COMP642

Object Oriented Programming

Lectorial 7 - Abstract Classes, Implementing Interface

Designing a Hierarchy (1)

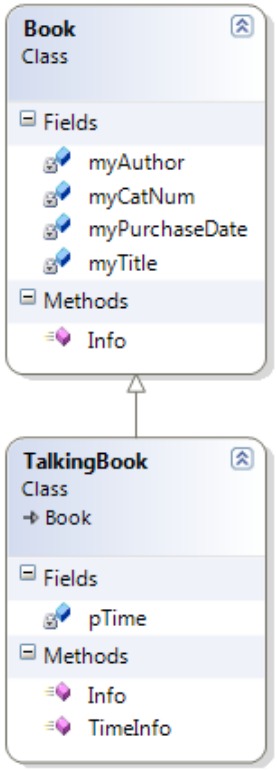
- Say we have our book hierarchy on the right:

  What if we want to include DVDs in our program?

  A DVD has a lot in common with a book and with a TalkingBook (what?).

  However, a **DVD IS NOT A book** (it doesn't have an author).

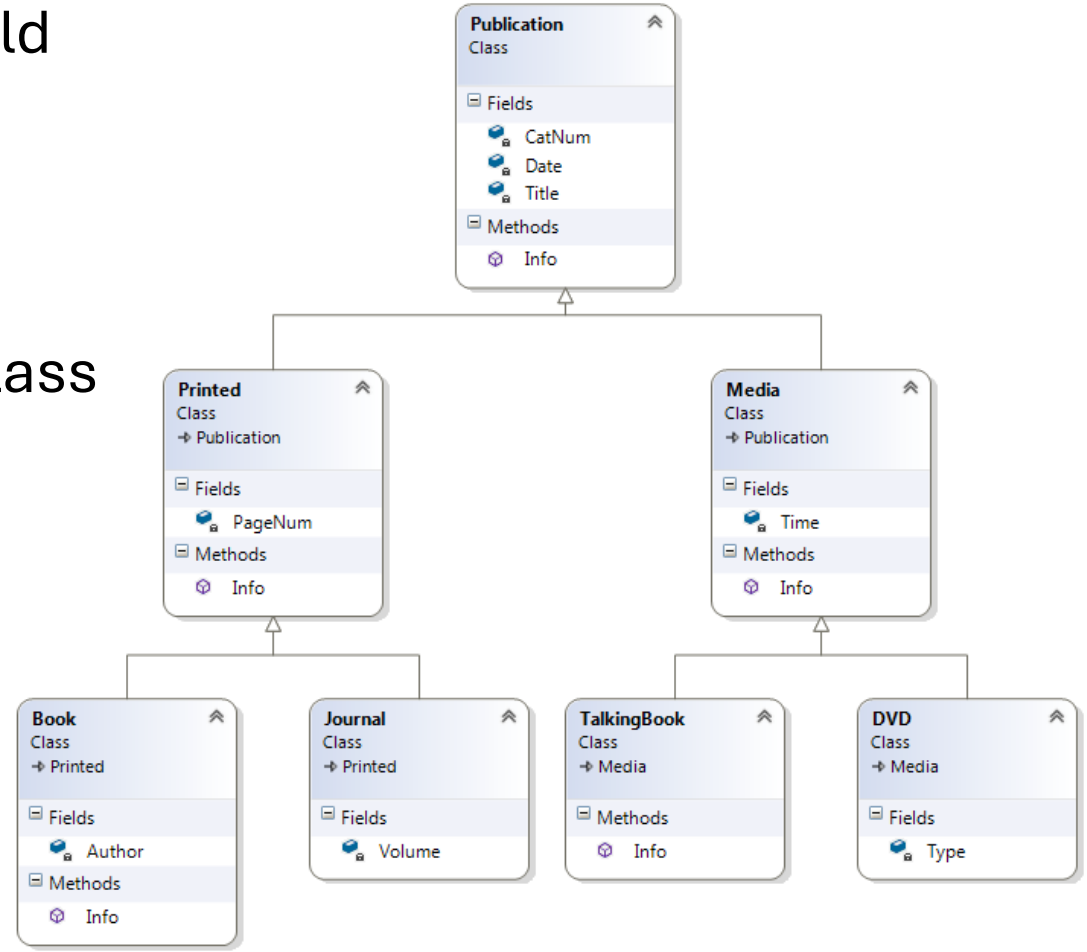
  Inheriting DVD from book is not a good idea.



