Lecture 15: Inheritance

2/27/2015
Guest Lecturer: Marvin Zhang

Some (a lot of) material from these slides was borrowed from John DeNero.
Announcements
Announcements

• Homework 5 due Wednesday 3/4 @ 11:59pm
Announcements

• Homework 5 due Wednesday 3/4 @ 11:59pm
• Project 3 due Thursday 3/12 @ 11:59pm
Announcements

• Homework 5 due Wednesday 3/4 @ 11:59pm
• Project 3 due Thursday 3/12 @ 11:59pm
• Midterm 2 on Thursday 3/19 7pm-9pm
Announcements

• Homework 5 due Wednesday 3/4 @ 11:59pm
• Project 3 due Thursday 3/12 @ 11:59pm
• Midterm 2 on Thursday 3/19 7pm-9pm
• Quiz 2 released Wednesday 3/4
Announcements

- Homework 5 due Wednesday 3/4 @ 11:59pm
- Project 3 due Thursday 3/12 @ 11:59pm
- Midterm 2 on Thursday 3/19 7pm-9pm
- Quiz 2 released Wednesday 3/4
  - Due Thursday 3/5 @ 11:59pm
Announcements

- Homework 5 due Wednesday 3/4 @ 11:59pm
- Project 3 due Thursday 3/12 @ 11:59pm
- Midterm 2 on Thursday 3/19 7pm-9pm
- Quiz 2 released Wednesday 3/4
  - Due Thursday 3/5 @ 11:59pm
  - Object-oriented programming
Announcements

• Homework 5 due Wednesday 3/4 @ 11:59pm
• Project 3 due Thursday 3/12 @ 11:59pm
• Midterm 2 on Thursday 3/19 7pm-9pm
• Quiz 2 released Wednesday 3/4
  • Due Thursday 3/5 @ 11:59pm
  • Object-oriented programming
  • Similar to homework 5
Announcements

• Homework 5 due Wednesday 3/4 @ 11:59pm
• Project 3 due Thursday 3/12 @ 11:59pm
• Midterm 2 on Thursday 3/19 7pm-9pm
• Quiz 2 released Wednesday 3/4
  • Due Thursday 3/5 @ 11:59pm
  • Object-oriented programming
  • Similar to homework 5
• Guerrilla section this Sunday 3/1 on mutation
Inheritance
Inheritance

• Powerful idea in Object-Oriented Programming
Inheritance

- Powerful idea in Object-Oriented Programming
- Way of *relating* similar classes together
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of \textit{relating} similar classes together

• Common use: a \textit{specialized} class inherits from a more \textit{general} class
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of relating similar classes together

• Common use: a specialized class inherits from a more general class

```python
class <new class>(<base class>):
...
```
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of relating similar classes together

• Common use: a specialized class inherits from a more general class

```python
class <new class>(<base class>):
    ...
```

• The new class shares attributes with the base class, and overrides certain attributes
Inheritance

• Powerful idea in Object-Oriented Programming

• Way of relating similar classes together

• Common use: a specialized class inherits from a more general class

  class <new class>(<base class>):
    ...

• The new class shares attributes with the base class, and overrides certain attributes

• Implementing the new class is now as simple as specifying how it’s different from the base class
Inheritance Example
Inheritance Example

class Account:
    """A bank account."""
    ...

Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  
  • an account holder
Inheritance Example

class Account:
    """A bank account.""
    ...

• Bank accounts have:
  • an account holder
  • a balance
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
      • an account holder
      • a balance
      • an interest rate of 2%
Inheritance Example

class Account:
  """A bank account."""
  ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
        • an account holder
        • a balance
        • an interest rate of 2%

    • You can:
        • deposit to an account
        • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
        • an account holder
        • a balance
        • an interest rate of 2%

    • You can:
        • deposit to an account
        • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

    • Checking accounts have:
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
      • an account holder
      • a balance
      • an interest rate of 2%

    • You can:
      • deposit to an account
      • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

    • Checking accounts have:
      • an account holder
Inheritance Example

class Account:
   """A bank account."""
   ...

   • Bank accounts have:
     • an account holder
     • a balance
     • an interest rate of 2%

   • You can:
     • deposit to an account
     • withdraw from an account

class CheckingAccount(Account):
   """A checking account."""
   ...

   • Checking accounts have:
     • an account holder
     • a balance
Inheritance Example

class Account:
    """A bank account."""
    ...

    • Bank accounts have:
        • an account holder
        • a balance
        • an interest rate of 2%

    • You can:
        • deposit to an account
        • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

    • Checking accounts have:
        • an account holder
        • a balance
        • an interest rate of 1%
Inheritance Example

class Account:
  """A bank account."""
  ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

class CheckingAccount(Account):
  """A checking account."""
  ...

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1
Inheritance Example

class Account:
    """A bank account."
    ...

