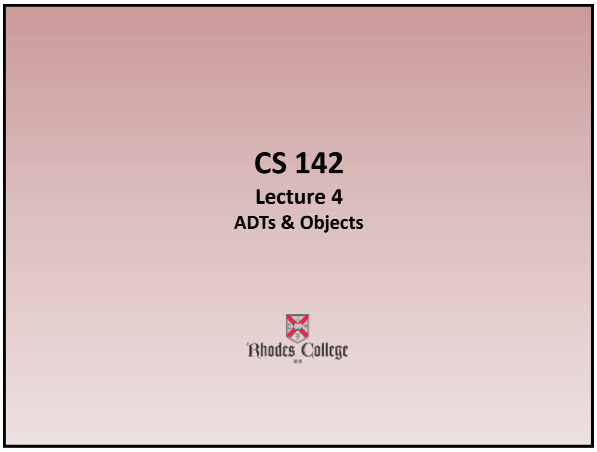
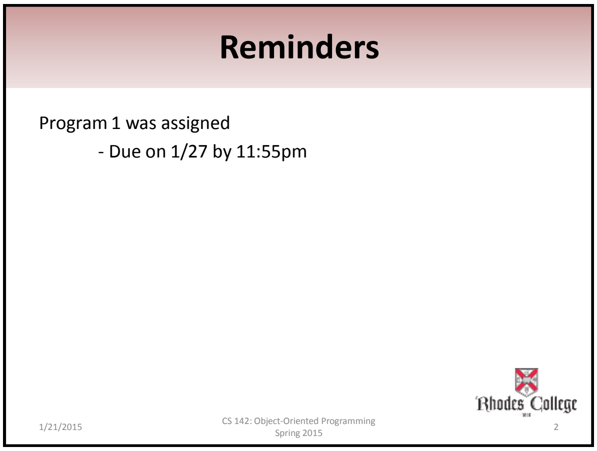


CS 142

Lecture 4

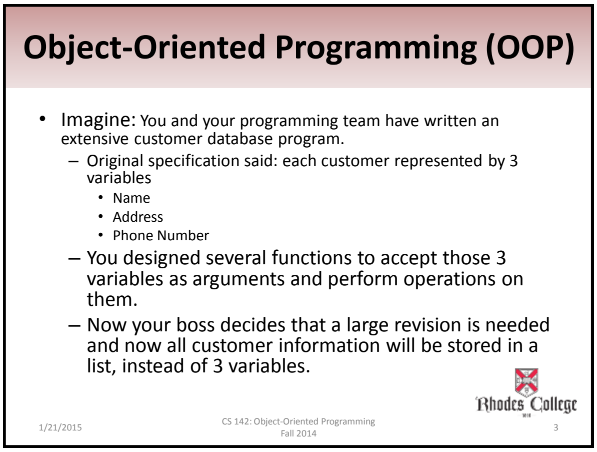
ADTs & Objects



Reminders

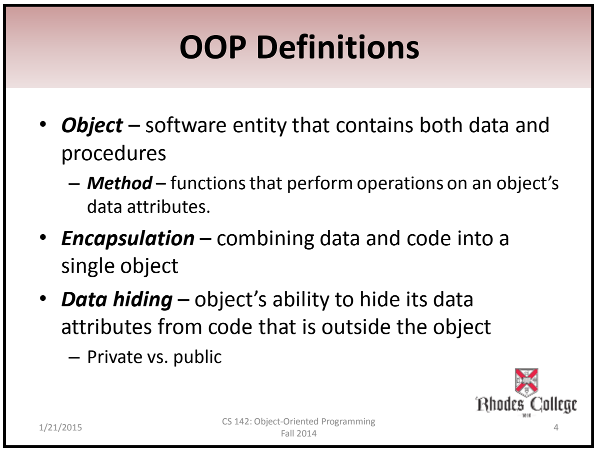
Program 1 was assigned

- Due on 1/27 by 11:55pm



Object-Oriented Programming (OOP)

- Imagine: You and your programming team have written an extensive customer database program.
  - Original specification said: each customer represented by 3 variables
    - Name
    - Address
    - Phone Number
  - You designed several functions to accept those 3 variables as arguments and perform operations on them.
  - Now your boss decides that a large revision is needed and now all customer information will be stored in a list, instead of 3 variables.



