Java Deployment

Albert Weichselbraun

Java Deployment

- Distribution, installation and maintenance of Java programs
  $\rightarrow$ user (and admin) friendly solutions
- approaches:
  - Java Archive files (JAR)
  - Java Web Start

Java Archive Files (JAR)

- a single JAR file contains the Java program code and all required files
  - JAR files are compressed archives (.zip/rar/tar) containing the Java project
  - they contain meta data such as a MANIFEST file, optional signatures, etc.
- executable via `java -jar program.jar`
  $\rightarrow$ a shell script is required

Java Web Start

- automatically installs and executes a program just by “clicking” on its link
- platform independent
- manages the used Java runtime environment (and acquires specific versions if required)
- integrates well into desktop environments
- automatically updates the application if necessary (the application may reside in a cache on the user’s computer)

Creating JAR files

- Eclipse: `File -> Export -> Runnable JAR file`
- your program code *must* refer to other resources using URLs (otherwise it will not be able to open them if they are distributed using the JAR file)
  - use `getClass()` to get a resource's URL
  - use a *Slash* (rather than a Backslash) to separate directory entries

Accessing Images

```
1   import java.net.URL
2   import java.imageio.*
3           ...

5   Image img;
6   try {
7     // create a URL pointing to the resource
8     URL imgUrl= getClass().
9                   getResource( 'path/myImage.png' );
10    // load image
11    img = ImageIO.read( imgUrl );
12  } catch (IOException e) {
13    e.printStackTrace();
14  }
```

Accessing Files

```
1   import java.net.URL
2   import java.imageio.*
3         ...

5   String line;
6   try {
7     // create a URL poitning to the resource
8     URL resourceUrl = getClass().
9                       getResource( 'dir/test.txt' );
10    // open the URL and create a BufferedReader
11    // object for the content of that URL
12    InputStream in=resourceUrl.openStream ();
13    BufferedReader inReader =
14      new BufferedReader (new InputStreamReader (in));
```

```
16        // read (and print) data
17        while ((line= inReader.readLine()) != null) {
18                System.out.println(line);
19        }
20        inReader.close();
21    } catch (IOException e) {
22        e.printStackTrace();
23    }
```

Java Web Start Applications

- export your project as a JAR file
- create a `.jnlp`-File describing your application
- specify the `codebase`, `Java version` and `Java Main Class`; you may add optional data such as `offline-allowed`, `vendor`, etc.
- security:
  - your application runs in a Java-Sandbox
    $\rightarrow$ no access to files and resources outside the JAR archive
  - exception: signed applications $\Rightarrow$ specification of fine grained access rights possible

Example

```
1  <?xml version="1.0" encoding="utf-8"?>

3  <jnlp spec="1.0"
4   codebase="http://www.ai.wu.ac.at/weichselbraun/jar/"
5   href="Java2DDemo.jnlp">

7   <information>
8     <title>Java 2D Demo</title>
9     <vendor>Albert Weichselbraun</vendor>
10    <homepage
11     href="http://www.ai.wu.ac.at/weichselbraun" />
12    <offline-allowed/>
13  </information>
```

```
15  <resources>
16    <jar href="Java2DDemo.jar"/>
17    <j2se version="1.4+"
18     href="http://java.sun.com/products/autodl/j2se"/>
19  </resources>

21  <application-desc main-class="java2d.Java2DDemo"/>
22 </jnlp>
```