Unit 1: Logging, Enumerations

APPLICATION PROGRAMMING IN JAVA

Basics

- Instructor: Jack Straub
- E-mail: jstraub@centurytel.net
- Phone: 425-88-9119
- Office hours: before and after class; by appointment

Grading

- 9 assignments in *Assignments Group*
  - 50% of your grade
  - Must receive 50%+ on each assignment
  - Must achieve cumulative 60%+ on all assignments
- 1 assignment in *All Assignments Complete Group*
  - 50% achieved on each of first 9 assignments? 100%
  - Less than 50% on any one of the first 9 assignments? 0%
- Must attend eight classes

- *See* Grading *page in the modules section of the class Canvas site*

Calendar

- See *Calendar* in syllabus

Grading – Synopsis

- Code must compile
  - Assignments that do not compile will receive NO credit
- You must complete ALL assignments
  - If you fail do not complete an assignment you will NOT pass the class
- All code is must be documented as required
- All coding conventions must be followed
- Every class must have a JUnit test

See also the *Grading* page in the Modules section of the class Canvas site

Topics Covered

- Logging
- Nested classes
- I/O streams
- Collections
- Lambdas
- Data Streams
- Serialization
- JavaBeans
- Database access
- Networking
- Threads

Concepts

LOGGING

Logger

- Logs messages to one or more destinations
  - Console
  - File
  - Network host
- Part of a hierarchy of loggers
- Must have a unique name
- Why use a logger?
  - You might not have a console
  - Configuration flexibility
  - Manage multiple destinations

Digression: Static Initialization Blocks

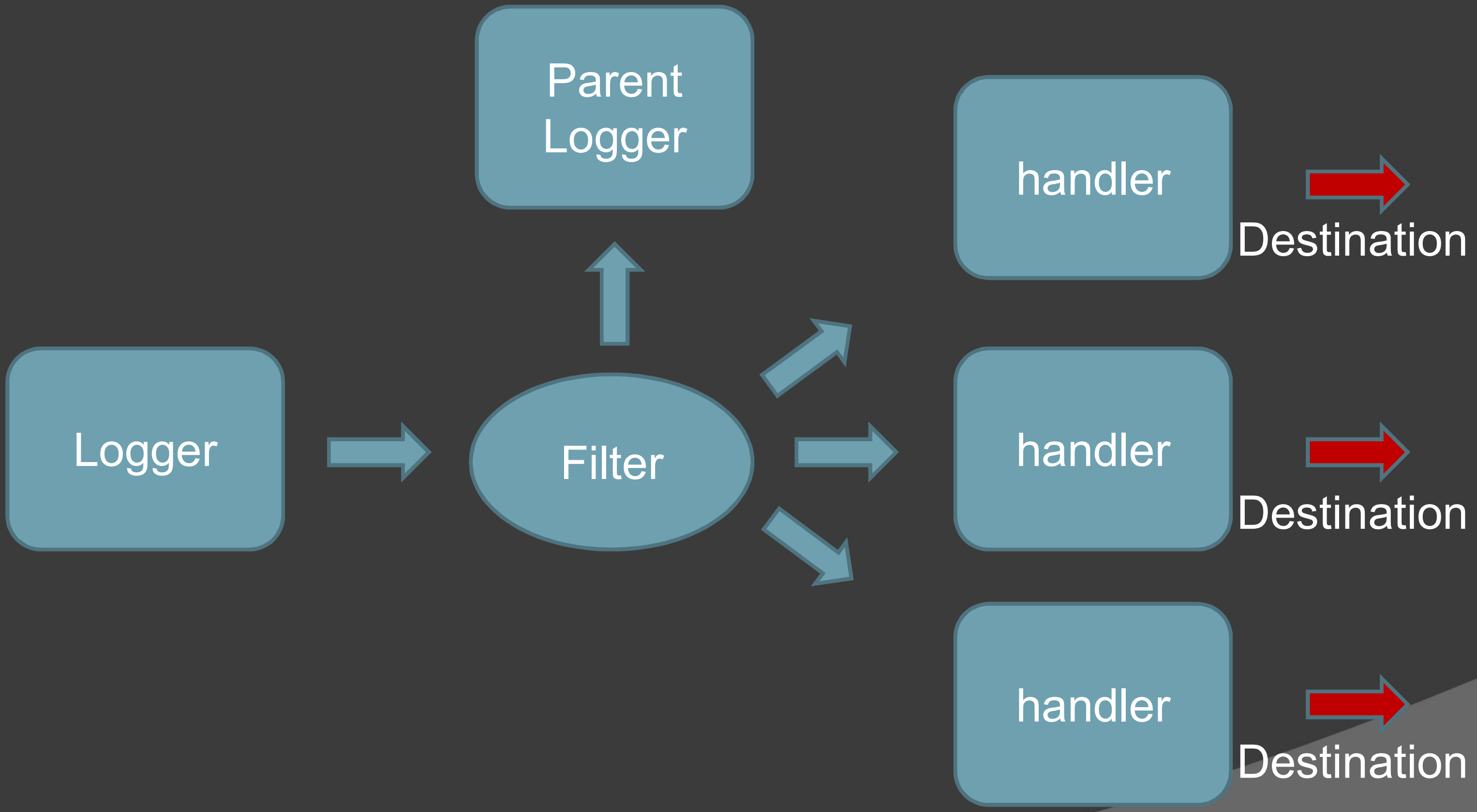
- Static initialization blocks can be used like a “constructor” for class members
- The static keyword and a {} block are placed directly in the body of the class

```
static
{
     ...
}
```

