Java Application Deployment

Deployment

- Application packaging
  - JAR files
- Internet-based application delivery
  - Applets
  - Java Web Start

JAR Files

- Give the user a ***single*** file
  - Not a directory structure filled with class files
- Java Archive (JAR) files
  - can contain both class files and other files such as image and sound files
  - using ZIP compression format

Making JAR files

```
jar options JARFileName File1 File2
...

e.g.
   jar cvf MyApp.jar *.class icon.gif
```

Option	Description
`c`	Creates a new or empty archive and adds files to it
`t`	Displays the table of contents
`f`	Specifies the JAR file name as the second command-line argument
`v`	Generates verbose output
`m`	Adds a manifest to the JAR file

- *Selected options are shown*
- *Similar to the options of the UNIX tar command*

Manifest

- A manifest file that describes special features of the archive
- The manifest file, **`MANIFEST.MF`**, is located in a special META-INF subdirectory of the JAR file
  - The last line in the manifest must end with a newline character
- Manifest entries

```
Manifest-Version: 1.0
lines describing this archive
Name: aClass.class
lines describing this file
Name: com/company/mypkg/
lines describing this package
```

Executable JAR Files

- Place all files that your application needs into a JAR file and then add a *manifest* entry that specifies the main class of your program

```
Main-Class: com/mcompany/mypkg/MainAppClass
```

- Running JAR-packaged program
  - JAR files as applications

    `java -jar` *MyProgram.jar* or `javaw -jar` *MyProgram.jar*
  - Applets packaged in JAR files

```
<applet code="MyApplet.class"
        archive="MyClasses.jar"
        width="100"  height="150">
```

Assessing Resources

- Resources
  - image and sound files, any other associated files
- Assessing resources from a JAR file
  - Get the Class object

```
getClass();
```

  - Get the resource location in the class path or the resource as a stream
    - For media files

```
URL url = getClass().getResource(filename);
Image img = Toolkit.getDefaultToolkit().getImage(url);
```

    - For data files

```
InputStream stream = getClass().getResourceAsStream(filename);
Scanner in = new Scanner(stream);
```

Sealing

- ***Seal*** a Java language package to ensure that no further classes can add themselves to it
  - You would want to seal a package if you use package-visible classes, methods, and fields in your code
  - Without sealing, other classes can place themselves into the same package and thereby gain access to its package-visible features
- In the JAR file manifest,
  - globally seal all packages

```
Sealed: true
```

  - Seal individual packages

```
Name: com/company/packageToBeSealed/
Sealed: true
```

Java Web Start

- Java Web Start applications
  - Typically delivered through a browser
    - once downloaded, it cab be started without using a browser
    - different from the launch of an applet
  - Do not live inside a browser and do not use the Java implementation of the browser
  - Digitally signed applications can be given arbitrary access rights on the local machine
    - Enabled to run outside of the Sandbox

