CS65: Introduction to Computer Science

Scope and Visibility
Aggregation vs Inheritance

Md Alimoor Reza
Assistant Professor of Computer Science

Exercise 1: Create a Bank Account Class

- Call it class 'AccountInsecure'
  - Attributes:
    - a bank account number (it could be a String)
    - a balance amount (it should be a number)
  - Methods:
    - `deposit()`: that can change old balance of your account by adding a new amount
    - `withdraw()`: that can change old balance of your account by removing an amount

Demo: Bank Account Class Creation

```
class AccountInsecure:

    def __init__(self, par_account_num, par_balance):

        self.account_num = par_account_num
        self.balance     = par_balance

    def __str__(self):

        str_var = "BankAccount ( " + self.account_num + ", " + str(self.balance) + " )"
        return str_var

    def deposit(self, amount):

        self.balance = self.balance + amount

    def withdraw(self, amount):

        self.balance = self.balance - amount
```

Demo: Bank Account Object Creation

```
# ----------------- creating object (instance of Class) ------------------------

account_a = AccountInsecure("AC#1234", 1000)

print("Before withdrawal: ", account_a)

account_a.withdraw(500)

# Unsecure access to your balance (1)
print("After withdrawal(500) account_a.balance: ", account_a.balance)

# Unsecure update to your balance (2)
account_a.balance = 0
print("After updating account_a.balance: ", account_a.balance)
```

Topics

- Scope and Visibility
- Aggregation vs Inheritance

Scope and Visibility

- Sometimes data is **<u>'sensitive'</u>** – either shouldn't be known outside an object, or at least shouldn't be directly modifiable by outsiders.
  - Eg, an Account object's balance should be modifiable but it <u>should be accessed in controlled fashion via methods</u>.

```
# ------------     insecure bank account     ------------
class AccountInsecure:

    def __init__(self, par_account_num, par_balance):

        self.account_num = par_account_num
        self.balance     = par_balance

    def __str__(self):

        str_var = "BankAccount ( " + self.account_num + ", " + str(self.balance) + " )"
        return str_var

    def deposit(self, amount):

        self.balance = self.balance + amount

    def withdraw(self, amount):

        self.balance = self.balance - amount
```

Insecure Access

- The `account_a.balance` shouldn't be accessed directly!

```python
# ---------------- creating object (instance of Class) -----------------------

account_a = AccountInsecure("AC#1234", 1000)

print("Before withdrawal: ", account_a)

account_a.withdraw(500)

# Unsecure access to your balance (1)
print("After withdrawal(500) account_a.balance: ", account_a.balance)

# Unsecure update to your balance (2)
account_a.balance = 0
print("After updating account_a.balance: ", account_a.balance)
```

```
>>> %Run bank_account_insecure.py

  Before withdrawal:  BankAccount ( AC#1234, 1000 )
  After withdrawal(500) account_a.balance:   500
  After updating account_a.balance:   0

>>>
```

Solution: Private Attribute

- Attributes whose identifiers begin with a double-underscore (__) are **private**: dot-operator access won't work from outside object
  - Eg, change the attribute balance to __balance

- In order to access **private** attributes, we need to define getter() and setter() methods

Solution: Private Attribute

```
# ------------      secure bank account     ------------
class AccountSecure:

    def __init__(self, par_account_num, par_balance):

        self.account_num = par_account_num
        self.__balance     = par_balance

    def __str__(self):

        str_var = "BankAccount ( " + self.account_num + ", " + str(self.__balance) + " )"
        return str_var

    def deposit(self, amount):

        self.__balance = self.__balance + amount

    def withdraw(self, amount):

        self.__balance = self.__balance - amount

    def get_balance(self):

        return self.__balance
```