OOP Definitions

- ***Object*** – software entity that contains both data and procedures
  - ***Method*** – functions that perform operations on an object’s data attributes.
- ***Encapsulation*** – combining data and code into a single object
- ***Data hiding*** – object’s ability to hide its data attributes from code that is outside the object
  - Private vs. public

An Everyday Example of an Object

- Data attributes: define the state of an object
  - Example: clock object would have `second`, `minute`, and `hour` data attributes
- Public methods: allow external code to manipulate the object
  - Example: `set_time, set_alarm_time`
- Private methods: used for object's inner workings
  - Example: `increment_hour, increment_minute`

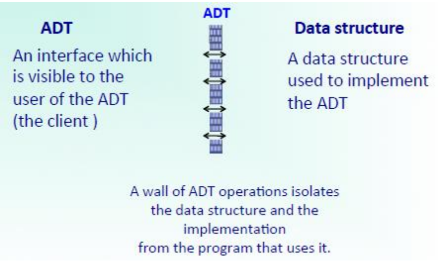
Rhodes College

Abstract Data Types (ADTs)

- Programming languages provide:
  - Some concrete data types
    - integers, characters, arrays,…
  - Some abstract data types
    - floating point, lists, tables, two dimensional arrays, records,…
  - Abstract data types are implemented using concrete datatypes
    - (but you don't need to know this to use them)
- Operations are provided for each datatype
  - e.g. creation, assignment, etc.
  - … but you cannot muck around with the internal representations
    - e.g. float is represented in two parts, but you cannot access these directly
  - But: some languages do allow you access to the internal representations
    - e.g. in C, you can use pointers to access the internals of arrays
    - this removes the distinction between the abstraction and the implementation
    - it destroys most of the benefits of abstraction
    - it causes confusion and error

Rhodes College

An ADT is Not a Data Structure


ADT
An interface which is visible to the user of the ADT (the client)
ADT
Data structure
A data structure used to implement the ADT

A wall of ADT operations isolates the data structure and the implementation from the program that uses it.

Rhodes College

Disadvantages/Advantages of an ADT

Disadvantages

initially there is more to consider, design issues, code to write and maintain, overhead of calling a method to access ADT information, greater initial time investment.

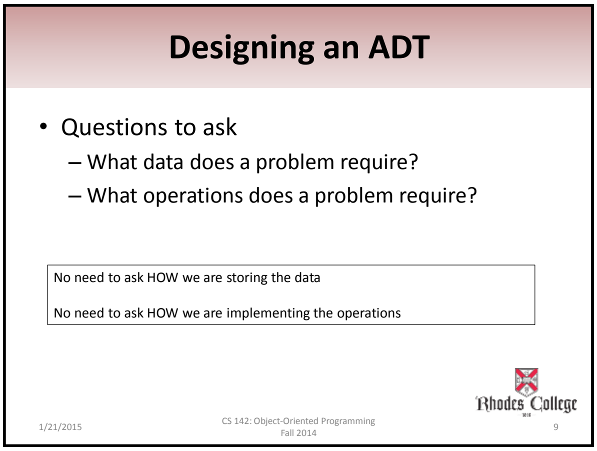
Advantages

A **client** (the application using the ADT) doesn't need to know about the implementation,

Maintenance of the application is easier,

The programmer can focus on problem solving and not worry about the implementation.

