STUDENT OUTLINE

Lesson 2 – Objects and Classes

INTRODUCTION: This lesson will introduce the basic structure of a Java application. Learning a computer language is very much like learning a **foreign language**. At first every concept seems new, but with effort and time you will eventually become fluent in a programming language. Java contains the fundamental features of high-level imperative language in addition to the tools of object-oriented programming.

The key topics for this lesson are:

A. The First Java Application
B. Program Components
C. Object Declaration, Creation, and Message Sending
D. Class Diagrams
E. The Difference Between Objects and Classes

VOCABULARY:
CONSTRUCTOR
PACKAGE
INSTANTIATE
`import` `new`
MESSAGE
RESERVED

KEYWORDS. PART OF THE LANGUAGE. NOTICE THE FONT... THESE ARE ACTUALLY CODE.

DISCUSSION:

A. The First Java Application

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

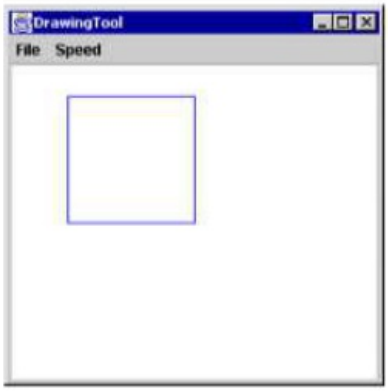
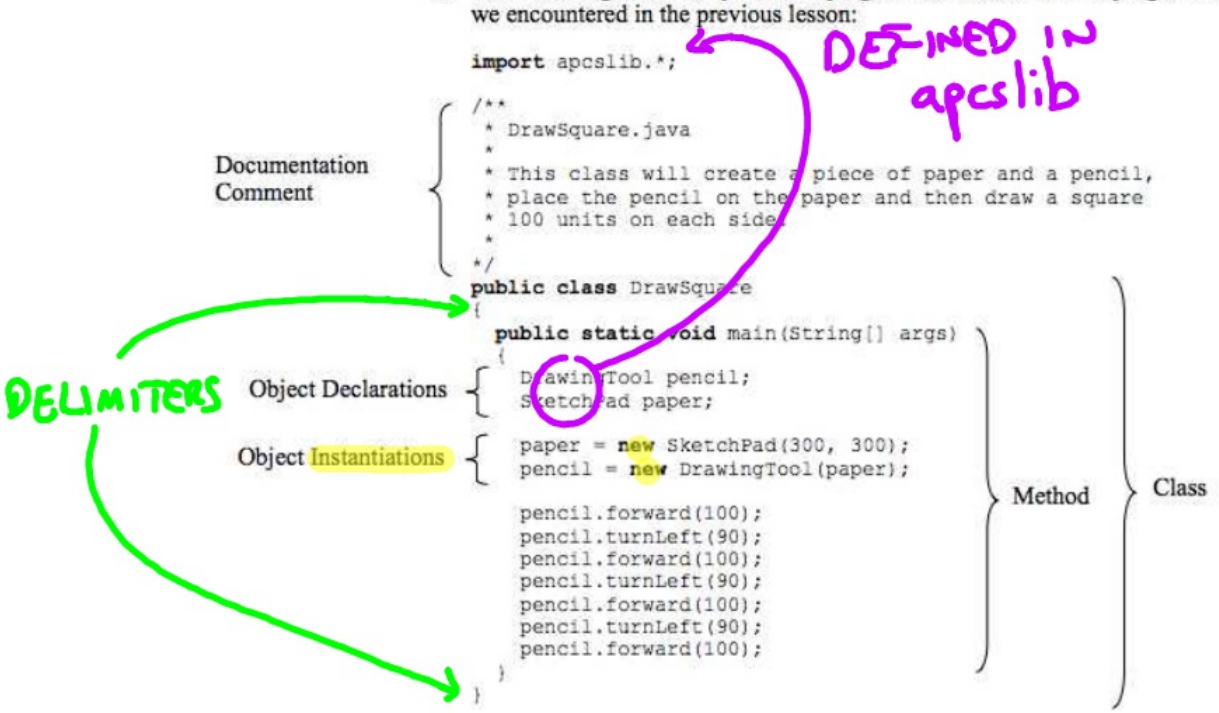