- Bank accounts have:
  - an account holder
  - a balance
  - an interest rate of 2%

- You can:
  - deposit to an account
  - withdraw from an account

class CheckingAccount(Account):
    """A checking account."
    ...

- Checking accounts have:
  - an account holder
  - a balance
  - an interest rate of 1%
  - a withdraw fee of $1

- You can:
Inheritance Example

class Account:
    
    """A bank account."""

    ...

    • Bank accounts have:
        • an account holder
        • a balance
        • an interest rate of 2%

    • You can:
        • deposit to an account
        • withdraw from an account

class CheckingAccount(Account):
    
    """A checking account."""

    ...

    • Checking accounts have:
        • an account holder
        • a balance
        • an interest rate of 1%
        • a withdraw fee of $1

    • You can:
        • deposit to a checking account
Inheritance Example

```python
class Account:
    """A bank account."""
    ...

    • Bank accounts have:
        • an account holder
        • a balance
        • an interest rate of 2%

    • You can:
        • deposit to an account
        • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

    • Checking accounts have:
        • an account holder
        • a balance
        • an interest rate of 1%
        • a withdraw fee of $1

    • You can:
        • deposit to a checking account
        • withdraw from a checking account (but there’s a fee!)
```
Inheritance Example

```python
class Account:
    """A bank account."""
    ...
    
    - Bank accounts have:
      - an account holder
      - a balance
      - an interest rate of 2%

    - You can:
      - deposit to an account
      - withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...
    
    - Checking accounts have:
      - an account holder
      - a balance
      - an interest rate of 1%
      - a withdraw fee of $1

    - You can:
      - deposit to a checking account
      - withdraw from a checking account (but there’s a fee!)
```
Inheritance Example

class Account:
  """A bank account."""
  ...
  
  • Bank accounts have:
    • an account holder
    • a balance
    • an interest rate of 2%
  
  • You can:
    • deposit to an account
    • withdraw from an account

class CheckingAccount(Account):
  """A checking account."""
  ...
  
  • Checking accounts have:
    • an account holder
    • a balance
    • an interest rate of 1%
    • a withdraw fee of $1
  
  • You can:
    • deposit to a checking account
    • withdraw from a checking account (but there’s a fee!)
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1

• You can:
  • deposit to a checking account
  • withdraw from a checking account (but there’s a fee!)
Inheritance Example

class Account:
    """A bank account."""
    ...

• Bank accounts have:
  • an account holder
  • a balance
  • an interest rate of 2%

• You can:
  • deposit to an account
  • withdraw from an account

class CheckingAccount(Account):
    """A checking account."""
    ...

• Checking accounts have:
  • an account holder
  • a balance
  • an interest rate of 1%
  • a withdraw fee of $1

• You can:
  • deposit to a checking account
  • withdraw from a checking account
    (but there's a fee!)
Attribute Look Up
Attribute Look Up

To look up a name in a class:
To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

>>> tom = CheckingAccount('Tom')
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
```

```python
>>> tom.interest     # Found in CheckingAccount
0.01

>>> tom.deposit(20)  # Found in Account
20

>>> tom.withdraw(5)  # Found in CheckingAccount
14
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest
```

0.01
 Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest                # Found in CheckingAccount
0.01
>>> tom.deposit(20)
```

Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes *are not copied* into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
>>> tom.deposit(20)  # Found in Account
20
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
>>> tom.deposit(20)  # Found in Account
20
>>> tom.withdraw(5)
```
Attribute Look Up

To look up a name in a class:

1. If the name is in the attributes of the class, return the corresponding value

2. If not found, look up the name in the base class, if there is one

Base class attributes are not copied into subclasses!

```python
>>> tom = CheckingAccount('Tom')  # Account.__init__
>>> tom.interest  # Found in CheckingAccount
0.01
>>> tom.deposit(20)  # Found in Account
20
>>> tom.withdraw(5)  # Found in CheckingAccount
14
```
Designing for Inheritance
Designing for Inheritance

• Don’t repeat yourself! Use existing implementations
Designing for Inheritance

• Don’t repeat yourself! Use existing implementations

• Reuse overridden attributes by accessing them through the base class
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Look up attributes on instances if possible
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Look up attributes on instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

    def withdraw(self, amount):
        return Account.withdraw(
            self, amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations.
- Reuse overridden attributes by accessing them through the base class.
- Look up attributes on instances if possible.

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(
        self, amount + self.withdraw_fee)
```
Designing for Inheritance

• Don’t repeat yourself! Use existing implementations

• Reuse overridden attributes by accessing them through the base class

• Look up attributes on instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(self, amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Look up attributes on instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Look up attributes on instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(self, amount + self.withdraw_fee)
```
Designing for Inheritance

- Don’t repeat yourself! Use existing implementations
- Reuse overridden attributes by accessing them through the base class
- Look up attributes on instances if possible

