13

Packaging and Deploying Applications

ORACLE

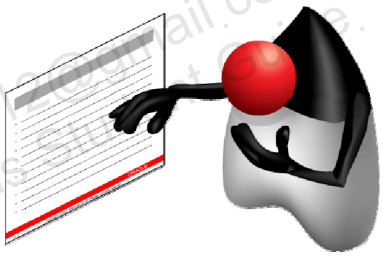
Objectives

After completing this lesson, you should be able to:

- Describe a JAR file and the steps for creating one
- Create a manifest file that uses headers
- Create a JAR by using development tools
- Deploy a stand-alone JAR
- Deploy a JAR as an applet
- Deploy a JAR by using Java Web Start

Topics

- Packaging
- Deployment
- JavaFX Deployment



Packaging Introduction

- Running a Java class
  - `java classname`
  - `java package.dir.classname`
- Examples:
  - `java BaseUi`
  - `java com.example.ui.BaseUi`
- What is the problem?
  - Not practical with hundreds of `.class` files
  - Not portable

Running a Java application is pretty straightforward. In a command line, type `java` and the class name: `java BaseUi`.

Note that if you include the `.class` extension, you will receive an error message, because Java interprets `BaseUi.class` as the "class" class in the BaseUi package.

Running your application this way works fine on a small scale, but what happens when you have dozens, if not hundreds, of class files? Then you need a better solution for organizing and distributing your files.

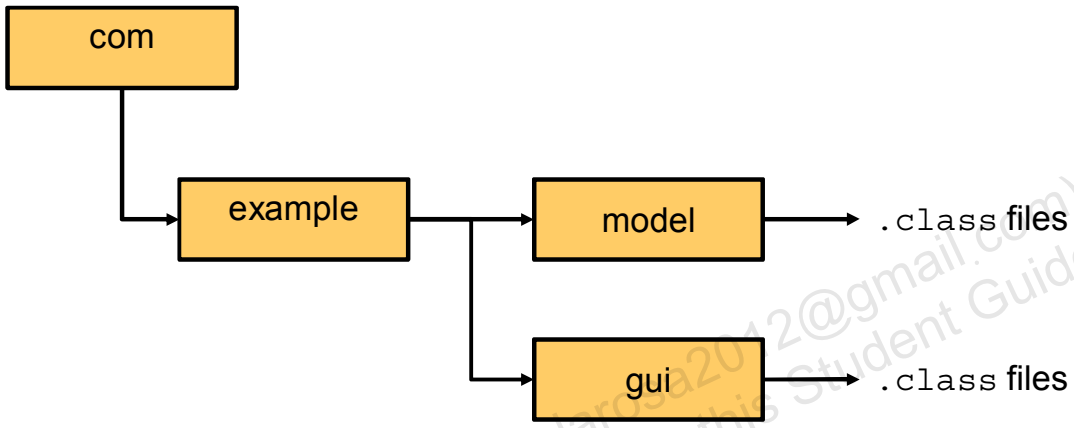
Java `jar` Files

- The `jar` utility is used to create JAR (or "`.jar`") files.
- What is a `.jar` file?
  - Essentially a `.zip` file
  - Contains all your `.class` files
  - Contains any resources needed by your program
  - Any special Java stuff
- How to you use the `jar` utility?
  - Command line
  - From another tool

Java applications are packaged and deployed using `.jar` files. These files are created using the `jar` tool. You can use the `jar` tool from the command line. However, typically the `jar` tool is most often used by other tools to create `.jar` files.

Application Example

A typical application and its packages:



This picture shows a typical directory structure for an application. Notice that the domain name precedes the directories, which contain the .class files.

`jar` Command Line

- Basic command line:
  - `jar cvf jarFileName dirName`
- Example:
  - `jar cvf BasicUi.jar com`
- Options:
  - `c`: Create the `.jar` file.
  - `v`: Produce verbose output.
  - `f`: Send the output to a file instead of to standard output.
- Running a `.jar` file:
  - `java -jar MyJar.jar`
  - Double-clicking the `.jar` file works in most situations, too.

The slide shows the basic command line options for `jar`. If you refer to the previous slide, the command in the example creates a .jar file from the "com" directory and all its subdirectories. If there are no other subdirectories in your current directory, you could also have used:

```
jar cvf BasicUi.jar *
```

However, how do we determine which class file to run?