Rhodes College



Designing an ADT

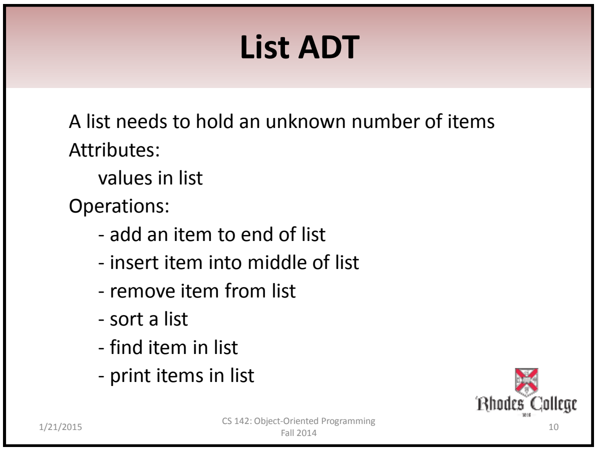
- Questions to ask
  - What data does a problem require?
  - What operations does a problem require?

No need to ask HOW we are storing the data

No need to ask HOW we are implementing the operations

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 9

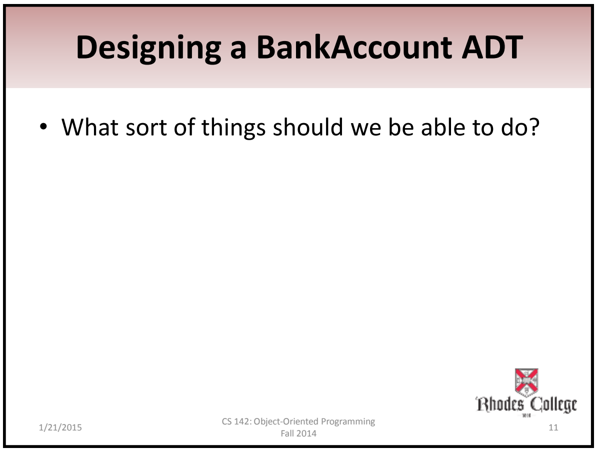


List ADT

A list needs to hold an unknown number of items
Attributes:
values in list
Operations:
- add an item to end of list
- insert item into middle of list
- remove item from list
- sort a list
- find item in list
- print items in list

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 10

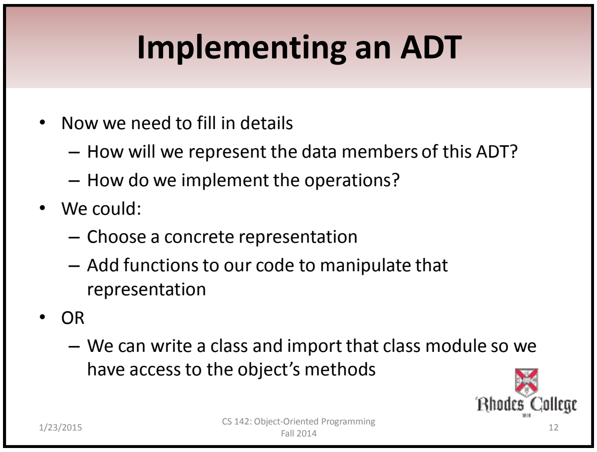


Designing a BankAccount ADT

- What sort of things should we be able to do?

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 11

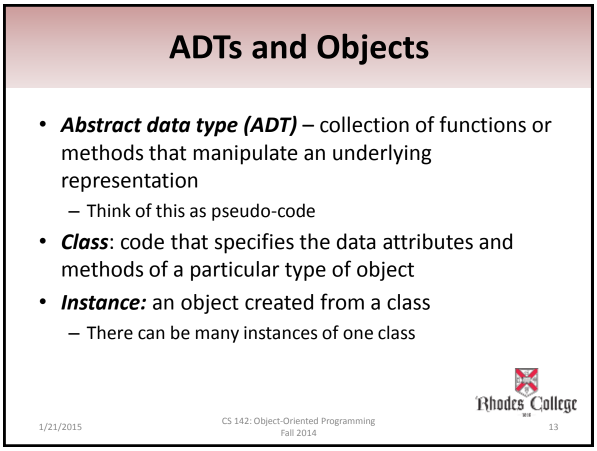


Implementing an ADT

- Now we need to fill in details
  - How will we represent the data members of this ADT?
  - How do we implement the operations?
- We could:
  - Choose a concrete representation
  - Add functions to our code to manipulate that representation
- OR
  - We can write a class and import that class module so we have access to the object’s methods