Static Initialization Block Example

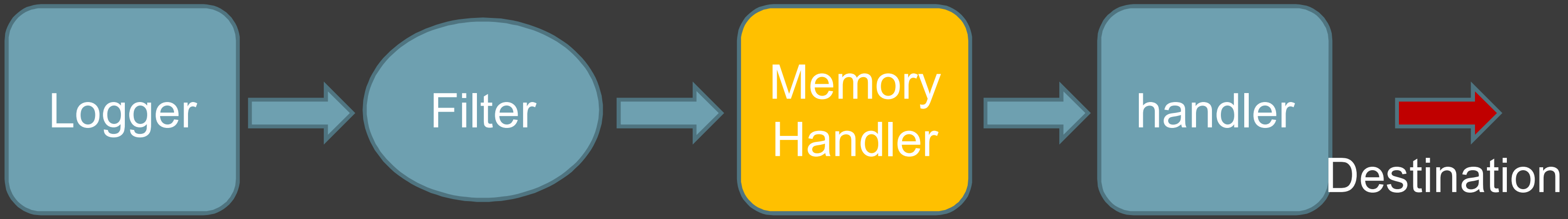
```
public class StaticInitBlockDemo
{
    private static final LocalDate      startTime;
    private static final List<String>   nameList;

    static
    {
        startTime = LocalDate.now();
        nameList = new ArrayList<>();
        nameList.add( "Manny" );
        nameList.add( "Moe" );
        nameList.add( "Jack" );
    }
    ...
```

Typical Logging Configuration 1



Typical Logging Configuration 2

- Memory handler keeps list of unused log records



Logger Name

JDK Sample

- A sequence of strings that define a unique name
- Usually represented in dot notation:

com.scg.logtest.LogTest

- Usually formed from the FQN of the class

```
// java.util.logging example
private static final Logger LOGGER  =
    Logger.getLogger( Demo1.class.getName() );
```

- You should get in the habit of creating a logger at the start of all your classes

Logger: Simple Example

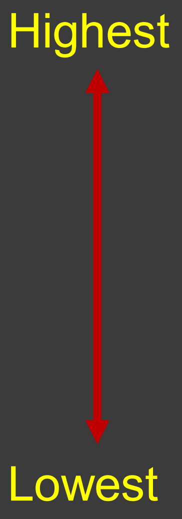
JDK
Sample

```
// java.util.logging example
public class PartProcessor
{
    private static final Logger LOGGER  =
        Logger.getLogger( PartProcessor.class.getName() );
    public void processPart( Part part )
    {
        stage1( part );
        stage2( part );
        stage3( part );
    }
    private void stage1( Part part )
    {
       LOGGER.info("Begin stage 1 processing for " + part);
        // do stage 1 processing
       LOGGER.info( "End stage 1 processing for " + part );
    }
    ...
```