Preparing Applications for Web Start

1. Compile **MyApp.java**.
2. Prepare a manifest file MyApp.mf with the line

```
Main-Class: MyApp
```

3. Produce a JAR file with the command

```
jar cvfm MyApp.jar MyApp.mf *.class
```

4. Prepare the launch file MyApp.jnlp with the following contents

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"codebase=http://localhost:8080/myapp/ href=“MyApp.jnlp">
  <information>
    <title>My Application</title>
    <vendor>My name</vendor>
    <description>My application</description>
    <offline-allowed/>
  </information>
  <resources>
    <j2se version="5.0+"/>
    <jar href=“MyApp.jar"/>
  </resources>
  <application-desc/>
</jnlp>
```

Preparing Applications for Web Start

5. Make a directory structure in the Web server document path

   `/webapps/MyApp`

   `/webapps/MyApp/WEB-INF`

   Place the following minimal web.xml file inside the WEB_INF subdirectory

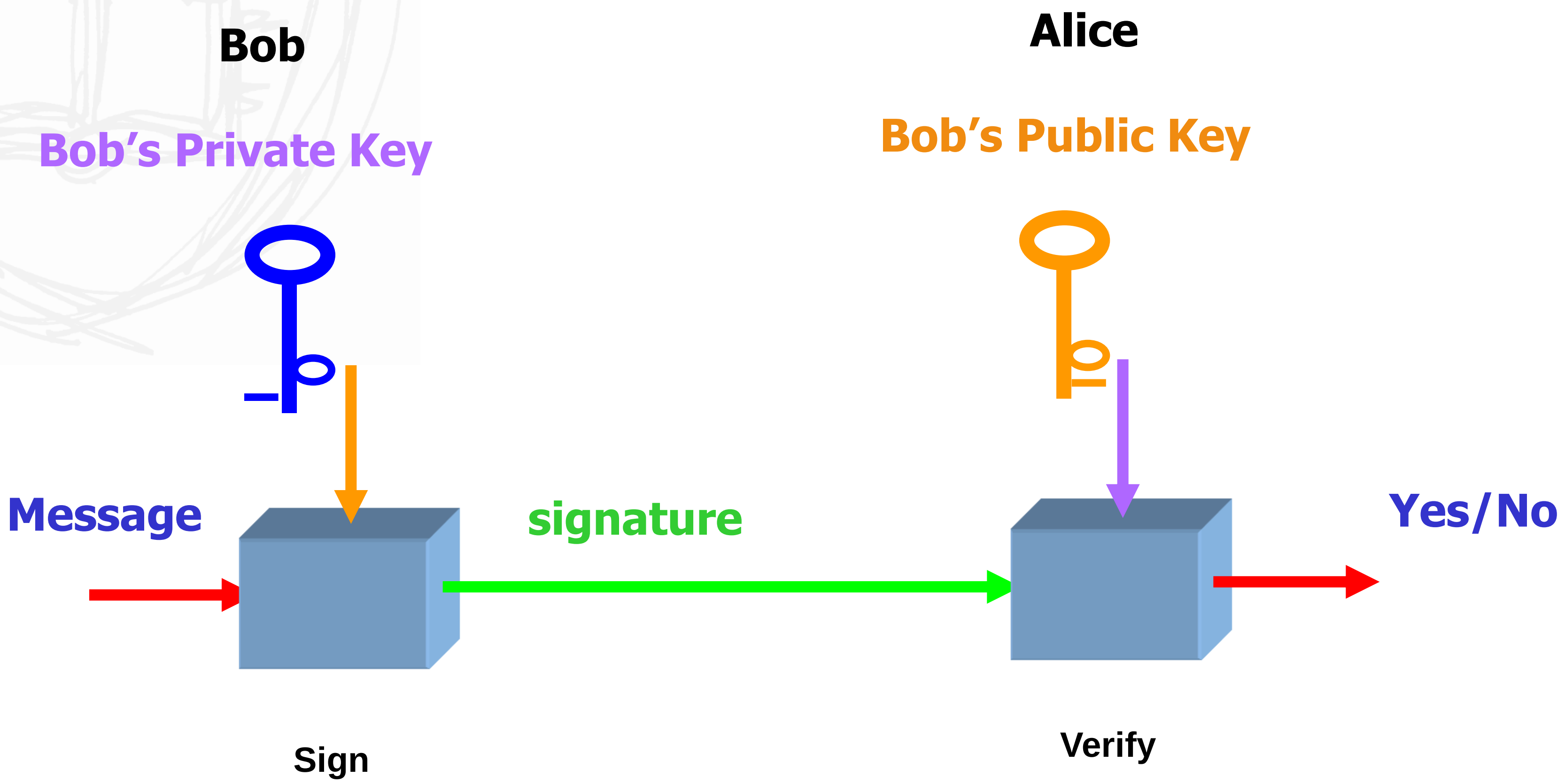
```
<?xml version="1.0" encoding="utf-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/nz/j2ee"
   xmlns:xsi=http:"//www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                              http://java.sun.com/xml/ns/j2ee/web-app_2_5.xsd">
</web-app>
```

6. Place the JAR file and the launch file on your web server so that the URL matches the codebase entry in the JNLP (Java Network Launching Protocol) file
7. Make sure that your browser has been configured for Java Web Start, by checking that the `application/x-java-jnlp-file` MIME type is associated with the `javaws` application
   – If you installed the JDK, the configuration should be automatic
8. Start the web server
9. Point your browser to the JNLP file.
   – For example, go to `http://localhost:8080/MyApp/MyApp.jnlp`

Digital Signatures

- A digital signature is generated by a mathematical algorithm
  - Authentication
- To generate a digital signature, you need a digital signature algorithm and a signing (private) "key"
- To verify a signature, you need to use the same algorithm and a verification (public) key
- Key pair
  - Private key - used to sign
  - Public key - used to verify
  - Public Key certificate is an authorised version of a public key. It is normally signed by a trusted third party.

Digital Signatures



Key for Digital Signatures

- Key generator: `keytool`

