UNIVERSITY of WISCONSIN
LA CROSSE
COMPUTER SCIENCE

# CS 224 Introduction to Python

Inheritance

1

## What is inheritance?

- Inheritance is the mechanism for creating a class that modifies or *specializes* an existing class.
- The existing class is the base class or superclass.
- The new class is the derived class or subclass.
- The subclass *inherits* all attributes of the superclass (data and methods).
- The subclass can redefine (override) anything it inherits.

2

# Recall our Account class

```
class Account(object):
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance

    def deposit(self, amt):
        self.balance += amt

    def withdraw(self, amt):
        self.balance -= amt

    def inquiry(self):
        return self.balance
```

3

# A class Derived from Account

```
class EvilAccount(Account):

    def inquiry(self):
        if random.randint(0, 4) == 1:
            return self.balance * 1.25
        else:
            return self.balance
```

overrides the method in Account

20% of the time, return an artificially high balance to "help" the account holder overdraw their account which increases fees paid to ~~Wells Fargo~~ the bank.

4

## How does it work?

An access (dot operator) of an attribute (data or method) for an instance of `EvilAccount`, looks first in `EvilAccount`. If the item isn't found there, Python looks in the superclass, `Account`.

It doesn't stop there. If the item isn't found in `Account`, Python looks in `Account`'s superclass. This continues until reaching `object`, the root of the class tree.

5

## Adding to `EvilAccount`

```
class EvilAccount(Account):

    def __init__(self, name, balance, ef):
        Account.__init__(self, name, balance)
        self.evil_factor = ef

    def inquiry(self):
        if random.randint(0, 4) == 1:
            return self.balance * self.evil_factor
        else:
            return self.balance
```

Superclass __init__ not automatically invoked.

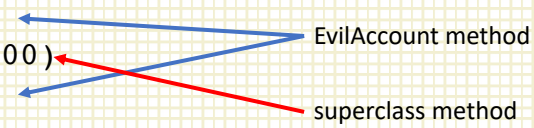
defining a new attribute

6

# Create an `EvilAccount` object

```
checking2 = EvilAccount('Jane', 10000, 1.1):

checking2.inquiry()
checking2.deposit(100)
checking2.inquiry()
```

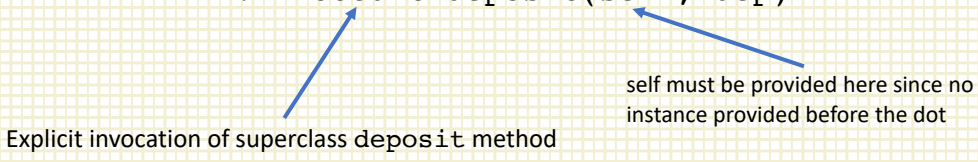



7

# `EvilerAccount`

```
class EvilerAccount(EvilAccount):

    def deposit(self, dep):
        self.withdraw(5)          # convenience fee
        EvilAccount.deposit(self, dep)
```




8

# EvilerAccount

A more general version:

```
class EvilerAccount(EvilAccount):

    def deposit(self, dep):
        self.withdraw(5)          # convenience fee
        super().deposit(dep)
```

You don't have to explicitly name the superclass

NOTE: `super()` is Python3 specific

9

# Multiple Inheritance

## What is it?

Allows a class to inherit features from more than one superclass.

## Why do we need it?

Some types exhibit characteristics of multiple other types so extending them both is desirable/necessary.

10

# Example

An iPhone 11 is a smart phone.
- It has some characteristics of a phone
- It has some characteristics of a computer

Phone alone doesn't describe it because not all phones are computers.

Computer alone doesn't describe it because not all computers are phones.

11

# Multiple Inheritance:

Consider these two new classes related to `Account`:

```
class DepositCharge(object):
    fee = 5
    def deposit_fee(self):
        self.withdraw(self.fee)

class WithdrawCharge(object):
    fee = 3
    def withdraw_fee(self):
        self.withdraw(self.fee)
```

12

# Multiple Inheritance:

Now we create a new account type:

multiple inheritance

```
class MoreEvilerAccount(EvilAccount, DepositCharge
                        WithdrawCharge):
    def deposit(self, amt):
        self.deposit_fee()
        super().deposit(amt)

    def withdraw(self, amt):
        self.withdraw_fee()
        super().withdraw(amt)
```

13

# Multiple Inheritance:

Let's use these new classes:

```
d = MoreEvilerAccount('Tim', 100000000, 1.2)
d.deposit(10)
d.inquiry()
```

deposit method of MoreEvilerAccount: calls deposit_fee method of DepositCharge class

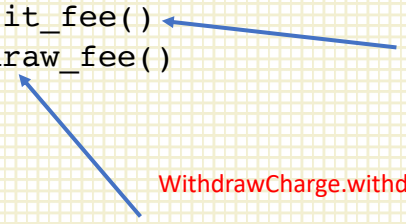
inquiry method of EvilAccount

14

# Hmm, this is subtle:

What's going on here?:

```
d = MoreEvilerAccount('Tim', 100000000, 1.2)
d.deposit_fee()
d.withdraw_fee()
```







$5 !!?

15

# Explanation

- fee is a class variable. It is defined twice: once in DepositCharge and once in WithdrawCharge.
- The value in DepositCharge was used in both calls. Why? Because the order of classes listed in the class definition for MoreEvilerAccount defines a priority. DepositCharge is listed before WithdrawCharge so it has higher priority.

16

# Polymorphism

## What is it?

Within the context of inheritance, ability to use an instance without regard to its type.

## Why do we need it?

- To simplify implementations of subclasses: there is no need to override every attribute of superclass A in subclass B since instances of B can directly access attributes in A.
- Allows passing object of subclass type B as a parameter when an object of superclass type A is expected (in general – doesn't apply to Python)

17

# Polymorphism

## How does it work?

It is handled entirely by the attribute lookup process.

Consider class `C` that is a subclass of `B`. `B` is, in turn, a subclass of `A`. Let `c` be an instance of `C`. In a reference to `c.attr`, `attr` is searched for in the following, in this order:

- instance `c`
- class `C`
- class `B`
- class `A`
- class `object`

18