As you arrive:

1. Start up your computer and plug it in.
2. ***Log into Angel*** and go to CSSE 120. Do the ***Attendance Widget*** – the PIN is on the board.
3. Go to the ***Course Schedule*** web page. Open the ***Slides*** for today if you wish.
4. Checkout today's project: `Session05_NumbersObjectsAndGraphics`

Session 5

Numbers, Objects and Graphics

Review

- Counted loops
  - FOR loops with RANGE expressions
- Accumulator loops

Objects

- Why, what, using them:
  - Constructing objects
  - Applying member functions
  - Accessing instance variable
  - Graphics (zellegraphics)

Checkout today's project:

`Session05_NumbersObjectsAndGraphics`

Are you in the Pydev perspective? If not:

`Window ~ Open Perspective ~ Other` then `Pydev`

Messed up views? If so:

`Window ~ Reset Perspective`

Troubles getting today's project? If so:

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

`Window ~ Show View ~ Other`
then `SVN ~ SVN Repositories`

1. ***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.
2. ***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.

Outline of today's session

Checkout today's project:

`Session05_NumbersObjectsAndGraphics`

- ***Review***
  - Loops: Counted loops. Accumulator loops.
- ***Numbers***
  - Integers versus Floating Point
- ***Objects***
  - What, why
  - Using objects:
    - Constructing objects
    - Applying methods to objects
    - Referencing instance variables (aka fields) of objects
  - UML object diagrams
- ***Graphics – Zellegraphics***

Numbers

- Integers
  - Infinite precision
- Floating point
  - Finite precision
  - Thus subject to roundoff error
- With your instructor, examine and run `m1_numbers.py`

Q1-5

Some Numeric Operations

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Remainder
//	Do integer division (even on floats)

Function	Operation
abs(x)	Absolute value of x
round(x, y)	Round x to y decimal places
int(x)	Convert x to the int data type
float(x)	Convert x to the float data type

Q6

Review – Counted Loops

A counted loop. The `range` statement makes `k` take on values 0, 1, 2, …. 9

```
for k in range(10):
    a = 0
    b = 0
    print("{:1} {:3} {:3}".format(k, a, b))
```

Does formatted printing. The three items printed (`k,a,b`) are printed in fields of widths 1, 3 and 3, respectively.

We'll learn more about formatted printing later.

Q7

What are objects?

- ***Traditional*** view, in languages like C
  - Data types are *passive*
    - They have values
    - There are operations that act on the data types
      - The data type itself cannot do anything
- ***Object-oriented*** view, in languages like Python (and most other modern languages)
  - Have *objects*, which are *active* data types. Objects:
    - ***Know stuff – they contain data***
      - The data that an object holds are its ***instance variables*** (aka ***fields***)
    - ***Can do stuff – they can initiate operations***
      - The operations that an object can do are its ***methods***

Q8

Traditional, non-object-oriented, design

- Break the problem into subproblems. That is:
  - To solve the problem I need to do: A, B, C, …
    - To solve A, I need to do: A1, A2, A3, …
      - To solve A1, I need to do A1a, A1b, A1c, …
      - To solve A2, I need to do A2a, A2b, A2c, …
        - etc
    - To solve B, I need to do: B1, B2, B3, …
    - etc, until the units are so small that you can just *do* them
- The units become ***functions***
- This process is called ***procedural decomposition***

Q9

Modern, *object-oriented*, design

- Basic idea of object-oriented (OO) development
  - *View a complex system as interaction of simple objects*
- In doing OO development, ask:
  1. What ***things*** (**objects**) are involved in the solution to my problem?

     The *types* of those things become our ***classes***

  2. For each type of thing (i.e., each *class*), what ***responsibilities*** does it have?

     What can it do? E.g. *A list can append stuff to itself.*

     These responsibilities become the *methods* of that class: `append`

  3. To carry out those responsibilities:
     a. What other objects does it need help from? *Relationships between classes*
     b. What objects does it have within? Become the *instance variables* of the class.

These *things* often come from *nouns* in the problem description, e.g.
single concepts *visual elements*
abstractions of real-life entities
actors utilities

These *responsibilities* often come from *verbs* in the problem description

Q10-12

Why is the *object-oriented* view useful?

