COE 302
Object Oriented Programming with Python III

Akinola Oluwole

soakinola@abuad.edu.ng

Content

- Abstraction
- UML

What is Abstraction

- Abstraction in Python is a programming methodology in which details of the programming codes are hidden away from the user, and only the essential things are displayed to the user.
- Abstraction is achieved in either Abstract classes or interface in Python.
- The solution to complexity is **abstraction**, also known as **information hiding**.
- Abstraction is simply the removal of unnecessary detail.
- The idea is that to design a part of a complex system, you must identify what about that part others must know in order to design their parts, and what details you can hide. The part others must know is the abstraction.

Benefits of Abstraction

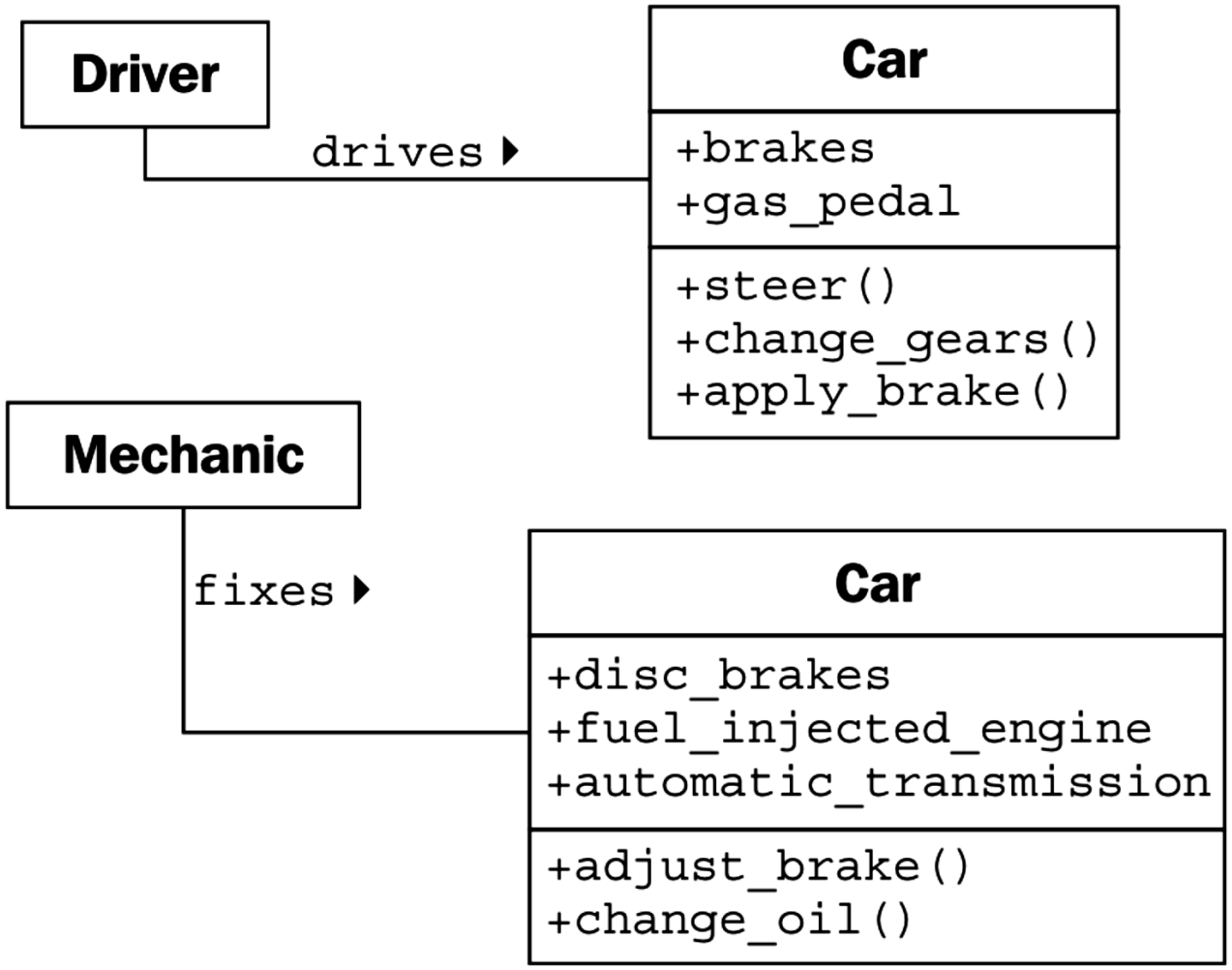
- The main purpose of abstraction is hiding the unnecessary details from the users
- It helps in reducing programming complexity
- Abstraction increase efficiency
- It is one of the most important concepts of OOPs

Example of Abstraction

- A car is a set of thousands of individual parts.
- A car is a well-defined object with its own unique behavior.
- This abstraction allows people to use a car to drive without knowing the complexity of the parts that form the car.
- They can ignore the details of how the engine transmission, and braking systems work. Instead, they are free to utilize the object as a whole.

