# Controlling App Execution and Environment

**Jim Wilson**
MOBILE SOLUTIONS DEVELOPER & ARCHITECT
@hedgehogjim  blog.jwhh.com

# Overview

- Command-line arguments
- Managing app and user properties
- Persisting and restoring properties
- Deploying property defaults in a package
- Default class loading behavior
- Working with class paths
- Execution environment information

# Factors That Affect App Behavior

**Apps require more than just our code**

- Our code is just the beginning

**Behavior affected by many factors**

- Startup options
- User preferences/actions
- Execution environment
- Support libraries

# Command-line Arguments

**Can pass info to app on command line**

- Easiest way to pass startup info
  - Target of app processing
    - Input/output files, URLs, etc.
  - Behavior options

**Arguments passed as a String array**

- Received by app's main function
- Each argument is a separate element
  - Separated by OS's whitespace
  - Honor's OS's value quoting

# Simple Command-line Processing

```
package com.jwhh.cmdline;
class Main {
  public static void main(                ) {
    if(args.length < 1)
      System.out.println("No arguments provided");
    else {
      for(String word:args)
        System.out.println(word);
    }
  }
}
```

```
java com.jwhh.cmdline.Main Hello there world
```

```
Hello
there
world
```

# Setting Command-line Arguments in Your IDE

**IDE's allow for ease of testing**

- Can set command line args
- Will automatically pass when run in IDE

**IntelliJ**

- http://bit.ly/intellijcmdlineargs

**NetBeans**

- http://bit.ly/netbeanscmdlineargs

# Managing Persistable Key/Value Pairs

**Apps often need persistable key/value pairs**

- Store app configuration information
- Track simple aspects of app state
- Track user preferences

**Need an easy way to manage key/value pairs**

- Set/retrieve value
- Store/load between app executions
- Provide default value when not set

**Use the java.util.Properties class**

# Properties Class

**Properties class**

- Inherits from HashTable class
- Keys and values are Strings

**Working with properties**

- setProperty method
  - Sets the current value for a key
  - Creates or updates key as needed
- getProperty method
  - Returns the current value for the key
  - Returns null if not found and no default
  - Can optionally provide default value

# Setting/Retrieving Properties

```
Properties props = new Properties();

props.setProperty("displayName", "Jim Wilson");

String name = props.getProperty("displayName");
    Jim Wilson
String acctNum = props.getProperty("accountNumber");
         null
String nextPosition = props.getProperty("position", "1");
           1
```

# Store and Load Property Values

**Properties can be persisted**

- Can be written to & read from a stream
- Can optionally include comments
- Supports 2 formats
  - Simple text
  - XML

# Properties Persisted as Simple Text

**Persist as simple text**

- Use store & load methods
  - Supports OutputStream/InputStream
  - Supports Writer/Reader
- Normally name file with .properties suffix

**One key/value pair written per line**

- Key/value normally separated by = or :
  - Whitespace surrounding =, : ignored
  - Whitespace acts as key/value separator if occurs without = or :
  - Can escape whitespace, =, or : with \
- Start a line with # or ! for comments
- Blank lines ignored

# Storing Properties as Simple Text

```
Properties props = new Properties();

props.setProperty("displayName", "Jim Wilson");
props.setProperty("accountNumber", "123-45-6789");

try(Writer writer = Files.newBufferedWriter(
                        Paths.get("xyz.properties"))) {

  props.store(writer, "My comment");
}
```

*xyz.properties*

```
#My comment
#Thu Apr 28 14:45:37 EDT 2016
displayName=Jim Wilson
accountNumber=123-45-6789
```

# Loading Properties from Simple Text

```
Properties props = new Properties();
try(Reader reader = Files.newBufferedReader(
                    Paths.get("myapp.properties"))) {
  props.load(reader);
}
String val1 = props.getProperty("val1");
String val2 = props.getProperty("val2");
String val3 = props.getProperty("val3");
String val4 = props.getProperty("val4");
```

hello world

goodnight moon

hi noon

night bobbi sue

*myapp.properties*

```
val1=hello world
val2 = goodnight moon
val3:    hi noon
val4 night bobbi sue
```

# Properties Persisted as XML

**Persist as XML**

- Use storeToXML & loadFromXML methods
  - Supports OutputStream/InputStream
- Normally name file with .xml suffix

