STUDENT LESSON

A6 – Libraries and APIs

STUDENT LESSON

A6 – Libraries and APIs

INTRODUCTION: Now that you have learned how to design your own classes, we will explore how to take advantage of the huge number of pre-made classes provided with Java. We will also learn how to read the APIs that come with those classes so that you will be able to take any class that comes with an API and teach yourself how to use that class. In this chapter, we will start with APIs, explore a few useful classes and their APIs, and then finish by learning how to write our own APIs so that other people can use our classes.

The key topics for this lesson are:

A. Understanding APIs
B. Final and Static
C. DrawingTool
D. Point2D.Double
E. Random
F. Math
G. Javadoc Tool

VOCABULARY:

API	PACKAGES
STATIC	FINAL
JAVADOC	

DISCUSSION:

A. Understanding APIs

1. API stands for Application Programming Interface and is one of the most useful tools you will have while working with Java. APIs show exactly how to use pre-made classes. The DrawingTool Class Specifications handout in Lesson A1 is an example of a simplified API. It lists the classes and constructors so that we know which methods are available. APIs do not tell us how the programmer dealt with a problem or what kind of formulas they used internally, but just tells us what methods we can access, how to interact with those methods, and what those methods will return back to us.

2. You can always access the Java APIs on the Web at java.sun.com. Click on API Specifications on the main page and then choose the version of Java you wish to retrieve the API for. You can also download the APIs to your computer for offline access. Many Java programming environments can be set up to access the APIs from within your code.

3. The Java APIs are organized both by package and by class. Packages are groups of related classes that are "packaged" together. When you use the code `import gpdraw.*;` you are adding the entire gpdraw package to your code. If you only need one or two classes from a package, you can add the classes individually with the code `import gpdraw.DrawingTool;`.

B. Final and Static

1. Before you look at these APIs in depth, you need to learn what the keywords *final* and *static* mean, as they frequently come up in the API documentation.

2. When used with a primitive data type, final means that the value of the variable will never change. This is useful in many cases, such as tax rates, math constants such as PI, and base values that are used in several places in your code. Identifiers with final are generally made with only capital letters so they are easily distinguishable from the rest of the code.

```
final double TAXRATE = 0.0825;
final double ROUNDS_IN_GAME = 100;
```

3. A program that repeatedly uses a constant identifier, such as TAXRATE, can be quickly modified by assigning a value to TAXRATE at the top of the program. If the value of the tax rate is hard coded everywhere it is used, then changing the tax rate would involve changing that value everywhere it appears in the program. This involves searching through your program and finding every single time that value is used, and then changing it to the new value. It would be easy to accidentally miss a value or two or possibly change something that was similar but wasn't supposed to be the tax rate. Using final values will not only save time but can also reduce the number of errors in your program.

4. Once a final variable is given a value within a program, that value may never change in that run. In order to change that value, you must change it within your code, recompile, and run the program again. You may still define this value within the constructors so that the value can be determined at run time; it need not be defined at the same time that the variable is declared.

5. Objects and methods can also take a final keyword. However, they behave differently than primitive data types. We have not yet discussed the concepts that these final objects and methods affect, but we will cover them in Lesson A10, *Inheritance*.

6. Using the keyword static means that the data member or method is attached to the class rather than to an object of that class.

7. With a static method, you only need to type the name of the class followed by the name of the method. You never need to create a new object when dealing with static methods. You can think of static methods as belonging to the class itself, whereas non-static methods are attached to the objects created from that class.

```
int jason = Math.pow(3,4);
```

`jason` receives and stores the result of 3 to the 4th power, which is 81.

Notice how there was never any need to create an object of type Math. Instead, we just have an int assigned the value created by the static pow method of the Math class.

8. Data members that are static may also be used without creating an object of that class. They also have specific behavior when using that value of the data member within the class. While we may make a hundred objects from one class, any static variables of that class will in fact be shared by each of those objects. If one object changes the value of that static variable, then the value is changed for all of the other objects of that type. This is because the variable and its value do not belong to any of the objects individually, but to the class itself.

C. DrawingTool

1. You have already been looking at an API for DrawingTool. The purpose of Handout A1.1 was to give you an introduction to the purpose of the class and how to use its various methods.

