Capturing Application Activity with the Java Log System

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

Overview

Log system management

Making log calls

Log levels

Types of log methods

Creating & adding log components

Built-in handlers and formatters

Log configuration file

Making the most of the log system

- Logger naming and hierarchy

Log System

We need a way to capture app activity

- Record unusual circumstances or errors
- Track usage info
- Debug

The required level of detail can vary

- Sometimes need lots of details
  - Newly deployed app
  - App is experiencing errors
- Generally need less detail
  - App is mature and stable

Java provides a built-in solution

- java.util.logging

Log System Management

Log system is centrally managed

- There is one app-wide log manager
- Manages log system configuration
- Manages objects that do actual logging

Represented by LogManager class

- One global instance
  - Access with static method LogManager.getLogManager

Making Log Calls

Logger class

- Provides logging methods

Access Logger instances with LogManager

- Use getLogger method
- Each instance named
  - We'll talk more about this shortly
- A global logger instance is available
  - Access using the Logger class' static field GLOBAL_LOGGER_NAME

Making Log Calls

```
public class Main {
  public static void main (String[] args) {
    LogManager lm = LogManager.getLogManager();
    Logger logger = lm.getLogger(Logger.GLOBAL_LOGGER_NAME);
    logger.log(Level.INFO, “My first log message”);
    logger.log(Level.INFO, “Another message”);
  }
}
```

Making Log Calls

```
public class Main {
  static Logger logger =
   LogManager.getLogManager().getLogger(Logger.GLOBAL_LOGGER_NAME);
  public static void main (String[] args) {
    logger.log(Level.INFO, “My first log message”);
    logger.log(Level.INFO, “Another message”);
  }
}
```

Logging Levels

Levels control logging detail

- Each log entry is associated with a level
  - Included with each log call
- Each Logger has a capture level
  - Use setLevel method
  - Ignores entries below capture level

Each Level has a numeric value

- 7 basic log levels
- 2 special levels for Logger
- Can define custom levels
  - Should generally be avoided

Logging Levels

Level	Numeric Value	Description
SEVERE	1000	Serious failure
WARNING	900	Potential problem
INFO	800	General info
CONFIG	700	Configuration info
FINE	500	General developer info
FINER	400	Detailed developer info
FINEST	300	Specialized developer info

Making Log Calls

```
public class Main {
  static Logger logger =
   LogManager.getLogManager().getLogger(Logger.GLOBAL_LOGGER_NAME);
  public static void main (String[] args) {
    logger.setLevel(Level.INFO);




  }
}
```

Logging Levels

Level	Numeric Value	Description
SEVERE	1000	Serious failure
WARNING	900	Potential problem
INFO	800	General info
CONFIG	700	Configuration info
FINE	500	General developer info
FINER	400	Detailed developer info
FINEST	300	Specialized developer info

Logger

Making Log Calls

```
public class Main {
  static Logger logger =
   LogManager.getLogManager().getLogger(Logger.GLOBAL_LOGGER_NAME);
  public static void main (String[] args) {
    logger.setLevel(Level.INFO);
    logger.log(Level.SEVERE, “Uh Oh!!”);
    logger.log(Level.INFO, “Just so you know”);
    logger.log(Level.FINE, “Hey developer dude”);
    logger.log(Level.FINEST, “You’re special”);
  }
}
```

Logging Levels

Level	Numeric Value	Description
SEVERE	1000	Serious failure
WARNING	900	Potential problem
INFO	800	General info
CONFIG	700	Configuration info
FINE	500	General developer info
FINER	400	Detailed developer info
FINEST	300	Specialized developer info

Logger

Making Log Calls

```
public class Main {
  static Logger logger =
   LogManager.getLogManager().getLogger(Logger.GLOBAL_LOGGER_NAME);
  public static void main (String[] args) {
    logger.setLevel(Level.FINE);
    logger.log(Level.SEVERE, “Uh Oh!!”);
    logger.log(Level.INFO, “Just so you know”);
    logger.log(Level.FINE, “Hey developer dude”);
    logger.log(Level.FINEST, “You’re special”);
  }
}
```

Logging Levels

Level	Numeric Value	Description
OFF	Integer.MAX_VALUE	Logger capture nothing
SEVERE	1000	Serious failure
WARNING	900	Potential problem
INFO	800	General info
CONFIG	700	Configuration info
FINE	500	General developer info
FINER	400	Detailed developer info
FINEST	300	Specialized developer info

