

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

ORACLE

Academy

Java Foundations

4-2

The `import` Declaration and Packages

ORACLE
Academy



Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Objectives

- This lesson covers the following objectives:
 - Access a class by using its fully qualified name
 - Describe the function of the `import` statement
 - Use the `import` statement to access a class in a package
 - Understand the purpose of an asterisk in an `import` statement
 - Identify packages that are automatically imported



Why Should You Reinvent the Wheel?

- Frequently, you may rewrite the same Java code for different programs
- As an alternative to rewriting the same code, you can use the Java-provided library, which organizes frequently used code
- This library is called the Java class library
- The Java class library documentation is available here:
– <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/module-summary.html>

Packages in the Java Class Library

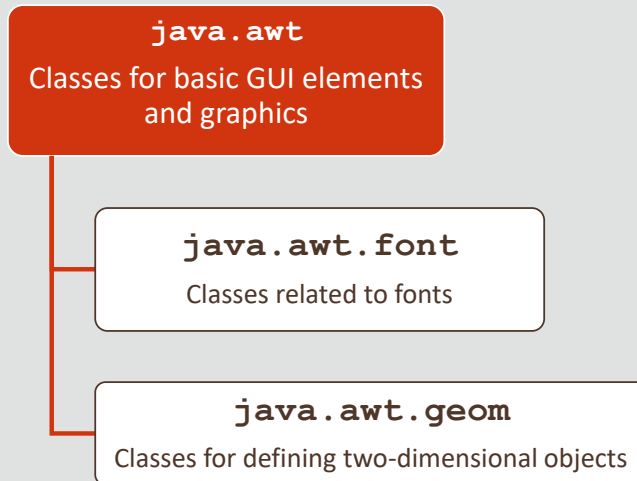
- The classes of the Java class library are organized into packages
- A package contains a group of related classes
- With a package, it becomes easier to locate the related classes

Packages in the Java Class Library

Package	Purpose
<code>java.lang</code>	Provides classes that are fundamental to the design of the Java language
<code>javax.swing</code>	Provides classes to build GUI components
<code>java.net</code>	Provides classes for networking applications
<code>java.time</code>	Provides classes for dates, times, instants, and durations

How Are the Packages Organized?

- The vast collection of classes are organized into a tree-like hierarchy, which allows packages to be divided into subpackages, like this:



Javadoc Tutorial

- From Section 4 of the Java Foundations Learning Path:
 - Access and complete the Javadoc Tutorial

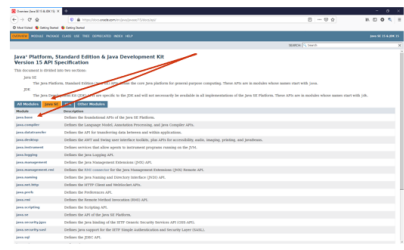
ORACLE
Academy

academy.oracle.com

Javadoc Tutorial

How to access and use the documentation for the Java SE Development Kit, or JDK - Versions 9 and later - for this tutorial, we will use JDK Version 15

1. Access and bookmark this link:
 - a. [Java Platform, Standard Edition & JDK Version 15 API Specification](https://docs.oracle.com/javase/15/docs/api/index.html):
<https://docs.oracle.com/javase/15/docs/api/index.html>
2. The documentation we will use in this course is under Java SE - Select the Java SE tab
3. The classes are grouped by module - in this course, you will be working with classes in the java.base module - Select the java.base link



Copyright © 2021, Oracle and/or its affiliates. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Using a Class from a Package

- To use a class from a package in your program, you need to specify its fully qualified name
- For example, to use the `Scanner` class to read a keyboard input, the fully qualified name for the `Scanner` class, which is defined in the `java.util` package, is:

`java.util.Scanner`

Package **Class Name**

Using the Fully Qualified Class Name

- As you can see, using the fully qualified name creates very long names for classes
- Long names reduce the readability of the code and also make coding difficult

```
public static void main(String[] args) {    Fully Qualified Class Name
    int num;
    java.util.Scanner keyboard = new java.util.Scanner(System.in);
    System.out.print("Enter a number");
    num = keyboard.nextInt();
    System.out.print("The number you entered is" + num);
} //end method main
```

Is There an Alternative to the Fully Qualified Name?

- Suppose that you have a friend whose name is Santi Inez Luis Vidal
 - How tedious is it to call him by his full name every time?
 - If you could simply refer to him as “Santi,” it would be so much more convenient
- Similarly, accessing Java classes by using fully qualified names is equally tedious in your programs
- Let’s see if there’s a way to specify only the name of the class instead of its fully qualified name

Using the `import` Statement

- You can avoid the fully qualified class name by using the `import` statement
- You place the `import` statement above your class definition. It looks like this:

- `import package.className`

- Example:

```
import java.util.Scanner;

public class AddNums {
    //class code goes here
} //end class AddNums
```

How Do You Import a Single Class?

