# STUDENT LESSON

A1 – Introduction to Object-Oriented Programming (OOP)

**INTRODUCTION:** Before we begin to write actual programs, we need to introduce a few basic concepts of *object-oriented programming*, the style 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 its conceptual foundation.

The key topics for this lesson are:

A. Classes and Objects

B. Methods

C. Objects in Software

D. Compiling and Running a Program

**VOCABULARY:** ARGUMENT ATTRIBUTES

BEHAVIORS CLASS

COMPILING CONSTRUCTOR

EDITOR INSTANCE

METHOD OBJECT

OBJECT-ORIENTED PROGRAMMING

SOURCE CODE

**DISCUSSION:** A. Classes and Objects

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

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 attributes and behaviors. Attributes are aspects of an object that describe it, while behaviors are things that the object can do. The attributes of a pencil are its drawing color, width of the line it draws, its location on the drawing surface, etc. Anything that *describes* an object is called an attribute. Its behaviors consist of drawing a circle, drawing a line in a forward or backward direction, changing its drawing direction, etc. Anything that an object *does* is called a behavior. Another aspect of an object has to do with *creation*, which determines the initial state of an object.

1. In order to use an object within a program, we need to provide a definition for the object. This definition is called a class. The class describes how the object behaves, what kind of information it contains, and how to create objects of that type. A class can be thought of as a mold, template, or blueprint that the computer uses to create objects.

4. When building a house, a construction crew uses a blueprint to define the aspects of the house. The blueprint gives the specifications on how many bedrooms there are, how to position the electrical wiring, the size of the garage, etc. However, even two houses built from the same blueprint may have different paint colors and will have different physical locations. Clients who are buying a house may make slight modifications to these blueprints. For example, they may want a bigger garage or a smaller porch. We can see the houses built from the blueprint as objects since they are all similar in structure, but each house has its own unique attributes. In the world of programming, we can view the blueprint just like a class, i.e. a tool for creating our objects.

B. Methods

1. While a program is running, we create objects from class definitions 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 an object to perform a task, we send a message to it.
2. In Java, we refer to these messages as methods.
3. An object can only receive a message that it understands, which means that the message must be defined within its class.

4. Suppose we take the DrawingTool class (provided by this curriculum in the package *gpdraw.jar*) and create an object myPencil. In OOP terminology, we say the object myPencil is an *instance* of the DrawingTool class. An object can only be an instance of one class. We can visually represent an object with an object diagram, as shown in Figure 1.1.

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

5. These are some of the behaviors that the DrawingTool class provides:

* forward
* turnLeft
* getColor

6. To draw a line of a specified length, we call the method forward along with passing the distance to move the pencil. A value we pass to an object’s method is called an *argument*. A diagram of calling a method is shown below in Figure 1.2.

|  |  |
| --- | --- |
|  |  |
|  | Figure 1.2 Calling the forward method of a DrawingTool object |

7. If we need to change the direction myPencil is facing, we can call the turnLeft method. This will bring a ninety-degree turn to the left. Two left turns can give us a complete reversal of direction, and three left turns essentially gives us a right turn. Notice that we do not need to send any arguments with the turnLeft method. A left turn is simply a left turn and does not need any additional information from the user. A diagram calling turnLeft is shown below in Figure 1.3.

|  |  |
| --- | --- |
|  |  |
|  | Figure 1.3 Calling the turnLeft method of a DrawingTool object |

8. The diagrams shown in Figures 1.2 and 1.3 illustrate situations in which an object carries out a request by the user but does not respond to the sender. Figure 1.2 requires arguments from the user because the user must specify how far to move, whereas Figure 1.3 operates without any specific details. However, in many situations we need an object to respond by returning a value to the sender. For example, suppose we want to know the current color that is being used for drawing. We can use the getColor method to return the value. The getColor method is illustrated returning a value to the sender in Figure 1.4 below.

|  |  |
| --- | --- |
|  |  |
|  | Figure 1.4 - The result of getColor is returned to the sender |

C. Objects in Software

1. A program is a collection of instructions that performs a particular task on a computer. Software is a collection of one or more programs. Code refers to the actual symbols that a programmer types in that tell the computer what instructions to execute. Individuals who write programs are called programmers, software-engineers, software-architects, and coders among many other terms.

