Abstraction

> **Prereq: Inheritance**

Abstraction is used to hide something too, but in a **higher degree (class, module)**. Clients who use an abstract class (or interface) do not care about what it was, they just need to know what it can do.

Abstract class contains one or more abstract methods, and is considered a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class.

Abstract method is a method that has a declaration but does not have an implementation.

- An abstract class is a class that is declared abstract — it may or may not include abstract methods.
- Abstract classes cannot be instantiated, but they can be subclassed. This is the differentiation from Inheritance
- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class

Be aware, Abstraction reduces code repetition but increases coupling.

https://www.geeksforgeeks.org/abstract-classes-in-python/

Working on Python Abstract classes

Here, This code defines an abstract base class called “Animal” using the ABC (Abstract Base Class) module in Python. The “Animal” class has a non-abstract method called “move” that does not have any implementation. There are four subclasses of “Animal” defined: “Human,” “Snake,” “Dog,” and “Lion.” Each of these subclasses overrides the “move” method and provides its own implementation by printing a specific movement characteristic.

Concrete classes contain only concrete (normal) methods whereas abstract classes may contain both concrete methods and abstract methods.

```
# abstract base class work
from abc import ABC, abstractmethod
```

```
class Animal(ABC):

    def move(self):
        pass

class Human(Animal):

    def move(self):
        print("I can walk and run")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

Concrete Methods in Abstract Base Classes (Super)

Concrete classes contain only concrete (normal) methods whereas abstract classes may contain both concrete methods and abstract methods.

The concrete class provides an implementation of abstract methods, the abstract base class can also provide an implementation by invoking the methods via super(). Let look over the example to invoke the method using super():

```
# Python program invoking a
# method using super()
```

```
from abc import ABC


class R(ABC):
    def rk(self):
        print("Abstract Base Class")


class K(R):
    def rk(self):
        super().rk()
        print("subclass ")


# Driver code
r = K()
r.rk()
```

```
Abstract Base Class
subclass
```

Abstract Property

- use the decorator to prevent users from calling abstract method

```
# Python program showing
# abstract properties

import abc
from abc import ABC, abstractmethod

class parent(ABC):
    @abc.abstractproperty
    def geeks(self):
        return "parent class"
class child(parent):

    @property
    def geeks(self):
        return "child class"


try:
    r =parent()
    print( r.geeks)
except Exception as err:
    print (err)

r = child()
print (r.geeks)
```

Can't instantiate abstract class parent with abstract methods geeks
child class

Protocol vs ABC

Revision #5
Created 26 February 2024 01:47:50 by victor
Updated 1 April 2024 03:20:11 by victor