

**Activity 1.4.1 Procedural Abstraction**

Introduction

|  |  |
| --- | --- |
| A lot of practical, reusable, published code is object-oriented code. The API of an object-oriented library defines the attributes and methods for each class. Attributes are data, and methods are procedures. Procedures, since they are instructions coded in language, are just a special kind of data!  The method's instructions are stored once, when the class is defined. The instructions can be used over and over for the entire class of objects. "Calling a method on an object” tells the computer to execute the class' instructions (the method) for that kind of object (class), using the attribute data that is specific to that particular object (instance). | File:PaperTapes-5and8Hole.jpg |

For an example think about the procedure you have for crossing a street. It can be executed at any given street of a particular kind.

   

One of the world's three hundred Univac 1108s running Algol at Carnegie-Mellon University in 1976

Images courtesy of J. Chris Hausler

Procedure

1. Greet your partner to practice professional skills. Establish team norms specific to the materials in this activity.
2. Computing relies on layers of abstraction. In two previous activities, you focused on abstraction.

* Data abstraction lets us *ignore the details* of how numbers, letters, sound, and images are represented. We can *generalize* a data type.
* Programs are data, too! Procedural abstraction lets us *ignore the details* of how instructions are executed. We can *generalize* an instruction.

Recall one or more steps or ideas from past activities and share with your partner.

**Part I: The abstraction of objects, at a low layer of abstraction**

1. In this activity we climb up and down the ladder of abstraction with object-oriented programming. Before working from the top of the ladder, let's take a moment to dive down near the bottom of the ladder of abstraction to the layer where data is stored. In the early days, that was on magnetic or hole-punched tape.

Alan Kay, one of the giants of computer science, coined the term "object-oriented" to describe ideas he learned from Trygve Reenskaug, a Norwegian graduate student at the University of Utah. Writing years later, Kay recalls analyzing Reenskaug's program, which allowed multiple people to work simultaneously on a huge drawing.

I actually walked into Dave Evans’ office looking for a job and a desk. On Dave’s desk was a foot-high stack of brown covered documents, one of which he handed to me: “Take this and read it.”

Alan Kay, *The Early History of Smalltalk*

© 1993 ACM

It was 1966. Kay recalls how every new graduate student got one of those files. He describes looking inside Reenskaug's documentation, which explained where Trygve had left off:

The title was “Sketchpad: A man-machine graphical communication system,” by Ivan Sutherland in 1963. What it could do was quite remarkable, and completely foreign to any use of a computer I had ever encountered.

...Head whirling, I found my desk. On it was a pile of tapes and listings, and a note: “This is the Algol for the 1108. It doesn’t work. Please make it work.”

...The documentation was incomprehensible. Supposedly, this was the Case-Western Reserve 1107 Algol–but it had been doctored to make a language called Simula. The documentation read like Norwegian transliterated into English, which in fact it was. There were uses of words like activity and process that didn’t seem to coincide with normal English usage. Finally, another graduate student and I unrolled the program listing 80 feet down the hall and crawled over it yelling discoveries to each other.

--Alan Kay, © 1993 ACM

Alan Kay and his colleague learned from Trygve Reenskaug the idea of storing data to be used by procedures alongside data for executing procedures.

**Part II: The abstraction of objects, at a high layer of abstraction**

1. The brown-covered envelope Kay refers to contained high-level documentation. High-level documentation for a program omits most of the details. Why is high-level documentation important? (answer this question…)

So the program can easily be followed by a person and they can quickly understand what it is doing.

1. High level documentation for object-oriented code shows **class diagrams** and indicates how the diagrams are related to each other. A class diagram shows three things:

|  |
| --- |
| class name |
| list of attribute names |
| list of method names |

Reenskaug's documentation included class diagrams very similar to today's standard. This image of Reenskaug's work reproduced from *The Early History of Smalltalk* is close to our modern standard!



In the 1990s the majority of programmers began to work in the object-oriented programming paradigm. The diagrams became standardized in **Unified Modeling Language (UML)**. UML was adopted as the standard for communicating and documenting software design in 1997 by the standards consortium OMG (Object Management Group). OMG had been created eight years earlier by eleven companies, including IBM, Apple, HP, Sun, American Airlines, and Data General.

What problems are created if people don't collaborate to create standards? (answer this question…)

Everyone will be doing things differently making it hard to work together and share things.