The Manifest File

- A manifest file is created whenever a `.jar` is created.
  - Located in `META-INF` directory
  - Named `MANIFEST.MF`
- Sample contents:

```
Manifest-Version: 1.0
Created-By: 1.7.0
```


When you create a `.jar` file, the `jar` utility automatically creates a manifest file unless you specify one. The slide shows sample contents of a manifest. Each line consists of a header, the text before the colon, and its value. The first line says that the file conforms to version 1.0 of the manifest file standard. The second line says that the `jar` file was created by JDK 7.

Modifying the Manifest

- Example command line:
  - `jar cfm BaseUi.jar Manifest.txt com`
- Common headers:
  - `Main-Class:`
  - `Class-Path:`
- Example:

```
Main-Class: com.example.ui.BaseUi
Class-Path: HelperLib.jar HelperLib2.jar
```


ORACLE

You can provide your own manifest file settings in a regular text file. Just pass the file on the command line and the headers will be added to the manifest in your `.jar`. Two of the most common headers are included in the slide.

Main-Class: Sets the class that should be run when the `.jar` file is executed.

Class-Path: Specifies the `.jar` files to be included in the CLASSPATH. This saves you from having to specify the `.jar` files in the command line.

Tools for Making `.jar` Files

Commonly, tools are used to create `.jar` files.

- Ant
  - "Another Neat Tool"
  - A Java build, test, and deployment tool
  - Written in Java
- JavaFX Packager
- NetBeans
  - Automatically creates the `.jar` file for you
  - Relies on Ant scripts for building and packaging

Quiz

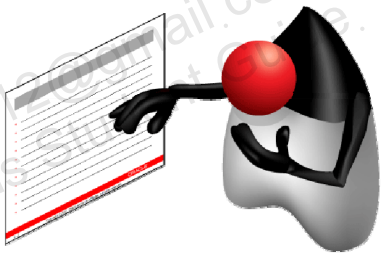
Which manifest header allows you to specify the class you should launch at run time?

a. `Run-Class:`
b. `Launch-Class:`
c. `Main-Class:`
d. `Super-Class:`

Answer: c

Topics

- Packaging
- Deployment
- JavaFX Deployment



Deploying Applications

- Distribute the `.jar` file.
  - Just make the `.jar` file available.
- Use an installer.
  - Use a professional installer to set up and install your `jar`.
- Embedded
  - Embed applications in a web page.
- Java Web Start
  - Launch a Java Application from a URL.

There are a number of ways to deploy an application. Some methods are very manual, other are much more automated.

Stand-Alone Deployment

- Double-click the `.jar` file on most platforms.
- Command line:
  - `java -jar` *`jarfilename`*`.jar`
- Put the `.jar` in an `.exe` wrapper.
- Security:
  - Full access to the system

Many Java applications are deployed as stand-alone applications. NetBeans is a great example of this type of deployment. When deployed stand-alone, a Java application has the same security access to your system resources as any other application (specifically, access to things like files, directories, and network shares).

On most operating systems, if Java (JDK or JRE) is installed, you can simply double-click the `.jar` file to launch it.

Installers and Wrappers

- Installers
  - install4j – Commercial
    - http://www.ej-technologies.com/products/install4j/overview.html
  - Izpack - Open Source (Apache)
    - http://izpack.org/
- Wrappers
  - launch4j - Open Source (BSD, MIT)
    - http://launch4j.sourceforge.net/
  - jsmooth -
    - http://jsmooth.sourceforge.net/

Note: The inclusion of products in this list is not an endorsement by Oracle.

Typically, many popular tools, such as NetBeans or Eclipse, are written in Java, but are installed just like a normal application. This is achieved through the use of an installation program or a wrapper program.

An installation program provides a complete framework for installing an application on a platform.

A wrapper program allows you to add platform-specific execution to your application. For example, you can execute your Java program on Windows by using an `.exe` file. In addition, wrapper programs can provide more functionality, such as detecting whether a JRE is present and auto-installing one if necessary.

Applets/Embedded

- Allows Java code to run within a web browser
- Introduced in the first version of Java
  - Very popular at the time (1995)
  - One of the main reasons for early Java adoption
  - Flash and AJAX caused popularity to diminish over time
- Applet functionality is still available in Java FX.
  - Applets can still be used in JDK 7.
  - However, JavaFX can be embedded in web pages.
  - Embedding Java FX is the preferred method for delivering Java in a browser.