```
keytool -genkey -alias theKey
     -keystore theKeyStore -keyalg DSA
```

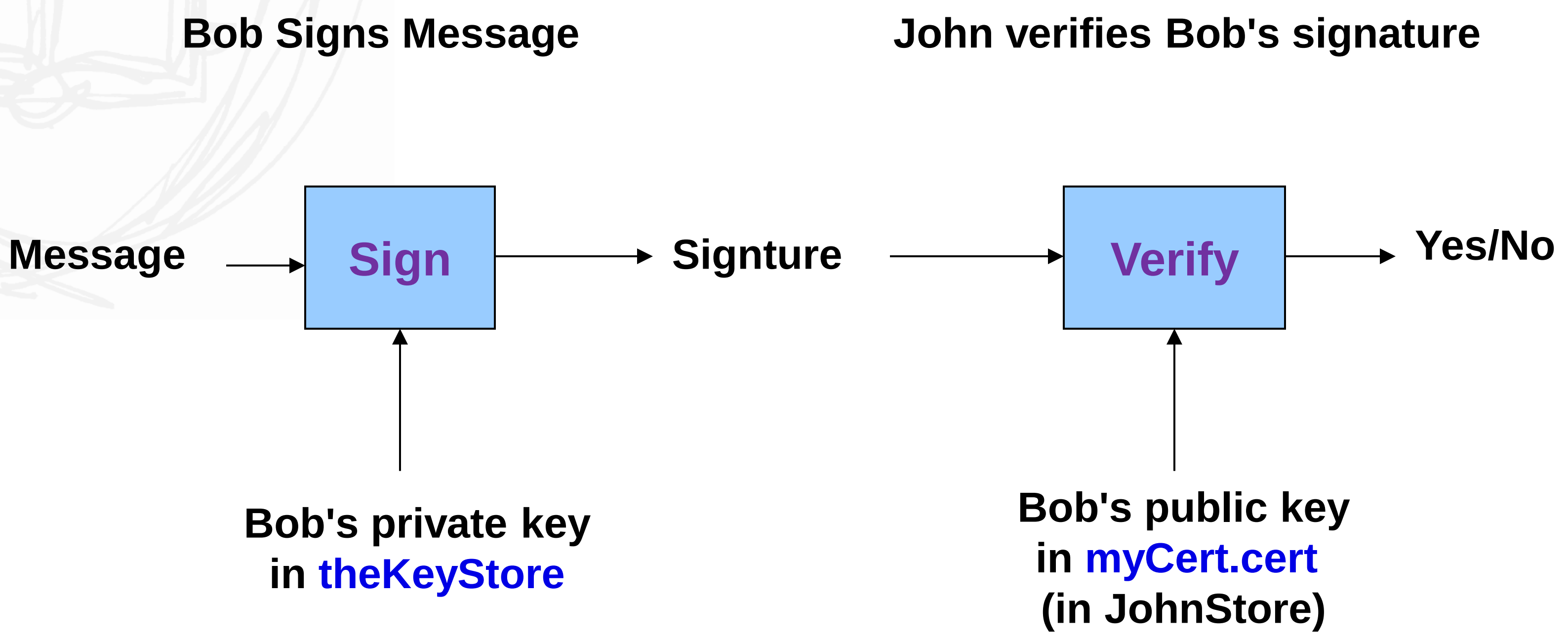
  - Key pair is stored in a file “`keyStore`"
  - The signature algorithm is DSA
  - The name of key is `theKey`

- Public key certificate generator:

```
keytool -export -alias theKey -keystore
  theKeyStore -file myCert.cert
```

  - `myCert.cert` is the certificate

How to sign/verify



Signing a Jar file

- Signing a jar file

Alias of the private key in the keystore

```
jarsigner [options] jar-file alias
```

e.g. signing **`MyApp.jar:`**

```
jarsigner -keystore theKeyStore -storepass mypasswd
          -keypass mypasswd MyApp.jar theKey
```

Verifying a Signed JAR File

```
jarsigner -verify jar-file
```

- This command will verify the JAR file's signature and ensure that the files in the archive haven't changed since it was signed
- If the verification fails, an appropriate message is displayed

```
jarsigner: java.lang.SecurityException: invalid SHA1
signature file digest for test/classes/Manifest.class
```