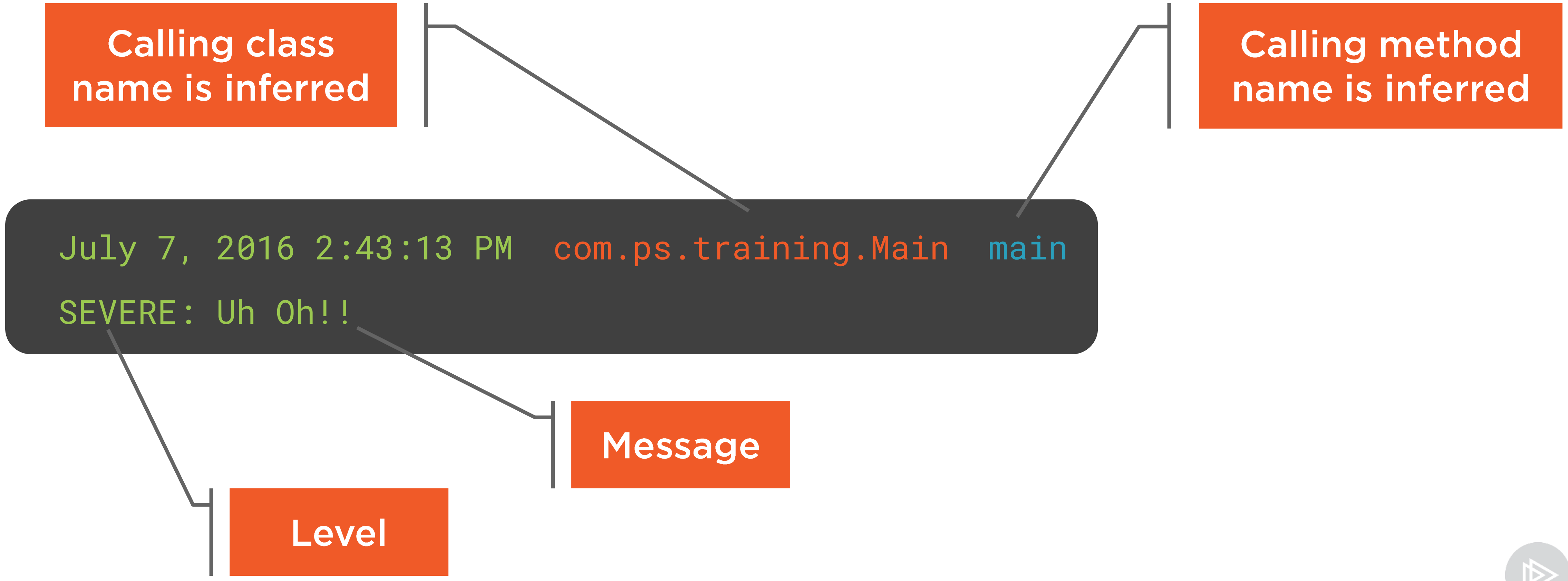
Types of Log Methods

Logger supports several logging methods

- Simple log method
- Level convenience methods
- Precise log method
- Precise convenience methods
- Parameterized message methods

Simple Log Method

```
logger.log(Level.SEVERE, “Uh Oh!!”);
```



```
July 7, 2016 2:43:13 PM  com.ps.training.Main  main
SEVERE: Uh Oh!!
```

Calling class name is inferred
Calling method name is inferred
Message
Level

Level Convenience Methods

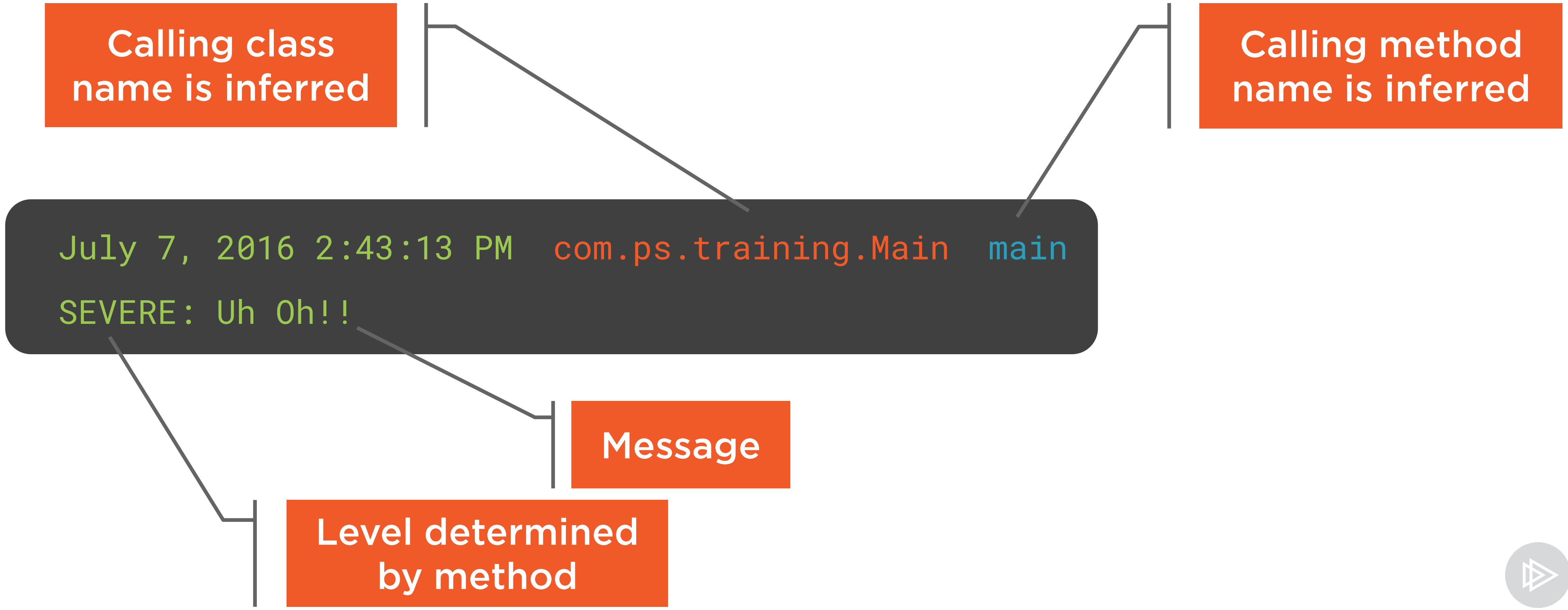
Level convenience methods

- Method name implies log level
- Only need to pass the message

Method	Level
severe	Level.SEVERE
warning	Level.WARNING
info	Level.INFO
config	Level.CONFIG
fine	Level.FINE
finer	Level.FINER
finest	Level.FINEST

