ORACLE

Academy

Java Programming

5-3

Deploying an Application

ORACLE
Academy

Objectives

- This lesson covers the following topics:
  - Describe the concept of packages
  - Describe how to deploy an application
  - Describe a complete Java application that includes a database back end

ORACLE Academy

Define a Package

- A package is a collection of Java classes
- Think of a package as a folder on your hard drive
- To define a package, specify the package name in the first line of a class definition using the keyword package

```
package graphics; //graphics is the name of our package

//Rectangle is part of the graphics package
class Rectangle{
   /* Class code is placed here */
}//end of class Rectangle
```

When creating applications in Java you create a package as a container to group all of your class files together.

Using Java Files Stored in a Package

- When you want your program to use the Java files that are stored in a package, make sure that the package is listed in the CLASSPATH setting
- This way the files can be found and used
- A class can only be in one package

Good organisation of your classes makes programming your applications easier. Knowing what packages all of your classes are in allows you to call them in multiple applications.

Naming a Package

- With programmers worldwide writing classes and interfaces using the Java programming language, it is likely that some will use the same name for different types
- In fact, the previous example (slide 4) does just that: It defines a Rectangle class when there is already a Rectangle class in the java.awt package
- Still, the compiler allows both classes to have the same name if they are in different packages

It is important to remember that your package makes up part of the fully qualified name of your class.

Fully Qualified Names and Package Names

- The fully qualified name of each Rectangle class includes the package name
- The fully qualified name of the Rectangle class in the graphics package is graphics.Rectangle

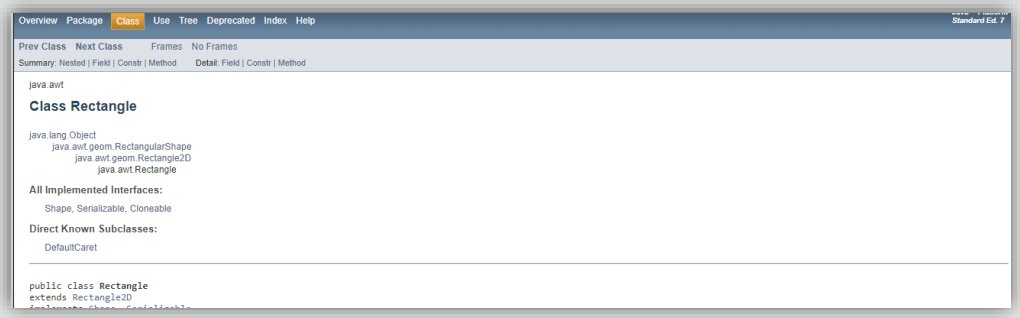
```
package graphics; //graphics is the name of our package

//Rectangle is part of the graphics package
class Rectangle{
   /* Class code is placed here */
}//end of class Rectangle
```

It is important to remember that your package makes up part of the fully qualified name of your class.

Fully Qualified Names and Package Names

- The fully qualified name of the Rectangle class in the java.awt package is java.awt.Rectangle



- This works well unless two independent programmers use the same name for their packages
- However, convention prevents this problem

Package Naming Conventions

- Package names are written in all lower case to avoid conflict with the names of classes or interfaces
- Companies use their reversed Internet domain name to begin their package names
- For example, com.example.mypackage for a package named mypackage created by a programmer at example.com
- If a programmer at code.org also created a package named mypackage, then the package name would need to be org.code.mypackage to follow the conventions established and to avoid collisions with
  - com.example.mypackage

Name Collisions

- Name collisions that occur within a single company need to be handled by convention within that company
- For example, include the region or the project name after the company name (com.example.region.mypackage)
- Packages in the Java language itself begin with java. or javax

At a local level (within an organization) the naming conventions should be specified in a standards document.