```python
class CheckingAccount(Account):
    withdraw_fee = 1
    interest = 0.01

def withdraw(self, amount):
    return Account.withdraw(self, amount + self.withdraw_fee)
```
Inheritance vs Composition
Inheritance vs Composition

- Inheritance: relating two classes through specifying similarities and differences
Inheritance vs Composition

- Inheritance: relating two classes through specifying *similarities and differences*
  - Represents “is a” relationships, e.g. a checking account is a specific type of account
Inheritance vs Composition

- Inheritance: relating two classes through specifying similarities and differences
  - Represents “is a” relationships, e.g. a checking account is a specific type of account

- Composition: connecting two classes through their relationship to one another
Inheritance vs Composition

- Inheritance: relating two classes through specifying similarities and differences
  - Represents “is a” relationships, e.g. a checking account is a specific type of account

- Composition: connecting two classes through their relationship to one another
  - Represents “has a” relationships, e.g. a bank has a collection of bank accounts
Inheritance vs Composition (demo)

- Inheritance: relating two classes through specifying similarities and differences
  - Represents “is a” relationships, e.g. a checking account is a specific type of account

- Composition: connecting two classes through their relationship to one another
  - Represents “has a” relationships, e.g. a bank has a collection of bank accounts
Multiple Inheritance
Multiple Inheritance

• In Python, a class can inherit from multiple base classes
Multiple Inheritance

- In Python, a class can inherit from multiple base classes
- This exists in many *but not all* object-oriented languages
Multiple Inheritance

- In Python, a class can inherit from multiple base classes
- This exists in many but not all object-oriented languages
- This is a tricky and often dangerous subject, so proceed carefully!
Multiple Inheritance

- In Python, a class can inherit from multiple base classes

- This exists in many **but not all** object-oriented languages

- This is a tricky and often dangerous subject, so proceed carefully!

class SavingsAccount(Account):
    deposit_fee = 2
    def deposit(self, amount):
        return Account.deposit(
            self, amount - self.deposit_fee)
Multiple Inheritance Example
Multiple Inheritance Example

• Bank executive wants the following:
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
Multiple Inheritance Example

• Bank executive wants the following:
  • Low interest rate of 1%
  • $1 withdrawal fee
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
  - $1 withdrawal fee
  - $2 deposit fee
Multiple Inheritance Example

- Bank executive wants the following:
  - Low interest rate of 1%
  - $1 withdrawal fee
  - $2 deposit fee
  - A free dollar for opening the account!
Multiple Inheritance Example

• Bank executive wants the following:

  • Low interest rate of 1%
  • $1 withdrawal fee
  • $2 deposit fee
  • A free dollar for opening the account!

```python
class BestAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1  # best deal ever
```
Multiple Inheritance Example

Account

CheckingAccount

SavingsAccount

BestAccount
Multiple Inheritance Example

```python
>>> such_a_deal = BestAccount('Marvin')
```

```
Account

CheckingAccount    SavingsAccount

BestAccount
```
Multiple Inheritance Example

```python
good_deal = BestAccount('Marvin')
good_deal.balance  # instance attribute
1
```
Multiple Inheritance Example

```python
class Account:
    pass

class CheckingAccount(Account):
    pass

class SavingsAccount(Account):
    pass

class BestAccount(CheckingAccount, SavingsAccount):
    def __init__(self, name):
        self.name = name

such_a_deal = BestAccount('Marvin')

>>> such_a_deal.balance  # instance attribute
1

>>> such_a_deal.deposit(20)  # SavingsAccount
19
```
Multiple Inheritance Example

```python
>>> such_a_deal = BestAccount('Marvin')
>>> such_a_deal.balance  # instance attribute
1
>>> such_a_deal.deposit(20)  # SavingsAccount
19
>>> such_a_deal.withdraw(5)  # CheckingAccount
13
```
Complicated Inheritance
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

![Diagram showing genetic inheritance from Mom and Dad to You]
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

![Diagram showing inheritance from Mom and Dad through Grampa and Gramma]

- Mom
- Dad
- You
- Grampa
- Gramma
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

```
some guy  |  Gramma  |  Gramps  |  Grandpop  |  Grandmom
          |  Half Aunt |  Mom     |  Dad       |

You
```
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

![Family Tree Diagram]

- Some guy
- Gramma
- Gramps
- Grandpop
- Grandmom
- Half Aunt
- Mom
- Dad
- You
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

[Diagram showing relationships among family members, including Mom, Dad, Gramma, Gramps, Grandmom, Grandpop, some guy, some other guy, Double, Half, Aunt, Half Cousin, Double Half, Uncle, and Double Half Cousin.]
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.
Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

```
Mom
Aunt

Gramma
Gramps
Grandpop
Grandmom

some guy

some other guy
Double Half Aunt

Quadruple Half Cousin

You

Double Half Uncle
```

Complicated Inheritance

To show how complicated inheritance can be, let’s look at an analogy through biological inheritance.

Moral of the story: inheritance (especially multiple inheritance) is complicated and weird. Use it carefully!