Java Libraries

Library definition, use, creation, APIs, Javadoc creation, Javadoc commenting standards

What is a library?

A **library** is a 3rd party collection of code that can be used in multiple programs.

- Libraries promote code reuse.
- Libraries offer tools to complete common tasks.
- Examples:
  - Java API
  - Junit

Java libraries are deployed as **JAR** files (Java ARchive). JARs are based on the zip file format.

How to use a library

1. Download the library.
2. Install the library
   - An executable (Java Libraries)
   - A jar executable to download rest of library (JAXB)
   - A jar file (JUnit, Apache libraries, JSON, Java3D, etc)
3. Include library on the classpath when compiling and executing your application.

Classpaths

The **classpath** is the path(s) where Java looks for libraries (referenced code) when compiling and running a program. To specify the classpath:

1. Use the `-cp` argument to the java command
    1. Separate directories/jars with a semi-colon (;)
    2. List each jar individually. You cannot just specify a directory of jars.
2. Alternatively, update the System's PATH environment variable. Make sure you restart your terminal so the new path will be available.

Using libraries on the command line

Assume the project is set up in directory structure like Eclipse project.

Commands to compile and run:

```
% javac  -d bin -sourcepath src -sourcepath test -cp
path1;path2;path3 src/pack/age1/*.java src/pack/age2/*.java
test/pack/age2/*.java

% java -cp ./bin;path1;path2;path3 pack.age1.ClassName
```

Creating a user library in Eclipse

1. Right-click on your project, select **Build Path > Configure Build Path.**
2. Select **Add Library > User Library > User Libraries.**
3. Select **New** and give your library a name. You do not need to add your library to the system path. But if you do add it to the system path, you can use it from the command line.
4. Select your library and click **Add Jars**.
5. Browse your file system and select the ***.jar** files appropriate for your library.
6. Click **OK**. Click **Finish**. Click **OK**.

Other library setups

- Some projects store all relevant *.jar files in a lib/ directory
  - Jars are added to lib in file system.
  - In Eclipse, select the Add Jar option and add the jar(s).
- Building applications such as Maven will download the required jar(s) from online repositories
- Building applications such as Ant and Maven simplify compilation and running when you use the command line rather than an IDE.

APIs

API = Application Programming Interface. APIs specify what their libraries can do. The API specification is a collection of web pages that describe:

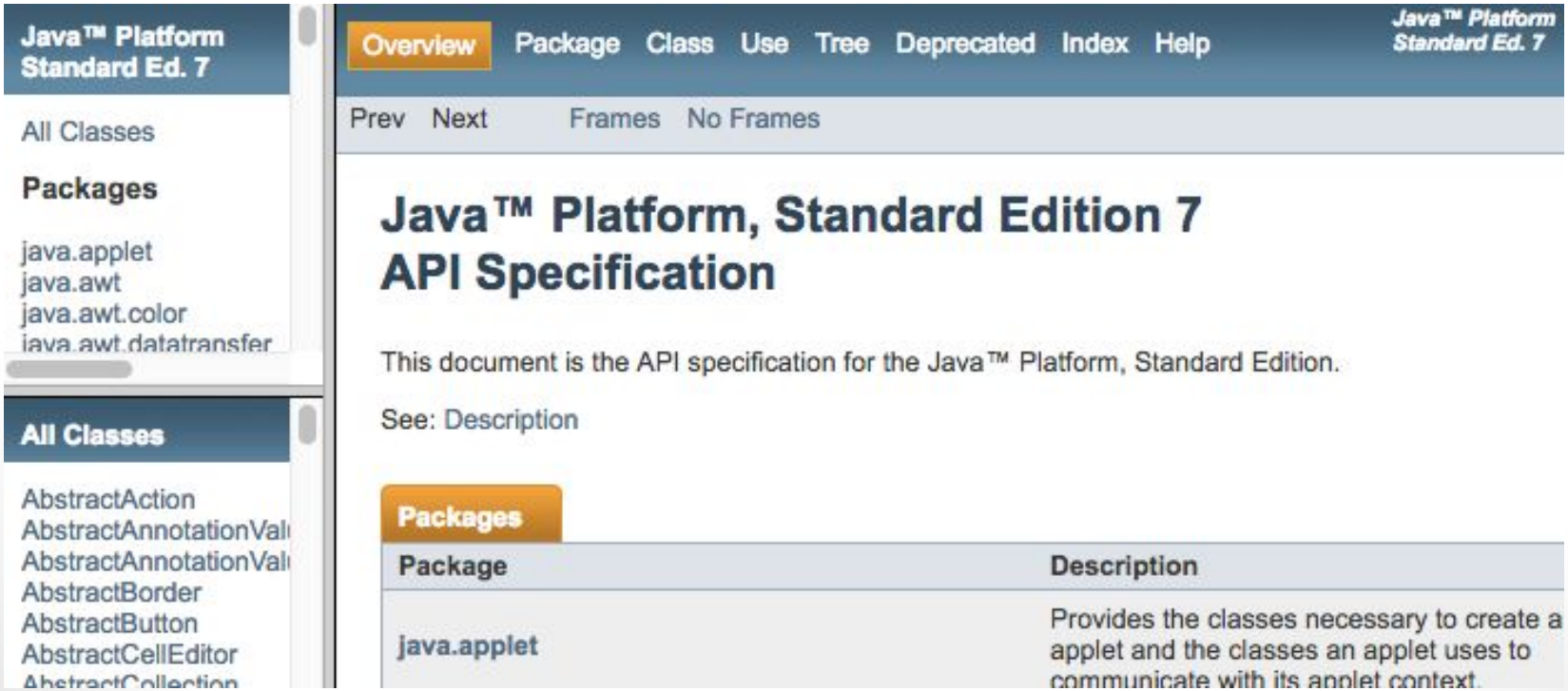
- The public classes, interfaces, and methods of the corresponding Java class libraries.
- What functionality is available for you (the client) to use from these libraries.