2. OOP is a strategy often employed by software developers. A programmer using an OOP strategy begins by selecting objects that can collectively solve the given problem.

3. To develop a particular program in an OOP fashion, the software developer might begin with a set of *program requirements*. For example:

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

4. A way to determine the objects needed in a program is to search for the nouns of the problem. This technique suggests that the above program should have three objects: a pencil, a piece of paper, and a square.

5. Ideally, a programmer *reuses* an existing class to create objects, 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. However, we don’t have a class for a square that is pre-made, so we must make our own.

6. Programming languages can be compared to a foreign language – 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. They will be covered in more detail in the next lesson.

**import** gpdraw.\*;

**public** **class** DrawSquare{

private DrawingTool myPencil;

object declarations

instructions

private SketchPad myPaper;

**public** DrawSquare(){

myPaper = **new** SketchPad(300, 300);

myPencil = **new** DrawingTool(myPaper);

}

**public** **void** draw(){

myPencil.forward(100);

myPencil.turnLeft();

myPencil.forward(100);

myPencil.turnLeft();

myPencil.forward(100);

myPencil.turnLeft();

myPencil.forward(100);

}

}

Code Sample 1.1 – DrawSquare.java

7. In OOP, we concern ourselves mostly with the objects themselves and how they relate to the other objects in the program. However, there must be a starting point for the program to begin creating objects, as the objects would obviously not be able to do anything if they did not exist. Code Sample 1.1 has no starting point and would therefore not be able to do anything by itself. Later on, we will learn how to utilize this code in an actual program.

1. The state of an object depends on its components. 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 myPencil and the SketchPad object is given the name myPaper.
2. A constructor is a method with the same name as the class. The first instruction will construct a new SketchPad object named myPaper with dimensions of 300 x 300 (read as 300 by 300). The next instruction will cause a new DrawingTool object named myPencil to be constructed using the SketchPad object named myPaper.

10. An object’s behavior is determined by *instructions* within its methods. When the method draw() for a DrawSquare() object is called, the instructions within the draw method will execute in the order they appear. There are seven instructions in the draw method. The first instruction will cause the myPencil to move forward 100 units drawing a line as it goes. The next line tells myPencil to turn left. The remaining 5 steps repeat the process of steps 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 objects and classes that a programmer writes or reuses. Classes are built from instructions, and these instructions are used in such a way that they manipulate objects to perform the desired tasks.

D. Compiling and Running a Program

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 you can simply use *source* or *code* individually. The source code is stored in a file called the *source file*. For example in the DrawSquare example given above, source code 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. Classes inside a package, such as the *gpdraw.jar*, have already been compiled into bytecode for you.

|  |  |
| --- | --- |
|  | Source Code  Editor    **Hi h;**  **h=new Hi();**  **h.hello();**  **h.bye();**    Compiler  Class files  Library files  Interpreter  Hello World!  Bye!  Java Bytecode  Program  Running Program |
|  | 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. These are the basic rules of languages that programmers must follow so that the interpreter understands what to do. It is similar to grammar in a spoken language and varies from language to language.

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*. Run-time errors are usually caused by a fault in the logic of the program, such as accidentally causing the computer to try and divide a number by zero.

|  |  |
| --- | --- |
|  | True  True  Begin  False  Edit Program  Compile program  Compiler errors?  Run program  Run-time errors?  End |
|  | Figure 1.6 – 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. Programmers can either define new classes for use in their program, or they can use pre-existing classes to create the objects for their application. |

**HANDOUTS AND**

**ASSIGNMENTS:** Handout A1.1, *DrawingTool Class Specifications*  
Lab Assignment A1.1, *DrawHouse*

Worksheet A1.1, *Object-Oriented Programming*