One cannot talk about Java deployment without mentioning applets. Applets are Java-based programs, typically GUIs, that are designed to run in a web browser. Applets were released with the first version of Java in 1995 and were a big reason for the growth of interest in the language and platform.

Although applets are included in JDK 7, they are not the preferred method for including a Java FX application in a web browser.

Applet Security

- Applets introduced the concept of a sandbox.
- Unsigned applets have very limited access.
  - Cannot access the local file system
  - Cannot access another server
  - Cannot load native libraries
  - Cannot change the security manager or create a ClassLoader
- Signed applets have access to the local system.
- Java web-based deployment technologies continue to follow this model.

Applets introduced the concept of a sandbox. This sandbox greatly limits the applications access to the local system. Limitations include:

- They cannot access client resources such as the local filesystem, executable files, system clipboard, and printers.
- They cannot connect to or retrieve resources from any third-party server (any server other than the server it originated from).
- They cannot load native libraries.
- They cannot change the SecurityManager.
- They cannot create a ClassLoader.
- They cannot read certain system properties.

The only way to get access to the local system is by signing the applet. The security model is still followed for newer systems like Java Web Start and the Deployment Kit.

Java Web Start

- Designed to launch full-featured Java applications from the Internet
- Complete client applications, such as a spreadsheet or editor
- How it works:
  - A web page
  - A link to a Java Network Launch Protocol (JNLP) file
  - An installation of Java on the client machine
- Uses a security model like applets

In contrast to applets, Java Web Start (JWS) was designed to launch full Java clients from a web page, not run in the web page.

Java Web Start Benefits

- The application can be deployed from a browser.
- The application is cached locally after the initial download.
- Shortcuts can be created to the locally cached apps.
- Updates can be distributed from the network automatically.

Java Web Start offers a number of benefits. Instead of being limited to a browser windows, Java Web Start can be used to distribute full-fledged applications to the desktop. Applications can be distributed using the network and cached locally. This makes distribution and updating very easy.

Quiz

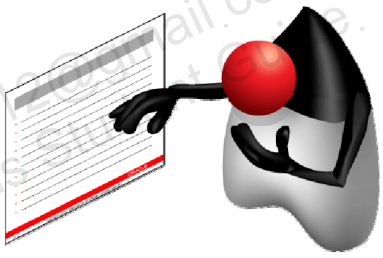
Which two deployment methods run their applications in a sandbox by default?

a. Browser-based deployment
b. Java Web Start
c. Stand-alone deployment
d. Wrapping your `.jar` in an `.exe` file

Answer: a, b

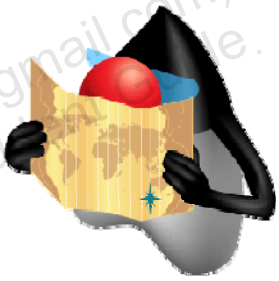
Topics

- Packaging
- Deployment
- JavaFX Deployment



Deployment Toolkit

- A set of JavaScript functions to help deploy Rich Internet Applications (RIA) written in Java
  - Correct HTML based on browser version and operating system
  - Checks that the required JRE is present
  - Prompts for installation if it is not
  - Introduced in Java 6 Update 10
- The JavaScript file is located at:
  - http://www.java.com/js/deployJava.js
  - https://www.java.com/js/deployJava.js
- For NetBeans development:
  - ./web-files/dtjava.js



The Deployment Toolkit helps deploy rich Internet applications written in Java. It corrects all the HTML so that Java and JavaFX apps will run correctly on any browser or operating system. The Deployment Toolkit is used to deploy a JavaFX application in a browser or by using Java Web Start.

If your application is deployed in an Internet-accessible environment, use the java.com URL to refer to the toolkit. However, when developing an application, NetBeans will include a copy of the Deployment Toolkit in the local directory.