Level Convenience Method

```
logger.severe("Uh Oh!!");
```



```
July 7, 2016 2:43:13 PM  com.ps.training.Main  main
SEVERE: Uh Oh!!
```

Calling class name is inferred
Calling method name is inferred
Message
Level determined by method

Precise Log Method

Standard log methods infer calling info

- Sometimes get it wrong

Use precise log methods to avoid issue

- Named logp
- Calling class and method names passed

Precise Log Method

```
logger.logp(Level.SEVERE,
            “com.jwhh.support.Other”
```

```
July 7, 2016 2:43:13 PM  com.jwhh.support.Other  myMethod
SEVERE: It broke!!
```

Precise Convenience Methods

Precise convenience methods

- Simplify logging common method actions
- Logs a predefined message
- Always logged as Level.FINER

Method	Message
entering	ENTRY

Precise Convenience Methods

```
void doWork() {
   logger.setLevel(Level.ALL);
   logger.entering("com.jwhh.support.Other", "doWork");
   logger.logp(Level.WARNING, "com.jwhh.support.Other", doWork", "Empty Function");
   logger.exiting("com.jwhh.support.Other", "doWork");
}
```

```
July 7, 2016 2:43:13 PM  com.jwhh.support.Other  doWork
FINER: ENTRY

July 7, 2016 2:43:13 PM  com.jwhh.support.Other  doWork
WARNING: Empty Function

July 7, 2016 2:43:13 PM  com.jwhh.support.Other  doWork
FINER: RETURN
```

Parameterized Message Methods

Some methods support message parameters

- log, logp
  - Parameter substation indicators explicitly appear within the message
    - Uses simple positional substitution
    - Zero-based index within brackets {*N*}
- entering, exiting
  - Values appear after default message
    - Space separated
- Values always passed as object
  - Accept individual object or object array

Parameterized Message Methods

```
logger.log(Level.INFO, “{0} is my favorite”, “Java”);

logger.log(Level.INFO, “{0} is {1} days from {2}”, new Object[]{“Wed”, 2, “Fri”});
```

```
July 7, 2016 2:43:13 PM  com.ps.training.Main  main
INFO: Java is my favorite

July 7, 2016 2:43:13 PM  com.ps.training.Main  main
INFO: Wed is 2 days from Fri
```

Parameterized Message Methods

```
doWork(“Jim”, “Wilson”);
```

```
void doWork(String left, String right) {
   logger.entering(“com.jwhh.support.Other”, “doWork”, new Object[]{left, right});
   String result = “<” + left + right + “>”;
   logger.exiting(“com.jwhh.support.Other”, “doWork”, result);
}
```

```
July 7, 2016 2:43:13 PM  com.jwhh.support.Other  doWork
FINER: ENTRY

July 7, 2016 2:43:13 PM  com.jwhh.support.Other  doWork
FINER: RETURN
```

Log System Divided into Components

Log system is divided into components

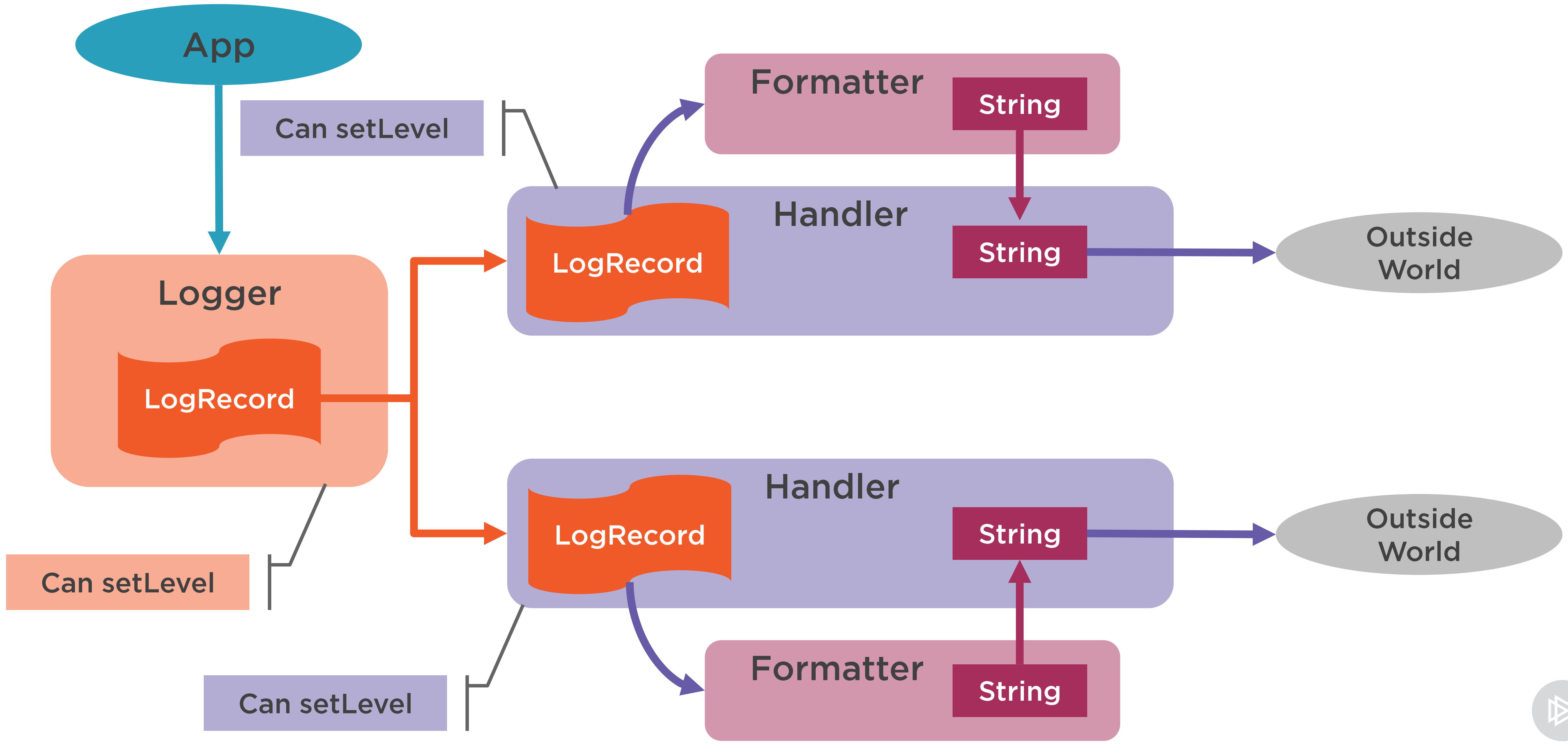
- Each component handles specific task
- Easy to setup common behaviors
- Provides flexibility