- Procedural decomposition is useful and forms an important part of OO design
- But for complex systems, we often find it easier to think about the complex system as the interaction of simple objects than to just "break it down into its parts"
- In practice, most complex software systems today are designed using OO design

How do you *use* objects?

Recall that objects:
- Know stuff (***instance variables***, aka ***fields***)
- Can do stuff (***methods***)

- To ***construct*** an object:

```
win = zg.GraphWin()
point1 = zg.Point(500, 450)
line = zg.Line(point1, zg.Point(30, 40))
circle = zg.Circle(point1, 100)
```

Constructor:
- Call it like a function, using the name of the ***class***
- Style: Class names begin with an ***uppercase*** letter
- The constructor ***allocates space*** for the object and does whatever ***initialization*** the class specifies

- To *ask an object to do something*, i.e. to apply its ***methods*** to it:

```
point1.draw(window)
line.move(45, -60)
x = point1.getX()
center = circle.getCenter()
```

Method call:
- Use the ***dot notation:***

`Who.Does_What(With_What)`

Just like a function call, except that the method has access to the object invoking the method.
So the object is an ***implicit argument*** to the method call

- To reference what the object knows (its ***instance variables***, aka ***fields***):

```
point1.x    circle.p1    circle.p2
```

Instance variable (aka *field*) reference:
- Use the ***dot notation*** but ***without parentheses*** `Who.Has_What`

How do objects interact?

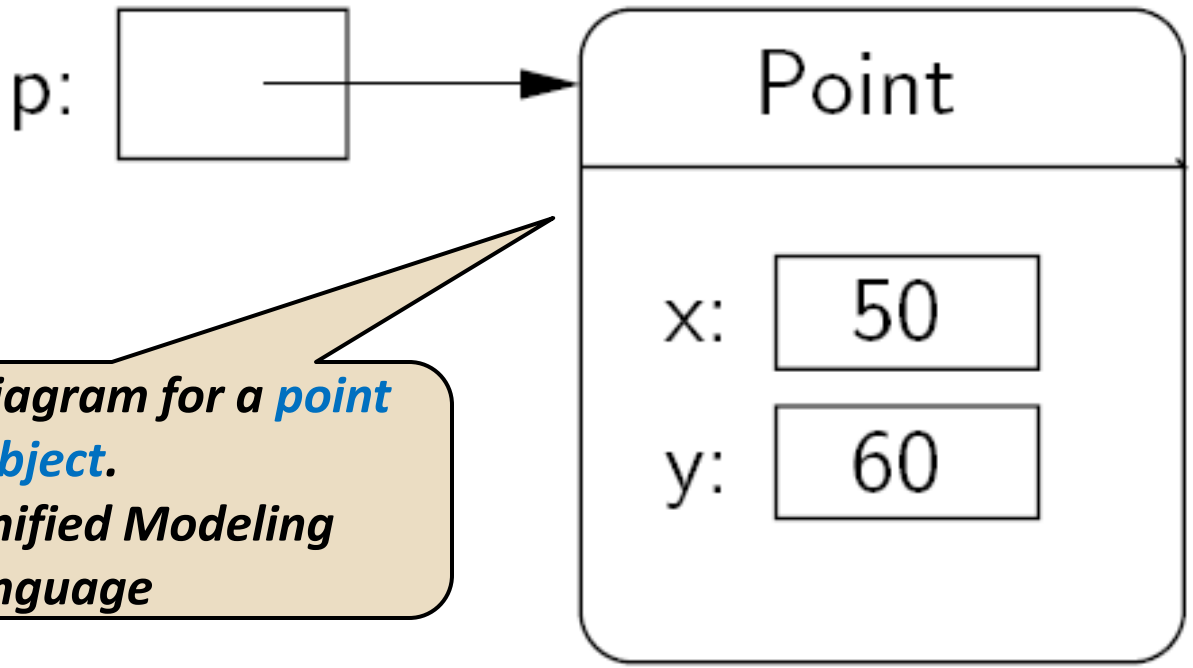
- Objects interact by sending each other messages
  - Message: request for object to perform one of its operations
  - Example: the brain can ask the feet to walk
  - In Python, messages happen *via* method calls.

```
window = zg.GraphWin()   # constructor
p = zg.Point(50, 60)     # constructor
p.getX()                 # accessor method
p.getY()                 # accessor method
p.draw(window)           # method
```