Log Levels

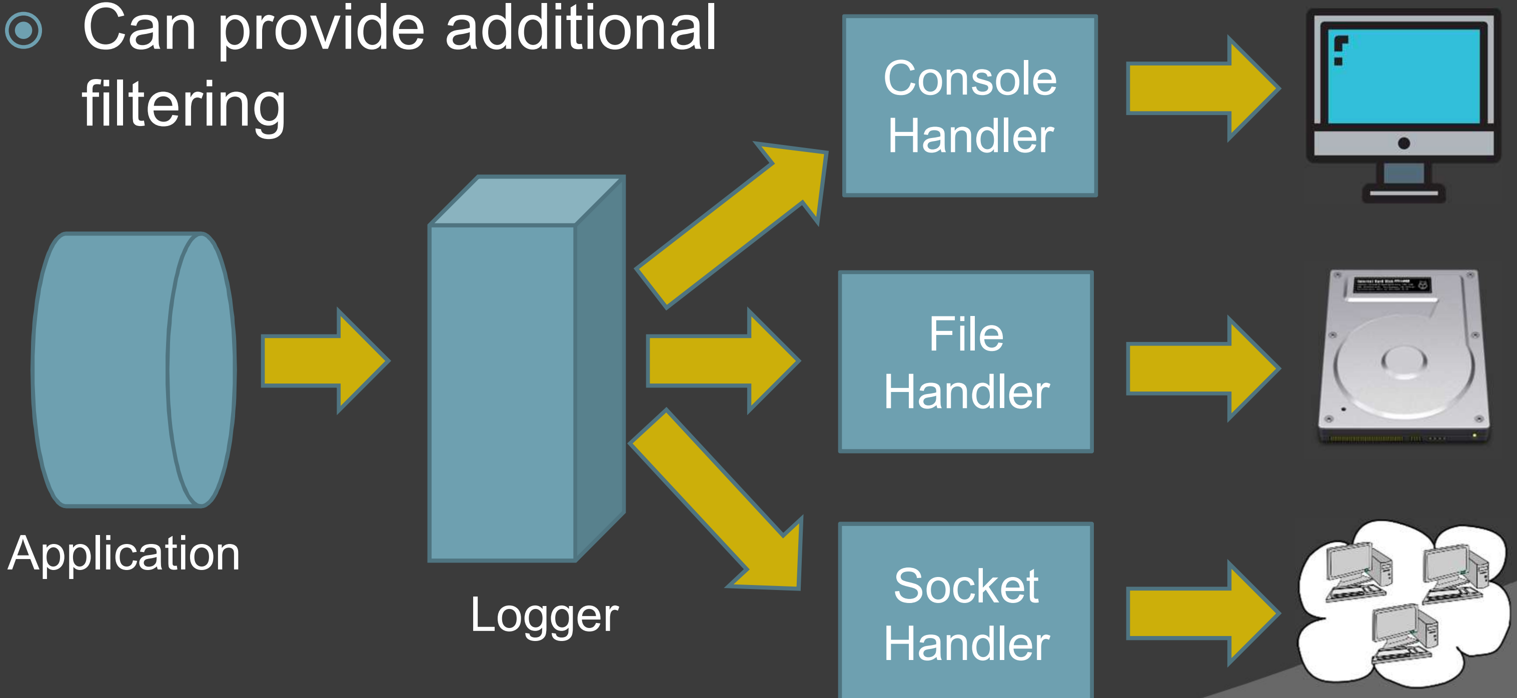
JDK Sample

- Determines how important a message is
- Logging is filtered by log level
- Example:
  - SEVERE
  - WARNING
  - INFO
  - CONFIG
  - FINE
  - FINER
  - FINEST



Handlers/Appenders

- *Handler* and *appender* refer to the same thing
- Format log message for different destinations
- Can provide additional filtering



Managing a Logger

JDK Sample

```
// java.util.logging example
public class Demo1
{
    private static final Logger     LOGGER  =
        Logger.getLogger( Demo1.class.getName() );

    static
    {
        LOGGER.setFilter( (r)->
            {return !r.getMessage().contains( "no log");});
        LOGGER.setLevel( Level.ALL );
        LOGGER.addHandler( new ConsoleHandler() );
    }
    ...
```

Equivalent Use of Anonymous Class

JDK Sample

- Alternative to lambda on previous slid

```
Filter filter = new Filter() {
    public boolean isLoggable( LogRecord record )
    {
        return record.getMessage().contains( "no log" );
    }
};
LOGGER.setFilter( filter );
```

Using Multiple Handlers

JDK Sample

```
static
{
    String  name    = "demo1.log";
    try
    {
        LogManager.getLogManager().reset();
        FileHandler     fHandler    =
            new FileHandler( name, true );
        logger.addHandler( fHandler );

        SocketHandler   sHandler    =
            new SocketHandler( "localhost", 1954 );
        logger.addHandler( sHandler );
    }
    catch ( IOException | SecurityException exc )
    ...
```