When Internet Domain Name Is Invalid

- In some cases, the internet domain name may not be a valid package name
- This can occur if:
  - The domain name contains a hyphen or other special character
  - The package name begins with a digit or other character that is illegal to use as the beginning of a Java name
  - The package name contains a reserved Java keyword, such as "int"
- When this happens, the suggested convention is to add an underscore

Legalizing Package Names

Domain Name	Package Name Prefix	Information About Conversion to Package Name
hyphenated-name.example.org	org.example.hyphenated_name	The hyphen in the domain name was replaced by an underscore in the package name.
example.int	int_.example	The word int in the domain name is a keyword so it is followed by an underscore in the package name.
123name.example.com	com.example._123name	The name with the number is prefaced with an underscore in the package name.

How to Use a Package

- To use a public package member from outside its package, you must do one of the following:
  - Refer to the member by its fully qualified name
  - Import the package member
  - Import the member's entire package

It is a common mistake to try to access a class from a different package without importing that package in your code. Always import the package to your application before attempting to use an external class.

Refer to a Package by Fully Qualified Name

- In the past, you may have imported packages to get certain classes for use in your programs
- You can skip importing and refer to a package by the fully qualified name
- The fully qualified name for the Rectangle class declared in the graphics package in the earlier examples is:

```
graphics.Rectangle
```

Refer Without Importing Class or Package

- To refer to it in your program without importing the class or package:

```
graphics.Rectangle myRect = new graphics.Rectangle();
```

- Qualified names are fine for infrequent use but should only be used when accessing the external class a limited number of times
- When a name is used repetitively, typing becomes tedious and produces difficult code to read
- As an alternative, you can import the member or its package and then use its simple name

Importing a Package Member

- To import a specific member into the current file, put an import statement at the beginning of the code before any type definitions but after the package statement (if there is one)
- Here is how you would import the Rectangle class from the graphics package created in the previous section

```
import graphics.Rectangle;
```

- Now you can refer to the Rectangle class by its simple name

```
Rectangle myRectangle = new Rectangle();
```

As you can see this way is much more meaningful and efficient.

Import Entire Package

- If you use many types from a package, you should import the entire package
- To import all the types contained in a particular package, use the import statement with the asterisk (*) wildcard character

```
import graphics.*;
```

Be careful when importing packages using the asterisk. This should only be done if you are using all or nearly all of the package, it should never be done just for ease of use.

Import Entire Package

- Now you can refer to any class or interface in the graphics package by its simple name
- The asterisk in the import statement can only be used to specify all the classes within a package

```
Circle myCircle = new Circle();
Rectangle myRectangle = new Rectangle();
```

Hierarchies of Packages

- At first, packages appear to be hierarchical, but they are not
- For example the Java API includes a
  - java.awt package
  - java.awt.color package
  - java.awt.font package
  - and many others that begin with java.awt
- The java.awt.color package, the java.awt.font package, and the other java.awt.xxxx packages are not included when importing the java.awt package

Hierarchies of Packages

- The prefix java.awt (the Java Abstract Window Toolkit) is used for a number of related packages to make the relationship evident, but not to show inclusion

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environmen
java.awt.image	Provides classes for creating and modifying images

Remember that the asterisk imports all of the classes at the level but not subpackages.

Importing Subpackages

- When you import a package, you only get the classes at that package level
- Sub packages are not included and will need to be imported separately
- Unused imported packages just adds unnecessary code to your application

It is good programming practice to only include the packages that you will directly use.

Importing java.awt.*

- Importing java.awt.* imports all of the types in the java.awt package, but it does not import java.awt.color, java.awt.font, or any other java.awt.xxxx packages
- If you plan to use the classes and other types in java.awt.color as well as those in java.awt, you must import both packages with all their files:

```
import java.awt.*;
import java.awt.color.*;
```

Deploying an Application

- Eventually you are ready to deploy your application so that users can run it from outside the development environment (IDE)
- There are two types of deployment technologies available in JavaTM 2 Standard Edition (J2SE) for deploying client-side Java applications on the desktop:
  - Java Plug-in
  - Java Web Start

Deploying an application allows you to make it available to others to use outside of the development environment.