Q13-14

How do objects interact? Point

p = Point(50, 60)

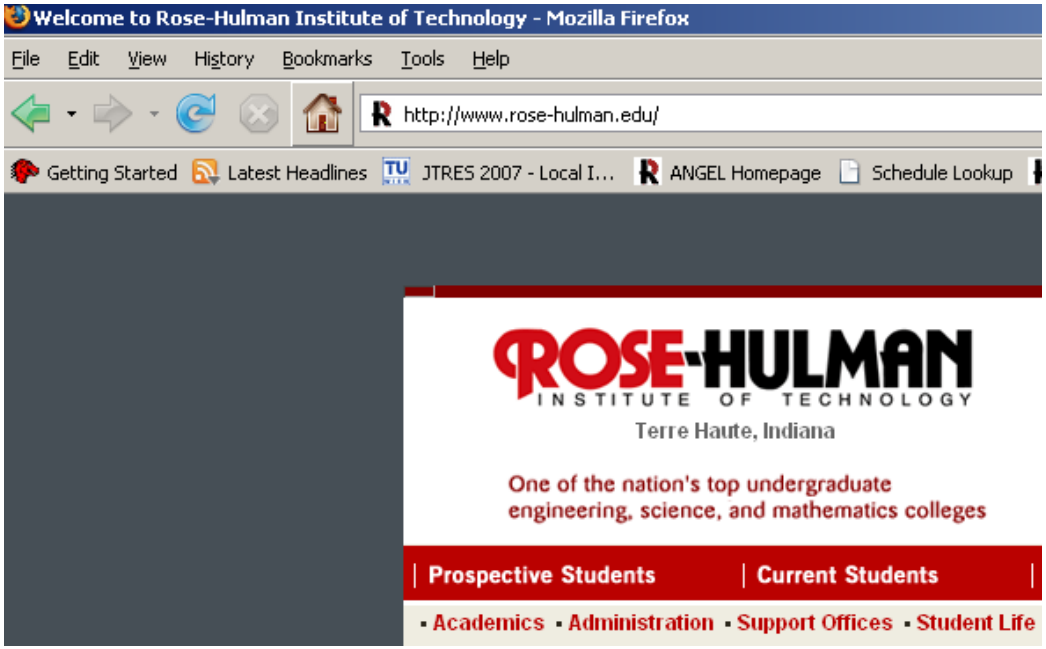


UML object diagram for a point object.
UML ➔ Unified Modeling Language

Q15-16

Simple graphics programming

- Graphics is fun and provides a great vehicle for learning about objects
- Computer Graphics: study of graphics programming
- Graphical User Interface (GUI)