Solution: Private Attribute

```
# ------------     secure bank account     ------------
class AccountSecure:

    def __init__(self, par_account_num, par_balance):

        self.account_num = par_account_num
        self.__balance     = par_balance

    def __str__(self):

        str_var = "BankAccount ( " + self.account_num + ", " + str(self.__balance) + " )"
        return str_var

    def deposit(self, amount):

        self.__balance = self.__balance + amount

    def withdraw(self, amount):

        self.__balance = self.__balance - amount

    def get_balance(self):

        return self.__balance
```

```
27  # ----------------- creating object (instance of Class) ------------------------
28
29  account_a = AccountSecure("AC#1234", 1000)
30
31  print("Before withdrawal: ", account_a)
32
33  account_a.withdraw(500)
34
35  # Unsecure access to your balance (1)
36  print("After withdrawal(500) balance is : ", account_a.get_balance())
37
38  # account_a.__balance
39  # (ERROR: AttributeError: 'AccountSecure' object has no attribute '__balance')
40
```

Shell

```
Python 3.7.9 (bundled)
>>> %Run bank_account_secure.py

 Before withdrawal:  BankAccount ( AC#1234, 1000 )
 After withdrawal(500) balance is :  500
```

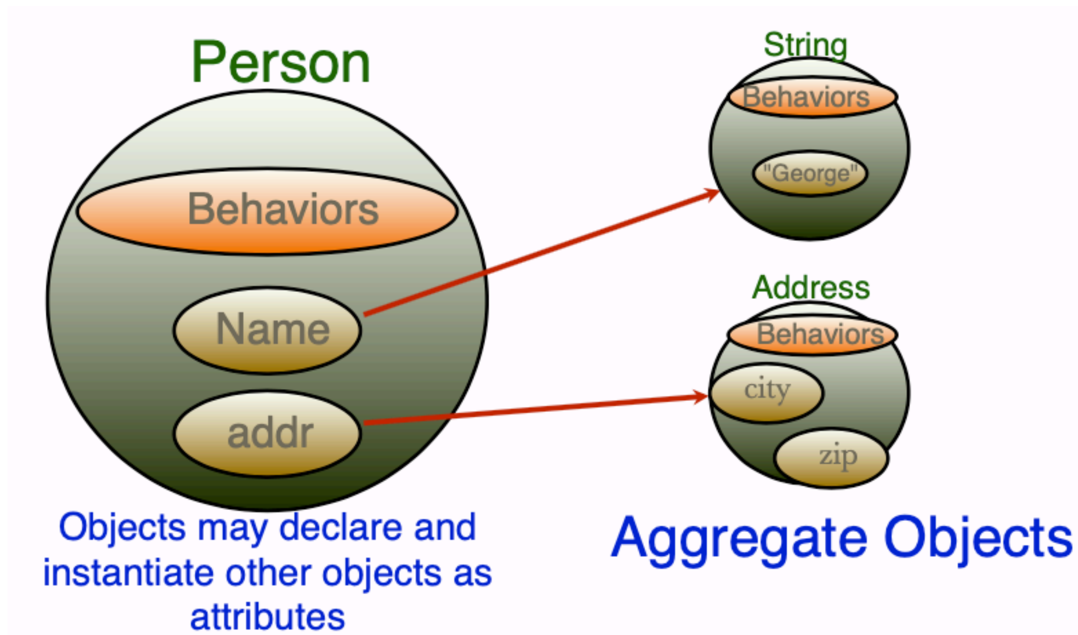
Drake UNIVERSITY

Exercise 2: Update Dog Class by Private Attributes

- A Dog has 'name', 'color', and 'breed'
  - Attributes: what changes are required?
  - Methods: what changes are required?

