DEFINING CLASSES

CSSE 120—Rose Hulman Institute of Technology

Review: Using Objects

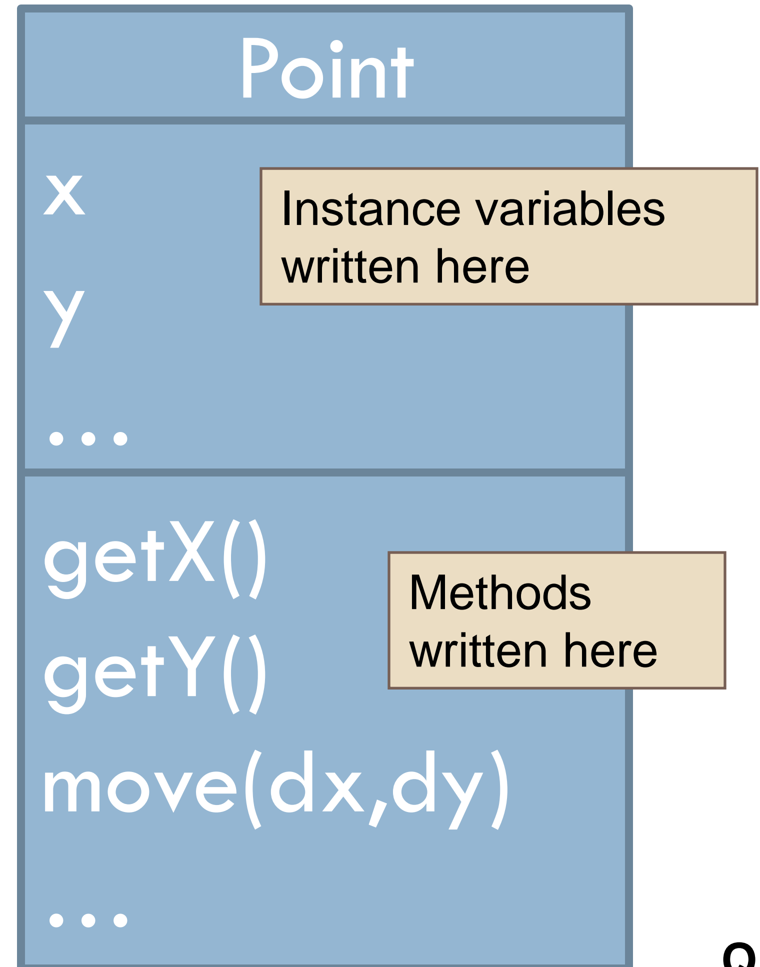
```
WIDTH = 400
HEIGHT = 50
REPEAT_COUNT = 20
PAUSE_LENGTH = 0.25
win = GraphWin('Giants Win!', WIDTH, HEIGHT)
p = Point(WIDTH/2, HEIGHT/2)
t = Text(p, 'NY Giants—2008 Super Bowl Champs!')
t.setStyle('bold')
t.draw(win)
nextColorIsRed = True
t.setFill('blue')
for i in range(REPEAT_COUNT):
    sleep(PAUSE_LENGTH)
    if nextColorIsRed:
        t.setFill('red')
    else:
        t.setFill('blue')
    nextColorIsRed = not nextColorIsRed
win.close()
```

Q1

Object Terminology

- Objects are *data types* that might be considered *active*
  - They **store information** in *instance variables*
  - They **manipulate their data** through *methods*
- Objects are *instances* of some *class*
- Objects are created by calling *constructors*

- UML class diagram:



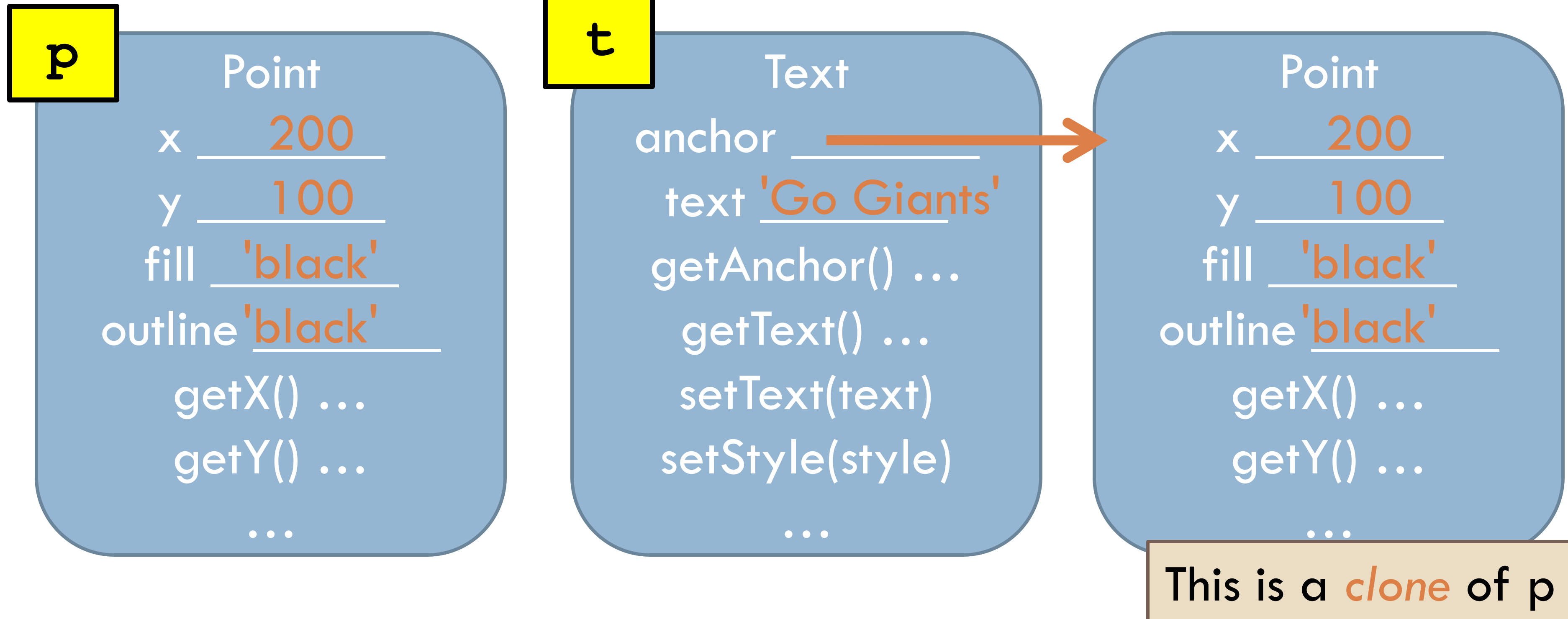
Q2

Key Concept!

- A class is an "object factory"
  - Calling the constructor tells the classes to make a new object
  - Parameters to constructor are like "factory options", used to set instance variables
- Or think of class like a "rubber stamp"
  - Calling the constructor stamps out a new object shaped like the class
  - Parameters to constructor "fill in the blanks". That is, they are used to initialize instance variables.

Examples

- `p = Point(200, 100)`
- `t = Text(p, 'Go Giants!')`



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

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

Declares a class named Card

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 fr
    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

    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
```

self parameter again, no other formal parameters

docstring for method

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

A sample method call:

c.getValue()

Card…

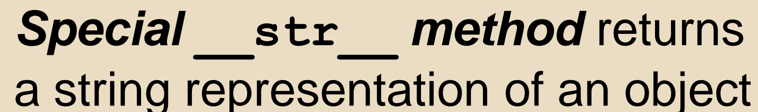
Q5

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)
```



Stepping Through Some Code

```
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
```

Sample use:

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

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

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))
  ...
```

Optional formal parameter, can construct a CardCollection three ways:

```
deck = CardCollection(True)
hand = CardCollection(False)
hand = CardCollection()
```

Instance variable containing a list

Self call, constructor calls another method of the CardCollection class

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

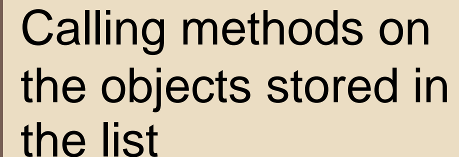
```
class CardCollection:
    """…"""
    def __init__(self, newDeck = False):
        """…"""
        self.name = None
        self.cardList = []
        self.hideFirst = True
        if newDeck:
            # Create an entire deck of cards
            for s in suits:
                for c in cardNames:
                    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
```



Encapsulation

- Insulating the rest of a program from some details by "hiding" them inside some structure
  - Top-down design encapsulates ***operations*** inside functions
  - Object-oriented design encapsulates ***data and operations*** inside classes
- Encapsulation is a form of abstraction—ignoring irrelevant details

Check It Out

- Use Eclipse *SVN Repository Exploring* perspective
  - Go to your personal repository: **http://svn.cs.rose-hulman.edu/repos/csse120-200920-*username***
  - Check-out the **BallSim** project
- Switch back to the *PyDev* perspective
- The **blackjackWithClasses** module in the Session15 project will serve as a great example of defining classes

That's the Way the Ball Bounces!

- We're going to begin working with defining classes by creating a simulation of an ideal gas
- Like simulating billiard balls on a frictionless table
- We hope this will be a straightforward and enjoyable assignment
- Begin by opening the instructions from ANGEL:
  - Lessons → Homework → Homework 15
- Try to get to step 4f in class.
  GET HELP IF YOU GET STUCK!