Lesson 2 – Object-Oriented Programming

**INTRODUCTION:** Java is a “high-level” computer programming language. High-level languages are more similar to English (or other human languages) than machine code. Programming in binary (ones and zeros) or Assembly would be considered low-level. In this section, we will continue to explore the world of OOP by looking at the example from Lesson A1 in more detail.

The key topics for this lesson are:

A. Our First Java Application

B. Program Components

C. Object Declaration, Creation, and Message Sending

D. Class Diagrams

E. The Difference Between Objects and Classes

**VOCABULARY:** CLASS DIAGRAM COMMENTS

CONSTRUCTOR DRIVER CLASS

IDENTIFIER **import**

**main** MESSAGE

**new** PACKAGE

UML

**DISCUSSION:** A. Our First Java Application

1. Our first bit of Java code in Lesson A1 will display a square in a window as shown in Figure 2.1. Although this program is very simple, it illustrates the fundamental strategy of an object-oriented program.



Figure 2.1 – DrawSquare

2. The following is the Java class that we encountered in the previous lesson:

**import** gpdraw.\*;

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

private DrawingTool myPencil;

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 2.1 *– DrawSquare.java*

3. The class is called DrawSquare and the class includes two methods. The first method, DrawSquare(), is named exactly the same as the class and is used in the construction of new instances of this class. The second method is called draw(), and it is where most of the action for this class takes place.

4. The DrawSquare constructor calls SketchPad’s constructor to create a new SketchPad object named myPaper with an initial size of 300 by 300 pixels. This will happen every time a new object of DrawSquare is created.

1. Another object called myPencil is created using the DrawingTool constructor with a drawing area represented by the myPaper object.
2. The method named draw() contains only those instructions directly related to the actual drawing of our square.
3. In order to run this program, we will create what is called a driver or throwaway class. This class serves the purpose of testing DrawSquare. The DrawSquare class should be tested and used in a small class to ensure that it is working as expected. We can then take the DrawSquare class and safely use it in other programs later on.

**public** **class** driver{

**public static void main**(String[] args){

DrawSquare sq = new DrawSquare();

sq.draw();

}

}

B. Program Components

1. Programming languages allow the inclusion of comments that are not part of the actual code. Documentation is an important aspect of writing programs as it helps the reader to understand what is going on. Java provides three different styles of comments.

/\* Multi-line comment. This style allows you to write

longer documentation notes as the text can wrap around to

succeeding lines.

\*/

// Single-line comment.

// This starts at the slashes and stops at the end of the line.

/\*\* Java-doc comment. This is a special type of comment used

to create APIs and will be discussed more thoroughly in a later lesson.

\*/

2. Programmers try to avoid "reinventing the wheel" by using predefined libraries of code. In Java, these predefined classes are grouped into *packages*. The **import** statement allows the program to use predefined classes.

3. Java comes with many packages. There are also many packages created by programmers that you can use. There are two non-standard packages supplied in this curriculum guide. The one we have been using so far is called gpdraw. In our example program, the DrawingTool and SketchPad classes are imported from the gpdraw package with the statement

**import** gpdraw.\*;

4. Every Java program contains a main method. However, not all classes need to contain a main method, nor should they.

5. Most classes will contain methods to define a class’s behavior. Here we will explore the basic syntax of a method. You will learn how to utilize all of these parts in Lesson A4 - *Object Behavior*. A method has the following general syntax:

***modifiers* *return\_type*** *method\_name* ( *parameters* ){

*method\_body*

}

* The ***modifiers*** refer to a sequence of terms designating different kinds of access to methods. (e.g. public, private)
* The ***method\_name*** is the name of the method. In the case of Code Sample 2.1, the name of the method is draw.
* The ***return\_type*** refers to the type of data a method returns. The data type can be one of the predefined types (e.g., **int**, **double**, **char,** **void**, **String**, etc.) or a user-defined type. In Code Sample 2.1, the draw method does not return a value and is therefore designated by the void type.
* The ***parameters*** list will allow us to send values to a method. In our example, we must tell the forward method how far to go. (myPencil.forward(100)).
* The ***method\_body*** contains statements to accomplish the work of the method. In this example, there are seven lines of code needed to draw the square.

C. Object Declaration, Creation, and Message Sending

1. Every object in a program must be declared. An object declaration designates the name of an object and the class to which the object belongs. Its syntax is:

*class\_name* *object\_name*

* *class\_name* is the name of the class to which these objects belong.
* *object\_name* is a sequence of object names separated by commas.

In the case of the DrawSquare example, the myPencil object is declared as

DrawingTool myPencil;

other examples:

Account checking;

Customer bob, betty, bill;

The first declaration declares an Account object named checking, and the second declares three Customer objects.