Example of Abstraction

Abstraction levels for a car



- A car's driver needs to interact with the steering, accelerator, and brakes.
- The workings of the motor, drive train, and brake subsystem don't matter to the driver.
- A mechanic, on the other hand, works at a different level of abstraction, tuning the engine and bleeding the brakes.

Abstraction for a Student Information System

- Creating a student information system, will require collecting about all the information required for the system.
- Assuming the following information was collected,

1. Name
2. Address
3. Matric Number
4. Department
5. Favourite food
6. Best Friend
7. Mother’s Maiden Name …

Abstraction for a Student Information System

1. Name
2. Address
3. Matric Number
4. Department
5. Favourite food
6. Best Friend
7. Mother’s Maiden Name …

Abstraction for a Student Information System

- Clearly, items 1-4 are relevant for our student information system but items 5-7 are not.
- You need to select only the useful information for your student application from that pool
- The process of filtering out relevant information from a large list represents an abstraction example in OOPs

Difference between Abstraction and Encapsulation

Abstraction	**Encapsulation**
Abstraction in OOP solves the issues at the design level.	Encapsulation solves it implementation level.
Abstraction in Programming is about hiding unwanted details while showing most essential information.	Encapsulation means binding the code and data into a single unit.
Data Abstraction in Python allows focussing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Abstraction Classes

- A class containing one or more abstract methods is called an abstract class.
- Abstract methods do not contain any implementation.
- Instead, all the implementations can be defined in the methods of sub-classes that inherit the abstract class.
- Example, an abstract class is created by importing a class named **Student** from the **student** module and inheriting the **Student** class. Next slide is the syntax for creating the abstract class.

Class Result(Student): #abstract class

```
from student import Student
Class Result(Student):
     def calculate_grade(self): #abstract method
          pass
```

Here, the class **Student** is imported to and inherited by the class **Result**. The class **Result** becomes an abstract class when we define abstract methods in it.

Abstract methods should not contain any implementation. So, we can define abstract methods using the '**pass** statement' as shown in the code.

Here, **calculate_grade** is the abstract method of the abstract class **Result**.

The implementation of this abstract class can be defined in the sub-classes that inherit the class Result.

Abstract Classes

- Abstract Class is a type of class in OOPs, that declare one or more abstract methods. These classes can have abstract methods as well as concrete methods.
- A normal class cannot have abstract methods. An abstract class is a class that contains at least one abstract method.
- Abstract classes help to describe generic types of behaviors and OOP class hierarchy.
- It also describes subclasses to offer implementation details of the abstract class.
- It also extends the same Abstract class and offers different implementations of the abstract methods.

Abstract Method

- Abstract Method is a method that has just the method definition but does not contain implementation.
- A method without a body is known as an Abstract Method.
- It must be declared in an abstract class.
- The abstract method will never be final because the abstract class must implement all the abstract methods.
- Abstract methods are mostly declared where two or more subclasses are also doing the same thing in different ways through different implementations.
- It also extends the same Abstract class and offers different implementations of the abstract methods.

Abstract Classes and Method Example

- To declare an Abstract class, we firstly need to import the **student** module. Let us look at an example.

```
from student import Student
class Result(Student):      #abstract methods
```

- Here, **Result** is the abstract class inside which abstract methods or any other sort of methods can be defined.

- As a property, abstract classes can have any number of abstract methods coexisting with any number of other methods. For example we can see below.

Abstract Classes and Method Example

```
from student import Student, abstractmethod
class Result(Student):      #abstract methods
      #normal method
      def method(self):
            #method definition
      @abstractmethod
      def Student_method(self):
         #Student_method definition
```

- Here, **method()** is normal method whereas **Student_method()** is an abstract method implementing **@abstractmethod** from the student module.

Abstract Classes and Method Example

```
from abc import ABC, abstractmethod
class Predator(ABC):
   @abstractmethod
   def eat(self, prey):
      pass
class Bear(Predator):
     def eat(self, prey):
          print(f'Mauling {prey}!')
class Owl(Predator):
     def eat(self, prey):
          print(f'Swooping in on {prey}!')
```

Abstract Classes and Method Example

- Inheriting from **ABC** makes this class an abstract base class.
- **@abstractmethod**: This indicates that the method must be defined on any subclasses
- **def eat(self, prey):** This method signature can be checked by IDEs in any subclasses.
- **Pass:** Abstract methods have no default implementation
- **class Bear(Predator)::** States your intent to implement the interface by subclassing the abstract base class
- **def eat(self, prey):**This method must be defined, or an exception will be raised.

QUIZ: Time 12 Mins.

Questions

- What is Abstraction in OOP?
- State a difference between Abstraction and Encapsulation
- How are Abstract Method different from normal method in a class?
- Define an abstract class for Bank account with a relevant abstract method and a normal method?

Reference

- https://ncert.nic.in/vocational/pdf/ivas103.pdf
- What is Abstraction in OOPs? Java Abstract Class & Method (guru99.com)
- Class and Object — Python Numerical Methods (berkeley.edu)