Q17

Import

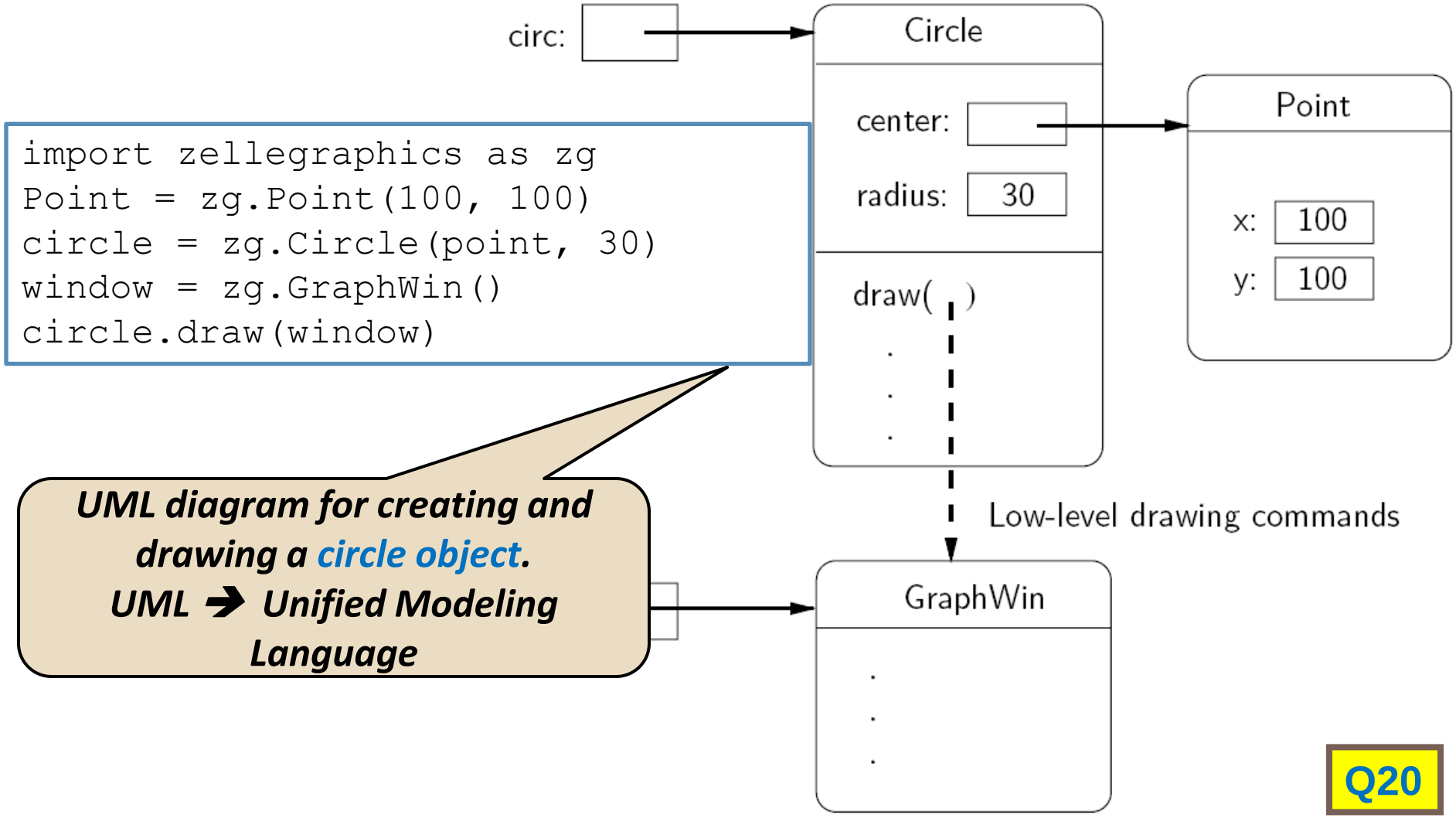
- Must import graphics library before accessing it
  - `import zellegraphics as zg`
  - `window = zg.GraphWin(...)`

Review: Class and object terminology

- Different types of objects
  - Point, Line, Rectangle, Oval, Text
  - These are examples of *classes*
- Different objects
  - head, leftEye, rightEye, mouth, message
  - Each is an *instance* of a class
  - Created using a *constructor*
  - Objects have *instance variables* (called *fields* in some languages)
  - Objects use *methods* to operate on instance variables
    - *Accessor methods* return data from the object

Q18-19

Object interaction to draw a circle



Q20

Interactive graphics

- *GUI*—Graphical User Interface
  - Accepts input
    - Keyboard, mouse clicks, menu, text box
  - Displays output
    - In graphical format
    - On-the-fly
- Developed using *Event-Driven Programming*
  - Program draws interface elements (*widgets*) and **waits**
  - Program responds when user does something

getMouse

- `win.getMouse()`
  - Causes the program to **pause, waiting** for the user to click with the mouse somewhere in the window
  - To find out where it was clicked, assign it to a variable:
    - `p = win.getMouse()`

Q21

Mouse Event Exercise

- Review **m2_objects_and_graphics_example.py** with your instructor
- Do **m3_click_me.py** with your instructor

Rest of Session

- ***Check your Quiz answers*** versus the solution
  - An assistant may check your Quiz to ensure you are using the Quizzes appropriately
- ***Work on today's homework***
  - Ask questions as needed!
- **Sources of help after class:**

```
CSSE lab: Moench F-217
          7 to 9 p.m.
Sundays thru Thursdays
```

  - **Assistants in the CSSE lab**
    - And other times as well (see link on the course home page)
  - **Email** `csse120-staff@rose-hulman.edu`
    - You get faster response from the above than from just your instructor