Designing hierarchies is DIFFICULT ☹

Designing a Hierarchy (2)

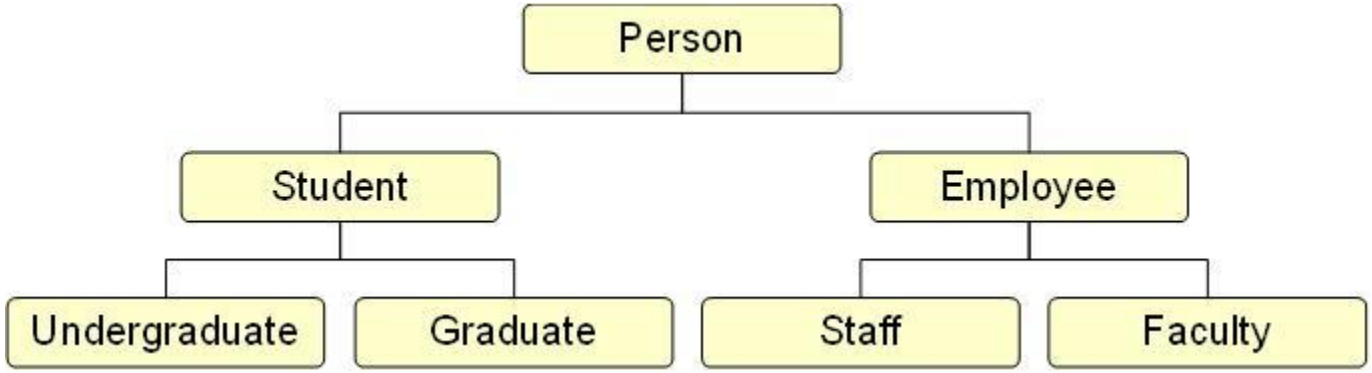
- The base class should be as general as possible.
- For library items, we might have a base class called Publication.
- We might then have printed and other publications.



Abstract Class (1)

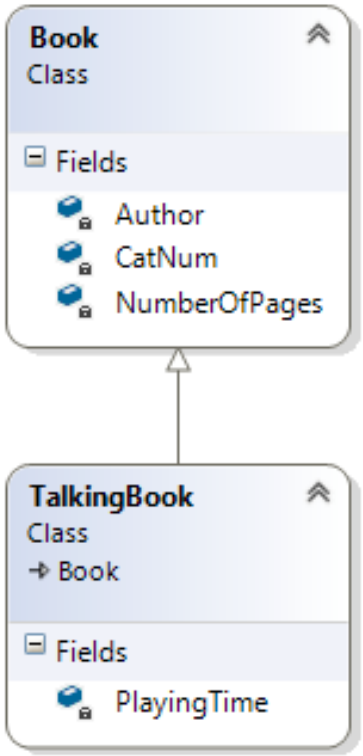
- The base class “Publication” would never have any objects.
- Any publication added to the library would be one of the derived classes (a book, or a DVD, etc).
- A class that never has any objects is called an **abstract class**. The other classes that have objects are called **concrete classes**.
- Every hierarchy should have an abstract class at the top – concrete classes should be the leaves.