**One key/value pair per XML element**

- Stored as element named entry
  - Key stored as key attribute
  - Value stored as element value
- Use comment element for comments

# Storing Properties as XML

```
Properties props = new Properties();

props.setProperty(“displayName”, “Jim Wilson”);
props.setProperty(“accountNumber”, “123-45-6789”);

try(OutputStream out = Files.newOutputStream(
                       Paths.get(“props.xml”))) {
  props.storeToXML(out, “My comment”);
}
```

# Properties as XML

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <comment>My comment</comment>
  <entry key="displayName">Jim Wilson</entry>
  <entry key="accountNumber">123-45-6789</entry>
</properties>
```

# Loading Properties from XML

```
Properties props = new Properties();

try(inputStream in = Files.newInputStream(
                     Paths.get("props.xml"))) {
  props.loadFromXML(in);
}

String val1 = props.getProperty("displayName");
    Jim Wilson
String val2 = props.getProperty("accountNumber");
    123-45-6789
```

# Providing Default Properties

**Often useful to provide default values**

- Simplifies configuration
- Provide initial values for user preferences
- Cumbersome to explicitly provide defaults for each getProperty call

**Can create Properties with defaults**

- Pass default Properties to constructor
  - Searched if key not found in current Properties instance
- Default properties take precedent over default value passed to getProperty

# Using Default Properties

```
Properties defaults = new Properties();
defaults.setProperty("position", "1");

Properties props = new Properties(defaults);

String nextPos = props.getProperty("position");
```

1

```
int iPos = Integer.parseInt(nextPos);
// do something with iPos
props.setProperty("position", Integer.toString(++iPos));

// do some other work
nextPos = props.getProperty("position");
```

2

# Including Default Properties in Package

**Default property file can be part of package**

- Create .properties file at development time
  - Build process includes file in package

**Can load file from package**

- Takes advantage of Java resource system
- Use getResourceAsStream method
  - Accessed through any class in package
    - *ClassName*.class
    - this.getClass()

# Class Loading

**Most applications do not stand alone**

- Rely on classes in other packages
- JDK packages located automatically
- May need help locating other packages

**Locating packages at development time**

- Specific to each IDE
- IntelliJ: bit.ly/intellijclasspath
- Netbeans: bit.ly/netbeansclasspath

**Locating packages at runtime**

- Java provides a number of options

# Default Class Loading

**By default Java searches current directory**

- Classes must be in .class files
  - Must be under package directories

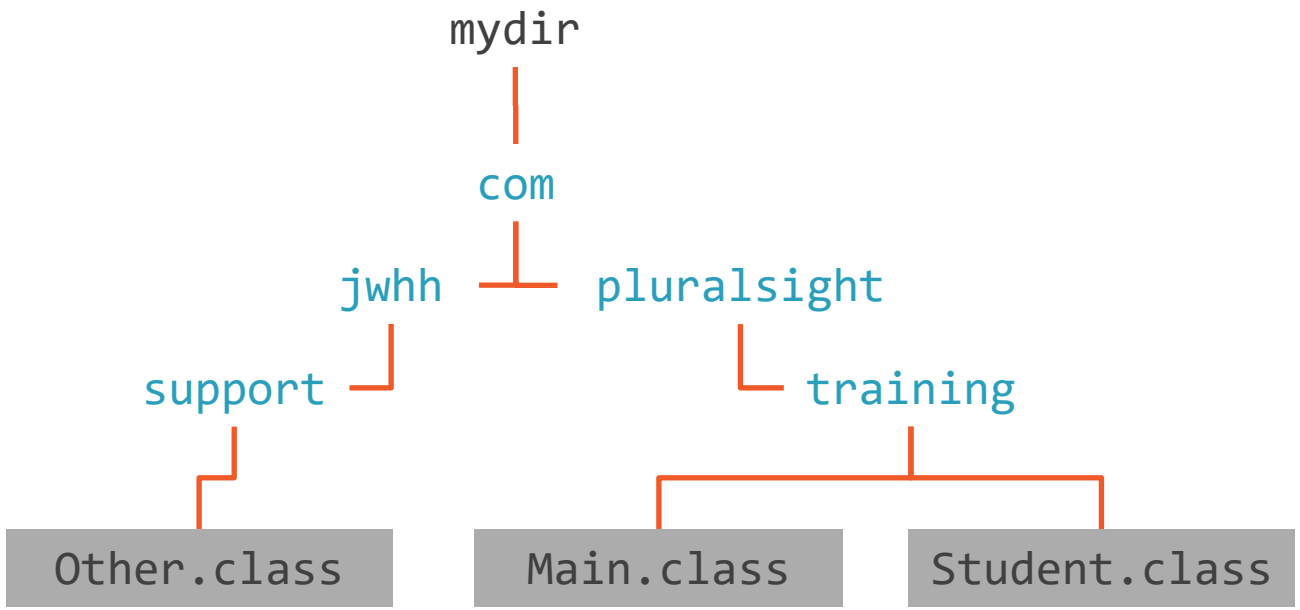
# Default Class Loading

