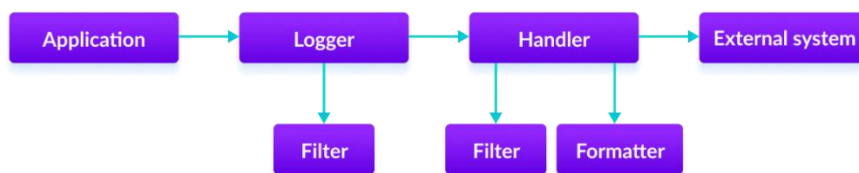


# Logging in Java

- Java allows us to create and capture log messages and files through the process of logging.
- Java has a built-in logging framework in the `java.util.logging` package which is the part of `java.logging` module.
- The Java Logging APIs, contained in the package `java.util.logging`, facilitate software servicing and maintenance at customer sites by producing log reports suitable for analysis by end users, system administrators, field service engineers, and software development teams. The Logging APIs capture information such as security failures, configuration errors, performance bottlenecks, and/or bugs in the application or platform.
- The core package includes support for delivering plain text or XML-formatted log records to memory, output streams, consoles, files, and sockets. In addition, the logging APIs are capable of interacting with logging services that already exist on the host operating system.

## Overview of the control flow

- Applications make logging calls on Logger objects. Logger objects are organized in a hierarchical namespace and child Logger objects may inherit some logging properties from their parents in the namespace.
- These Logger objects allocate LogRecord objects which are passed to Handler objects for publication. Both Logger and Handler objects may use logging Level objects and (optionally) Filter objects to decide if they are interested in a particular LogRecord object. When it is necessary to publish a LogRecord object externally, a Handler object can (optionally) use a Formatter object to localize and format the message before publishing it to an I/O stream.
- Each Logger object keeps track of a set of output Handler objects. By default all Logger objects also send their output to their parent Logger. But Logger objects may also be configured to ignore Handler objects higher up the tree.
- Some Handler objects may direct output to other Handler objects. For example, the MemoryHandler maintains an internal ring buffer of LogRecord objects, and on trigger events, it publishes its LogRecord object through a target Handler. In such cases, any formatting is done by the last Handler in the chain.



## Log Levels

The log levels control the logging details. They determine the extent to which depth the log files are generated. Each level is associated with a numeric value and there are 7 basic log levels and 2 special ones. The logging levels are:

| Level   | Value | Used for                       |
|---------|-------|--------------------------------|
| SEVERE  | 1000  | Indicates some 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     |

|     |                   |                      |
|-----|-------------------|----------------------|
| OFF | Integer.MAX_VALUE | Capturing nothing    |
| ALL | Integer.MIN_VALUE | Capturing Everything |



## The need for Log capture

- There are multiple reasons why we may need to capture the application activity. Recording unusual circumstances or errors that may be happening in the program. Getting the info about what's going in the application.
- The details which can be obtained from the logs can vary. Sometimes, we may want a lot of details regarding the issue, or sometimes some light information only. Like when the application is under development and is undergoing testing phase, we may need to capture a lot of details.

## Java's Log System

- The log system is centrally managed. There is only one application wide log manager which manages both the configuration of the log system and the objects that do the actual logging.
- The Log Manager Class provides a single global instance to interact with log files. It has a static method which is named `getLogManager`.

## Logger Class

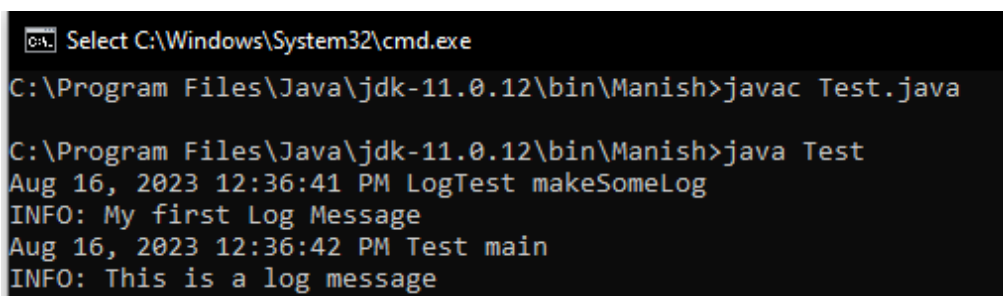
- The logger class provides methods for logging. Since `LogManager` is the one doing actual logging, its instances are accessed using the `LogManager`'s `getLogger` method.
- The global logger instance is accessed through `Logger` class' static field `GLOBAL_LOGGER_NAME`. It is provided as a convenience for making casual use of the Logging package.

```
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.*;
```

```
class LogTest
{
    private final static Logger obj1 = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);

    public void makeSomeLog()
    {
        obj1.log(Level.INFO, "My first Log Message");
    }
}
```

```
public class Test
{
    public static void main(String[] args)
    {
        LogTest obj2 = new LogTest(); obj2.makeSomeLog();
        LogManager lm = LogManager.getLogManager();
        Logger obj3 = lm.getLogger(Logger.GLOBAL_LOGGER_NAME);
        obj3.log(Level.INFO, "This is a log message");
    }
}
```



```
Select C:\Windows\System32\cmd.exe

C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Aug 16, 2023 12:36:41 PM LogTest makeSomeLog
INFO: My first Log Message
Aug 16, 2023 12:36:42 PM Test main
INFO: This is a log message
```