Java Plug-In

- Java Plug-in:
  - Is a tool used to deploy Java applets that run inside a web browser
  - Uses Web Start to deploy standalone Java applications on the desktop, using JNLP (Java Network Launching Protocol)
- Supported web browsers include:
  - Internet Explorer, Mozilla Firefox, Google Chrome and Apple Safari and Microsoft Edge all run Java although some will require some modifications to the browser settings
  - Most of the modern browsers have moved away from allowing the running of Java Applets so Oracle recommends Microsoft Internet Explorer 11 that still supports NPAPI plug-ins

Steps to Deploying an Application (Using the command line)

1. Compile your application's Java code and make sure that all class files and resources such as images are in a separate directory
2. Create a JAR file containing your application's class files and resources by running appropriate command line commands:

```
% cd path/to/classes
% jar cvf  Sample.jar  SampleCode
```

3. Create a JNLP file that describes how your application/applet should be launched
4. There are different versions for applets and applications.
   - Refer to the following reference site for various code samples:
     - http://docs.oracle.com/javase/tutorial/deployment/

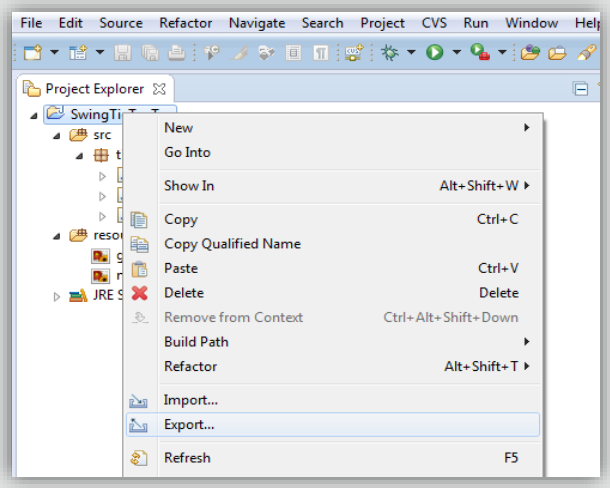
Steps to Deploying an Application (Using the command line)

4. Create the HTML page from which your application/applet will be launched
5. Invoke Deployment Toolkit functions to deploy the Java Web Start application/applet
6. Again refer to the site in Step 3 for code samples
7. Place the application's JAR file, JNLP file, and HTML page in the appropriate folders
8. Open the application's HTML page in a browser to view the application/applet
9. Check the Java Console log for error and debugging messages

Steps to Deploying an Application (Using the Eclipse IDE)

- Creating a runnable JAR file in Eclipse
- This is a relatively straight forward process

1. Choose File or Right click the project file folder on the project explorer window and choose Export from the menu

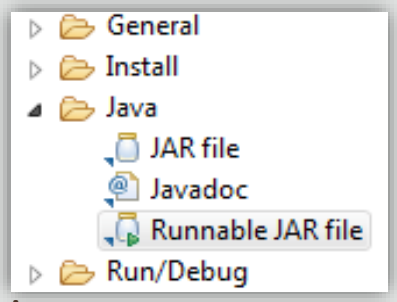


ORACLE Academy

The JAR file sits on your computer system similar to an exe file and can be run through the operating system.

Steps to Deploying an Application (Using the Eclipse IDE)

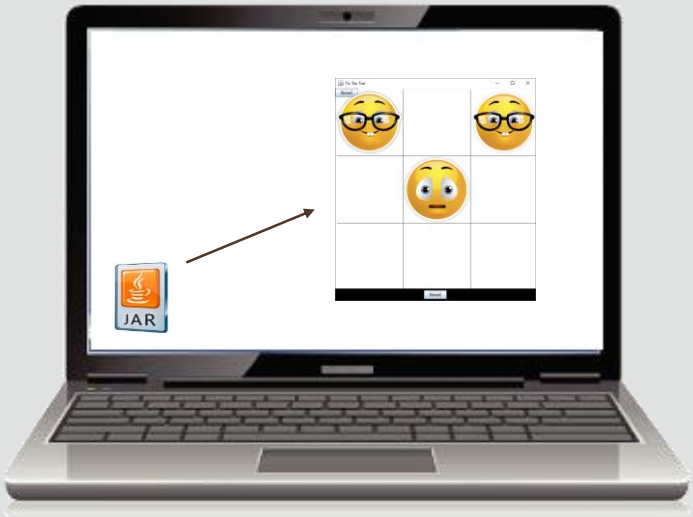
2. From the dialog box open the Java option and then choose runnable JAR file from the list