Configuration Management

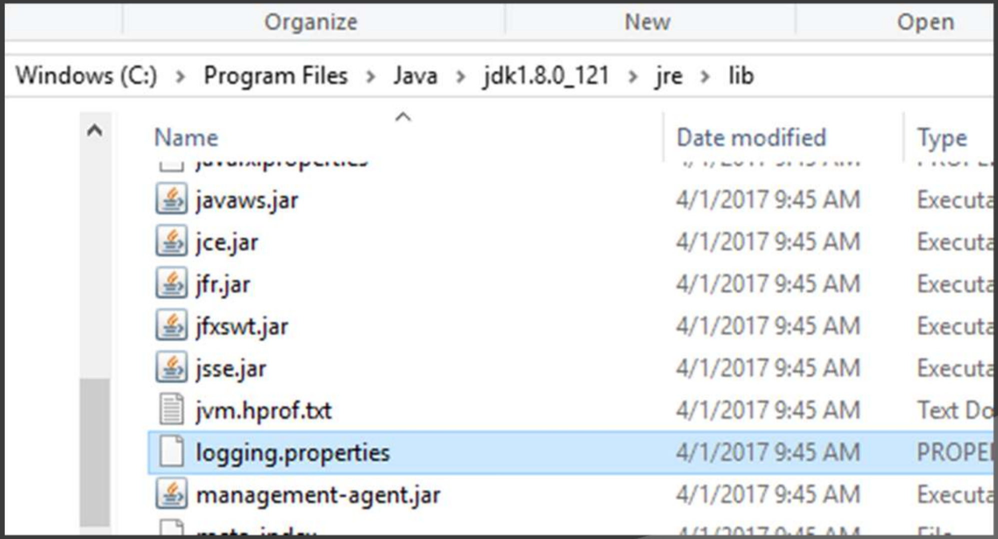
Foreshadowing

- Configuration management via JDK
- Runtime operation via SLF4J API

java.util.logging Configuration

- Copy the default logging configuration file from your JDK to the src/main/resources directory of your project
  - You'll find it in jdk1.8xxx/jre/lib/logging.properties

Copy from:



Windows (C:) › Program Files › Java › jdk1.8.0_121 › jre › lib

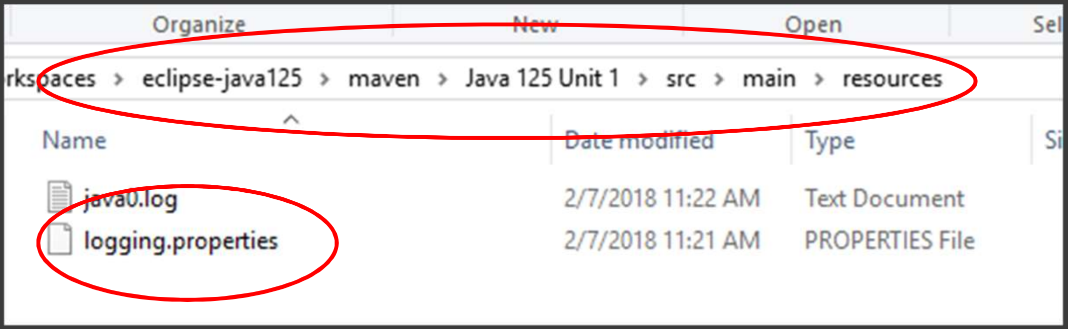
Name	Date modified	Type
javaws.jar	4/1/2017 9:45 AM	Executa
jce.jar	4/1/2017 9:45 AM	Executa
jfr.jar	4/1/2017 9:45 AM	Executa
jfxswt.jar	4/1/2017 9:45 AM	Executa
jsse.jar	4/1/2017 9:45 AM	Executa
jvm.hprof.txt	4/1/2017 9:45 AM	Text Do
logging.properties	4/1/2017 9:45 AM	PROPE
management-agent.jar	4/1/2017 9:45 AM	Executa

Continued on next slide

Continued from previous slide

java.util.logging Configuration

Copy to:



rkspaces › eclipse-java125 › maven › Java 125 Unit 1 › src › main › resources

Name	Date modified	Type	Si
java0.log	2/7/2018 11:22 AM	Text Document	
logging.properties	2/7/2018 11:21 AM	PROPERTIES File	

Continued on next slide

Continued from previous slide

Configure the FileHandler

- Edit the logging.properties file in your resource directory:
  - Comment out (using the #) the “handlers=“:

```
#handlers= java.util.logging.ConsoleHandler
```

  - Uncomment the line that includes FileHandler:

```
# To also add the FileHandler, use the following…
handlers= java.util.logging.FileHandler, java.util.log…
```

Continued on next slide

Continued from previous slide

Configure the FileHandler

- Edit the logging.properties file in your resource directory:
  - Change …FileHandler.pattern from this:

```
%h/java%u.log
```

  - To this:

```
src/main/resources/java%u.log
```

Continued on next slide

Continued from previous slide

Configure the FileHandler

- Add this VM argument to your project run configuration (all on one line with no spaces):

```
-Djava.util.logging.config.file=
      src/main/resources/logging.properties
```