2. When instructed to draw a circle, you probably looked at Handout A1.1 and saw this:

```
public drawCircle (double r);
```

postcondition

- If the object is in drawing mode, a circle of radius `r` is drawn around the current location using the current `width` and `color`.

This tells us exactly what we need to know to use this method. We have the name of the method and the type of argument it takes. We also know what will happen after the method is called.

3. This is not the official Java format for an API, but it accomplishes the same thing. Without this handout, how would you have known how to draw a circle? How about when you made the picture of the house? How would you have known to use the forward method, turnLeft, down, or up? As you can see, APIs are an essential tool that must be looked at before a programmer can understand how to use pre-made classes.

4. Now that you understand how to read the DrawingTool API, take a look at a sample (the Pizza Parlor assignment) in the Javadoc folder (provided by the teacher) for this lesson. Open up the *index-all.html* file in your Web browser to see a basic package view. Click on the Help Link at the top to access a page on "How This API Document is Organized". Links that do not work are simply placeholders. The feature that you are clicking on does not exist for that class.

D. Point2D.Double

1. The Point2D class is useful for storing locations on a two dimensional space. It also contains several methods that can be used to do certain calculations. There are two sub-classes of Point2D, but you will generally only want to use Point2D.Double.

2. Consider an application where you need to track the locations of two mice in a flat-bottomed box. You could use two Point2D.Double objects to keep track of their locations.

3. What if we want to be able to tell how far apart the two mice are at any given time? We could use the distance formula from Geometry to calculate the distance. However, if we take a quick look at the Point2D API, we can find this method:

`double`	**`distance`**`(Point2D pt)` Returns the distance from this `Point2D` to a specified `Point2D`.

A line of code as simple as

```
double distance = rat1.distance(rat2);
```

will give us the distance between the two rats. This is much simpler than trying to do all the calculations ourselves. Remember, whenever possible we want to avoid writing code that has already been written. By doing a slight bit of research, we have saved ourselves the hassle of writing and then debugging code.

E. Random

1. As the name suggests, the java.util package is full of utility classes that you will find very useful. Many of them are going to be far too advanced for what you need at this stage, but as you progress in skill you should browse through the classes and see which ones you begin to understand. During this course, you will learn several of the util classes in detail, but for now we will just concentrate on this one, java.util.Random.

2. Random is probably the most fun of all the classes in the java.util package. Any sort of game or in depth simulation (such as inspecting a transportation system to see how efficient it is) will generally need some sort of randomness in order to work the way we want. That's where the Random class comes in. Let's take a look at it now.

Constructor Summary

`Random``()` Creates a new random number generator.

Method Summary	
double	**nextDouble**() Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.
int	**nextInt**(int n) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

3. These are the methods that you will be likely to find the most useful. The constructor is very basic and requires no arguments. The word "pseudorandom" just indicates that the number is not completely random. Computers are not physically capable of creating true random numbers. Therefore, in Computer Science, we refer to computer generated random numbers as "pseudorandom."

4. Let's look at a situation where we might use this class. Consider an electronic raffle for a prize. There are 200 participants with one number each between 1 and 200. We can create an object of type Random to determine who the winner is.

```
Random chooser = new Random();
int winner = chooser.nextInt(200) + 1;
```

   This code will give us a value between 1 and 200 in our winner variable.

5. The Random class has many uses. Most board games you have played probably use six sided dice to give the game an element of chance. Video and computer games also use randomness to determine whether you hit your enemy or not, modified by the skill of the character you are using. Think about the lottery as well. Without an element of chance, it would be pretty boring to buy lottery tickets and the game would cease to exist.

F. Math

1. The Math class in the java.lang package contains class methods for commonly used mathematical functions. Java loads the java.lang package automatically, so no special actions are required to access these.

2. The Math class contains both methods and the numerical values for two important mathematical constants, e and Pi. These constant values are accessed the same way as normal variables, but they can never be modified directly by your code.