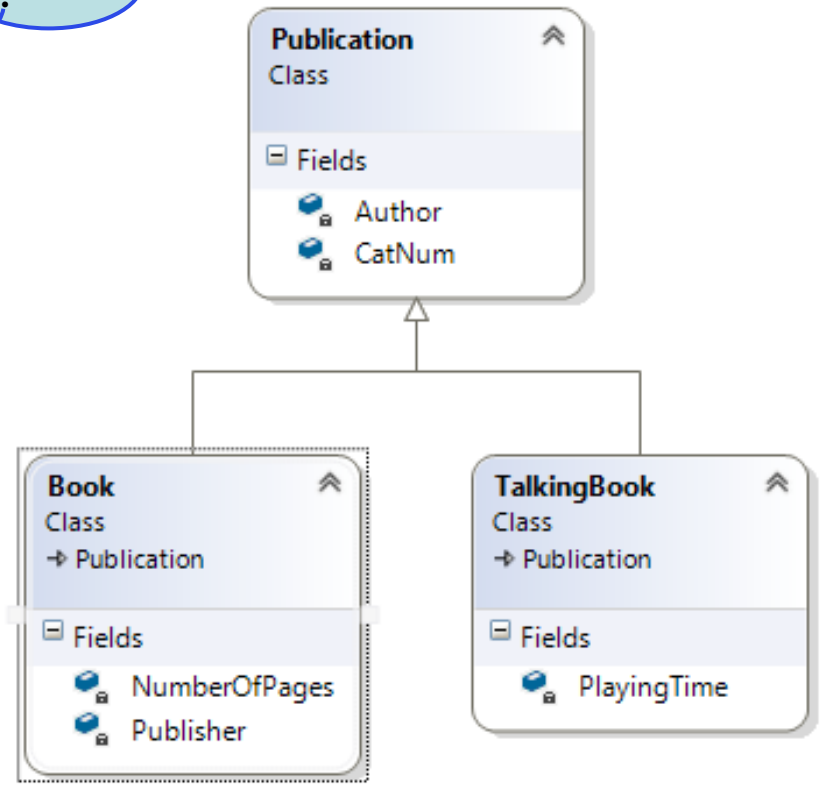
Abstract Class (2)



Abstract Class (3)

What if we need to add number of pages to book?



