Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Abstract Classes and Methods

Table of Contents

In the Object-Oriented Programming, there are 4 main concepts:

- Encapsulation
- Inheritance
- Abstraction
- Polymorphism

As we have covered the topics of Encapsulation and Inheritance in the previous lectures/ notes. Now, we will learn about the concept of Abstraction. So, let us see what it is.

ABSTRACTION

Abstraction is the process where a programmer hides every data except for the significant ones about an object to avoid complexity and to increase efficiency. To say this in a simpler way, it is basically the hiding of certain details and showing only the essential information to the user.

For example, you need to withdraw some money from an ATM machine. To do so, you would have to insert your card in it, input the amount, input your PIN number and finally the ATM machine will dispense some cash. Now, these are what you and any other user gets to see but the internal workings of the machine, that is, how does the machine know what to do, is hidden from you. This is abstraction.

Abstraction can be achieved by using Abstract classes and methods.

ABSTRACT CLASS

We know that a class is a blueprint or a design for creating objects. Similarly, an abstract class is considered as the blueprint for creating other classes.

Abstract classes are incomplete classes because they have methods which have no body. So, methods are only declared in these classes. And these methods with no body/implementation are known as abstract methods. Implementation of these abstract methods are provided in the subclasses.

Now, before we proceed further, let us see how to write an abstract class:

```
1  from abc import ABC, abstractmethod
2
3  class Shape(ABC):
4
5      # abstract method
6
7
```

Notice line 1 in the code above. Python actually does not provide abstract classes. But python does have a module called abc that provides an infrastructure for defining Abstract Base Classes (ABCs). Thus, the first line to begin with for writing an abstract class is to import ABC from the module abc. Then, write the

class you want to make abstract that takes ABC as an argument in the declaration. Here, the Shape class is a base class which is abstract.

ABSTRACT METHOD

An abstract method is a method that is declared but has no implementations. The abstract classes are classes that contain one or more abstract methods.

Abstract methods are decorated with the keyword @abstractmethod.

To add more, ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base.

Now, let us see how to write abstract methods.

```
1  from abc import ABC, abstractmethod
2
3  class Shape(ABC):
4
5      # abstract method
6      @abstractmethod
7      def square(self):
8          pass
9
10     @abstractmethod
11     def circle(self):
12         pass
```

The Shape class now has two methods that are abstract defined by the @abstractmethod decorator. Now, remember we said abstract methods have no body? This is why the pass keyword is used (as again we know that in python a method's body cannot be empty).

So, what is the point of such classes and methods and where are the method implementations?

Well, we use an abstract class as a template and according to out requirements we extend it and build on it before we can use it. It allows you to implement the abstract methods within any child/sub classes of the abstract class. This avoids the duplication of similar codes and hence, one common code can be used many times and in different ways based on how it is implemented in the sub classes.

INSTANTIATION OF ABSTRACT CLASSES

An abstract class is not a concrete class, and so it cannot be instantiated. When you try to create an object of an abstract class it will give an error.

Code

```
from abc import ABC, abstractmethod

class Shape(ABC):

    # abstract method
    @abstractmethod
    def square(self):
        pass

    @abstractmethod
    def circle(self):
        pass

obj_1 = Shape()
```

Output

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-644a7399e2c0> in <module>
     12         pass
     13
---> 14 obj_1 = Shape()

TypeError: Can't instantiate abstract class Shape with abstract methods circle, squa
re
```

Wait, what is a concrete class?

A concrete class is a class that has all its methods implemented. Specifically, for python, a concrete class is a subclass of an abstract class that implements **all** its abstract methods. Let us see a code for this:

```
from abc import ABC, abstractmethod

class Shape(ABC):

    @abstractmethod
    def square(self):
        pass

    @abstractmethod
    def circle(self):
        pass

class Square_Shape(Shape):

    def square(self):
        return f"I am a square."

    def circle(self):
        return f"I am NOT square."

obj_1 = Square_Shape()
```

In the code above, the **Shape** class is an abstract class because the methods in it have no implementations. But the **Square_Shape** class is a concrete class because it implements all the methods of the abstract class that it extends.

In this same code, if the **Square_Shape** class did not implement any one method among the two it will raise an error as shown below when the circle method is not implemented in it.

Code

```
from abc import ABC, abstractmethod

class Shape(ABC):

    # abstract method
    @abstractmethod
    def square(self):
        pass

    @abstractmethod
    def circle(self):
        pass


class Square_Shape(Shape):

    def square(self):
        return f"I am a square."


obj_1 = Square_Shape()
```

Output

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-2f9adf1a5a70> in <module>
     20
     21
---> 22 obj_1 = Square_Shape()

TypeError: Can't instantiate abstract class Square_Shape with abstract methods circl
e
```

When the abstract circle method is not implemented, the **Square_Shape** class is not concrete but abstract because it is inheriting an abstract method. To make it concrete, we need to override this circle method in the **Square_Shape** class.

So, in short, Abstract classes cannot be instantiated, and require subclasses to provide implementations for all the abstract methods. A class that is derived from an abstract class cannot be instantiated unless all of its abstract methods are overridden.

Now, we learned that abstract methods have no implementations but that does not mean it cannot have any implementations at all. It can but yet it **must** be overridden in its child classes. Thus, having implementations in the abstract class doesn't really make sense.

EXAMPLE

Code

```
from abc import ABC, abstractmethod

# Employee Class
class Employee(ABC):

    @abstractmethod
    def monthly_pay(self):
        pass

    @abstractmethod
    def annual_pay(self):
        pass

# Hourly_Employee Class
class Hourly_Employee(Employee):

    hourly_earnings = 10

    def monthly_pay(self):
        return Hourly_Employee.hourly_earnings * 4 * 30


    def annual_pay(self):
        return Hourly_Employee.hourly_earnings * 4 * 30 * 12

# Salaried_Employee Class
class Salaried_Employee(Employee):

    monthly_earnings  = 20000

    def monthly_pay(self):
        return Salaried_Employee.monthly_earnings


    def annual_pay(self):
        return Salaried_Employee.monthly_earnings*12


david = Hourly_Employee()
print("David's monthly pay is",david.monthly_pay())
print("David's annual pay is",david.annual_pay())

print("============================")

sarah = Salaried_Employee()
print("Sarah's monthly pay is",sarah.monthly_pay())
print("Sarah's annual pay is",sarah.annual_pay())

```

Output

```
David's monthly pay is 1200
David's annual pay is 14400
============================
Sarah's monthly pay is 20000
Sarah's annual pay is 240000
```

In the above code, there is one abstract class **Employee** and 2 of its derived class **Hourly_Employee** and **Salaried_Employee**. Notice in each of the subclasses, the methods **monthly_pay** and **annual_pay** are implemented but, in each class, it is implemented differently. Hence, we can see that the benefit of using abstraction is that one common code can be reused and implemented in various ways for various purposes.