1. The *Python®* programming language uses a dot notation to refer to the attributes of an object and to call the class' methods on an object. For example we used the AxesSubplot class and called set\_title() on ax, which was an AxesSubplot:

ax.set\_title('My picture')

Here is the AxesSubplot class described with a UML diagram:

|  |  |
| --- | --- |
| AxesSubplot | class name |
| title  ... | attribute names |
| set\_title()  ... | method names |

What is another method from AxesSubplot that would be listed in this UML class diagram? (answer this question…, it might require a little bit of thinking back to old code you have seen/used)

1. Consider the following class:

|  |  |
| --- | --- |
| PingPongBall | class name |
| radius  color | attribute names |
| draw\_circle() | method names |

Many programming languages support programming in the object-oriented **programming** **paradigm**. In all of the languages that support object-oriented programming, if you ***instantiate*** an object in the class described by this UML, the computer stores the object's attribute data. You can then call the class' methods on the object.

In the *Python* programming language, you might use the PingPongBall class like this:

|  |  |
| --- | --- |
| In[a]:  In[b]:  In[c]:  In[d]: | **import** **realityArtist as** **fiat**  mike = fiat.PingPongBall()  **print** mike.color  mike.draw\_circle() |

We will explore these four commands to understand what they do. The import statement executes the class definitions in realityArtist and gives it the nickname fiat.

|  |  |
| --- | --- |
| In[a]: | **import** **realityArtist as** **fiat** |

Where is the class definition? It is code in the realityArtist module, and it might look like this. This is how a new class is created. You won't have to create classes in this course, but we're glancing at the magic behind the curtain.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | *# module realityArtist.py*  **class** **PingPongBall**(object):  **def** \_\_init\_\_(self, myRadius=20, myColor='#BB7700'):  """ Creates a new PingPongBall    radius expressed in millimeters  color is expressed in a 6-digit hexadecimal  color defaults to xBB red and x77 green (orangish)  """  *# Implement with a human so that:*  self.radius = myRadius  self.color = myColor    **def** draw\_circle(self, diameter=10, color='#FF0000', fill=False):  """ Draws a circle on a random location on the ball  uses colored pencil  diameter is expressed in millimeters  color is expressed in a 6-digit hexadecimal string  default color red  """  *# Implement with human* |

Once this code is executed with the import statement, the computer knows how to create objects of the class. The constructor function in *Python*, which instantiates the class, is always the special function \_\_init\_\_(). Instead of being called with \_\_init\_\_(), it is called by using the class name. To reinforce concepts with a tactile activity, we have made up (tongue-in-cheek) a blend of "People" and *Python*. **Lines 10 and 22 comment that a human will implement part of the code!**

**Computers aren't used in this activity. It is an "unplugged" activity.**

How many arguments does the \_\_init\_\_ function definition have? (answer this question…)

3

The ***self*** argument is always first in class function definitions. It refers to the object being instantiated or the object on which the method is called.

1. Now we'll instantiate a PingPongBall.

|  |  |
| --- | --- |
| In[b]: | sagar = fiat.PingPongBall() |

Why is fiat there?

It is calling a method from the imported fiat realityArtist class

1. Record what you think the next line would print.

|  |  |
| --- | --- |
| In[c]: | **print** sagar.color |

The color of the PingPongBall named sagar

1. Before we instantiate more PingPongBalls, let's see sagar draw a circle.

|  |  |
| --- | --- |
| In[d]: | sagar.draw\_circle() |

According to the docstring above, this code draws a random circle on the ball. According to the default parameters, what size and color will be drawn?

Diameter of 10 and red

1. What lines of code would instantiate a ball called alexa and then draw the black, red, and gray circles as shown in the picture below? (answer this question…. And Record the code below. )



|  |
| --- |
| In[e]:  In[f]:  In[g]:  In[h]:  ... : |

E: alexa = fiat.PingPongBall()

F: alexa.draw\_circle(diameter = 40, color = #000000)

G: alexa.draw\_circle()

H: alexa.draw\_circle()

I: alexa.draw\_circle()

J: alexa.draw\_circle(diameter = 30, color = #D3D3D3)

1. Now we'll make an aggregator to remember all the PingPongBalls in the classroom.

|  |  |
| --- | --- |
| In[i]: | items = [sagar, alexa] |

As a classroom of students, instantiate more PingPongBalls.

|  |  |
| --- | --- |
| In[j]:  ... : | **for** i **in** range(6):  items.append(PingPongBall()) |

What is items[1] ?

Ping pong ball alexa

What is items[3].color?

The color of the 4th ping pong ball

1. As a classroom, implement this code.

|  |  |
| --- | --- |
| In[k]: | items[3].draw\_circle(myColor='#00FF00', diameter=6) |

Describe what it did.

Drew a green circle with a diameter of 6

1. As a classroom, implement this code.

|  |  |
| --- | --- |
| In[l]:  ... : | **for** pingpong **in** items:  pingpong.draw\_circle(diameter=6, fill=True) |

Describe what it did.