3. The Math class is most useful for complex mathematical formulas, for example:

$$\frac{1}{2}\sin\left(x-\frac{\pi}{y^3}\right)$$

Using the Math class, we can create a line of code to solve this calculation much like you would type the same equation into your calculator:

```
(1.0/2.0) * Math.sin(x – Math.PI / Math.pow(y, 3));
```

4. Let's see what other kind of methods the Math class provides for us. Go ahead and take a look at the Java APIs. (Remember: Access the Java APIs on the Web at java.sun.com – click on API Specifications in the left column, and then pick the version of Java, such as J2SE 1.4.2, that you wish to see.) Find the Math class within the Java.lang package. Hint: You can find it either in the long list of classes on the left hand side by scrolling down to the M section, or you can access the Java.lang package first in the upper left section and then look for the Math class. You should find something similar to this:

java.lang

Class Math

java.lang.Object
  └ **java.lang.Math**

```
public final class Math
extends Object
```

Field Summary

static double	**E** The double value that is closer than any other to *e*, the base of the natural logarithms.
static double	**PI** The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Method Summary

static double	abs(double a) Returns the absolute value of a double value.
static int	abs(int a) Returns the absolute value of an int value.
static double	pow(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	sqrt(double a) Returns the correctly rounded positive square root of a double value.

5. The list shown here is much shorter than you will find online. However, let's look at the layout. At the very top is java.lang, the name of the package this class is in. Below that is the name of the class, followed by a series of class names.

You will learn more about this later, but for now just think of the classes listed there as the parents of the current class. Below that comes the description of the class, which gives an introduction to the purpose of the class. This allows programmers to quickly decide if this class will do what they need. Next is the list of attributes and behaviors, labeled as “Field Summary" and "Method Summary.”

6. The “Field Summary” section contains two items in it. In the left side of the table, we can see they are both labeled as static double. This tells us the type of the variable so we know how to use it. On the right, we see the name of the variables followed by a brief description of it. If you are looking at this on the Internet, you can click on the name label (E or PI) to be taken to a more detailed description of the variable. We don’t generally need more information on variables, but the links are provided in the API just in case.

7. The “Method Summary” section has all of the available methods provided by the Math class. Once again, we can see that the table is laid out in a similar manner, with a small section on the left and a larger section on the right. With the first listed method, abs, we can see the left section has the same label (static double) as we found with the E and PI values. However, this time it is telling us the return type of the methods. On the right, we again get the name and a brief description but we also get the arguments that must be passed to the method. Clicking on the name to go to the full description of methods is often much more useful than with the variables. If we click on the name `abs` we find much more information than we originally had with the short description:

abs

```
public static double abs(double a)
```

Returns the absolute value of a `double` value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned. Special cases:

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is the same as the value of the expression:

```
Double.longBitsToDouble((Double.doubleToLongBits(a)<<1)>>>1)
```

Parameters:
`a` - the argument whose absolute value is to be determined

Returns:
the absolute value of the argument.

8. Here we gain much more information about the abs method and what its purpose is. We learn some basics about what it does to the value. It even gives us some special cases that, while rare, can still occur. Note: NaN means Not a Number.

9. Here are some examples using the abs method.

Math.abs(25) -> 25	Math.abs(-25) -> 25
Math.abs(0) -> 0	Math.abs(17/5) -> 3

G. Javadoc Tool

1. The basics of creating your own APIs are pretty simple. When you add comments in your code, you can use the tag /**…*/ before each class, variable, constructor, and method to create block comments. These comments work within your code in essentially the same way as the regular block comment tag /*…*/. However, once we run the Javadoc tool, APIs will be created based on these comments.

2. The first line of the comment should be a quick description that sums up what it is in front of. This first line will turn into the quick description that we discussed earlier. The rest of your paragraph should consist of a more detailed description of the item.

3. When you run your javadoc.exe program on your Java class file (as discussed in Handout A6.1, *Javadocs*), it will create a few .html files in the local directory. If you open up index.html with your Web browser, you will find yourself looking at an API created for your class.

SUMMARY/ REVIEW:

As a Java programmer, you want to avoid redoing work already done. This is why we try to design methods and classes that do small jobs and can therefore be reused in other spots. Pre-made java classes help with this process by providing a huge library of commonly needed classes. APIs are our instruction books, which we use to understand the purpose and applicability of these classes.

ASSIGNMENT:

Lab Assignment A6.1, *Taxes*
Lab Assignment A6.2, *RegularPolygon*
Handout A6.1, *Javadocs*
Worksheet A6.1, *API Search*
Worksheet A6.2, *Static Review*