```
package com.pluralsight.training;
import com.jwhh.support;
public class Main {
 public static void main {
   Student s = new Student;
   Other o = new Other;
   // do something with s and o
 }
}
```

```
package com.pluralsight.training;
public class Student {...}
```

```
package com.jwhh.support;
public class Other {...}
```

```
C:\mydir> java com.pluralsight.training.Main
```



# Specifying Class Path

**Can provide the list of paths to search**

- Searched in the order they appear
- Current directory used only if in list

**Two options for specifying class path**

- Environment variable
- Java command option

# Specifying Class Path as Environment Variable

**Can specify as an environment variable**

- Variable named CLASSPATH

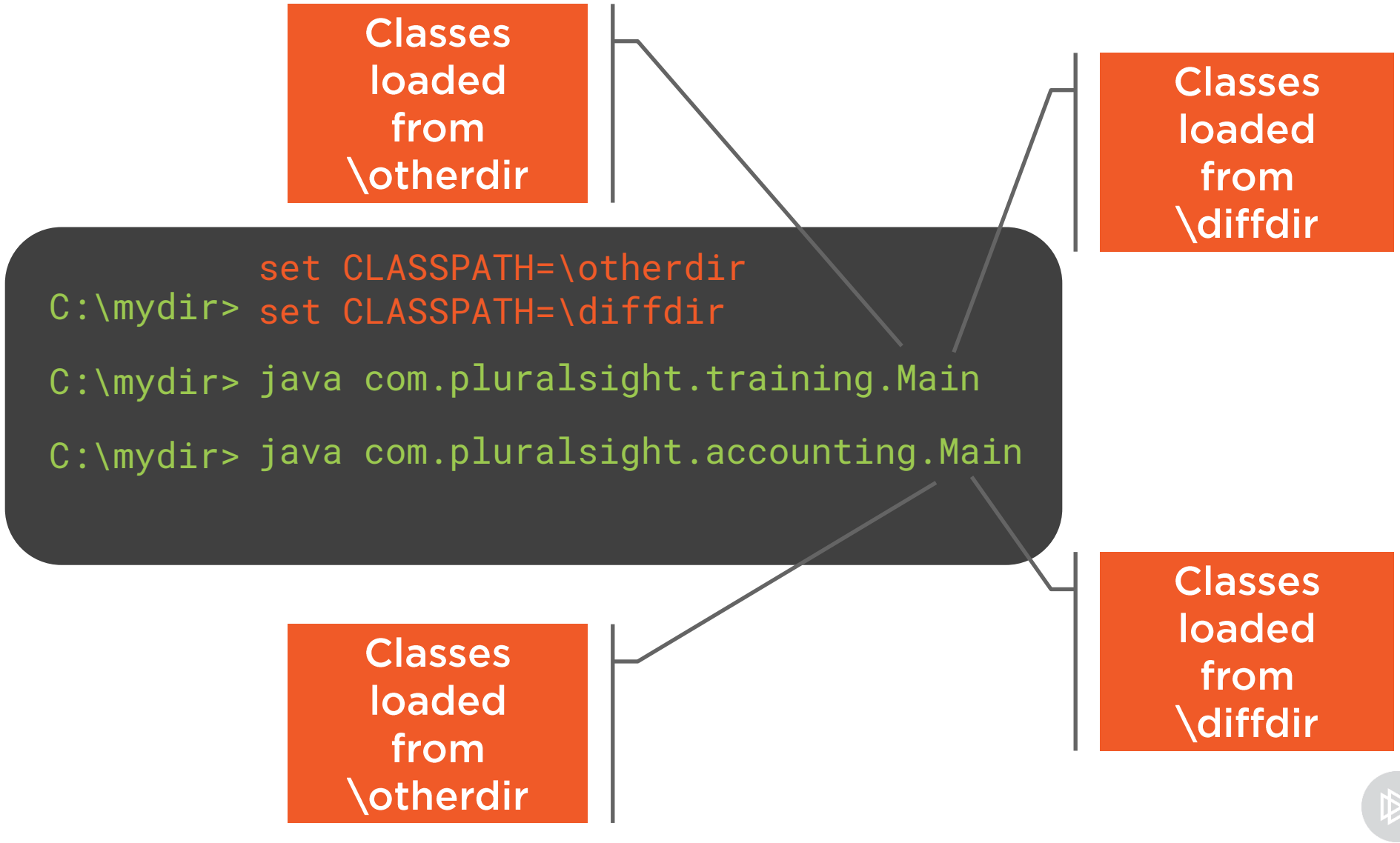
**Becomes default path**

- Used by all programs that don't provide a specific path

**Use environment variable with caution**

- Changing for one program can break another

# Specifying Class Path as Environment Variable



# Specifying Class Path with Java Command

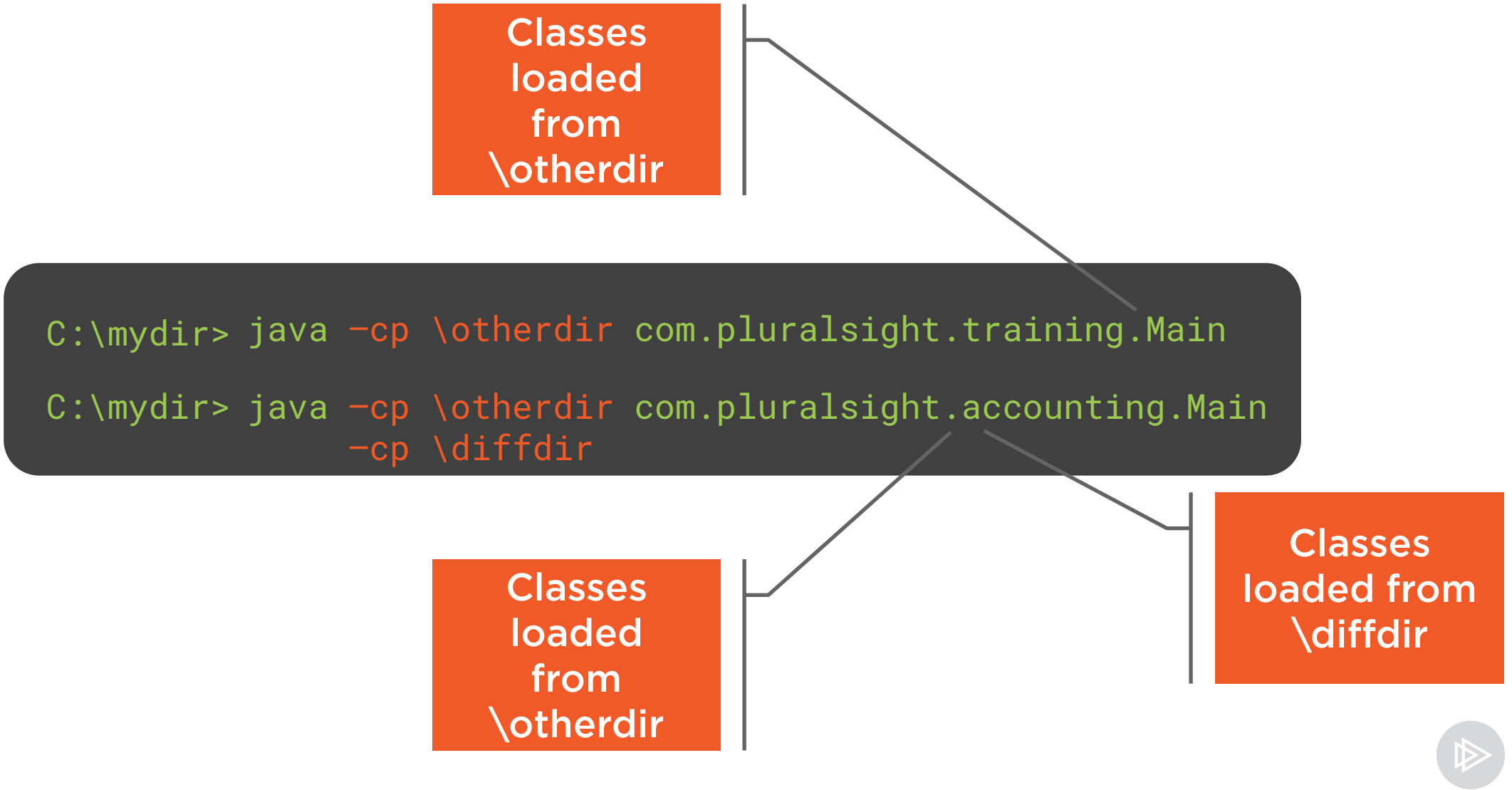
**Can specify as part of java command**

- Use –cp option
- Any CLASSPATH value is ignored