Figure 2.1 – `DrawSquare`

2. The following is the complete Java program and is based on the program that we encountered in the previous lesson:



Program 2.1 – *DrawSquare.java*

3. The program declares one class called `DrawSquare`, and the class includes one method called `main`. The `main` method is required for the program to run. It is where execution begins and must appear somewhere in your project. Write `main` exactly as it appears here every time.

4. From this main method, the `DrawSquare` class creates a `SketchPad` object named `paper` with an initial size of 300 by 300 pixels.

5. Another object called `pencil` is created using the `DrawingTool` class and named `pencil` with a drawing area represented by the `paper` object. A sequence of messages to place and move the `pencil` object are sent to draw the square.

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 allows for two styles of comments:

```
/* This style will allow you to write longer
   documentation notes as the text can wrap around to
   succeeding lines.
*/

// This style automatically ends at the end of the line.
```

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, and one is supplied in this curriculum guide called `apcslib`. In our example program, the `DrawingTool` and `SketchPad` classes are imported from the `apcslib` package with the statement

```
import apcslib.*;
```

4. A Java program is composed of one or more classes. The syntax for declaring a class is

```
class class_name
{
  class_member_declarations;
}
```

   a. The word **`class`** is a Java reserved word, which means it is not available for programmers to use. It is used to mark the beginning of a class ~~declaration~~.
   b. *`class_name`* is the name of the class. Any valid identifier can be used to name the class. (The complete set of rules for writing valid identifiers in Java will be covered in a later lesson.)
   c. *`class_member_declarations`* is a sequence of either data values or methods.

5. Every Java program consists of a main method. A method has the following general syntax:

```
modifiers return_type method_name ( arguments )
{
  method_body
}
```

a. The *modifiers* refer to a sequence of terms designating different kinds of methods. These will be discussed gradually in later lessons
b. The *return_type* refers to the type of data a method returns. The data type can be one of the predefined types (**int**, **double**, **char**, **void**) or a user-defined type. These will be discussed further in a later lesson.
c. The *method_name* is the name of the method. In the case of Program 2.1, the name of the method is `main`.
d. The *arguments* list will allow values to be sent to a method. This will be covered in future lessons.
e. The *method_body* contains statements to accomplish the work of the method.

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
```

a. *class_name* is the name of the class to which these objects belong.
b. *object_name* is a sequence of object names separated by commas.

In the case of the `DrawSquare` example, the `pencil` object is declared as

```
DrawingTool pencil;
```

other examples:

```
Account checking;
Customer bob, betty, bill;
```

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

CLASS NAMES ALWAYS BEGIN WITH A CAPITAL LETTER; METHODS NEVER START WITH A CAPITAL

ACTUALLY INVOKES THE CONSTRUCTOR METHOD OF A CLASS

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. An object is created by using the **new** operation. The syntax for **new** is

```
object_name = new class_name ( parameters ) ;
```

a. *object_name* is the name of the declared object.
b. *class_name* is the name of the class to which the object belongs.
c. *parameters* is a sequence of zero or more values passed to the **new** operation.

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

```
paper = 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( parameters ) ;
```

a. *object_name* is the name of the declared object.
b. *method_name* is the name of a method of the object.
c. *parameters* is a sequence of zero or more values passed to the object.

EXAMPLES OF METHOD CALLS

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

```
pencil.forward(100);
pencil.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

Figure 2.2 – General form of a Class diagram

2. A software class consists of two groups of members:

   - attributes (think of these as nouns)
   - methods (think of these as verbs)

3. An attribute, often represented by an *instance variable*, names a single instance of an object's state. A method is an operation that can be performed by 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 two 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.