Deployment Toolkit Methods

- Key JavaScript deployment methods:
  - `dtjava.embed(app, platform, callbacks)`
    - Embeds the application into the browser
  - `dtjava.launch(app, platform, callbacks)`
    - Launches applications outside of the browser
- Utility methods:
  - `dtjava.install(platform, callbacks)`
    - Initiates installation of required components
  - `dtjava.validate(platform)`
    - Validates that the user environment satisfies platform requirements

The Deployment Toolkit API provides several important methods:

- `dtjava.embed(app, platform, callbacks)` embeds the application into the browser based on a given application descriptor. If Java Runtime or JavaFX Runtime installation is required, it will invite the user to click to install it.
- `dtjava.launch(app, platform, callbacks)` launches applications that are not embedded in the browser, based on a given application descriptor. If Java Runtime or a JavaFX Runtime installation is required, then it makes attempts to start the installer.
- `dtjava.install(platform, callbacks)` initiates the installation of required components according to platform requirements.
- `dtjava.validate(platform)` validates that the user environment satisfies platform requirements (for example, if a required version of JavaFX is available). It returns PlatformMismatchEvent, which describes problems, if any.

Java Web Start Deployment

Here is an example of the JavaScript included in the browser:

```
<script>
    function launchApplication(jnlpfile) {
        dtjava.launch(                  {
                url : 'TextViewer.jnlp'
            },
            {
                javafx : '2.0+'
            },
            {}
        );
        return false;
    }
</script>
```

.jnlp file

Versions supported

The Deployment Toolkit uses JavaScript object literal notation to specify the key information needed to launch the application. Note that the URL to the `.jnlp` file is included. In addition, you must specify which version of JavaFX is supported; in this case, any version 2.0 or later of JavaFX.

Java Web Start HTML Link

HTML launch link:

```
<b>Webstart:</b>
<a href='TextViewer.jnlp' onclick="return
launchApplication('TextViewer.jnlp');">click to launch
this app as webstart</a>
<br/>
<hr/>
<br/>
```


Even with the launch script included in the header of the HTML document, you must still add a link to launch the application in the body of the document. This link in the body of the document calls the JavaScript and launches your JavaFX application.

JNLP Application Descriptor

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0" xmlns:jfx="http://javafx.com" href="TextViewer.jnlp">
  <information>
    <title>TextViewer</title>
    <vendor>Your Name Here</vendor>
    <description>Sample JavaFX 2.0 application.</description>
    <offline-allowed/>
  </information>
  <resources os="Windows">
    <jfx:javafx-runtime version="2.0+"
href="http://javadl.sun.com/webapps/download/GetFile/javafx-latest/windows-
i586/javafx2.jnlp"/>
  </resources>
  <resources>
    <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>
    <jar href="TextViewer.jar" size="17431" download="eager" />
  </resources>
  <applet-desc  width="800" height="600" main-
class="com.javafx.main.NoJavaFXFallback"  name="TextViewer" />
  <jfx:javafx-desc  width="800" height="600" main-
class="com.example.gui.TextViewer"  name="TextViewer" />
  <update check="background"/>
</jnlp>
```


This slide shows the JNLP application descriptor for the TestViewer application. Note that elements are included for embedded launch (the `applet-desc` tag) and Java Web Start. The screen size is also included here.

Web Browser Deployment