Core Log Components

Consists of 3 core components

- Logger
  - Accepts app calls
- Handler
  - Publishes logging information
  - A Logger can have multiple
- Formatter
  - Formats log info for publication
  - Each Handler has 1 Formatter

Core Logging Component Relationship



Creating/Adding Log Components

Creating a Logger

- Use Logger.getLogger static method
- Loggers named with a string
- Once created accessible in LogManager

Adding a Handler

- Java provides built-in Handlers
- Add with Logger.addHandler

Adding a Formatter

- Java provides built-in Formatters
- Add with Handler.setFormatter

Creating/Adding Log Components

```
public class Main {
  static Logger logger = Logger.getLogger("com.pluralsight");
  public static void main (String[] args) {
    Handler h = new ConsoleHandler();
    Formatter f = new SimpleFormatter();
    h.setFormatter(f);
    logger.addHandler(h);
    logger.setLevel(Level.INFO);
    logger.log(Level.INFO, "We're Logging!");
  }
}
```

Built-in Handlers

Java provides several built-in Handlers

- Inherit directly or indirectly from Handler

Commonly used built-in Handlers

- ConsoleHandler
  - Writes to System.err
- StreamHandler
  - Writes to specified OutputStream
- SocketHandler
  - Writes to a network socket
- FileHandler
  - Writes to 1 or more files

FileHandler

FileHandler output options

- Can output to a single file
- Can output to a rotating set of files

Working with rotating set of files

- Specify approximate max size in bytes
- Specify max number of files
- Cycles through reusing oldest file

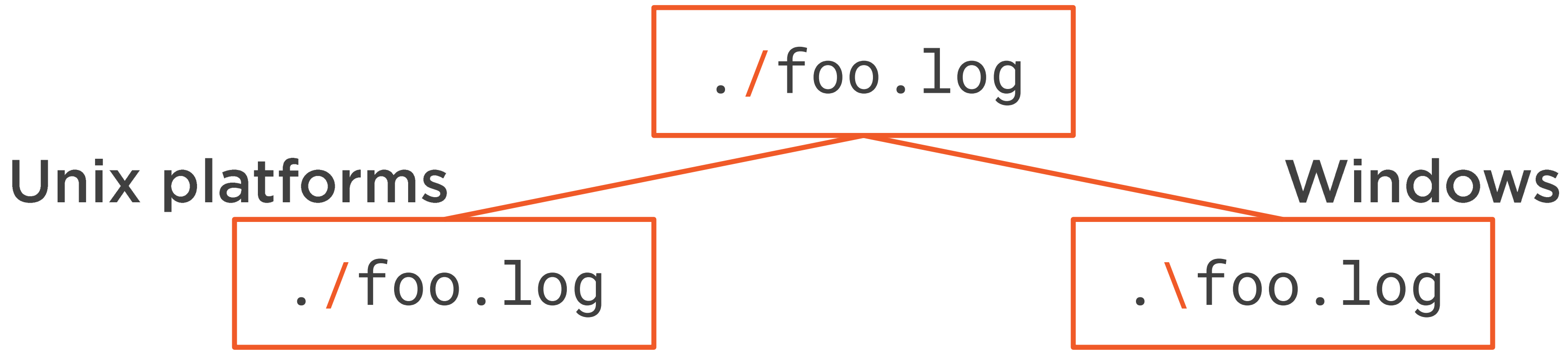
FileHandler Substitution Pattern

Supports a substitution-based file naming

- Reduces issues related to system and configuration differences
- Automates rotating file set naming