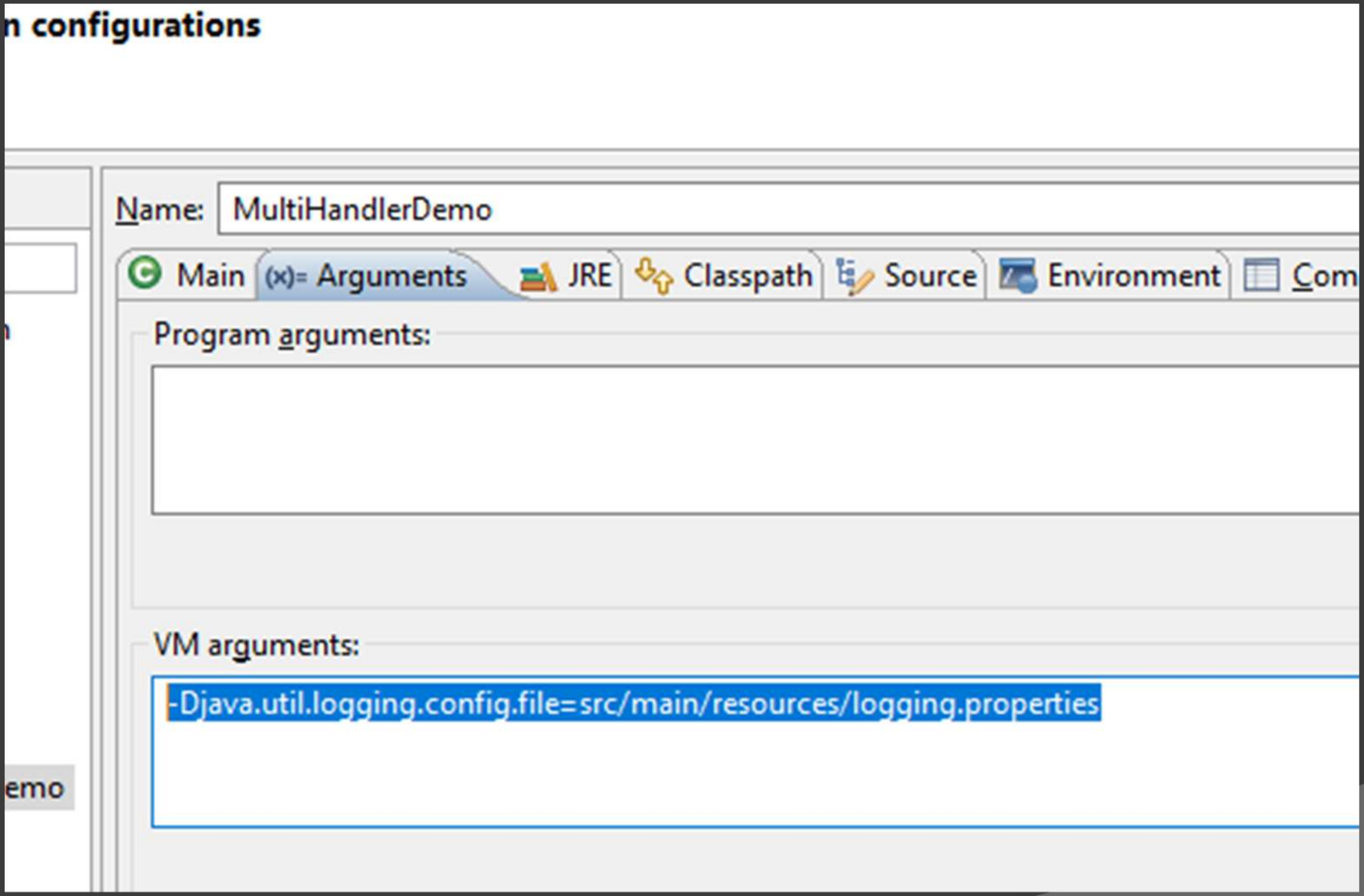
```
-Djava.util.logging.config.file=src/main/resources/logging.properties
```

Must be entered on a single line with no spaces

Continued on next slide

Continued from previous slide

Configure the FileHandler



Name: MultiHandlerDemo

Program arguments:

VM arguments:

```
-Djava.util.logging.config.file=src/main/resources/logging.properties
```

Continued on next slide

Popular Loggers

- java.util.logging
- org.apache.commons.logging (Log4j)
- ch.qos.logback.classic.Logger (Logback)