```
<script>
    function javafxEmbed() {
        dtjava.embed(
            {
                url : 'TextViewer.jnlp',
                placeholder : 'javafx-app-placeholder',
                width : 800,
                height : 600,
                jnlp_content : 'PD94bWwgdmVyc2lvbj0iMS4w'
            },
            {
                javafx : '2.0+'
            },
            {}
        );
    }
    <!-- Embed FX application into web page once page is loaded -->
    dtjava.addOnloadCallback(javafxEmbed);
</script>
```

Base64 encoded JNLP file

Here is the JavaScript for deploying your application in a browser. Notice a callback method is used to launch the application once the page loads. Notice the "jnlp content" attribute in the middle of the page. This allows you to Base64 encode your `.jnlp` file and include it in your page. This should give you a minor speed improvement on initial page load, because the browser does not have to make a second request to get the `.jnlp` file.

Note: In this example, the Base64 data has been truncated so that the code example fits in the slide.

Deployment Toolkit Callbacks

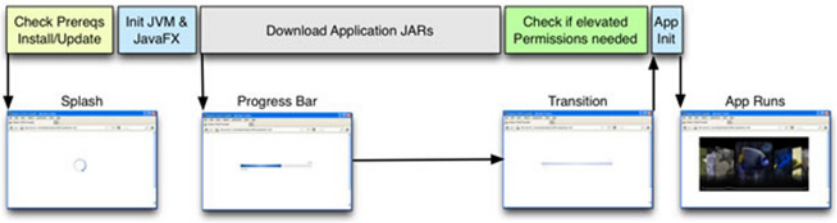
The deployment toolkit includes callback functions. Here are a few of them:

- `onDeployError: function(app, mismatchEvent)`
  - Called when platform requirements are not met
- `onInstallFinished: function(placeholder, component, status, relaunchNeeded)`
  - Called once installation of a required component is completed
- `onInstallNeeded: function(app, platform, cb, isAutoinstall, needRelaunch, launchFunc)`
  - Called if embedding or launching an application needs additional components to be installed
- `onInstallStarted - function(placeholder, component, isAuto, restartNeeded)`
  - Called before installation of the required component is triggered

Callback JavaScript functions are included to allow you handle a number of event and situations that might arise when trying to deploy your application. The examples provided here are just a subset of what you can use. See the *Java FX Deployment Guide* for details:

http://docs.oracle.com/javafx/2.0/deployment/jfxpub-deployment.htm

Application Startup Process



- Initialization
- Loading and preparation
- Application-specific initialization
- Application execution

JavaFX application startup can be broken up into the following phases.

Phase 1: Initialization

Initialization of Java Runtime and an initial examination identifies components that must be loaded and executed before starting the application.

Phase 2: Loading and preparation

The required resources are loaded from either the network or a disk cache, and validation procedures occur. All execution modes see the default or a custom preloader.

Phase 3: Application-specific initialization

The application is started, but it may need to load additional resources or perform other lengthy preparations before it becomes fully functional. An example of this is checking whether elevated permissions are needed and displaying the appropriate request for permission to the user.

Phase 4: Application execution

The application is displayed and is ready to use.

Preloaders

- The application provides visual information that your application is loading.
- Java FX includes default preloaders:
  - Splash screens
  - Progress bars
- JavaFX preloader is customizable.
  - Can be customized with CSS

Preloaders are small applications that provide visual information that your application is loading. Typically, they provide splash screens, usually with a logo, and progress bars, indicating that your application is loading.

JavaFX provides a default preloader. However, you can customize the preloader with your own logos and progress bars. In addition, as with other JavaFX components, you can style the preloader with CSS.

Default Information

- Java Web Start

Splash Screen

Loading Dialog

- HTML

Splash Screen

Loading Dialog

Pictured are some of the default images displayed depending upon your deployment option.

Packaging Tools

- NetBeans
  - Builds a Web Start and web browser deployment for each app
  - Totally automated
  - Digital signing
- Ant
  - For more control and customization options
    - Can create multiple JARs
    - Custom HTML and preloaders
    - Digital signing
- JavaFX Packager
  - A convenience utility
  - Does not provide as much flexibility or as many options as Ant tasks

NetBeans automatically creates deployment files for Web Start and web browser distribution. You can always start with this files and customize them for your needs.

If you need more power, Ant tasks are also provided with JavaFX, giving you much finer-grained control of your `.jar` files.

Both NetBeans and Ant can be used to digitally sign your applications.

Breaking Out of the Sandbox

How do you break out of the sandbox?

- Digitally sign each application.
- What steps are involved?
  - Use public/private key encryption.
  - Publish your public key with a known certificate authority.
    - Requires a yearly fee
    - Generate a certificate
- Covered in detail in the security lessons.

Quiz

Which two components are required to launch a JavaFX application from a web page?

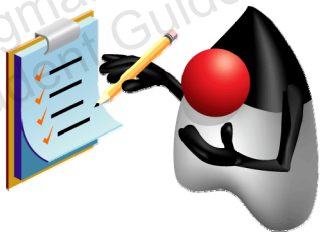
a. A JavaScript launch script
b. RSA encryption
c. A security certificate
d. A `.jnlp` file

Answer: a, d

Summary

In this lesson, you should have learned how to:

- Describe a JAR file and the steps for creating one
- Create a manifest file that uses headers
- Create a JAR by using development tools
- Deploy a stand-alone JAR
- Deploy a JAR as an applet
- Deploy a JAR using Java Web Start



Practice Overview

- Practice 13-1: Deploying an Application in a Web App
- Practice 13-2: Deploying an Embedded Application Only (Browser Only)
- Practice 13-3: Deploying Java Web Start Only