Objectives

- Designing our own classes
  - Representing attributes/data
  - What functionality to provide
- Using our defined classes

Where We Are

- With what you now know (OO programming)
  - Opens up the possibilities for what you kinds of programs you can write
  - Just about anything computational is possible
- Example: Car
  - Data to model for a Car?
  - API for a Car?

Review: Classes and Objects

- Car class
- Each car has these **attributes**:
  - Make
  - Model
  - Year
  - Transmission
  - Exterior color

Cars all have these attributes, different values for the attributes

- Methods
  - getYear()
  - setGear()
  - ...

Each car is an **instance of** the Car class

Review: Classes and Objects

```
c1 = Card(14, "spades")
c2 = Card(13, "hearts")
```

Object c1 of type Card
_rank = 14
_suit = "spades"

Object c2 of type Card
_rank = 13
_suit = "hearts"

Instance variables, attributes, or fields

c1 and c2 are **instances of** the Card class

Instance variables: named beginning with _

Card Class (Incomplete)

Doc String

```
class Card:
    """ A class to represent a standard playing card.
    The ranks are ints: 2-10 for numbered cards, 11=Jack,
12=Queen, 13=King, 14=Ace.
    The suits are strings: 'clubs', 'spades', 'hearts',
'diamonds'."""
    def __init__(self, rank, suit):
        """Constructor for class Card takes int rank and
             string suit."""
        self._rank = rank
        self._suit = suit

    def getRank(self):
        "Returns the card's rank."
        return self._rank

    def getSuit(self):
        "Returns the card's suit."
        return self._suit
```

Methods

Methods are like *functions* defined in a *class*

card.py

Defining the Constructor

- **__init__** method is like the ***constructor***
- In constructor, define ***instance variables***
  - **Data** contained in every object
  - Also called **attributes** or **fields**
- Constructor **never *returns*** anything

First parameter of *every* method is `self`
- pointer to the object that method acts on

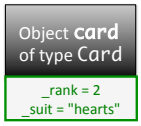
```
def __init__(self, rank, suit):
    """Constructor for class Card takes int rank
     and string suit."""
    self._rank = rank
    self._suit = suit
```

Instance variables

Using the Constructor

```
def __init__(self,
             rank, suit):
```

- As defined, constructor is called using `Card(<rank>,<suit>)`
  - Do not *pass* anything for the `self` parameter
  - Python handles for us, passing the parameter automatically
- Example:
  - `card = Card(2, "hearts")`
  - Creates a 2 of Hearts card
  - Python passes `card` as `self` for us



Accessor Methods

- Need to be able to get information about the object

```
def getRank(self):
    "Returns the card's rank."
    return self._rank

def getSuit(self):
    "Returns the card's suit."
    return self._suit
```

- Have `self` parameter
- Return data/ information

- These methods will get called as `card.getRank()` and `card.getSuit()`
  - Python plugs `card` in for `self`

Another Special Method: `__str__`

- Returns a *string* that describes the object
- Whenever you `print` an object, Python checks if the object's `__str__` method is defined
  - Prints result of calling `__str__` method
- `str(<object>)` also calls `__str__` method

```
def __str__(self):
    """Returns a string
       describing the card as
       'rank of suit'."""
    result = ""
    if self._rank == 11:
        result += "Jack"
    elif self._rank == 12:
        result += "Queen"
    elif self._rank == 13:
        result += "King"
    elif self._rank == 14:
        result += "Ace"
    else:
        result += str(self._rank)
    result += " of " + self._suit
    return result
```

self is a Card object

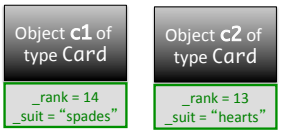
Using the `Card` Class

Invokes the `__str__` method

```
def main():
    c1 = Card(14, "spades")
    print(c1)
    c2 = Card(13, "hearts")
    print(c2)
```

Displays:

Ace of spades
King of hearts



Creating a Deck Class (Partial)

- List of Card objects

```
from card import *

class Deck:
    def __init__(self):
        self._listOfCards = []
        for suit in ["clubs","hearts","diamonds","spades"]:
            for rank in range(2,15):
                self._listOfCards.append(Card(rank, suit))

    def __str__(self):
        deckRep= ""
        for c in self._listOfCards:
            deckRep += str(c) + "\n"
        return deckRep
```

Initialize instance variable, `self._listOfCards`

Creates and returns a string

Displays cards on separate lines

Deck Class

- What does the Deck API look like so far?

Deck API

- `Deck()` Constructor
- `__str__()`

Deck API

- What additional methods should our Deck class provide?
- What do the method headers look like?
  - Deck's API
- What should they return?
- How do we implement them?

Deck API

- `Deck()` ← Constructor
- `shuffle()`
- `draw()`
- `deal(num_cards)`
- `numRemaining()`
- `isEmpty()`
- `__str__()`

Algorithm for Creating Classes

1. Identify need for a class
2. Identify state or attributes of a class/an object in that class
   - Write the constructor (`__init__`) and `__str__` methods
3. Identify methods the class should provide
   - How will a user call those methods (parameters, return values)?
     - Develop API
   - Implement methods

Review: Object-Oriented Programming

- Why do we want to define classes/new data types?
- What is the keyword to create a new class?
- How do you define a method?
  - What parameter is needed in every method?
- How do you create a new object of a given class?
  - What method does this call?
- How do we access instance variables in other methods?

Looking Ahead

- Lab 9: Analysis of student names at W&L
- Broader Issue