Choosing a Logger

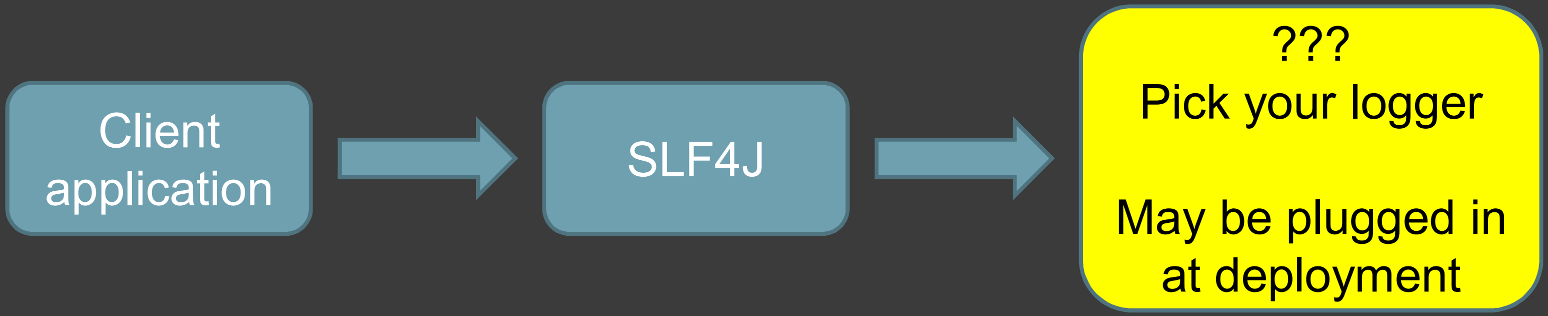
- A lot of people don't seem to like java.util.logging
- Different loggers can be good for different things
- Ease-of-use vs functionality

Difference Between Loggers, Example

- Log levels
  - Java: severe, warning, info, config, fine, finer, finest
  - Log4j: fatal, error, warn, info, debug, trace
- Log an exception
  - `Java: LOG.log(Level.SEVERE,e.getMessage(),e);`
  - Log4j: `logger.error( e );`
- Create a logger
  - Java:
    ```
    Logger LOGGER  =
            Logger.getLogger(Clazz.class.getName());
    ```
  - Log4j: `Logger LOG = Logger.getLogger(getClass());`

SLF4J

- Simple Logging Facade for Java (SLF4J)
- Uses the *façade* pattern



SLF4J Maven Configuration

- Simple Logging Facade for Java (SLF4J)
- Uses the *façade* pattern
- Must add two dependencies:
  - Dependency for SLF4j itself
  - Dependency for SLF4j *binding*
- Binding determines which library to use
  - We will bind to the *java.util.logging*

Continued on next slide

Continued from previous slide

SLF4J Maven Configuration

- The SLF4j dependency:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.22</version>
</dependency>
```

- The java.util.logging binding:

```
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-jdk14</artifactId>
    <version>1.7.22</version>
    <scope>runtime</scope>
</dependency>
```

SLF4J LoggerFactory

slf4j
Sample

- Given:

```
public class Slf4jFormatDemo
```

- getLogger(Class<?>)
  Obtain a logger using a Class instance

```
private static final Logger LOGGER  =
    LoggerFactory.getLogger( Slf4jFormatDemo.class );
```

- getLogger(String name)
  Obtain a logger using the given name

```
private static final Logger LOGGER  =
    LoggerFactory.getLogger( “app.Slf4jFormatDemo” );
```

SLF4J Logger Common Methods

slf4j
Sample

Method	Usage
trace(String msg)	Log *msg* at the trace level
debug(String msg)	Log *msg* at the debug level
info(String msg)	Log *msg* at the info level
warn(String msg)	Log *msg* at the warning level
error(String msg)	Log *msg* at the error level
fatal(String msg)	Log *msg* at the fatal level
info(String msg, Throwable exc)	Log an exception
debug(String msg, Throwable exc)	Log an exception
etc.	

SLF4J Logger Common Methods, Example

slf4j
Sample

```
public class Slf4jCommonDemo
{
    private static final Logger LOGGER  =
        LoggerFactory.getLogger( Slf4jCommonDemo.class );

    public static void main(String[] args)
    {
        LOGGER.info( "Info message" );
        LOGGER.error( "Error message" );
        LOGGER.debug( "Debug message" );

        int age = getAge();
        if ( age < 0 )
            LOGGER.info( "Invalid age given" );
        else
            LOGGER.info( "Age = " + age );
    }
    ...
```

Continued on next slide

SLF4J Logger Common Methods, Example

Continued from previous slide

slf4j Sample

```
...
private static int getAge()
{
    String  str =
        JOptionPane.showInputDialog( "Enter your age" );
    int     num = -1;
    try
    {
        num = Integer.parseInt( str );
    }
    catch ( NumberFormatException exc )
    {
        LOGGER.warn( "Invalid operator entry", exc );
        num = -1;
    }

    return num;
}
```

More SLF4J Logger Methods

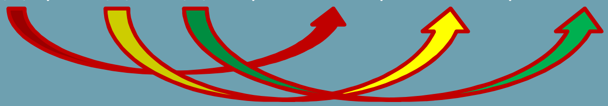
slf4j
Sample

- info(String format, Object arg)
- info(String format, Object arg1, Object arg2)
- info(String format, Object... arguments)
- debug(String format, Object arg)
- debug(String format, Object arg1, Object arg2)
- debug(String format, Object... arguments)
- error(String format, Object arg)
- etc.

SLF4J Formatting Example

slf4j
Sample

