CODING PARADIGM (WAY WE APPROACH, WAY OF THINKING)

STUDENT OUTLINE

Lesson 1 - Introduction to Object Oriented Programming

INTRODUCTION: Before we begin to write actual programs, we need to introduce a few basic concepts of *object-oriented programming*, the paradigm of programming you will learn throughout this curriculum guide. The purpose of this lesson is to give you a feel for object-oriented programming and to introduce a conceptual foundation of object-oriented programming.

The key topics for this lesson are:

A. Classes and Objects
B. Messages and Methods
C. Objects in Software
D. Compiling and Running a Program

VOCABULARY:

OBJECT
INSTANCE
METHOD
CLASS
MESSAGE
ARGUMENT

DISCUSSION: A. Classes and Objects

1. Object-Oriented Programming (OOP) represents an attempt to make programs more closely model the way people think about and deal with the world. In object-oriented programming, a program consists of a collection of interacting objects. To write such a program you need to describe different types of objects: what they can do, how they are created, and how they interact with other objects.

NOUNS

2. The world in which we live is filled with objects. For example, an object we are all familiar with is a drawing tool such as a pencil or pen. A drawing tool is an object, which can be described in terms of its state and behaviors. The attributes (state) of a pencil are its drawing color, width of the line it draws, its location on the drawing surface, etc. Its behaviors consist of drawing a circle, drawing a line in a forward or backward direction, changing its drawing direction, changing the color, etc.

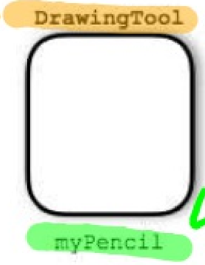
VERBS

3. An object in programming is an abstraction for a real-world object. For example, a drawing tool is an attempt to model the attributes and behaviors of a pencil or pen.

KIND OF OBJECT I HAVE

(CODE IN A CLASS)

"BLUEPRINT"



AN IDENTIFIER

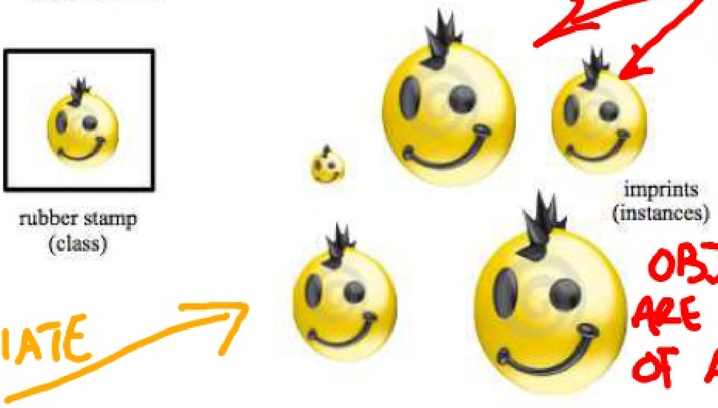
Figure 1.1 - A `DrawingTool` object named `myPencil`

4. To create an object inside the computer program, we must provide a definition for objects - how they behave and what kinds of information they maintain is called a *class*. A class is a kind of mold or template that the computer uses to create objects.

5. A class is like a rubber stamp that can be used many times to make many imprints. Each imprint is an object and each one has its own individual properties such as "size" and "position." Different stampings may have different characteristics, even though they were all made with the same rubber stamp.

DEFINITION IN CLASS (FIRST)

THEN, WE CAN INSTANTIATE



ACTUAL OBJECTS

OBJECTS ARE INSTANCES OF A CLASS

Figure 1.2 - A class and fives instances having five different values for instance variables "`size`" and "`position`".

6. In OOP terminology, we say the `DrawingTool` object `mypencil` is an *instance* of the `DrawingTool` class. An object can only be an instance of one class. In effect, an instance of the class *belongs to* the class.

B. Messages and Methods

1. In writing object-oriented programs we first define classes, and while the program is running, we create objects from these classes to accomplish tasks. A task can range from drawing in a paint program, to adding numbers, to depositing money in a bank account. To instruct a class or an object to perform a task, we send a message to it.

FUNCTION (C)

2. You can send a message only to the classes and objects that understand the message. For an object to process the message it receives, it must possess a matching method, which is a sequence of instructions an object follows to perform a task.

3. For example, consider what kind of operations you can carry out with a pencil. You can

WHAT A PENCIL CAN DO

- draw a line in the forward direction
- change the drawing direction by turning left
- get the current drawing color

4. Suppose you have an object `myPencil` of type `DrawingTool`. You could represent the behaviors of the `DrawingTool` class with the methods

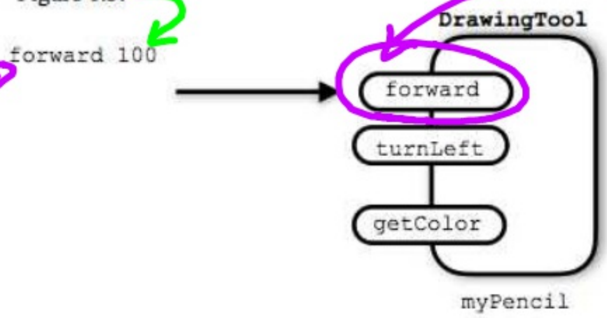
SENDING A MESSAGE ==

CALLING A FUNCTION (C)

- `forward`
- `turnleft`
- `getcolor`

5. To draw a line of a specified length, we send the message `forward` along with the distance to move the pencil. A value we pass to an object is called an *parameter* of a message. A diagram of sending a message is shown below in Figure 1.3.

