More on Identifiers

After covering the basics in Lesson_02, we know how to give proper names or “identifiers” in our programs, and that these identifiers should be both *unique* and *descriptive*. In addition to following Java’s rules, there are many “best practice” rules we as programmers can follow to make our code readable. For example, Identifiers that have 2 or more words can be difficult to read. Since we can’t use spaces, starting new words in our identifiers with capital letters helps distinguish between the two words.

For example, `firstName` might be easier to read than `firstname`. In addition, we can use underscores instead of spaces between words in a multi-word name. Using the previous example, you can see that `first_name` is much more readable as well.

Importing Packages and Classes

We learned in Lesson_01 that Java programs are contained in Classes. Remember that we enclose our programs in `public class className {}`. We will discuss classes in detail later on in this course. For now you only need to know that a class is a container for a program. Sometimes Java programs are organized into groups of classes called packages. Even Java itself could be considered a package (called java). However, Java has many other packages and classes we can use in our programs. In order to use these packages, we must write an **import statement** that imports them into our program. An import statement for an entire package is formatted as shown below:

```
import packageName.*;
```

This tells the program to import the package `package`. The `.*` tells the program to import all of the classes inside `package`.

When we want to import a single class inside a package we write:

```
import packageName.ClassName;
```

This tells Java that we want to import the class `Class` inside of the package `package`. The Java packaging system can also use subpackages within other packages, creating a simple yet robust hierarchy of packages, subpackages, and classes. We use the same dot notation to address subpackages. For example, to import the class Class in the subpackage subPackage inside of the package package, we use the following notation.

```
import packageName.subPackageName.ClassName
```

Each dot in the notation denotes another step down in the hierarchy.

The Scanner Class

Scanner is one of a number of Java’s built – in classes. It is contained in the package `util`, which like all other Java packages is a subpackage of `java`. To import the Scanner class, we add the following to the top of our program:

```
import java.util.Scanner;
```

The Scanner scans various forms of user input and parses them. In programming, the term **parse** means to analyze a block of data and return the primitive values that it contains. Therefore, the Scanner class is designed to get primitive data values from user input.

Creating a Scanner Object

Before we go further, we need to have a deeper understanding of the concept of a class. Think of classes as the DNA of objects. A class is not an object itself, but simply a framework for building one particular type of object. To use the DNA example, I have a friend named Ulysses. We could take his DNA and make clones (though I don’t think anyone would want to). But Ulysses’ DNA itself would not allow me to hear his voice or give him a high five (also, high fives are lame). In other words, Ulysses’ DNA would not give us the use any of the functions of Ulysses himself.

Classes work much the same way. In order to use the Scanner class and all of its functions, we create a new object (like a clone) that is an **instance** of the class. Below is a block of example code that instantiates a new Scanner object.

```
Scanner keyboard = new Scanner(System.in);
```

This code tells Java first that we are initializing a `Scanner` object called `keyboard`, and that it will be a new `Scanner` object. Inside the parentheses, `System.in` tells Java that `Scanner` will read text from the system input, which is the text typed into the keyboard.

Prompting the User for Input

In order to use the scanner object, we need to prompt the user for input. For keyboard input, we just use a simple `println` statement.

```
System.out.println("[Prompt]");
```

Methods

In Lesson_01 we learned that primitive data values describe the state of our objects. But on object also has behavior. In object-oriented programming, a **method** describes the behavior of an object. We can control the behavior of our scanner object by calling its methods. We call an object’s methods by typing:

```
object.method()
```

With the Scanner object, we set a variable equal to the method call in order to use the data. For example, if we wanted our `keyboard` object to take `int` values from our keyboard input, we would type:

```
int num = keyboard.nextInt();
```

This declares a new `int` variable called `num` and initializes it to the value of the next integer that `keyboard.nextInt()` finds in our input from the keyboard. We can now use the integer, in this case by just printing it out. See the complete program below.

```java
import java.util.Scanner; //import statement

public class Packages
{
        public static void main(String[]args)
        {
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Please Enter an integer: ");
        int num = keyboard.nextInt();

        System.out.println(num);

        }
}
```

This program yeilds the following output:

```
Please Enter an integer:
45
You typed the number 45
```

Please watch the video below to see a demonstration of the use of Scanner objects in Intellij.

We will go over methods in greater detail in the next lesson, and even write our own. For now, just understand that the Scanner object has the following methods that we can use to control how it behaves.

Scanner Methods in the AP Subset

Name	Return Value
`nextInt()`	returns the next int value in the input
`nextDouble()`	returns the next double value in the input
`nextFloat()`	returns the next float value in the input
`nextLong()`	returns the next long value in the input
`nextByte()`	returns the next byte value in the input
`nextShort()`	returns the next short value in the input
`next()`	returns the next one word string value in the input
`nextLine()`	returns the next multi-word String value in the input

APLab_04

EX_01

Create a new class called RudeAI. Write the program to ask you the following questions:

```
What is your name?

How old are you?

What do you do for fun?

What kind of music do you like?

How many siblings do you have?

What do you want to be when you grow up?
```

Then write print statements to give rude responses such as the ones below. Use variables in your print statements so that the responses are specific to user inputs. You can use the same responses as me or createyour own

```
Hi, my name is RudeAI.

I'd like to ask you a few questions.

What is your name?

Professor_Handsome

Professor_Handsome?!!! Why would anyone name a baby that?

How old are you, Professor_Handsome?

20

Oooooo!!! 20 is getting up there.

What do you do for fun, Professor_Handsome?
```

```
Write_Code
I thought only nerds like to Write_Code?
What kind of music to you like?
Hip-Hop
Boooo!!! I wouldn't wish the sound of Hip-Hop on my worst enemy.
How many siblings do you have?
3
3? Wow, I hope the rest of your family is better looking.
What do you want to be when you grow up?
Computer_Science_Teacher
I think you'd have to be smarter to be a Computer_Science_Teacher.
So Professor_Handsome, you're 20...
You like to Write_Code and listen to Hip-Hop...
Good luck becoming a Computer_Science_Teacher.
I'm only kiddin' Professor_Handsome.
```

EX_02

Create a BMI calculator with user input.

Your code should do the following:

1. Ask for the users height and weight
2. Perform the BMI calculation (google search “BMI” formula)
3. Print the results

EX_03

Create a program that makes a calculation you choose. Keep it simple. Just focus on collecting user input to make calculations. Your code should:

1. Collect user input into variables
2. Perform a function with the data
3. Print the results
4. Be cool