

As you arrive:

1. Start up your computer and plug it in
2. **Log into Angel** and go to CSSE 120
3. Do the **Attendance Widget** – the PIN is on the board
4. Go to the course **Schedule Page**
5. Open the **Slides** for today if you wish
6. Check out today's project:

Session18_BlackJackWithClasses

Plus lots of in-class time to work on team project.

Defining classes part 2

- Blackjack with classes
- Defining and exploring the card class
- Integrating other classes
- Object interaction

Project work:

Work in your team to complete next milestone

**Troubles getting
today's project?**

If so: →

Are you in the Pydev perspective? If not:

- **Window ~ Open Perspective ~ Other**
then **Pydev**

Messed up views? If so:

- **Window ~ Reset Perspective**

No SVN repositories view (tab)? If it is not there:

- **Window ~ Show View ~ Other**
then **SVN ~ SVN Repositories**

In your SVN repositories view (tab), expand your repository (the top-level item) if not already expanded.

- If no repository, perhaps you are in the wrong Workspace. Get help as needed.

Right-click on today's project, then select *Checkout*.
Press OK as needed.

The project shows up in the

Pydev Package Explorer

to the right. Expand and browse the modules under **src** as desired.

Review of Key Ideas

- **Constructor:**
 - ▣ Defined with special name `__init__`
 - ▣ Called like `ClassName()`
- **Instance variables:**
 - ▣ Created when we assign to them
 - ▣ Live as long as the object lives
- **`self` formal parameter:**
 - ▣ Implicitly get the value *before the dot* in the call
 - ▣ Allows an object to "talk about itself" in a method

Creating Custom Objects: Defining Your Own Classes

- Custom objects:
 - ▣ Hide complexity
 - ▣ Provide another way to break problems into pieces
 - ▣ Make it easier to pass information around
- Example: Cards, Decks, and Hands

- ▣ Recall from BlackJack:

```
suits = ['Clubs', 'Diamonds', 'Hearts', 'Spades']  
cardNames = ['Ace', 'Deuce', '3', '4', '5', '6', '7',  
             '8', '9', '10', 'Jack', 'Queen', 'King']
```

Code to Define a Class

Declares a class
named Card

```
class Card:
```

```
    """This class represents a card from a standard deck."""
```

docstring
describes class,
used by help()
function and by
Eclipse help

Code to Define a Class

Special name, `__init__`
declares a **constructor**

```
class Card:
```

```
    """This class represents a card for"""
```

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

```
        self.cardName = card
```

```
        self.suitName = suit
```

Special **self** parameter
is the first formal
parameter of each
method in a class.
self always refers to
the current object

Create instance variables just
by assigning to them

c

Card

cardName _____

suitName _____

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

```
    self.cardName = card
```

```
    self.suitName = suit
```

A sample constructor call:

c = Card('Ace', 'Hearts')

'Ace'

'Hearts'

Code to Define a Class

```
class Card:
```

```
    """This class represents a card from a standard deck."""
```

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

```
        self.cardName = card
```

```
        self.suitName = suit
```

self parameter again, no other formal parameters

docstring for method

```
    def getValue(self):
```

```
        """Returns the value of this card in BlackJack.
```

```
        Aces always count as one, so hands need to adjust to count aces as 11."""
```

```
        pos = cardNames.index(self.cardName)
```

```
        if pos < 10:
```

```
            return pos + 1
```

```
        return 10
```

use `self.<varName>` to read instance variable

A sample method call:
c.getValue()

Card...

Code to Define a Class

```
class Card:
```

```
    """This class represents a card from a standard deck."""
```

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

```
        self.cardName = card
```

```
        self.suitName = suit
```

```
    def getValue(self):
```

```
        """Returns the value of this card in BlackJack.
```

```
        Aces always count as one, so hands need to adjust  
        to count aces as 11."""
```

```
        pos = cardNames.index(self.cardName)
```

```
        if pos < 10:
```

```
            return pos + 1
```

```
        return 10
```

```
    def __str__(self):
```


```
        return self.cardName + " of " + self.suitName
```

Sample uses of `__str__` method:

```
print c
```

```
msg = "Card is " + str(c)
```

Special `__str__` method returns
a string representation of an object



Stepping Through Some Code

Sample use:

```
card = Card('7', 'Clubs')
print card.getValue()
print card
```

```
class Card:
```

```
    """This class represents a card
```

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

```
    self.cardName = card
```

```
    self.suitName = suit
```

```
def getValue(self):
```

```
    """Returns the value of this card in BlackJack.
```

```
    Aces always count as one, so hands need to adjust  
    to count aces as 11."""
```

```
    pos = cardNames.index(self.cardName)
```

```
    if pos < 10:
```

```
        return pos + 1
```

```
    return 10
```

```
def __str__(self):
```

```
    return self.cardName + " of " + self.suitName
```

Another Example:

Lists of Objects, Lists in Object

```
class CardCollection:
```

```
    """This class represents a collection of cards, either a  
    single hand or the whole deck."""
```

```
    def __init__(self, newDeck = False):
```

```
        """Creates a new collect  
        it's empty, but if newDe  
        collection is a full sta
```

```
        self.name = None
```

```
        self.cardList = []
```

```
        self.hideFirst = True
```

```
        if newDeck:
```

```
            # Create an entire deck of cards
```

```
            for s in suits:
```

```
                for c in cardNames:
```

```
                    self.insert(Card(c, s))
```

Instance variable
containing a list

...

Optional formal parameter, can
construct a CardCollection three ways:

```
deck = CardCollection(True)
```

```
hand = CardCollection(False)
```

```
hand = CardCollection()
```

Self call, constructor calls another
method of the CardCollection class

Another Example (continued): Lists of Objects, Lists in Object

```
class CardCollection:
```

```
    """..."""
```

```
    def __init__(self, newDeck = False):
```

```
        """..."""
```

```
        self.name = None
```

```
        self.cardList = []
```

```
        self.hideFirst = True
```

```
        if newDeck:
```

```
            # Create an entire deck of cards
```

```
            for s in suits:
```

```
                for c in cardNames:
```

```
                    self.insert(Card(c, s))
```

```
    def insert(self, card)
```

```
        """Adds the given card to this collection."""
```

```
        self.cardList.append(card)
```

insert method uses append()
method to mutate the list instance
variable

We can put objects into lists.

Why not just use a list of cards?

```
class CardCollection:
```

```
...
```

```
def getValue(self):
```

```
    """Returns the best score for this collection  
    in the game of BlackJack."""
```

```
    score = 0
```

```
    hasAce = False
```

```
    for card in self.cardList:
```

```
        val = card.getValue()
```

```
        score += val
```

```
        if val == 1:
```

```
            hasAce = True
```

```
    if score <= winningScore - 10 and hasAce:
```


```
        score += 10
```

```
    return score
```

Iterating over the list



Calling methods on
the objects stored in
the list



Work on your team project

- Meet with your project team
 - Finish up what is due for session 18 milestone
 - Continue working on next milestone
 - Decide on time/venue for next meeting