**Less prone to errors than CLASSPATH**

- Lists exact path for this app to use
- Not exposed to side-effects of other apps

# Specifying Class Path with Java Command



# Class Path Structure

**Paths provided as delimited list**

- Windows: separate with ; (semicolon)
- Unix platforms: separate with : (colon)
- Searched in the order they appear

**To reference classes in .class files**

- Path to folder containing package root

**To reference classes in jar files**

- Path to the jar file
  - Including jar file name

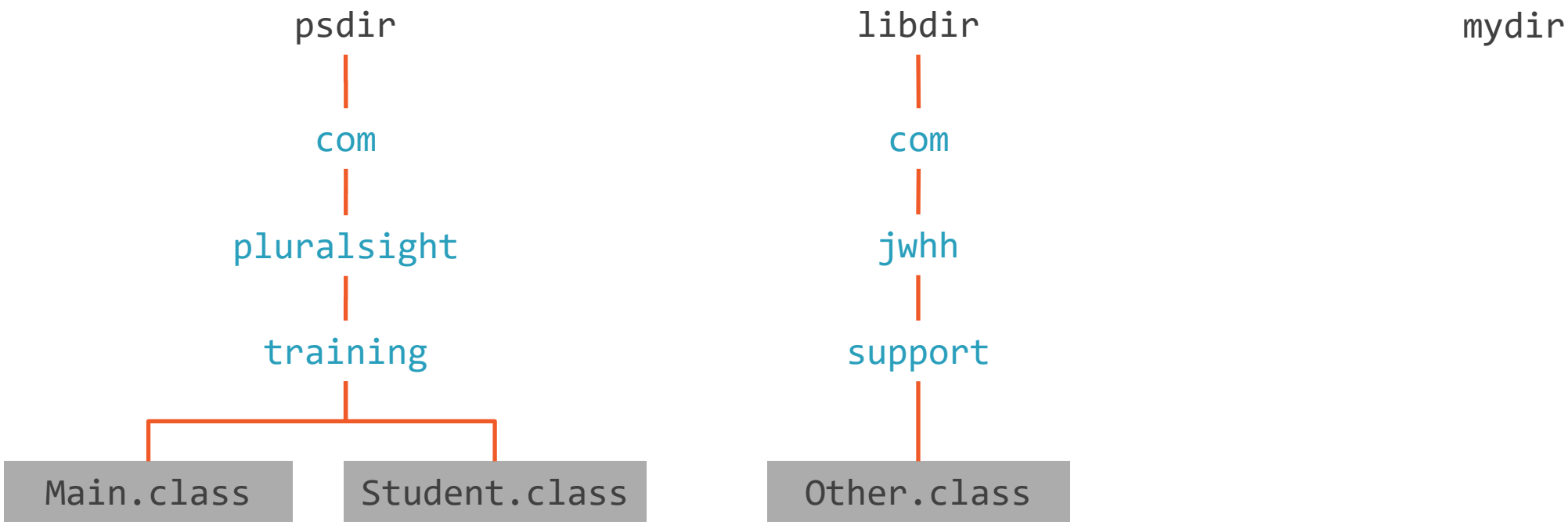
# Class Path Structure

**Windows**