An API tells you what a class can do, but doesn't tell you how the class does it.

Note: The API is the specification. The library is an implementation of the API.

API standard layout

- Upper left frame: packages. Lower left frame: classes, interfaces.
- Main frame:
  - List of packages and descriptions
  - List of classes and descriptions
  - List of class members and descriptions: inheritance hierarchy, fields, methods



Creating your own API

- Sometimes you're the client of a library.
  - Java API
  - 3rd Party Libraries – JFreeChart, JUnit, JAXB, etc.
- Sometimes you're writing code that others will use.
  - An API gives those clients information about how to use the library.
- Sometimes you're working with a large team and other team members may need to know what you've done.
  - You need to provide high level details about your work.

Javadoc

Javadoc is a tool for generating documentation for Java source code. It is a **doclet**, which is a Java program that produces standard API documentation.

- Javadoc produces HTML files.
- Java supports three different kinds of comments:
  - C-style comments, enclosed in **/* … */**
  - Line comments, begin with **//**
  - Javadoc comments, enclosed in **/** .. */.**
- Eclipse can help:
  - It will stub out some Javadoc comments for you.
  - It will also generate the API documentation for your projects from Javadoc comments in your code:

    Project > Generate Javadoc
- Command to create API documentation from the terminal:

```
% javadoc [comma-separated list of source files]
```

Javadoc comments

For your projects, you must create Javadoc comments for all classes, interfaces, fields, and methods -- including test classes and methods!

- Javadoc comments are placed immediately before whatever they are commenting. No extra blank lines.
- Javadoc tags start with @ and specify additional information about code that is handled in a specific way.
- CSC Style Guidelines have directions on how to write Javadoc comments for your code.

Javadoc comment format

Javadoc comments typically are placed before every public class, interface, and class member. The format is as follows.

```
/**
 * Explanation of the purpose of this class/method.
 *
 * @author  your_name  (For class declarations only)
 * @param  parameter1  description of the first parameter
 * @param  parameter2  description of second parameter
 * ...
 * @return  Description of the return value
*/
```

What to avoid

Oracle has a good example of ineffective and effective documentation.

The description on this page merely repeat the method declaration.

```
/**
 * Sets the tool tip text.
 *
 * @param text  the text of the tool tip
 */
public void setToolTipText(String text) {
```

The sample here illustrates bad documentation. It does not reveal anything that one cannot understand just from reading the method declaration.

Good documentation instead

```
/**
 * Registers the text to display in a tool tip.   The text
 * displays when the cursor lingers over the component.
 *
 * @param text  the string to display.  If the text is null,
 *              the tool tip is turned off for this component.
 */
public void setToolTipText(String text) {
```

When you describe a method, start with a verb.

Html generated from the good documentation example

Method Detail

setToolTipText

```
public void setToolTipText(java.lang.String text)
```

Registers the text to display in a tool tip. The text displays when the cursor lingers over the component.

Parameters:

`text` - the string to display. If the text is null, the tool tip is turned off for this component.

Class level comments

```
package edu.ncsu.csc216.project5.utils;

import java.util.Scanner; // Or other imports

/**
 * A description of what the class does. The more detailed,
 * the better.
 * @author Author Name
 * @version 1.0
 */
public class MyClass { … }
```

Method level comments

```
/**
 * Describe what the method does at the top. If it is not
 * void, describe what it returns or the significance of the
 * return value.
 * @param paramName1 description of first parameter.
 * @param paramName2 description of second parameter.
 * @return description of return
 */
public <return> myMethod(<type> paramName1,
                         <type> paramName2) { … }
```

Instance, static field comments

```
public class MyClass {

    /** Describe myField1 */
    private <type> myField1;
    /**
     * Describe myField2
     */
    private <type> myField2;
    …
}
```

References

- Sarah Heckman, CSC 216 slides.
- Oracle, “How to Write Doc Comments for the Javadoc Tool,” https://www.oracle.com/technetwork/java/javase/tech/index-137868.html