



Java Programming

4-1

Deploying an Application



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

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`.
- For example:

```
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
```

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.

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.

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`.
- 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.

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.

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.
- When a name is used repetitively, however, typing the name repeatedly becomes tedious and the code becomes difficult 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 file before any type definitions but after the package statement, if there is one.
- For example, 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();
```

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.*;
```

- 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, a `java.awt.color` package, a `java.awt.font` package, and many others that begin with `java.awt`.
- However, the `java.awt.color` package, the `java.awt.font` package, and other `java.awt.xxxx` packages are not included in the `java.awt` package.
- 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.

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.

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 Java™ 2 Standard Edition (J2SE) for deploying client-side Java applications on the desktop:
 - Java Plug-in
 - Java Web Start

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.

Steps to Deploying an Application (command line)

- Compile your application's Java code and make sure that all class files and resources such as images are in a separate directory.
- 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
```

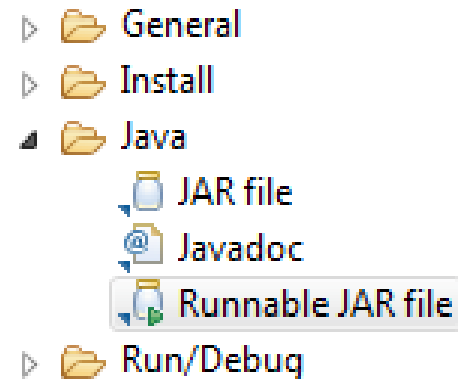
- Create a JNLP file that describes how your application/applet should be launched. There are different versions for applets and applications.
- Refer to the following reference site for various code samples:
<http://docs.oracle.com/javase/tutorial/deployment/>

Steps to Deploying an Application (command line)

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

Steps to Deploying an Application (eclipse)

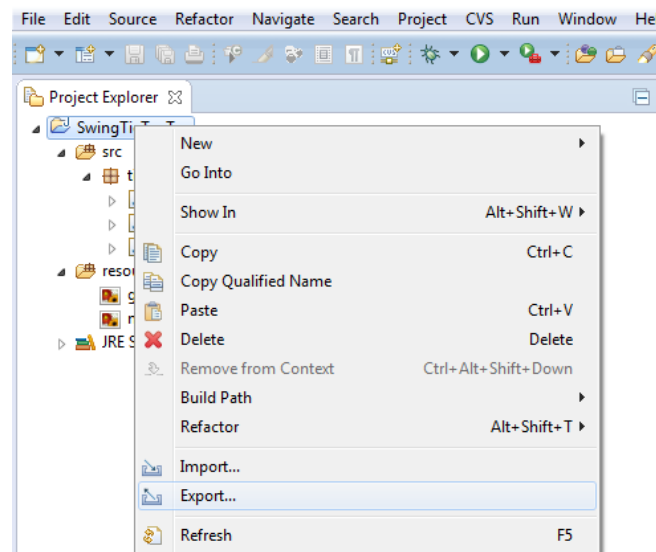
- From the dialog box open the Java option and then choose runnable JAR file from the list.



- Click on the next button.
- On the screen configure the options as you want.
- Go to your save location and double click the JAR file.
- Your program should run.

Steps to Deploying an Application (eclipse)

- Creating a runnable JAR file in Eclipse
- This is a relatively straight forward process.
- Choose File or Right click the project file folder on the project explorer window and choose Export from the menu.



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.

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.

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.

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.

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