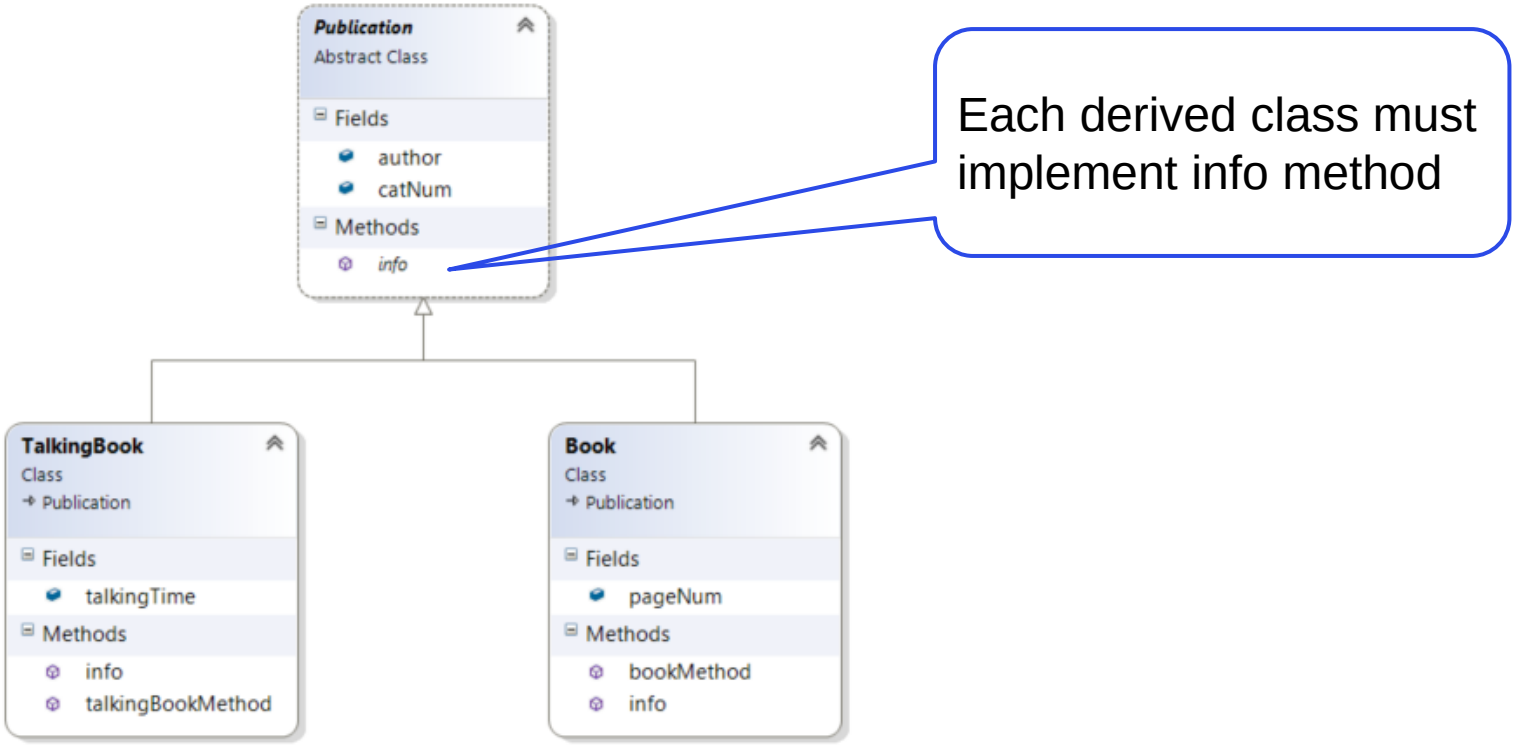


Abstract Class (4)

- Provides a blueprint for concrete classes.
- Does not contain implementation.
- Provides an interface to make sure that derived concrete classes are properly implemented.
- Abstract based classes cannot be instantiated. They are inherited and extended by the concrete subclasses.
- Subclasses derived from specific abstract base must implement the methods and properties provided in the abstract base class.

Abstract Class (5)

- The abstract base class (publication) provides the minimum attributes and methods that every publication object (of whatever type) must have.



Abstract Class (6)

- Publication class can be defined as follows:

```
class Publication(ABC):
    def __init__(self, catnum, author):
        self._catnum = catnum
        self._author = author

    @abstractmethod
    def info(self):
        pass
```

1. Import ABC and abstractmethod from the Python abc module
2. Derive the Publication class from ABC
3. Add the @abstractmethod to the info method

Abstract Class (7)

```
aBook = Publication("1234", "Jim")
```

- If we try to create an instance of Publication, we get the following error:

```
    aBook = Publication("1234", "Jim")
TypeError: Can't instantiate abstract class Publication with abstract methods info
```

Abstract Class Inheritance (1)

- Each derived class **must** have an implementation of the abstract method info().

```
class Book(Publication):
    def __init__(self, catnum, author, pnum, pub):
        self._pnum = pnum
        self._pub = pub
        super().__init__(catnum, author)

    def info(self):
        print("This is a book")
```

```
aBook = Book("1234", "Smith", 200, "ABC")
aBook.info()
```

```
This is a book
```

Abstract Class Inheritance (2)

- If info() is not defined in the Book class, we will get the following error:

```
    aBook = Book("1234", "Smith", 200, "ABC")
TypeError: Can't instantiate abstract class Book with abstract methods info
```

- Derived class must provide concrete implementation for all the abstract methods of the base class. In this instance, the derived classes can also be referred to as concrete classes.
- Concrete class is required to override the abstract methods defined in the abstract class.

Using super to Call a Method from an Abstract Class

- An abstract method does not have to be completely empty.
- It can contain some implementation that can be reused by derived classes by calling the abstract method with super(), including the constructor.

```
class Publication(ABC):
    def __init__(self, catnum, author):
        self._catnum = catnum
        self._author = author

    @abstractmethod
    def info(self):
        print("This is a publication")
```

```
class Book(Publication):
    def __init__(self, catnum, author, pnum, pub):
        self._pnum = pnum
        self._pub = pub
        super().__init__(catnum, author)

    def info(self):
        super().info()
        print("This is a book")
```

Implementing an Abstract Property

- We can define method to read and modify the value of the attributes (catnum, author) in Publication.
- Enforces their implementation in every subclass

Publication

```python
@property
@abstractmethod
def CatNum(self):
    pass

@CatNum.setter
@abstractmethod
def CatNum(self, value):
    pass

@property
@abstractmethod
def Author(self):
    pass

@Author.setter
@abstractmethod
def Author(self, value):
    pass
```

