1.00, "red")
shape2 = Rectangle(3.00, 2, "green")

print(isinstance(shape1, Circle))
print(isinstance(shape1, Rectangle))
print(isinstance(shape1, Shape))
```

```
True
False
True
```

Overriding a Method

- If method defined in subclass has the same name as a method in its superclass
  - Child's method will override parent's method

Classwork 10. Class of Employees

- Define a class of employees with their first name and last names and two methods:
  - initial: the first letters of the first and last names
  - number_letter: count the total number of letters in their names.
- The class should be capable of the following. Upload the .py file and the output screenshot on Canvas.

```
>>> employee1 = Employee("Norm", "Macdonald")
>>> employee1.first
'Norm'
>>> employee1.last
'Macdonald'
>>> employee1.initial()
'NM'
>>> employee1.number_letter()
13
```