```
java -cp \psdir;\libdir com.pluralsight.training.Main
```

**Unix platforms**

```
java -cp /psdir:/libdir com.pluralsight.training.Main
```



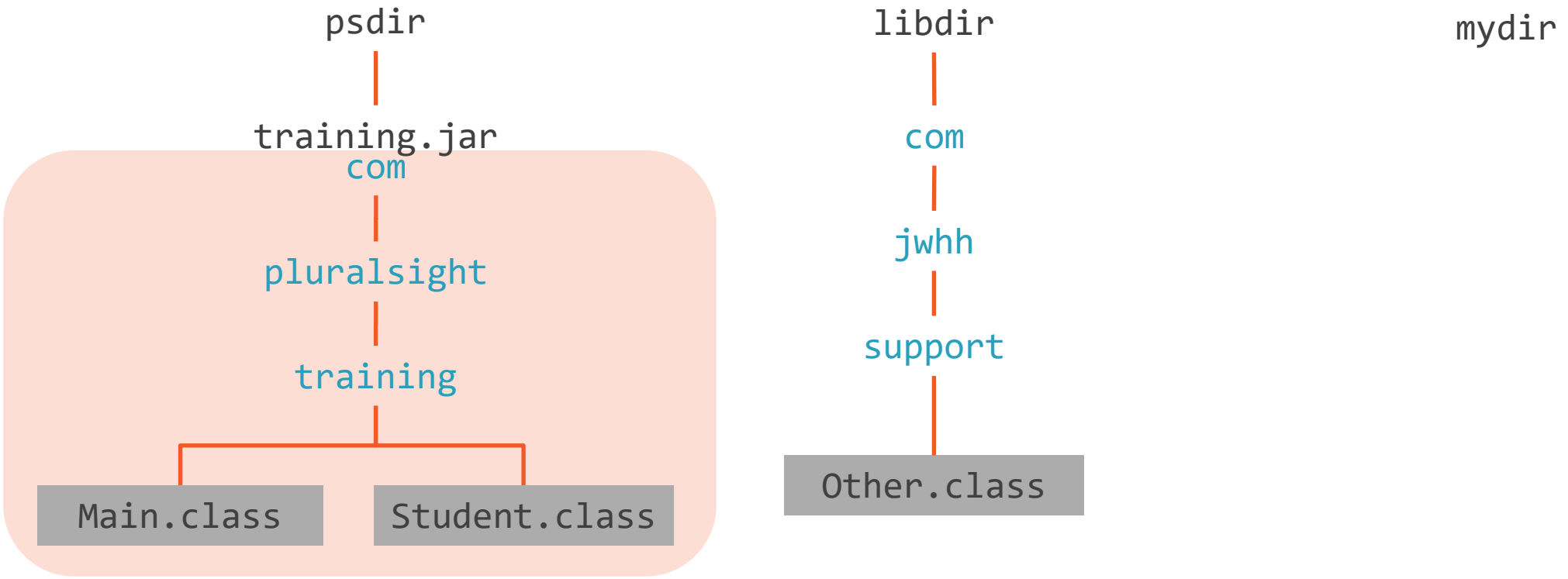
# Class Path Structure

Windows

```
java  -cp \psdir\training.jar;\libdir  com.pluralsight.training.Main
```

Unix platforms

```
java  -cp /psdir/training.jar:/libdir  com.pluralsight.training.Main
```



# Class Loading with Java –jar Option

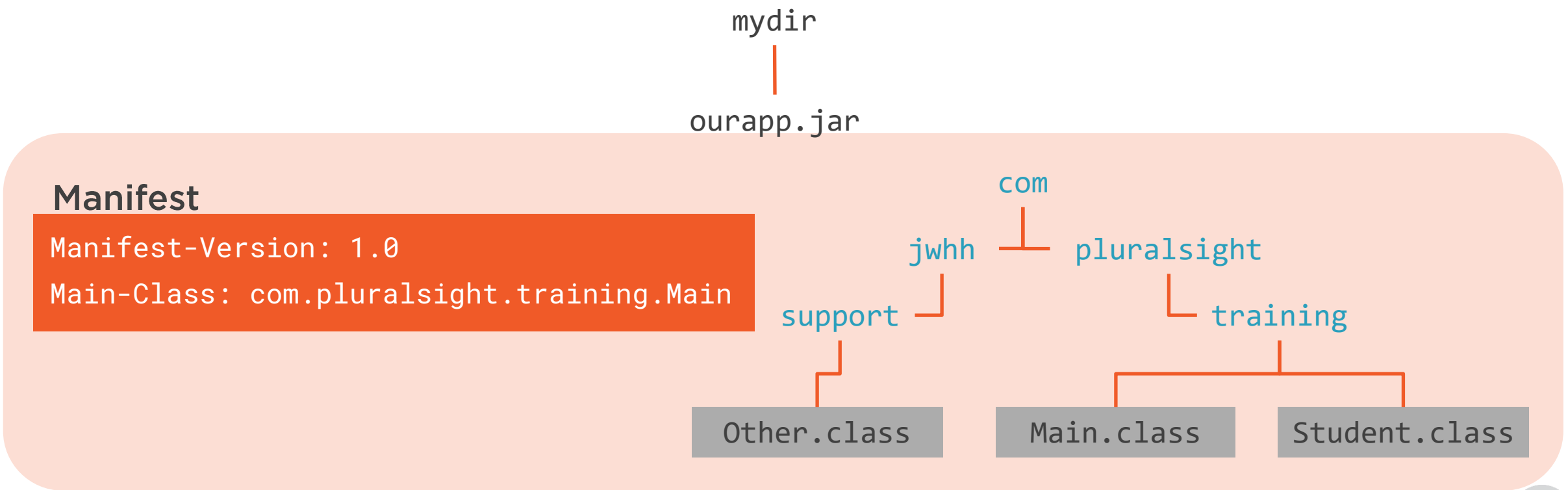
**Java –jar option locks down class loading**

- Class loading totally controlled by jar file
- No other class loading source is used

**Provides tight control over class loading**

# Class Loading with Java –jar Option