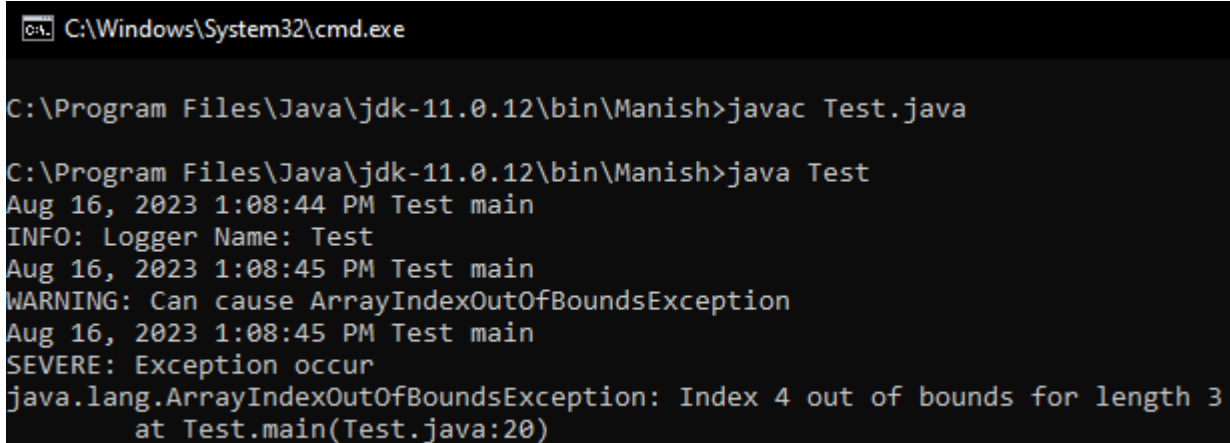
## Logger and Level

```
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

class Test
{
    private static final Logger LOGGER = Logger.getLogger(Test.class.getName());
    public static void main(String[] args) throws SecurityException, IOException
    {
        LOGGER.info("Logger Name: "+LOGGER.getName());
        LOGGER.warning("Can cause ArrayIndexOutOfBoundsException");

        //An array of size 3
        int []a = {1,2,3};
        int index = 4;
        LOGGER.config("index is set to "+index);

        try
        {
            System.out.println(a[index]);
        }
        catch(ArrayIndexOutOfBoundsException ex)
        {
            LOGGER.log(Level.SEVERE, "Exception occur", ex);
        }
    }
}
```



```
C:\Windows\System32\cmd.exe

C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Aug 16, 2023 1:08:44 PM Test main
INFO: Logger Name: Test
Aug 16, 2023 1:08:45 PM Test main
WARNING: Can cause ArrayIndexOutOfBoundsException
Aug 16, 2023 1:08:45 PM Test main
SEVERE: Exception occur
java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 3
    at Test.main(Test.java:20)
```