2. No objects are actually created by the declaration. An object declaration simply declares the name (identifier) that we use to refer to an object. Calling a constructor using the new operator creates an object. The syntax for **creating an object is:**

*object\_name* = **new** *class\_name* ( *arguments* ) ;

* *object\_name* is the name of the declared object.
* *class\_name* is the name of the class to which the object belongs.
* *arguments* is a sequence of zero or more values passed to the **constructor**.

In the DrawSquare example, the paper object is created (instantiated) with the statement

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

3. After the object is created, we can start sending messages to it. The syntax for sending a message to an object is

*object\_name***.***method\_name*( *arguments* ) ;

* *object\_name* is the name of the declared object.
* *method\_name* is the name of a method of the object.
* *arguments* is a sequence of zero or more values passed to the method.

In the DrawSquare example, the myPencil object is sent a sequence of messages; forward with an argument of 100, and turnLeft with an argument of 90.

myPencil.forward(100);

myPencil.turnLeft(90);

D. Class Diagrams

1. Pictures are often helpful when designing software. One particularly useful picture is the class diagram. A class diagram shows the key features of a class including:

* the class name
* the class attributes
* the class methods

# **Class Name**

Attributes

## Methods

Figure 2.2 – General form of a Class diagram

1. A software class consists of attributes (think of these as nouns) and methods (think of these as verbs).
2. An attribute, or *instance variable*, represents a property of an object.
3. A method is an operation that can be performed upon an object. It is useful to picture the attributes and methods as a class diagram with the following general form.
4. The class diagram is a rectangle with three compartments separated by horizontal lines. The top compartment contains the name of the class. The middle compartment lists the attributes of the class, and the bottom compartment shows the class methods. This class notation is part of the *Unified Modeling Language (UML)*. UML is the most widely used set of notations in today’s software engineering industry. A diagram for the DrawSquare class is shown below.

|  |  |
| --- | --- |
|  | **DrawSquare**  SketchPad  DrawingTool  DrawSquare()  draw()  (Nouns)  (Verbs) **Class Name** Attributes Methods |
|  | Figure 2.3 – Class diagram for the DrawSquare class |

1. The methods of the class are listed in the bottom compartment of the class diagram. One of the methods in the DrawSquare class has the same name as the class (DrawSquare()). It may seem strange for a method to have the same name as its class, but this is how you give instructions on how to create objects of this class. This is called a constructor.
2. The DrawSquare class also makes use of another class: DrawingTool (the class of the myPencil object). The class diagram for this class is shown in Figure 2.4 below. This figure illustrates a couple of new notations that are typical of class diagrams.

**DrawingTool**

(Attributes)

<<constructor>>

DrawingTool()

...

<<modifier>>

void down()

void forward(int d)

void turnLeft(double degrees)

...

|  |  |
| --- | --- |
|  | Figure 2.4 – Class diagram for the DrawSquare class |

The “...” notation shown within the class diagram indicates that the list of methods is incomplete. There are more methods in the DrawingTool class that are not shown, because they are not relevant to the discussion.

UML diagrams frequently include labels within “<< >>” symbols to categorize the class methods. In Figure 2.4, the DrawingTool() method is categorized as a *constructor* method while down, forward, and turnLeft are *modifier* methods. The distinction between constructor and modifier categories will be covered in a later lesson.

E. The Difference Between Objects and Classes

1. An object is very closely associated with the class to which it belongs. An object has attributes as defined by its class. An object’s behavior is restricted by the methods that are included in its class. However, there are significant differences between objects and classes.

A class:

* is a blueprint that defines attributes and methods.
* is written by a programmer.
* cannot be altered during program execution.
* is named by a class name.

An object:

* must belong to some class.
* is an instance of that class.
* exists during the time that a program executes.
* must be explicitly declared and constructed by the executing program.
* has attributes that can change in value and methods that can execute during program execution. (The class to which the object belongs defines theses attributes and methods.)
* is most often referenced using an identifier.

1. Classes can be compared to a blueprint. The purpose of a blueprint is to provide a guide for creating things, just as the purpose of a class is to guide the creation of objects. A single blueprint is designed to produce a single basic type of object. Similarly, the objects from the same class all share common characteristics.
2. Each object must belong to one particular class, and the object is said to be a member of the class to which it belongs. The myPencil object that was created earlier belongs to the DrawingTool class. This means that the myPencil object is permitted to perform DrawingTool methods (operations). This also means that the myPencil object is *not* permitted to perform DrawSquare methods; these methods are designed for a different kind (class) of object.

|  |  |
| --- | --- |
| **SUMMARY/ REVIEW:** | A programmer designs and writes classes that can later be used in other classes to create objects of that class. The objects then work together to solve your problem. Breaking down a large problem into a collection of smaller problems in this way is a very useful programming technique. When designing a class, we must consider the behaviors and attributes of the objects created from that class, as well as how we will create those new objects. |

**ASSIGNMENTS:** Lab Assignment A2.1, *Benzene*

Worksheet A2.1, *Objects and Classes*

Worksheet A2.2, *Review OOP*