Rhodes College

1/23/2015 CS 142: Object-Oriented Programming Fall 2014 12

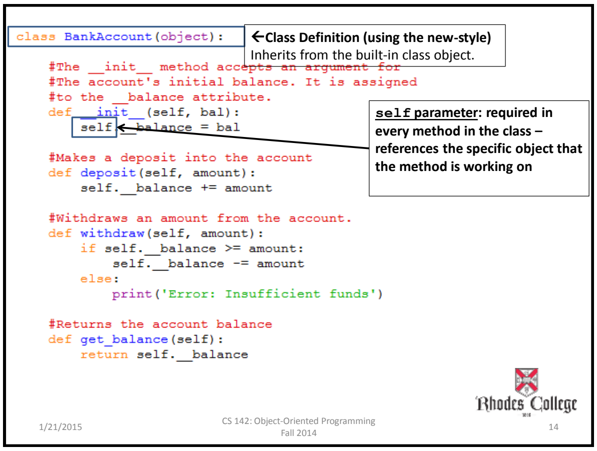


ADTs and Objects

- ***Abstract data type (ADT)*** – collection of functions or methods that manipulate an underlying representation
  - Think of this as pseudo-code
- ***Class***: code that specifies the data attributes and methods of a particular type of object
- ***Instance:*** an object created from a class
  - There can be many instances of one class

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 13



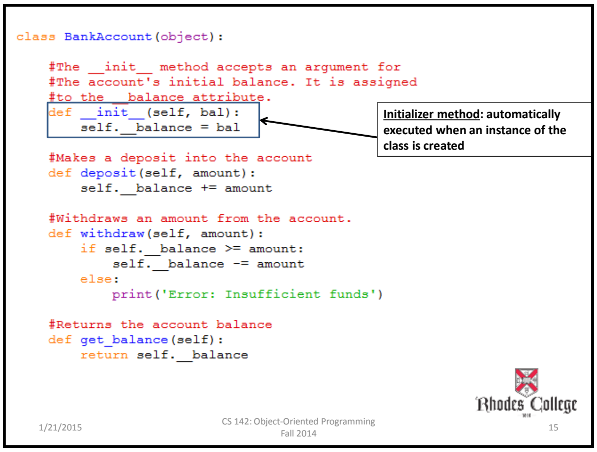
```
class BankAccount(object):

    #The __init__ method accepts an argument for
    #The account's initial balance. It is assigned
    #to the __balance attribute.
    def __init__(self, bal):
        self.__balance = bal

    #Makes a deposit into the account
    def deposit(self, amount):
        self.__balance += amount

    #Withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    #Returns the account balance
    def get_balance(self):
        return self.__balance
```

←Class Definition (using the new-style) Inherits from the built-in class object.

<u>`self` parameter</u>: required in every method in the class – references the specific object that the method is working on

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 14

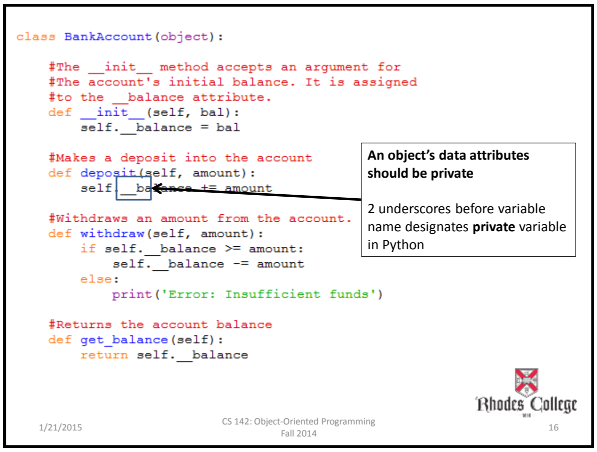


```
class BankAccount(object):

    #The __init__ method accepts an argument for
    #The account's initial balance. It is assigned
    #to the __balance attribute.
    def __init__(self, bal):
        self.__balance = bal

    #Makes a deposit into the account
    def deposit(self, amount):
        self.__balance += amount

    #Withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    #Returns the account balance
    def get_balance(self):
        return self.__balance
```