METHOD



MESSAGE

CALL (C)

FUNCTION DEFINITION (C)

Figure 1.3 - Sending a `forward` message to a `DrawingTool` object

6. The diagram shown in Figure 1.3 illustrates a situation in which an object carries out a request (it draws a line 100 units long) but does not respond to the message sender. In many situations, we need an object to respond by returning a value to the message sender. For example, suppose we want to know the current color that is being used for drawing. We can use the `getColor` message to return the value. A method that returns a value to a message sender is illustrated in Figure 1.4 below.

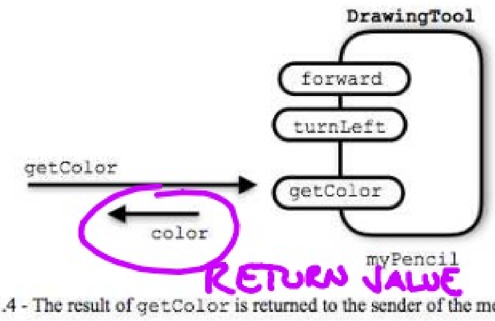


Figure 1.4 - The result of `getColor` is returned to the sender of the message.

C. Objects in Software

1. A program is a collection of instructions that, when performed, cause a particular task to be performed on a computer. Individuals who write programs are therefore called programmers. The terms software and code refer to a collection of one or more programs, so programmers are also referred to as software developers.

2. Today, the strategy (or paradigm) most often employed by software developers is called *object-oriented-programming (OOP)*. A programmer using an object-oriented strategy begins by selecting objects that can collectively solve the given problem.

1. USE PRE-MADE CLASSES (API's)
2. DESIGN OUR OWN CLASS

3. To illustrate how a particular program might be developed in an OOP fashion, the software developer begins with a set of *program requirements* that specifies the desired task for a program. For example:

   Write a program to draw a square on a piece of paper with a pencil.

4. The program requirements suggest that there are two objects, namely a pencil and piece of paper. One way to determine the objects needed in a program is to search for the nouns of the problem. In our draw square problem, the pencil and paper are examples of such nouns.

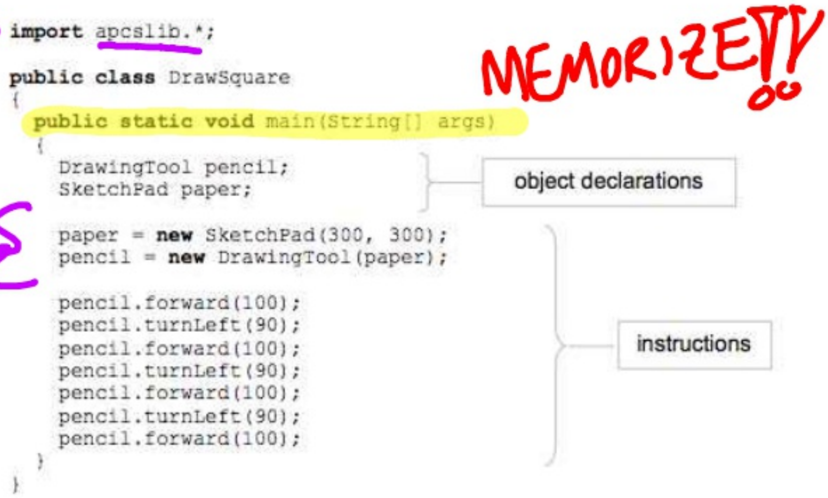
5. Once a programmer identifies the objects in the program, the next step is to find or create a class corresponding to each object. Classes are essential because they serve as the places where the code of an objectoriented program resides.

ALL OF OUR CODE WILL BE WRITTEN INSIDE OF CLASS

DEVELOPED BY Dr. CHRIS NEVISON AT COLGATE UNIVERSITY

6. Ideally, a programmer *reuses* an existing class, as opposed to writing code for a new class. For the purposes of our drawing example, we will use the preexisting `DrawingTool` and `SketchPad` classes for the pencil and paper objects.

7. Programming languages are like other foreign languages – the first exposure to a written example is bound to seem pretty mysterious. You don't have to understand the details of the program shown below, we'll go over them in the next lesson.



```
import apcslib.*;

public class DrawSquare
{
  public static void main(String[] args)
  {
    DrawingTool pencil;
    SketchPad paper;

    paper = new SketchPad(300, 300);
    pencil = new DrawingTool(paper);

    pencil.forward(100);
    pencil.turnLeft(90);
    pencil.forward(100);
    pencil.turnLeft(90);
    pencil.forward(100);
    pencil.turnLeft(90);
    pencil.forward(100);
  }
}
```

MEMORIZE!!