```
public class Slf4jFormatDemo
{
    private static final Logger LOGGER  =
        LoggerFactory.getLogger( Slf4jFormatDemo.class );

    public static void main(String[] args)
    {
        String  fir = "George";
        String  mid = "M.";
        String  las = "Cohen";
        LOGGER.info( "{} has logged in", las );
        LOGGER.info( "Send to {}, {} {}", las, fir, mid );
    }
}
```



SLF4J Formatting Example Output

slf4j Sample

```
Feb 11, 2018 11:21:38 AM app.Slf4jFormatDemo main
INFO: Cohen has logged in
Feb 11, 2018 11:21:38 AM app.Slf4jFormatDemo main
INFO: Send to Cohen, George M.
```

Enumerations

Enumerations

- *Enum* creates a special kind of type
- An enum type encapsulates a set of pre-defined constants
- Like a class, the name of the file containing the enum must be the same as the enum

```
public enum Apple
{
    GALA,
    FUJI,
    GRANNY_SMITH,
    RED_DELICIOUS;
}
```

Enumerations, Example 1

```
public enum TimeUnit
{
    DAY,
    HOUR,
    MINUTE,
    SECOND,
    MILLISECOND,
    NANOSECOND;
}
```

Continued on next slide

Continued from previous slide

Enumerations, Example 1

```
public static void sleep(long duration, TimeUnit timeUnit)
{
    long    timeOut = 0;
    int     nanos   = 0;
    switch ( timeUnit )
    {
    case DAY:
        timeOut = duration * MILLIS_PER_DAY;
        break;
    case HOUR:
        timeOut = duration * MILLIS_PER_HOUR;
        break;
    case MINUTE:
        timeOut = duration * MILLIS_PER_MINUTE;
        break;
        ...
```

Continued on next slide

Continued from previous slide

Enumerations, Example 1

```
...
case MINUTE:
    timeOut = duration * MILLIS_PER_MINUTE;
    break;
case SECOND:
    timeOut = duration * 1000;
    break;
case MILLISECOND:
    timeOut = duration;
    break;
case NANOSECOND:
    timeOut = duration / 1000;
    nanos = (int)(duration % timeOut);
    break;
default:
    throw new IllegalArgumentException();
}
```

Continued on next slide

Continued from previous slide

Enumerations, Example 1

```
        long    wakeUp  =
            System.nanoTime() + timeOut * 1000 + nanos;
        while ( System.nanoTime() < wakeUp )
        {
            Object  waiter  = new Object();
            synchronized ( waiter )
            {
                try
                {
                    waiter.wait( timeOut, nanos );
                }
                catch ( InterruptedException exc )
                {
                }
            }
        }
    }
}
```

Continued on next slide

Continued from previous slide

Enumerations, Example 1

```
public static void main(String[] args)
{
    System.out.println( "10 seconds" );
    sleep( 10, TimeUnit.SECOND );

    System.out.println( "1 minute" );
    sleep( 1, TimeUnit.MINUTE );

    System.out.println( "2,000 milliseconds" );
    sleep( 2000, TimeUnit.MILLISECOND );

    System.out.println( "2,000,500 nanoseconds" );
    sleep( 2000500, TimeUnit.NANOSECOND );

    System.out.println( "done" );
}
```

Continued on next slide

Comparisons

- Except for equality, enumerated constants cannot be compared.

```
private void
compare( TimeUnit unit1, TimeUnit unit2 )
{
    if ( unit1 == unit2 )
        System.out.println( "units are equal" );
    else if ( unit1 < unit2 )
        System.out.println( "unit1 < unit2" );
    else
        System.out.println( "unit1 > unit2" );
}
```

Compiler Error

The ordinal() Method

- The *ordinal* values of enumerated constants may be compared like any two ints

```
public
void compare( TimeUnit unit1, TimeUnit unit2 )
{
    int unit1Val    = unit1.ordinal();
    int unit2Val    = unit2.ordinal();

    if ( unit1 == unit2 )
        System.out.println( "units are equal" );
    else if ( unit1Val < unit2Val )
        System.out.println( "unit1 < unit2" );
    else
        System.out.println( "unit1 is > unit2" );
}
```

The valueOf() Method

- The valueOf() method obtains an enumerated constant given its name

```
public class EnumValueOfDemo
{

    public static void main(String[] args)
    {
        TimeUnit unit = TimeUnit.valueOf( "NANOSECOND" );
        System.out.println( unit );
    }
}
```

The values() Method

- Use the values() method to obtain an array of all constants in an enum

```
public class EnumValuesDemo
{
    public static void main(String[] args)
    {
        TimeUnit[]  allUnits    = TimeUnit.values();
        for ( TimeUnit unit : allUnits )
            System.out.println( unit );
    }
}
```

Output:

```
DAY
HOUR
MINUTE
SECOND
MILLISECOND
NANOSECOND
```

Enums and Methods

- Enums can have methods

```
public enum TimeUnit
{
    DAY,
    HOUR,
    MINUTE,
    SECOND,
    MILLISECOND,
    NANOSECOND;

    public void sleep( long duration )
    {
        Sleeper.sleep( duration, this );
    }
}
```

```
public static
void main(String[] args)
{
    TimeUnit.SECOND.sleep( 5 );
    TimeUnit.SECOND.sleep( 5 );
}
```

Continued on next slide

Enums and Methods, Example

```
public enum Month
{
    JANUARY,
    FEBRUARY,
    MARCH,
    APRIL,
    MAY,
    JUNE,
    JULY,
    AUGUST,
    SEPTEMBER,
    OCTOBER,
    NOVEMBER,
    DECEMBER;
    ...
```

Continued on next slide

Continued from previous slide

Enums and Methods, Example

```
    ...
    public String getAbbreviation()
    {
        String          name    = name(); // this.name
        StringBuilder   bldr    = new StringBuilder();
        bldr.append( name.charAt( 0 ) );

        String temp = name.substring( 1, 3 ).toLowerCase();
        bldr.append( temp );

        return bldr.toString();
    }
}
```

Continued on next slide

Continued from previous slide

Enums and Methods, Example

```
public class EnumMethodDemo2
{
    public static void main(String[] args)
    {
        Month[] months  = Month.values();
        for ( Month month : months )
            System.out.println( month.getAbbreviation() );
    }
}
```

Enums and Constructors

- Enumerations can have constructors
- Constructors must be private
- Required constructor arguments are placed on each enumeration constant

Enums and Constructors, Example 1

```
public enum Apple
{
    GALA( 5 ),
    FUJI( 4 ),
    GRANNY_SMITH( 4 ),
    RED_DELICIOUS( 3 );

    private int rating;
    private Apple( int rating )
    {
        this.rating = rating;
    }
    public String getRating()
    {
        String  str = rating + " stars";
        return str;
    }
}
```

Continued on next slide

Enums and Constructors, Example 1

Continued from previous slide

```
public class EnumConstructorDemo1
{
    public static void main(String[] args)
    {
        Apple[]    apples   = Apple.values();
        for ( Apple apple : apples )
        {
            String  rating   = apple.getRating();
            System.out.println( apple + ": " );
        }
    }
}
```

Output:

```
GALA: 5 stars
FUJI: 4 stars
GRANNY_SMITH: 4 stars
RED_DELICIOUS: 3 stars
```

Enums and Constructors, Example 2

```
public enum Planet
{
    MERCURY (.3303e+24, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27,   7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

    // universal gravitational constant  (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    private final double mass;   // in kilograms
    private final double radius; // in meters
    ...
```

Continued on next slide

Enums and Constructors, Example 2

Continued from previous slide

```
...
Planet(double mass, double radius)
{
    this.mass = mass;
    this.radius = radius;
}
double mass()
{
    return mass;
}

double radius()
{
    return radius;
}
...
```

Continued on next slide

Enums and Constructors, Example 2

Continued from previous slide

```
    ...
    double surfaceGravity()
    {
        return G * mass / (radius * radius);
    }

    double surfaceWeight(double otherMass)
    {
        return otherMass * surfaceGravity();
    }
}
```

Continued on next slide

Enums and Constructors, Example 2

Continued from previous slide

```
private static void printStats( Planet planet )
{
    final double    kgsPerPound     = 0.453592;
    final double    poundsPerKilo   = 2.20462;

    double  radius     = planet.radius();
    double  mass       = planet.mass();
    double  sGravity   = planet.surfaceGravity();
    double  otherMass  =
        (100 * kgsPerPound)/Planet.EARTH.surfaceGravity();
    double  sWeight    = planet.surfaceWeight( otherMass );
    double  pounds     = sWeight * poundsPerKilo;

    StringBuilder   bldr    = new StringBuilder();
    Formatter       form    = new Formatter( bldr );
    ...
```

Enums and Stealth Methods

- Two methods are added by the compiler:
  - values()
  - valueOf()
- For full test coverage, these methods must be tested

Stealth Methods, Testing Example

```
@Test
public void test()
{
    TimeUnit[]  expValues   =
    {
        TimeUnit.DAY,
        TimeUnit.HOUR,
        TimeUnit.MINUTE,
        TimeUnit.SECOND,
        TimeUnit.MILLISECOND,
        TimeUnit.NANOSECOND,
    };

    TimeUnit[]  actValues   =  TimeUnit.values();
    assertEquals( expValues.length, actValues.length );
    assertArrayEquals( expValues, actValues );
}
```