- You can import a single class or an entire package
- To import a single class into your program, you write an `import` statement like this:

```
import javax.swing.JOptionPane;
```

Package Name

Class Name

`import` keyword
followed by the name of
the package dot, the
name of the class

`javax.swing` Package

- Java has an extensive library for constructing GUIs
- This library, called `swing`, can be imported into your program to give you access to Java's GUI functionality
- The `swing` library is in the `javax.swing` package

Accessing a Class from the `swing` Package

- The `swing` package has a `JOptionPane` class
- This class creates a pop-up window that can be used to display strings of text to the user
- To use the `JOptionPane` class, you must first import it into your class:

```
import javax.swing.JOptionPane;
```

```
public class Welcome {  
    //class code to go here  
} //end class Welcome
```

import statement that imports a single class, `JOptionPane`, from the `swing` package

Importing the JOptionPane Class

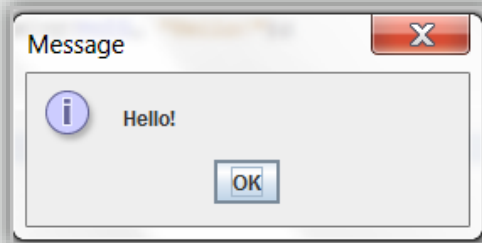
- You can then use `JOptionPane` to display text by calling the `showMessageDialog` method inside the `JOptionPane` class

```
import javax.swing.JOptionPane;

public class Welcome {

    public static void main(String[] args) {
        JOptionPane.showMessageDialog(null, "Hello!");
    } //end method main
} //end class Welcome
```


Output Looks Like This!



How Do You Import All Classes in a Package?

- You can import all classes in a particular package by using the `*` wildcard character in the `import` statement

How Do You Import All Classes in a Package?

- Let's say you want to extend the previous example by creating an instance of the `JFrame` class and add its reference to `JOptionPane`, like this:

```
import javax.swing.JOptionPane;
import javax.swing.JFrame; }
```

Importing 2 classes from the
swing package

```
public class Welcome {

    public static void main(String[] args) {
        JFrame frame = new JFrame();
        JOptionPane.showMessageDialog(frame, "Hello!");
    } //end method main
} //end class Welcome
```

Accessing All Classes from the `swing` Package

- As you access more classes from the `swing` package in your program, the number of `import` statements also increases

Accessing All Classes from the `swing` Package

- To avoid this, you can import all classes from the `swing` package by using the `*` wildcard character in the `import` statement, like this:

```
import javax.swing.*;
```

Replace all `import` statements of the previous example with one `import` statement

```
public class Welcome {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame();  
        JOptionPane.showMessageDialog(frame, "Hello!");  
    } //end method main  
} //end class Welcome
```

Including Multiple `import` Statements

- You can include multiple `import` statements in a Java program to access classes in the same package or in different packages
- For example:

```
import java.util.Date;
import java.util.Calendar;
import javax.swing.*;

public class DisplayDate {
    //class definition here
} //end class DisplayDate
```

Importing classes from the same package

Importing classes from different packages

Identify Packages That Are Automatically Imported

- So far, you have used `System.out.println()` to print text to the console

```
public class DisplayOutput {  
    public static void main(String[] args) {  
        System.out.println("Hello, how are you today?");  
    } //end method main  
} //end class DisplayOutput
```

- However, you didn't import a package for using this method in your program
- So how does Java know what to do when you call it?

java.lang Package

- If you look at the Java library, you'll see that the `System` class is organized in the `java.lang` package
- By default, the `java.lang` package is automatically imported into all Java programs

Exercise 1

- Create a new project and add the `AddImport.java` file to the project
- Examine `AddImport.java`
 - Perform the following:
 - Replace the fully qualified name to access the `JLabel` component with an `import` statement
 - To import classes from the `util` package, replace multiple `import` statements with a single `import` statement

Summary

- In this lesson, you should have learned how to:
 - Access a class by using its fully qualified name
 - Describe the function of the `import` statement
 - Use the `import` statement to access a class in a package
 - Understand the purpose of an asterisk in an `import` statement
 - Identify packages that are automatically imported



The Oracle Academy logo is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by two horizontal dark gray bars, one at the top and one at the bottom.

ORACLE

Academy