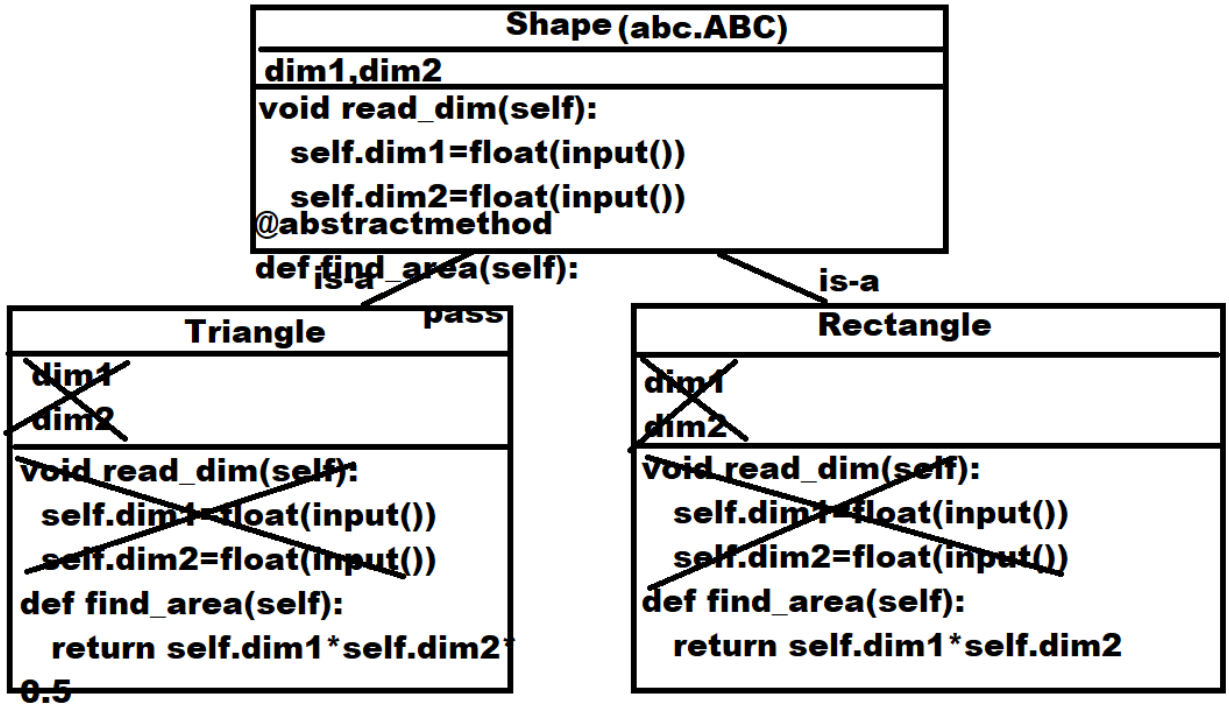


When more than one sub class having similar role, with different implementation that method is declared abstract.

Example:

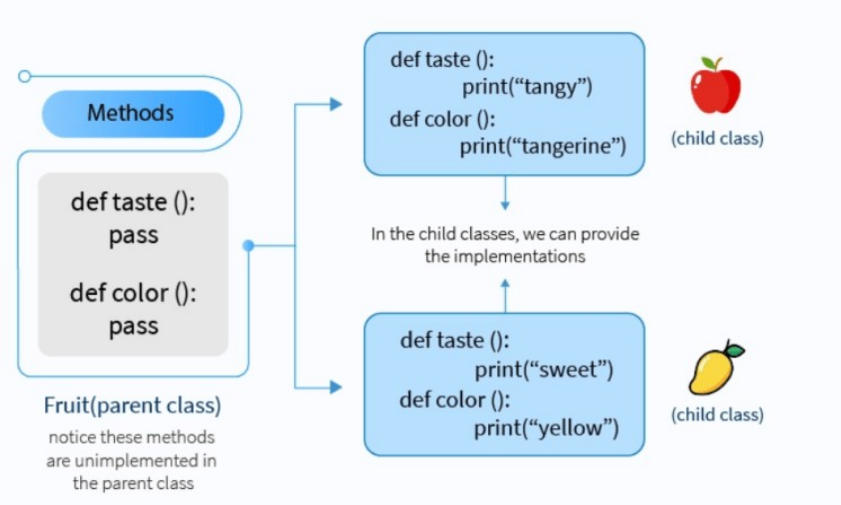
```
import abc

class Shape(abc.ABC):
    def __init__(self):
        self._dim1=None # Protected
        self._dim2=None # Protected
    def read_dim(self):
        self._dim1=float(input("Enter Dim1 :"))
        self._dim2=float(input("Enter Dim2 :"))
    @abc.abstractmethod
    def find_area(self):
        pass

class Triangle(Shape):
    def find_area(self):
        return self._dim1*self._dim2*0.5

class Rectangle(Shape):
```