# Logger and Handler

```
import java.io.IOException;
import java.util.logging.ConsoleHandler;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Test
{
    private static final Logger LOGGER = Logger.getLogger(Test.class.getName());
    public static void main(String[] args)
    {
        Handler consoleHandler = null;
        Handler fileHandler = null;
        try
        {
            //Creating consoleHandler and fileHandler
            consoleHandler = new ConsoleHandler();
            fileHandler = new FileHandler("./Test.log");

            //Assigning handlers to LOGGER object
            LOGGER.addHandler(consoleHandler);
            LOGGER.addHandler(fileHandler);

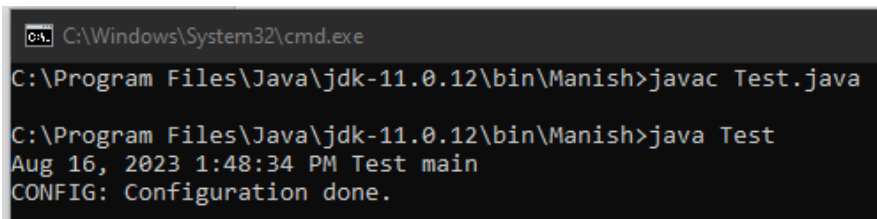
            //Setting levels to handlers and LOGGER
            consoleHandler.setLevel(Level.ALL);
            fileHandler.setLevel(Level.ALL);
            LOGGER.setLevel(Level.ALL);

            LOGGER.config("Configuration done.");

            //Console handler removed
            LOGGER.removeHandler(consoleHandler);

            LOGGER.log(Level.FINE, "Finer logged");
        }
        catch(IOException exception)
        {
            LOGGER.log(Level.SEVERE, "Error occur in FileHandler.", exception);
        }

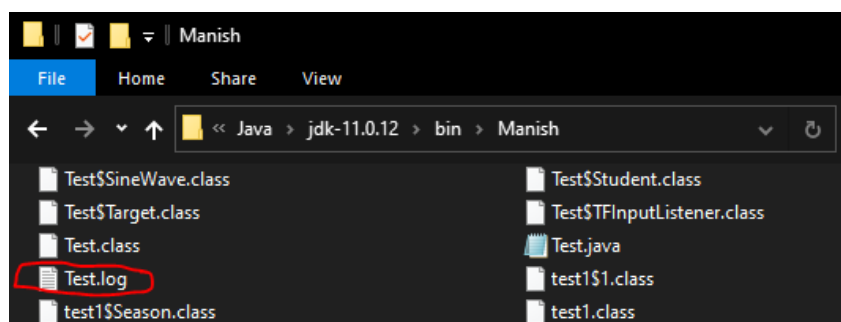
        LOGGER.finer("Finest example on LOGGER handler completed.");
    }
}
```



```
C:\Windows\System32\cmd.exe

C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Aug 16, 2023 1:48:34 PM Test main
CONFIG: Configuration done.
```



## Test.log

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2023-08-16T08:18:34.079156Z</date>
  <millis>1692173914079</millis>
  <nanos>156000</nanos>
  <sequence>0</sequence>
  <logger>Test</logger>
  <level>CONFIG</level>
  <class>Test</class>
  <method>main</method>
  <thread>1</thread>
  <message>Configuration done.</message>
</record>
<record>
  <date>2023-08-16T08:18:34.141103500Z</date>
  <millis>1692173914141</millis>
  <nanos>103500</nanos>
  <sequence>1</sequence>
  <logger>Test</logger>
  <level>FINE</level>
  <class>Test</class>
  <method>main</method>
  <thread>1</thread>
  <message>Finer logged</message>
</record>
<record>
  <date>2023-08-16T08:18:34.142099700Z</date>
  <millis>1692173914142</millis>
  <nanos>99700</nanos>
  <sequence>2</sequence>
  <logger>Test</logger>
  <level>FINER</level>
  <class>Test</class>
  <method>main</method>
  <thread>1</thread>
  <message>Finest example on LOGGER handler completed.</message>
</record>
</log>
```

## Logger and Formatter

```
import java.io.IOException;
import java.util.logging.Formatter;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class Test
{
    private static final Logger LOGGER = Logger.getLogger(Test.class.getName());

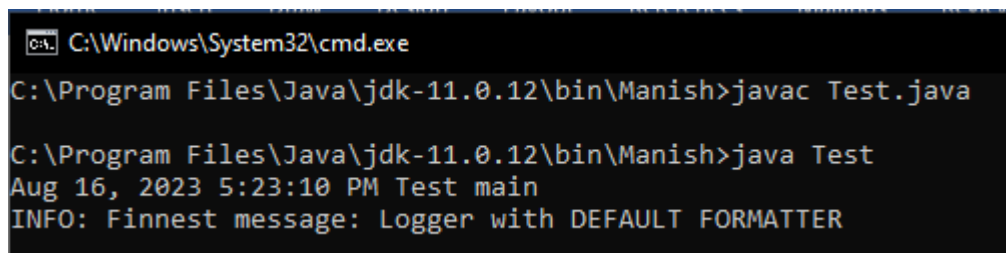
    public static void main(String[] args)
    {
        Handler fileHandler = null;
        Formatter simpleFormatter = null;

        try
        {
            fileHandler = new FileHandler("./Test.formatter.log");
            simpleFormatter = new SimpleFormatter();