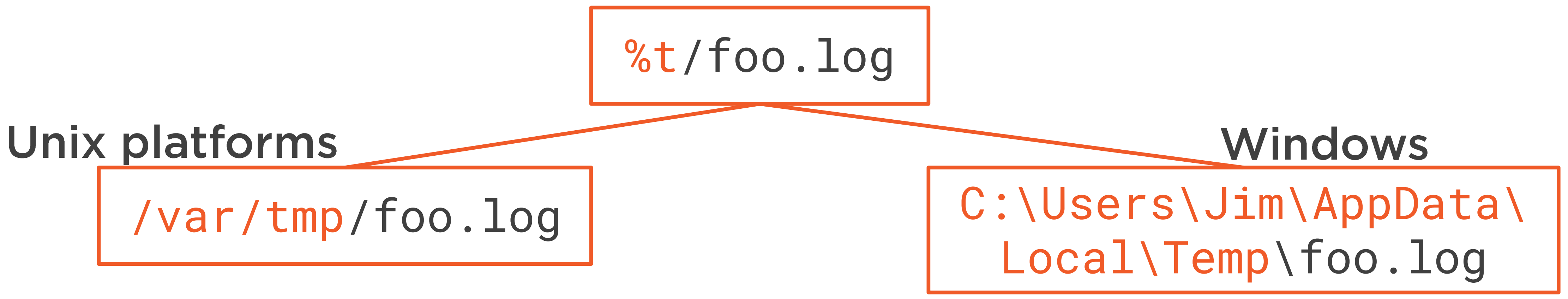
FileHandler Substitution Pattern Values

Value	Meaning
/	Platform slash\backslash



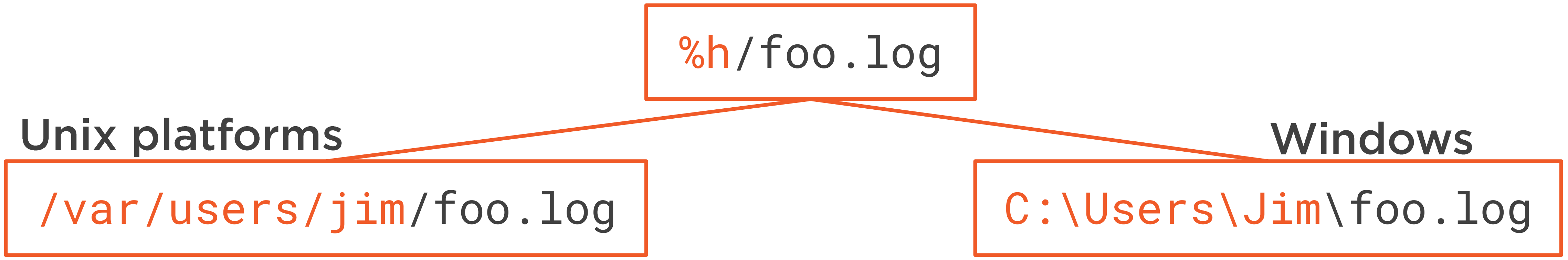
FileHandler Substitution Pattern Values

Value	Meaning
/	Platform slash\backslash
%t	Temp directory



FileHandler Substitution Pattern Values

Value	Meaning
/	Platform slash\backslash
%t	Temp directory
%h	User's home directory



```
%h/foo.log
```

Unix platforms

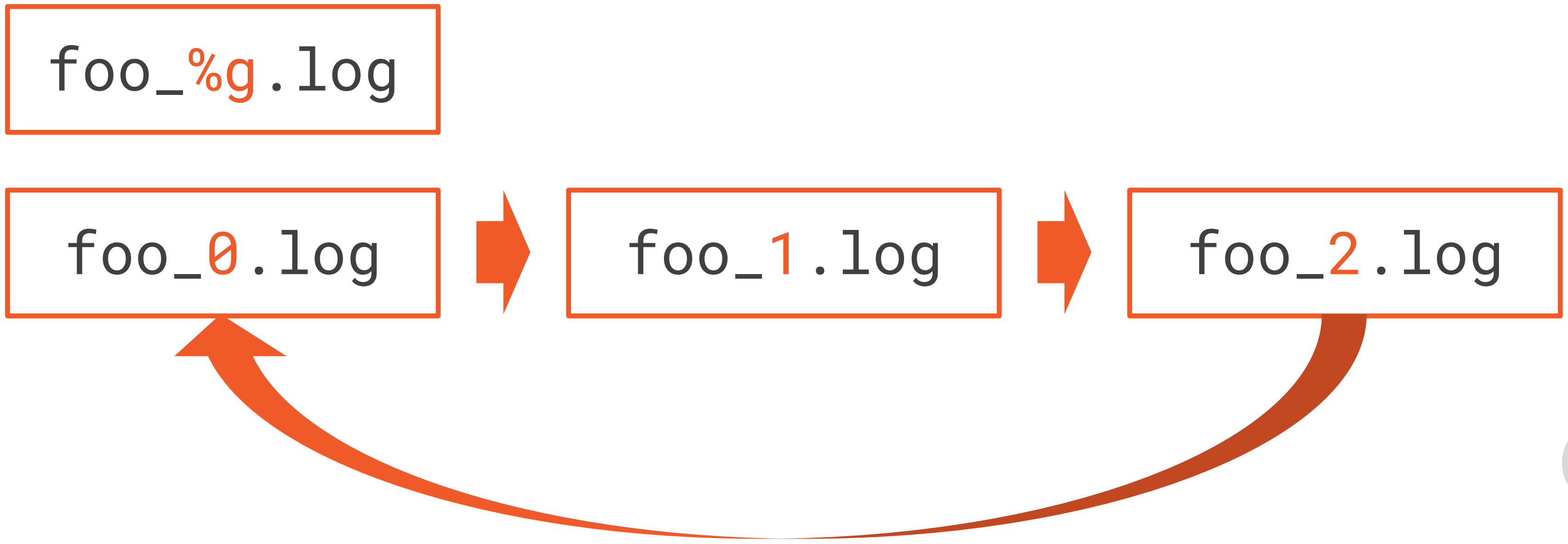
```
/var/users/jim/foo.log
```

Windows

```
C:\Users\Jim\foo.log
```

FileHandler Substitution Pattern Values

Value	Meaning
/	Platform slash\backslash
%t	Temp directory
%h	User's home directory
%g	Rotating log generation