Book

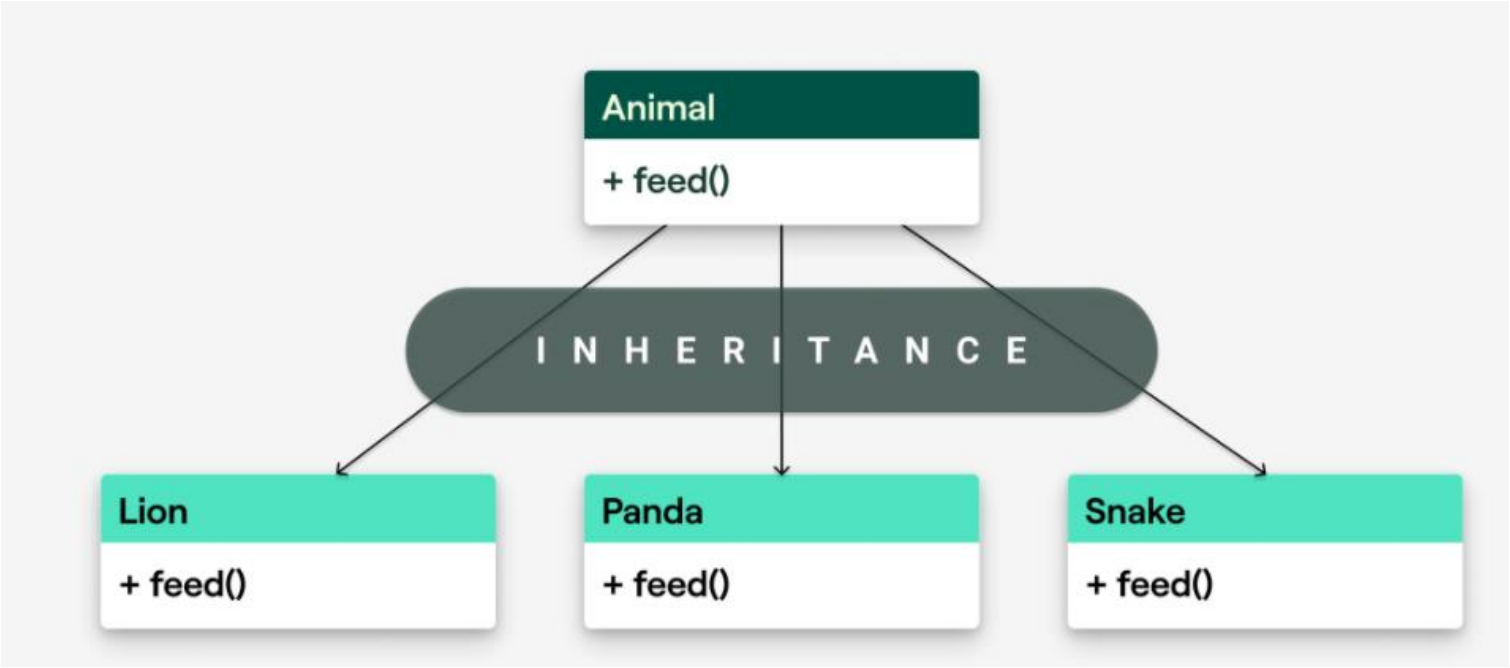
```python
@property
def CatNum(self):
    return self._catnum

@CatNum.setter
def CatNum(self, value):
    self._catnum = value

@property
def Author(self):
    return self._author

@Author.setter
def Author(self, value):
    self._author = value
```