```
java –jar ourapp.jar
```



# Execution Environment Information

**Apps often need environment information**

- User information
- System information
- Java configuration information
- Application specific information

**Java provides two common solutions**

- System properties
- Environment variables

# System Properties

**Java provides info about environment**

- Accessed with System.getProperty

**Information includes**

- User information
- Java installation information
- OS configuration information

**Each value accessed via a string name**

- List of commonly used properties:
  - bit.ly/javasystemprops

# System Properties

```
String userName = System.getProperty("user.name");
```

Jim

```
String userHome = System.getProperty("user.home");
```

C:\Users\Jim

```
String osArchitecture = System.getProperty("os.arch");
```

amd64

```
String javaVendor = System.getProperty("java.vendor");
```

Oracle Corporation

# Environment Variables

**Most OS's support environment variables**

- Provide configuration information
- Many variables are set by OS
- Can provide app-specific variables

**Apps can access environment variables**

- Access all with System.getenv()
  - Returns Map<String, String>
- Access one with System.getenv(*name*)
  - Returns value of specific variable

# Environment Variables

```
package com.pluralsight.app;
public class Main {
 public static void main {
  String compName = System.getenv("COMPUTERNAME");
  String sysRoot = System.getenv("SystemRoot");


  System.out.println(compName);
  System.out.println(sysRoot);


 }
}
```

```
C:\mydir> java com.pluralsight.app.Main
   JIM_Y50
   C:\windows
```

# Environment Variables

```
package com.pluralsight.app;
public class Main {
 public static void main {
  String compName = System.getenv("COMPUTERNAME");
  String sysRoot = System.getenv("SystemRoot");
  String author = System.getEnv("COURSE_AUTHOR");

  System.out.println(compName);
  System.out.println(sysRoot);
  System.out.println(author);
 }
}
```

```
C:\mydir> java com.pluralsight.app.Main
   JIM_Y50
   C:\windows
   null
```

# Environment Variables

```
package com.pluralsight.app;
public class Main {
 public static void main {
   String compName = System.getenv("COMPUTERNAME");
   String sysRoot = System.getenv("SystemRoot");
   String author = System.getEnv("COURSE_AUTHOR");

   System.out.println(compName);
   System.out.println(sysRoot);
   System.out.println(author);
 }
}
```

```
C:\mydir> set COURSE_AUTHOR=Jim Wilson

C:\mydir> java com.pluralsight.app.Main
   JIM_Y50
   C:\windows
   Jim Wilson
```

# Summary

**Apps require more than just code**

- Behavior is affected by many factors

**Command line arguments**

- Received as parameter to main method

**Properties class provides name/value pairs**

- Can persist across app executions
- Useful for preferences or simple app state
- Supports providing defaults
- Defaults can be included in app package

# Summary

**Class path controls where classes are found**

- Can set with CLASSPATH
  - Use caution
  - Changing for one app can affect others
- Better to use Java command –cp option
  - Sets path specific to each app

**Execution environment info is available**

- Java provides info in system properties
- OS environment variables accessible
  - Can provide app-specific variables