Logging with FileHandler

```
public class Main {
  static Logger logger = Logger.getLogger("com.pluralsight");
  public static void main (String[] args) {

    FileHandler h = new FileHandler("%h/myapp_%g.log", 1000, 4);

    h.setFormatter(new SimpleFormatter());
    logger.addHandler(h);
    // Do something
  }
}
```

Rotating set of 4

Each about 1000 bytes

```
C:\Users\Jim\myapp_0.log
C:\Users\Jim\myapp_1.log
C:\Users\Jim\myapp_2.log
C:\Users\Jim\myapp_3.log
```

Built-in Formatters

Java provides two built-in Formatters

- Both inherit directly from Formatter

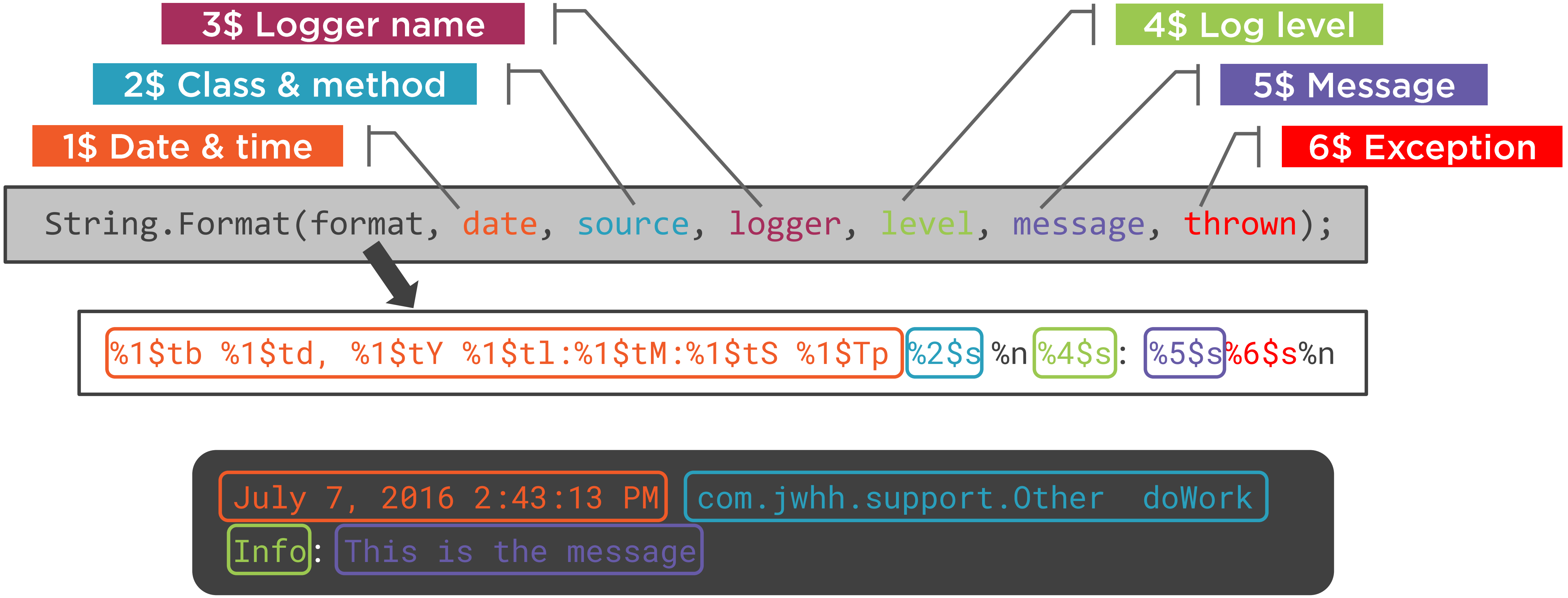
XMLFormatter

- Formats content as XML
- Root element named log
- Each entry in element named record

SimpleFormatter

- Formats content as simple text
- Format is customizable
    - Uses standard formatting notation

SimpleFormatter Formatting



```
String.Format(format, date, source, logger, level, message, thrown);
```

```
%1$tb %1$td, %1$tY %1$tl:%1$tM:%1$tS %1$Tp %2$s %n %4$s: %5$s%6$s%n
```

```
July 7, 2016 2:43:13 PM com.jwhh.support.Other  doWork
Info: This is the message
```