INSTANTIATIONS

Program 1.1 – `DrawSquare.java`

8. The execution of an object-oriented program begins with an initial object. This initial object serves as the starting point for the entire program. For the program in Program 1.1, the initial object belongs to the `DrawSquare` class.

9. The state of an object depends on its components (objects). The `DrawSquare` object includes one `DrawingTool` object declared in the line that begins with the word `DrawingTool` and a `SketchPad` object declared in the line that begins with `SketchPad`. The `DrawingTool` object is given the name `pencil` and the `SketchPad` object is given the name `paper`.

10. An object's behavior is determined by *instructions*. When a program executes, the program's instructions are performed. There are nine instructions for the `DrawSquare` object that are found following the object declaration lines.

    a. The first instruction will construct a new `SketchPad` object named `paper` with dimensions of 300 by 300.
    b. The next instruction will cause a new `DrawingTool` object named `pencil` to be constructed on the `SketchPad` object named `paper`.
    c. The next line of code will cause the `pencil` to move forward 100 units drawing a line as it goes.
    d. The next line of code will cause the `pencil` to turn to the left 90 degrees.
    e. The remaining 5 steps repeat the process of steps c and d to draw the remaining three sides of the square.

11. The `DrawSquare` example illustrates the tools that a programmer uses to write a program. A program is built from classes that a programmer writes or reuses. Classes are composed from instructions, and these instructions are used in such a way that they manipulate objects to perform the desired tasks.

D. <u>Compiling and Running a Program</u>

1. A programmer writes the text of a program using a software program called an *editor.* The text of a program in a particular programming language is referred to as *source code*, or simply *source*. The source code is stored in a file called the *source file.* For example in the `DrawSquare` example given above, source program would be created and saved in a file named `DrawSquare.java`.

2. Compiling is the process of converting a program written in a high-level language into the *bytecode* language the Java interpreter understands. A Java compiler will generate a *bytecode file* from a source file if there are no errors in the source file. In the case of `DrawSquare`, the source statements in the `DrawSquare.java` source file would be compiled to generate the bytecode file `DrawSquare.class`.

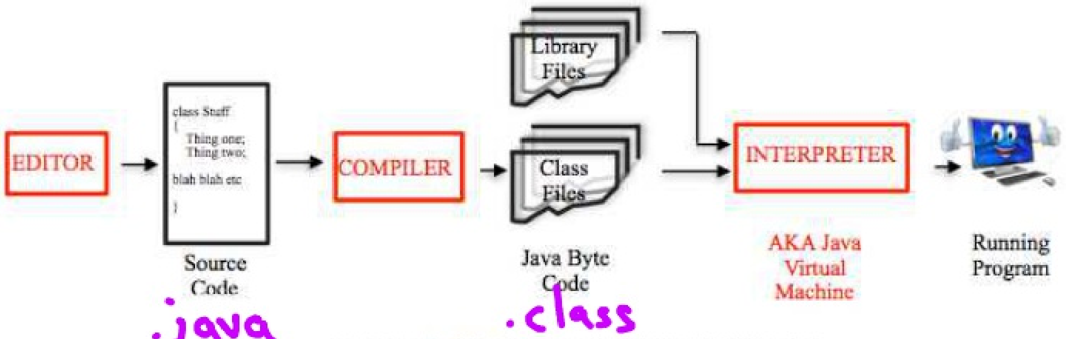


Figure 1.5 - From Source Code to Running Program

3. Errors detected by the compiler are called *compilation errors*. Compilation errors are actually the easiest type of errors to correct. Most compilation errors are due to the violation of syntax rules.

4. The Java interpreter will process the bytecode file and execute the instructions in it.

5. If an error occurs while running the program, the interpreter will catch it and stop its execution. Errors detected by the interpreter are called *run-time errors*.

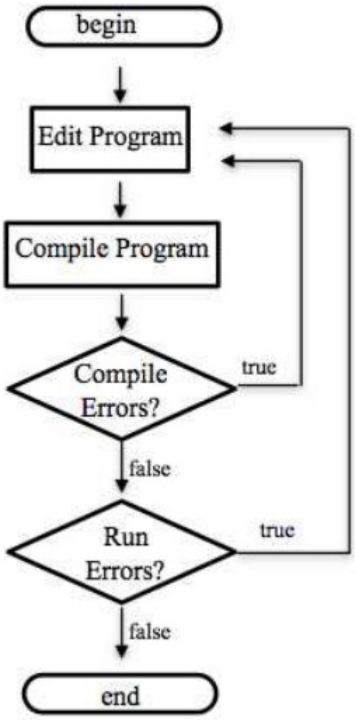


Figure 1.6 -
The Edit-Compile-Run
Cycle for a Java Program

SUMMARY/ REVIEW: One can think of an OOP application as a simulated world of active objects. Each object has a set of methods that can process messages of certain types, send messages to other objects, and create new objects. A programmer creates an OOP application by defining classes of objects.

ASSIGNMENT: Lab Exercise, L.A.1.1, *DrawHouse*