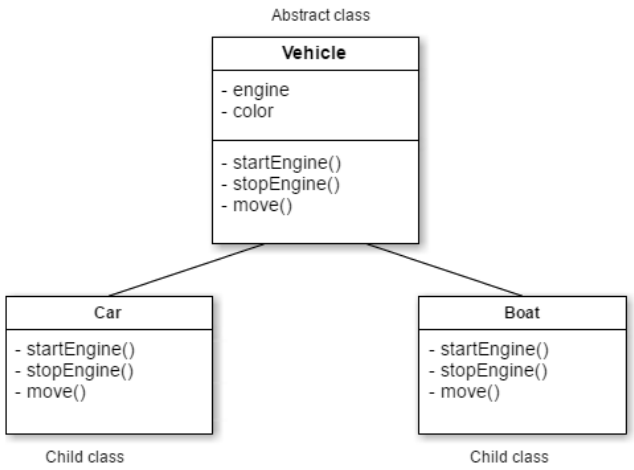
```
    def find_area(self):
        return self._dim1*self._dim2

t1=Triangle()
t1.read_dim()
area1=t1.find_area()

r1=Rectangle()
r1.read_dim()
area2=r1.find_area()
print(f'Area of triangle is {area1:.2f}')
print(f'Area of rectangle is {area2:.2f}')
```

Output

```
Enter Dim1 :1.2
Enter Dim2 :1.3
Enter Dim1 :1.5
Enter Dim2 :1.7
Area of triangle is 0.78
Area of rectangle is 2.55
```

Abstract class is called abstract data type; a data type which allows building similar data types is called abstract data type.





Example:

```
import abc
class Vehicle(abc.ABC):
    @abc.abstractmethod
    def start_engine(self):
        pass
    @abc.abstractmethod
    def stop_engine(self):
        pass
    @abc.abstractmethod
    def move(self):
        pass

class Car(Vehicle):
    def start_engine(self):
        print("Car Start....")
    def stop_engine(self):
        print("Car Stop")
    def move(self):
        print("Car Move")

class Boat(Vehicle):
    def start_engine(self):
        print("Boat Start...")
    def stop_engine(self):
        print("Boat Stop...")
    def move(self):
```

```
        print("Boat Move...")

car1=Car()
car1.start_engine()
car1.stop_engine()
car1.move()

boat1=Boat()
boat1.start_engine()
boat1.stop_engine()
boat1.move()
```

Output

```
Car Start....
Car Stop
Car Move
Boat Start...
Boat Stop...
Boat Move...
```

Interface

Interface is an abstract class contains only abstract methods. Interface defines specifications, which has to be implemented by every derived class.

Interface is pure abstract class.

Using interface, we can achieve

1. Abstraction
2. Runtime Polymorphism

What is abstraction?

Hiding implementations by giving only specifications is called abstraction.

What is runtime polymorphism?

An ability of a reference variable change its behavior based on the type of object assigned is called runtime polymorphism.

RBI

```
class Debitcard(abc.ABC):
    @abc.abstractmethod
    def withdraw(self):
        pass
```

HDFC

```
class HdfcDebitcard(Debitcard):
    def withdraw(self):
        print("withdraw 50000")
```

SBI

```
class SBIDebitcard(Debitcard):
    def withdraw(self):
        print("withdraw 10000")
```

ICICI

```
class ICICIATM:
    def insert(self,d):
        d.withdraw()
```

Example:

```
import abc

class Debitcard(abc.ABC):
    @abc.abstractmethod
    def withdraw(self):
        pass

class HdfcDebitcard(Debitcard):
    def withdraw(self):
        print("withdraw of hdfc bank")

class SbiDebitcard(Debitcard):
    def withdraw(self):
        print("withdraw of sbi bank")

class IciciAtm:
    def insert(self,d):
        d.withdraw()



card1=HdfcDebitcard()
card2=SbiDebitcard()
```

```
atm1=IciciAtm()
atm1.insert(card1)
atm1.insert(card2)
```

Output

```
withdraw of hdfc bank
withdraw of sbi bank
```

What is duck typing?

A programming style which does not look at an object's type to determine if it has the right interface; instead, the method or attribute is simply called or used (“If it looks like a duck and quacks like a duck, it must be a duck.”) By emphasizing interfaces rather than specific types.

Example:

```
class Car:
   def drive(self):
      print("I'm driving a Car!")


class GolfClub:
   def drive(self):
      print("I'm driving a golf club!")

class Alpha:
    def drive(self):
        print("i'm not driving")

def test_drive(item) :
   item.drive() # don't care what it is, all i care is that it can
"drive"

car = Car()
test_drive(car) #=> "I'm driving a Car"

club = GolfClub()
test_drive(club) #=> "I"m driving a GolfClub"

a=Alpha()
```

```
test_drive(a)
```

Example:

```
import abc
class Sim(abc.ABC):
    @abc.abstractmethod
    def connect(self):
        pass

class JioSim(Sim):
    def connect(self):
        print("Connect to Jio Network")

class AirtelSim(Sim):
    def connect(self):
        print("Connect to Airtel Network")

class Mobile:
    def insert(self,s):
        s.connect()

card1=JioSim()
card2=AirtelSim()

mobile1=Mobile()
mobile1.insert(card1)
mobile1.insert(card2)
```

Output

```
Connect to Jio Network
Connect to Airtel Network
```