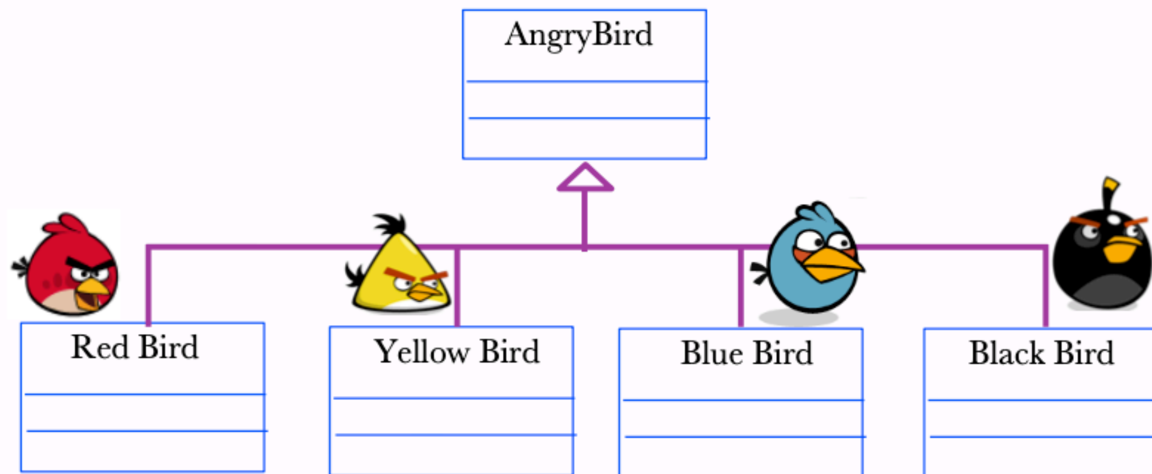
OOP: Aggregation

- Aggregation is the collecting of other (possibly complex) values. An object can have other objects as data, just like a list can have other lists as items inside of it.



OOP: Inheritance

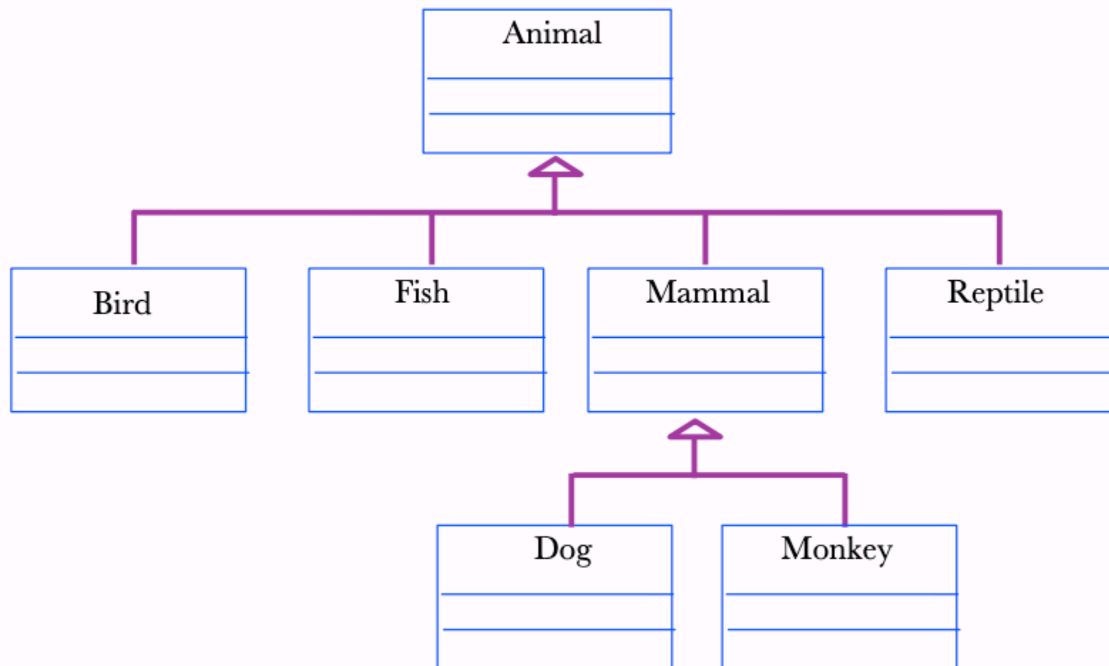
- Inheritance: a mechanism to derive a new class from existing ones
- The existing class is called the superclass/parent class/base class
- The derived class is called the subclass/child class/extended class
- Inheritance defines child class and parent class relations



- Parent class defines the general features
- Child class extends with special characteristics

OOP: Inheritance

- Inheritance defines child class and parent class relations



- Parent class hierarchy of multiple levels
- Child class inherits from immediate parent as well as the ancestors

Inheritance (is a) vs Aggregation (has a)

- Every object of child class is a value of the parent class type
  - Student is a Person
  - Cat is an Animal
- Inheritance depicts the ”is a” relationship
  - Child is a more specific version of parent

- Aggregation embodies the “has a” relationship
  - Student “has a” GPA
  - A Car “has a” Wheel