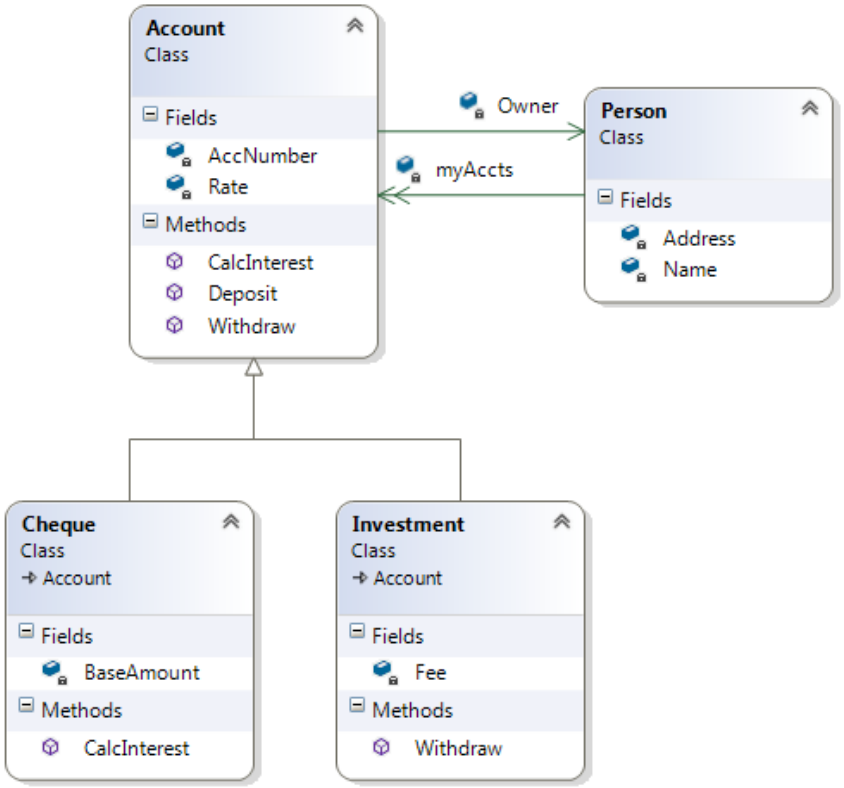
Writing Cleaner Code using Abstract Class




```
zoo = [Lion(), Panda(), Snake()]
for animal in zoo:
    animal.feed()
```

Test Your Knowledge

- What difference would all this make to the account class in the previous lectures?



Interface

- A programming structure/syntax that allows the computer to enforce certain properties on object (class)
- They are classes without code.
- A description of how an object behaves.
- It is a set of publicly accessible methods of an object which can be used by other parts of the program to interact with that object.

What is Interface in Python?

- Abstract class may contain some abstract methods as well as non-abstract method.

```
@abstractmethod
def get_salary(self):
    pass

def info(self):
    print("Info is a non-abstract method")
```

```
anEmp = FulltimeEmployee("John Doe", 6000)
print(anEmp.get_salary())
anEmp.info()
```

```
6000
Info is a non-abstract method
```

- Interface is an abstract class which contains only abstract methods.
- Python does not provide interface explicitly; we have to use abstract classes to create interface manually.

Interface in Python (1)

- Python has a completely different typing philosophy to Java or C#.
- No native support for interfaces.
- Generally, two solutions:
  - Use some third-party framework that adds a notion of interfaces (such as zope.interface package) – **not covered here**
  - Use of abstract base classes

Interface in Python (2)

```python
from abc import ABC, abstractmethod

class myClass(ABC):
    @abstractmethod
    def connect(self):
        pass

    @abstractmethod
    def disconnect(self):
        pass
```

Interface in Python (3)

```python
class mySQL(myClass):
    def connect(self):
        print("Connecting to mySQL database...")

    def disconnect(self):
        print("Disconnecting from mySQL database...")

class POSTGRES(myClass):
    def connect(self):
        print("Connecting to POSTGRES database...")

    def disconnect(self):
        print("Disconnecting from POSTGRES database...")
```

Interface in Python (4)

```python
dbstr = input("Enter the database name: ")

className = globals()[dbstr]

aConn = className()

aConn.connect()
aConn.disconnect()
```

```
Enter the database name: POSTGRES
Connecting to POSTGRES database...
Disconnecting from POSTGRES database...
```

Test Your Knowledge

Suppose that you need to develop a payroll program for a company. The company has two groups of employees:

- full-time employees get a fixed salary
- hourly employees get paid by hourly wages

The payroll program needs to print out a payroll that includes employee names and their monthly salaries.