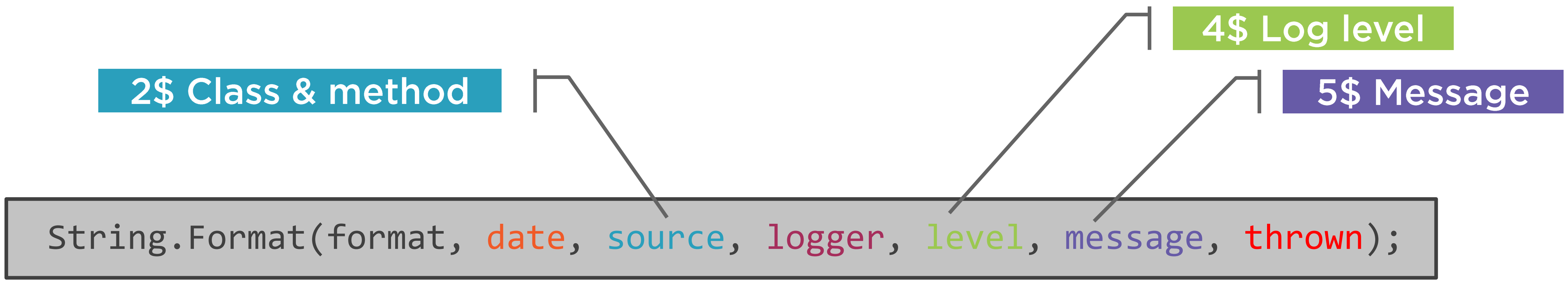
Customizing the Format String

Set format string with a system property

- java.util.logging.SimpleFormatter.format
- Pass value with Java –D option

SimpleFormatter Formatting

```
C:\> java -Djava.util.logging.SimpleFormatter.format=%5$s,%2$s,%4$s%n
     com.pluralsight.training.Main
```



```
String.Format(format, date, source, logger, level, message, thrown);
```

```
This is the message,com.jwhh.support.Other  doWork,INFO
```

Log Configuration File

Configuration info can be set in a file

- Follows standard properties file format
- Can replace code-based config
- Can be used with code-based config

Set file name with a system property

- java.util.logging.config.file
- Pass value with Java –D option

Identifying Configuration Values

Specific values depend on classes

- Most code-based options available

Naming of values for Handlers & Formatters

- Fully qualified class name
- Followed by a “dot” and the value name

Naming of values for Loggers

- Name of Logger as passed to getLogger
- Followed by a “dot” and the value name

Logging Code-based Configuration

```
public class Main {
  static Logger logger = Logger.getLogger("com.pluralsight");
  public static void main (String[] args) {
    Handler h = new ConsoleHandler();
    h.setLevel(Level.ALL);
    h.setFormatter(new SimpleFormatter());
    logger.addHandler(h);
    logger.setLevel(Level.ALL);
    logger.log(Level.INFO, "We're Logging!");
  }
}
```

```
java -Djava.util.logging.SimpleFormatter.format=%5$s,%2$s,%4$s%n
com.pluralsight.training.Main
```

Logging Configuration File

log.properties

```
java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
com.pluralsight.handlers = java.util.logging.ConsoleHandler
com.pluralsight.level = ALL
java.util.logging.SimpleFormatter.format = %5$s,%2$s,%4$s%n
```

Logging Configuration File

```
java -Djava.util.logging.config.file=log.properties com.pluralsight.training.Main
```

```
public class Main {
  static Logger logger = Logger.getLogger("com.pluralsight");
  public static void main (String[] args) {

  }
}
```

Logging Configuration File

log.properties

```
✓ java.util.logging.ConsoleHandler.level = ALL
✓ java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
✓ com.pluralsight.handlers = java.util.logging.ConsoleHandler
✓ com.pluralsight.level = ALL
✓ java.util.logging.SimpleFormatter.format = %5$s,%2$s,%4$s%n
```

Logging Configuration File

```
java -Djava.util.logging.config.file=log.properties com.pluralsight.training.Main
```

```
public class Main {
  static Logger logger = Logger.getLogger(“com.pluralsight”);
  public static void main (String[] args) {
    logger.log(Level.INFO, “We’re Logging!”);
  }
}
```

Logging Configuration File

log.properties

```
java.util.logging.ConsoleHandler.level = ALL
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
com.pluralsight.handlers = java.util.logging.ConsoleHandler
com.pluralsight.level = ALL
java.util.logging.SimpleFormatter.format = %5$s,%2$s,%4$s%n
```

Logger Naming

Naming implies a parent-child relationship

- LogManager links Loggers in a hierarchy based on each Logger’s name

Logger naming

- Should following hierarchical naming
- Corresponds to type hierarchy
  - Each “dot” separates a level
- Generally tied to a class’ full name

Logger Naming

```
package com.ps.training;
public class Main {
  static Logger pkgLogger = Logger.getLogger("com.ps.training");
  static Logger logger = Logger.getLogger("com.ps.training.Main");
  public static void main { ... }
}
```