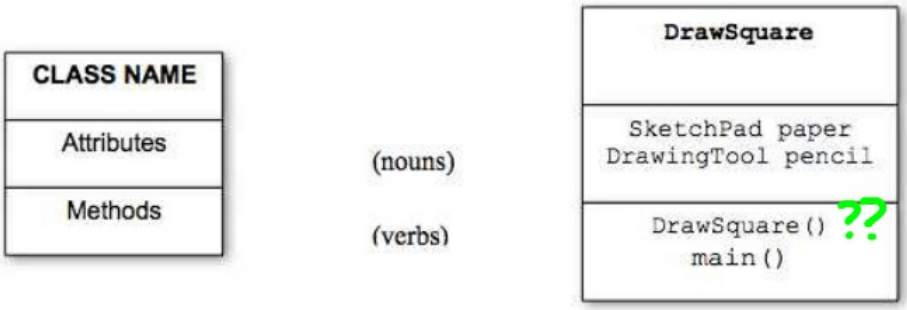


Figure 2.3 – Class diagram for the `DrawSquare` class

5. The methods of the class are listed in the bottom compartment of the class diagram. The only method 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 a common occurrence in some programming languages. This is known as the *default constructor* and the compiler will create it for you automatically if it does not exist. Details of this will be covered in a future Lesson.

THE COMPILER WILL BUILD ONE FOR YOU!!

6. The `DrawSquare` class also makes use of another class: `DrawingTool` (the class of the `pencil` object). The class diagram for this class is shown in Figure 2.4. This figure illustrates a couple of new notations that are typical of class diagrams

SEE H.A. 1.1 !

"PLAY WITH LEGOS"

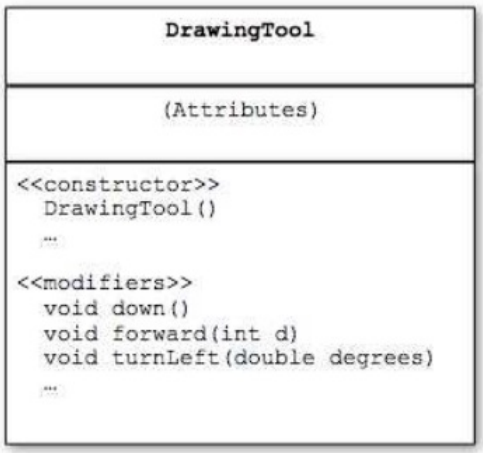


Figure 2.4 – Class diagram for the `DrawSquare` class

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 bracket within "<< >>" symbols to categorize the class methods. In Figure 2.4, the `DrawingTool()` method is categorized as a *constructor* method while `down`, `forward`, and `left` are *update (or modifier)* methods. The distinction between constructor and update 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.

BLUEPRINT

a. A class: (DEFINITION)
   - is a template that defines attributes and methods.
   - is written by a programmer as a part of a program.
   - does not exist when programs execute, except in the form of one or more member objects.
   - its code cannot be altered during program execution.
   - is named by a class name.

HOUSE BUILT

b. An object (INSTANCE)
   - must belong to some 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 often referenced using a variable name.

2. Classes can be compared to a rubber stamp. The purpose of a rubber stamp is to produce imprints, just as the purpose of a class is to produce objects. A single rubber stamp is designed to produce a single basic type of imprint. Similarly, the objects from the same class all share common characteristics.

3. 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 `pencil` object that was created earlier belongs to the `DrawingTool` class. This means that the `pencil` object is permitted to perform `DrawingTool` methods (operations). This also means that `pencil` object is *not* permitted to perform `DrawSquare` methods; these methods are designed for a different kind (class) of object.

SUMMARY/ REVIEW: One can think of an Object-Oriented Programming (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.2.1, *Benzene*