Drew a filled red circle of diameter 6 on every ball

1. As a classroom, implement this code.

|  |  |
| --- | --- |
| In[m]:  ... : | **for** pingpong **in** items:  pingpong.append(PingPongBall()) |

Describe what it did.

Not sure on this, but I would guess it errors because pingpong is not a list

1. Now we will define a new class, shown in UML and *Python* below.

|  |  |
| --- | --- |
| GolfBall | class name |
| radius  color | attribute names |
| draw\_circle() | method names |

The *Python* below continues the realityArtist module shown above.

|  |  |
| --- | --- |
| 1  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | *# module realityArtist.py*  **class** **GolfBall**(object):  **def** \_\_init\_\_(self, myRadius=38, myColor='#FFFFFF'):  """ Creates a new practice-style GolfBall    myRadius expressed in millimeters  myColor is expressed in a 6-digit hexadecimal  """  *# Implement with a human so that:*  self.radius = myRadius  self.color = myColor    **def** draw\_circle(self, diameter=5, color='#00FF00', fill=True):  """ Draws a circle on a random location on the ball  uses colored pencil  diameter is expressed in millimeters  color is expressed in a 6-digit hexadecimal string  """  *# Implement with human* |

What are some differences between the GolfBall and PingPongBall classes? (answer this question…)

Ball size is automaticaly larger and is white, circles are automatically smaller green and filled

1. With your partner, implement this code.

|  |  |
| --- | --- |
| In[n]:  ... : | **for** i **in** range(8):  items.append(GolfBall()) |

Describe what it does.

Makes 8 golfballs in a list items

1. With your partner, implement this code.

|  |  |
| --- | --- |
| In[o]:  ... : | **for** ball **in** items:  ball.draw\_circle() |

Describe what it does.

**Draws a circle on every ball**

**Part III: The abstraction of objects, at another high layer of abstraction**

1. As a classroom, design an abstraction for a class of objects. Record the abstraction in UML here.

|  |  |
| --- | --- |
| dinosaur | class name |
| Color  food | attribute names |
| Draw\_Spikes | method names |

1. *Abstraction* is defined by the following two characteristics.

* Discarding some details
* Generalizing

Describe some of the details that were lost and describe what generality was gained with the abstraction from the previous step.

Looses exactly how it is done/works, but is a lot easier to look at and comprehend

1. With your partner, design an abstraction for a class of objects. Record the abstraction in UML here.

|  |  |
| --- | --- |
| hamburger | class name |
| Toppings  burnedness | attribute names |
| Add\_patty  Add\_seaseme\_seed | method names |

1. Software developers often use UML diagrams to describe their ideas during the early stages of software development. Suppose you and your partner are working on a word processor.
   1. Describe what you would want the user to be able to do with the word processing software.

Check spelling, check grammar

* 1. Circle all nouns in your description.

Spelling, grammar

* 1. For each noun that you think would make sense as a class, make up a class name and title a UML class diagram with it.

|  |  |
| --- | --- |
| spelling | class name |
| Prefix  Suffix  root | attribute names |
| Check\_dictionary | method names |

|  |  |
| --- | --- |
| grammar | class name |
| Sentence  paragraph | attribute names |
| checkSentance  checkParagraph | method names |

* 1. In your UML class diagram, record attributes and methods that would make sense with objects in these classes.

Conclusion

1. Think of an example from your daily life where you use abstraction. Describe some of the details you discard and some of the generality you gain by using the abstraction. (be specific to get full credit)

When asking to go to the restroom. I say I need to go and sometimes specify the urgency, but I typically leave out the details of what I need to do there like flushing or washing my hands.

1. What is the difference between procedural abstraction and data abstraction?(you might need to look back through this packet to help answer this question)

Data abstraction generalizes how data is represented into data types. Procedural abstraction generalizes how something gets done.

1. The GUI (Graphical User Interface) was first developed in 1961 by Ivan Sutherland for his Ph.D. at M.I.T. Watch a 1964 video produced by MIT, pay attention to the demo of Sutherland's work starting at 3:20, at <http://www.youtube.com/watch?v=USyoT_Ha_bA> .

Bill Gates at Microsoft got inspiration for Windows from Apple's Steve Jobs.

Steve Jobs at Apple got inspiration for Macintosh from Xerox's Alan Kay.

Alan Kay at Xerox got inspiration for Star from Trygve Reenskaug.

Trygve Reenskaug created the program Autokon with a graphical user interface in 1963 to design ships.

Trygve Reenskaug got inspiration for Autokon from M.I.T.'s Ivan Sutherland.

All along the way, GUI programming, object-oriented programming, and abstraction have been intertwined. Why do you think GUIs, objects, and abstraction have been connected like this in the history of computer science?(make sure you are specific in order to get full credit.)

All of these make programming easier for the programmer by simplifying things and making it easier to see and understand what is going on.