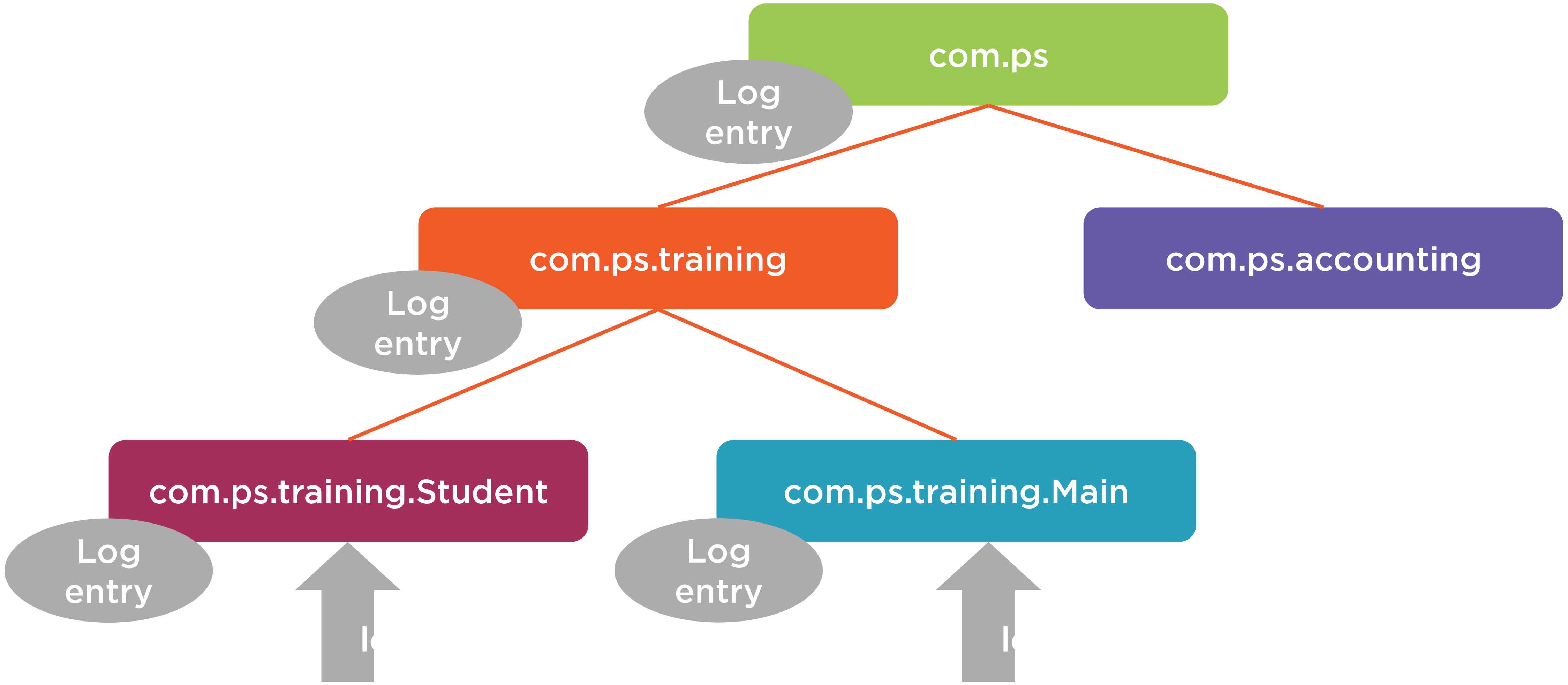
com.ps.training

com.ps.training.Main

```
package com.ps.training;
public class Student {
  static Logger logger = Logger.getLogger("com.ps.training.Student");
  // . . .
}
```

com.ps.training.Student

Logger Naming Hierarchy



Leveraging Logger Naming Hierarchy

Making the most of the hierarchical system

- Focus on capturing important info
  - With the option to get details if needed
- Manage setup primarily on parents
- Manage log calls primarily at children

Logging Hierarchy and Levels

Loggers do not require their to be level set

- Log level can be null
  - Will inherit parent level
- Primarily set level on parents
  - Normally somewhat restrictive level
- Set more detail level on child if needed

Logging Hierarchy and Handlers

Loggers do not require handlers

- A Logger doesn’t log if no handler
  - But does pass up to parent Logger
- Primarily add Handlers to upper parents
- Add Handlers to child if needed

Logger Naming

```
package com.ps.training;
public class Main {
 static Logger pkgLogger = Logger.getLogger("com.ps.training");
 static Logger logger = Logger.getLogger("com.ps.training.Main");
 public static void main {
   logger.entering("com.ps.training", "Main");
   logger.log(Level.INFO, "We're Logging!");
   logger.exiting("com.ps.training", "Main");
 }
}
```

Not logged

Logged to com.ps.training

Not logged

```
com.ps.training.handlers=java.util.logging.ConsoleHandler
com.ps.training.level=INFO
```

Logger Naming

```
package com.ps.training;
public class Main {
 static Logger pkgLogger = Logger.getLogger("com.ps.training");
 static Logger logger = Logger.getLogger("com.ps.training.Main");
 public static void main {
   logger.entering("com.ps.training", "Main");
   logger.log(Level.INFO, "We're Logging!");
   logger.exiting("com.ps.training", "Main");
 }
}
```

Not logged

Logged to com.ps.training

Logged to com.ps.training.Main

Not logged

```
com.ps.training.handlers=java.util.logging.ConsoleHandler
com.ps.training.level=INFO
java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.pattern=./main_%g.log
com.ps.training.Main.handlers=java.util.logging.FileHandler
```

Logger Naming

```
package com.ps.training;
public class Main {
 static Logger pkgLogger = Logger.getLogger("com.ps.training");
 static Logger logger = Logger.getLogger("com.ps.training.Main");
 public static void main {
   logger.entering("com.ps.training", "Main");
   logger.log(Level.INFO, "We're Logging!");
   logger.exiting("com.ps.training", "Main");
 }
}
```

Logged to com.ps.training.Main

Logged to com.ps.training

Logged to com.ps.training.Main

Logged to com.ps.training.Main

```
com.ps.training.handlers=java.util.logging.ConsoleHandler
com.ps.training.level=INFO
java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.pattern=./main_%g.log
com.ps.training.Main.handlers=java.util.logging.FileHandler
com.ps.training.Main.level=ALL
```

Summary

Log system is centrally managed

- One app-wide manager
- Represented by LogManager class

Logger class

- Represents each individual logger
- Provides log methods

Levels indicate relative importance of entry

- Each entry recorded with a level
- Each Logger has a capture level
  - Ignores entries below capture level

Summary

Loggers rely on other components

- Handlers
  - Publish log info
  - A Logger can have multiple handlers
- Formatters
  - Format log info for publication
  - Each Handler has 1 formatter

Log configuration

- Can be handled in code
- Can be handled with a file
  - File name passed with system property

Summary

Loggers are hierarchical

- Hierarchy established through naming
- Loggers can pass log entries to parent
- Loggers can inherit parent log level

Getting the most from the log system

- Manage setup primarily on parent loggers
- Make log calls primarily on child loggers