<u>Initializer method</u>: automatically executed when an instance of the class is created

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 15



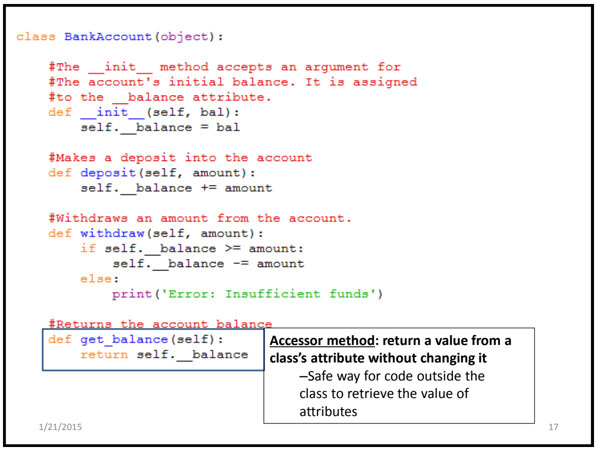
```
class BankAccount(object):

    #The __init__ method accepts an argument for
    #The account's initial balance. It is assigned
    #to the __balance attribute.
    def __init__(self, bal):
        self.__balance = bal

    #Makes a deposit into the account
    def deposit(self, amount):
        self.__balance += amount

    #Withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    #Returns the account balance
    def get_balance(self):
        return self.__balance
```

An object's data attributes should be private

2 underscores before variable name designates **private** variable in Python

Rhodes College

1/21/2015 CS 142: Object-Oriented Programming Fall 2014 16

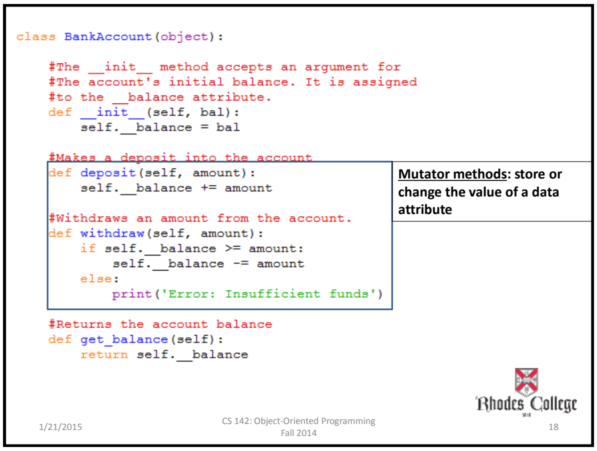


```
class BankAccount(object):

    #The __init__ method accepts an argument for
    #The account's initial balance. It is assigned
    #to the __balance attribute.
    def __init__(self, bal):
        self.__balance = bal

    #Makes a deposit into the account
    def deposit(self, amount):
        self.__balance += amount

    #Withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    #Returns the account balance
    def get_balance(self):
        return self.__balance
```

<u>Accessor method</u>: return a value from a class's attribute without changing it

–Safe way for code outside the class to retrieve the value of attributes

1/21/2015 17



```
class BankAccount(object):

    #The __init__ method accepts an argument for
    #The account's initial balance. It is assigned
    #to the __balance attribute.
    def __init__(self, bal):
        self.__balance = bal

    #Makes a deposit into the account
    def deposit(self, amount):
        self.__balance += amount

    #Withdraws an amount from the account.
    def withdraw(self, amount):
        if self.__balance >= amount:
            self.__balance -= amount
        else:
            print('Error: Insufficient funds')

    #Returns the account balance
    def get_balance(self):
        return self.__balance
```

<u>Mutator methods</u>: store or change the value of a data attribute

1/21/2015

CS 142: Object-Oriented Programming
Fall 2014

18