3. Click on the next button
4. On the screen configure the options as you want
   - Launch Configuration - This is the driver class of your application
   - Export Destination - Where you want to save the JAR file
   - Library Handling - Leave this as the default
   - ANT File allows you to create documentation that will recreate the JAR file for you in the future, save this in your workspace

Steps to Deploying an Application (Using the Eclipse IDE)

5. Go to your save location and double click the JAR file
6. Your program should run



Creating a JavaFX JAR file

- Open JavaFX_Tutorial.pdf for instructions to complete the tasks for creating a JavaFX JAR file
- You will require the JavaFX_Graphics.zip file to complete the tutorial!
- Complete all the tasks in the tutorial
  - Download and install JavaFX in Eclipse
  - Create a JavaFX appliction
  - Test the application
  - Create and a Runnable JAR file for the application
  - Run the application outside the development environment

Two Tier and Three Tier Architecture

- Up to this point, you have written programs that are one tier architecture systems
- One tier architecture systems:
  - Run on one computer at a time
  - Are more complex to program as you add layers and functionality
- Functionality types may include:
  - Server and client processes
  - Database storage and retrieval

One tier applications run on a single computer system and can be thought of as stand alone applications.

Tier Architecture

- The concept of tiers provides a convenient way to group different classes of computer architecture
- If your application is running on a single computer, it has a one tier architecture
- If your application is running on two computers, such as a typical web server application that runs on a Web browser (client) and a Web server, then it has two tiers

Most Java applications are created to run on two or three tier architectures as Java makes it relatively simple to join multiple applications and data storage systems together.

Two and Three Tier Systems

- In a two tier system, you have a client program and a server program
- The server responds to requests from many different clients
- The clients usually initiate the requests for information from a single server
- A three tier application adds a third program to the mix, such as a database, in which the server stores its data

These applications are usually coded so that the tier architecture is transparent to the user.

Two Tier Architecture Deployment

- A two tier system usually consists of a client and a server program
- The server program runs on a web server
- The client programs could be applets running via html webpages
- Applets would communicate with the server program to get instructions

Two Tier Architecture Deployment Example

- An example of two tier architecture is a two player chess game
- Each user would access the client application, but a server application would be needed to handle the communications between the clients and determine game play
- To deploy such a system one would need to set the server application to auto start on the web server and have the Java applets communicate with the server via http protocols

Three Tier Architecture Deployment Example

- To turn the chess game into a three tier system, we could add a database to store the results of a chess game for later rankings and retrieval of user information
- Java uses JDBC connectivity to communicate with a variety of databases
- To deploy this implementation:
  - A database needs to run on a server
  - The Java server application would need to have connectivity using the JDBC driver and appropriate ports to access the database

Three Tier Architecture Deployment Example

- Using a database as the main storage for data is now the most common way of storing and retrieving information within applications
- The method of doing this has been greatly simplified over the years and it is now a fairly straightforward procedure to connect your application to a database back end
- This will be explored in more detail in the next section!

Terminology

- Key terms used in this lesson included:
  - Deployment
  - HTML files
  - Jar files
  - JNLP files
  - Package
  - Three tier architecture
  - Two tier architecture

Summary

- In this lesson, you should have learned how to:
  - Describe the concept of packages
  - Describe how to deploy an application
  - Describe a complete Java application that includes a database back end


ORACLE
Academy