Introduction to Libraries

by Sophia

WHAT'S COVERED

In this lesson, you will review the use of Java libraries that you have already learned about. You will then learn about the use of a few new libraries that are part of the standard Java language. Specifically, this lesson covers:

1. Using import to Access Library Classes
2. Working with Standard Library Classes and Libraries Used Already
3. The java.io and java.nio Library Packages

1. Using import to Access Library Classes

From early in the course, you have been making use of Java's standard libraries. To begin working with a `Scanner` to read user input, you have needed to add a line like this to the code:

EXAMPLE

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

This statement makes the `Scanner` class from the standard **library classes** available for use in the program. The import keyword is followed by the package `java.util`, which identifies the section of the `Java` library we are accessing, and the capitalized class name, `Scanner`, indicates the specific library class.

There are thousands of classes included in the libraries that come with Java. The **packages** are both a way of organizing the libraries and a way to avoid name collisions.

IN CONTEXT

You might think of the package name as being equivalent to an area code in a phone number. A number of people around the country (or the world) may have the local number 234-5678. To make

sure the correct person is reached, the area code is included. The 212 area code reaches the person in Manhattan with the phone number 234-5678, while the 312 area code is used to reach the subscriber with that number in Chicago.

KEY CONCEPT

Instead of specifying the specific class in a package that needs to be imported, we can use a wildcard character to import all classes in a package.

It is also possible to use the wildcard character * to specify that all classes in a package should be imported, as demonstrated in the following example:

EXAMPLE

```
import java.util.*;
```

This line imports the `Scanner` class along with everything else in the `java.util` package. It is sometimes the case that different packages may contain classes with the same name. In such cases, using the wildcard * to import all classes in a package may lead to "name collisions." It may not be clear which class is actually needed, so the compiler can't finish its work. In such cases, using more specific imports of specific classes can be useful.

TERMS TO KNOW

Package
A package is a collection of interrelated classes in a particular section of the Java class library.

Library Class
A class that is included in the libraries that come with the Java Virtual Machine and developer tools and is available on all machines running a given version of Java.

2. Working with Standard Library Classes and Libraries Used Already

Once a class (or all of the classes in a package) have been imported, the objects and methods provided by the class are available for use like local classes in the application. From early on in the course, you have made use of the java.util `Scanner` class (part of the java.util package).

EXAMPLE A number of programs have featured this statement:

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

In an earlier discussion of exception handling using try and catch, the code used a very general `Exception` in the catch clause (which did not require any further imports).

EXAMPLE You might, though, have added the import for:

```
import java.util.InputMismatchException;
```

If you added the import, it would have been possible to catch a more specific exception.

Your program could look like this:

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

public class ScannerException {

  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Please enter your age in years: ");
    int age = 0;
    try {
      age = input.nextInt();
    }
    catch(InputMismatchException ex) {
      System.out.println("You did not enter a valid integer.");
      System.out.println("Please run again & enter a valid age.");
    }
    System.out.println("You are " + age + " years old.");
  }
}
```

As noted in the previous lesson, this code could be simplified by one line if it used a wildcard when importing library classes, as demonstrated below:

EXAMPLE

```
import java.util.*;
```

We have also met the `java.util.Arrays` utility class when printing the contents of arrays, sorting them, and copying them. Note the use of the following line that you used earlier in the import class:

EXAMPLE

```
import java.util.Arrays;
```

As a reminder, here is a brief program that makes use of `Array.sort()` and `Arrays.toString()`.

```
import java.util.Arrays;

class NumbersArraySort {
  public static void main(String[] args) {
    int[] numArray = {2, 45, 9, 17, 1, 2};
    System.out.println("Original array: " + Arrays.toString(numArray));
    Arrays.sort(numArray);
    System.out.println("Sorted array: " + Arrays.toString(numArray));
  }
}
```

If you enter this code in the IDE (in a file named NumbersArraySort.java) and run it, the results should look like this:

```
Original array: [2, 45, 9, 17, 1, 2]
Sorted array: [1, 2, 2, 9, 17, 45]
```

The Arrays class has a plural name to indicate that it includes utility methods that are useful when working with Java arrays. In the next section, we will be briefly introduced to a Files class that includes common utility methods for working with files.

The `java.util` library package also includes collections that we have worked with, such as `ArrayList`, `HashMap`, and `HashSet`.

Here is an example that we saw back in lesson 2.1.5 that uses a `HashSet`, and so has to have the import statement for `java.util.HashSet`:

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

public class ScoresHashMap {

  public static void main(String[] args) {
    // HashMap holds key-value pairs.
    // The key (user ID) is a String (case sensitive).
    // The value (score) is an Integer (int)
    HashMap<String, Integer> scores = new HashMap<>();
    scores.put("ssmith04", 88);
```

```
    scores.put("tlang01", 100);
    scores.put("glewis03", 99);
    System.out.println("Scores: " + scores.toString());

    Scanner input = new Scanner(System.in);

    System.out.print("Enter an ID: ");
    String id = input.nextLine();
    // Check if the HashMap contains the key (id)
    if(scores.containsKey(id)) {
      // Only safe to use get() to retrieve value if key exists in HashMap
      int score = scores.get(id);
      System.out.println(id + " has a score of " + score + ".");
    }
    else {
      System.out.println("There is no score for " + id + ".");
    }
  }
}
```

If entered into a file named ScoresHashMap.java in the IDE, the program should produce output like the following:

```
Scores: {ssmith04=88, glewis03=99, tlang01=100}
Enter an ID: tlang01
tlang01 has a score of 100.
```

As the examples from earlier in the course show, the standard libraries included with Java have already been useful.

3. The java.io and java.nio Library Packages

In its history of more than 25 years, the Java libraries have evolved and changed. The devices that run Java have changed as well. The upcoming lessons will deal with file input and output. The libraries for working with files have changed over the years. To accommodate this growth and change, Java has ended up with classes for working with files that are organized into two library packages: `java.io` and `java.nio`.

The `java.io` package is the older of the two and goes back to the first version of Java. This package includes the File class that we will use in many of the upcoming programs.

The `java.io` package also provides important exception types when working with files: `FileNotFoundException` and `IOException`. Since the File object will be used for all of the code we will write for working with files, this library will play a key role.

BIG IDEA

The `java.nio` package provides the `Files` class. The plural name of this class indicates that it provides common utility methods for working with files. The functionality provided by java.nio provides simpler approaches to working with files that we will cover in the upcoming lessons. As you move further on in Java (after this course), you will want to look into older approaches to working with files using input and output streams, but for our purposes, the simpler approaches provided by `java.nio Files` will be ideal.

HINT

The File and Files classes are two different classes in different packages—even though the names are similar.

SUMMARY

In this lesson, you learned how to **access Java library packages and classes**. You learned how they are added to a program to be used. You have worked with classes provided by the **standard libraries** from early on in the course, by necessity. In this tutorial, you learned about a few common package examples. Finally, we looked ahead to a couple of key libraries that we will use for working with files in the upcoming lessons, which include **java.io** and **java.nio**.

TERMS TO KNOW

Library Class

A class that is included in the libraries that come with the Java Virtual Machine and developer tools and is available on all machines running a given version of Java.

Package

A package is a collection of interrelated classes in a particular section of the Java class library.