            LOGGER.addHandler(fileHandler);
            LOGGER.info("Finnest message: Logger with DEFAULT FORMATTER");

            fileHandler.setFormatter(simpleFormatter);
            fileHandler.setLevel(Level.ALL);

            LOGGER.setLevel(Level.ALL);
            LOGGER.finest("Finnest message: Logger with SIMPLE FORMATTER");
        }
        catch(IOException exception)
        {
            LOGGER.log(Level.SEVERE, "Error occur in FileHandler.", exception);
        }
    }
}
```



```
C:\Windows\System32\cmd.exe
C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java
C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Aug 16, 2023 5:23:10 PM Test main
INFO: Finnest message: Logger with DEFAULT FORMATTER
```

## Test.formatter.log

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2023-08-16T11:53:10.856827100Z</date>
  <millis>1692186790856</millis>
  <nanos>827100</nanos>
  <sequence>0</sequence>
  <logger>Test</logger>
  <level>INFO</level>
  <class>Test</class>
  <method>main</method>
  <thread>1</thread>
  <message>Finnest message: Logger with DEFAULT FORMATTER</message>
</record>
```

Aug 16, 2023 5:23:10 PM Test main

FINEST: Finnest message: Logger with SIMPLE FORMATTER

## Logger and Filter

```
import java.util.logging.Filter;
import java.util.logging.LogRecord;
import java.util.logging.Logger;

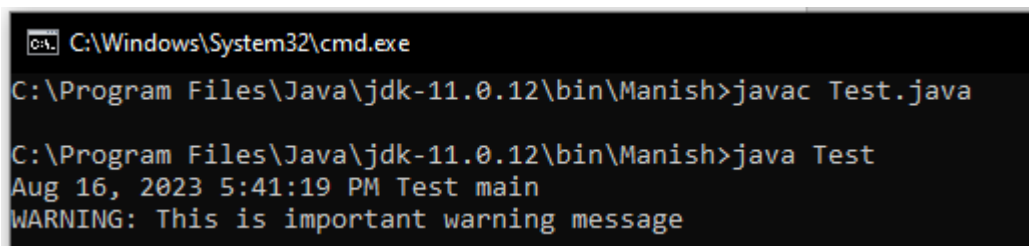
public class Test implements Filter
{
    private static final Logger LOGGER = Logger.getLogger(Test.class.getName());

    public static void main(String[] args)
    {
        LOGGER.setFilter(new Test());
        LOGGER.severe("This is SEVERE message");
        LOGGER.warning("This is important warning message");
    }

    @Override
    public boolean isLoggable(LogRecord record)
    {
        if(record == null)
            return false;

        String message = record.getMessage()==null?"":record.getMessage();

        if(message.contains("important"))
            return true;
        return false;
    }
}
```



```
C:\Windows\System32\cmd.exe
C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Aug 16, 2023 5:41:19 PM Test main
WARNING: This is important warning message
```



# Logger and Configuration

## Java Code file: Test.java

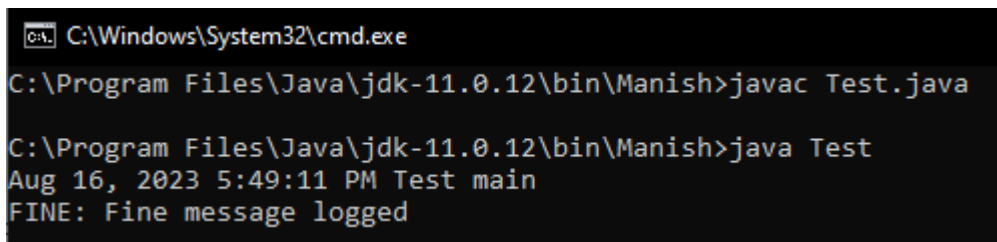
```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.LogManager;
import java.util.logging.Logger;

public class Test
{
    private static final LogManager lm = LogManager.getLogManager();

    private static final Logger L1 = Logger.getLogger("confLogger");

    static
    {
        try
        {
            lm.readConfiguration(new FileInputStream("../Test.properties"));
        }
        catch (IOException exception)
        {
            L1.log(Level.SEVERE, "Error in loading configuration",exception);
        }
    }

    public static void main(String[] args)
    {
        L1.fine("Fine message logged");
    }
}
```



```
C:\Windows\System32\cmd.exe
C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Aug 16, 2023 5:49:11 PM Test main
FINE: Fine message logged
```

## Configuration file: Test.properties

```
handlers=java.util.logging.ConsoleHandler
.level=ALL
java.util.logging.ConsoleHandler.level=ALL
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
confLogger.level=ALL
```