Logger: ist ein Mechanismus, der in der Programmierung verwendet wird, um Laufzeitinformationen zu protokollieren (logging). **Logger** können verwendet werden, um **Informationen über** den **Programmablauf**, **Fehlerzustände** oder **sonstige Ereignisse** während der **Laufzeit** aufzuzeichnen. Diese Informationen können für Debugging, Fehlersuche oder Überwachung von Anwendungen verwendet werden.

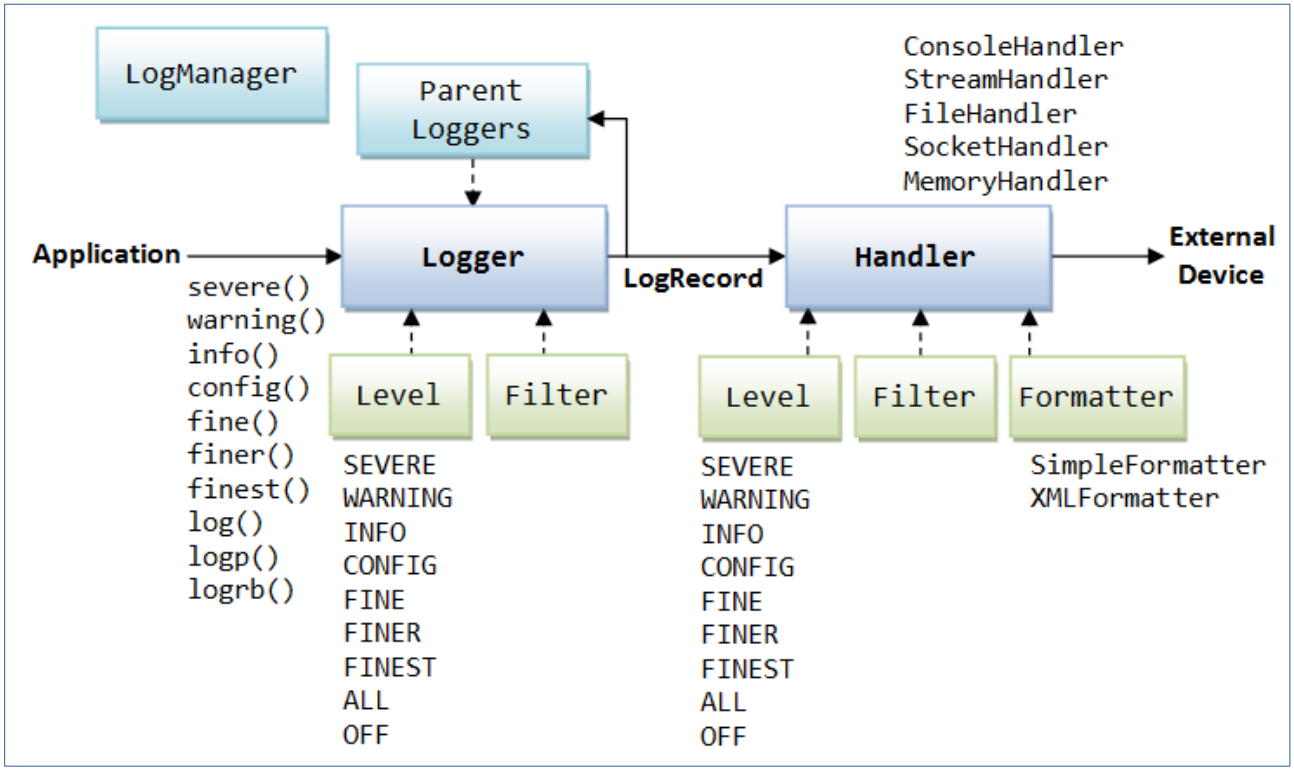
Aufbau des Logging-Systems:

Das Java-Logging-System besteht aus drei Hauptkomponenten:

1. **Logger**: Die Klasse, die verwendet wird, um Nachrichten zu protokollieren. Ein Logger wird in der Regel mit einem Namen erzeugt, der oft dem Paketnamen oder Klassennamen entspricht.
2. **Handler**: Ein Handler ist verantwortlich für die Ausgabe der geloggten Nachrichten an verschiedene Ziele wie die Konsole, Dateien, Datenbanken, etc.
3. **Formatter**: Ein Formatter definiert das Format, in dem die Nachrichten protokolliert werden.

```
private static final Logger logger = Logger.getLogger(LoggingBeispiel.class.getName());
```

Schaubild 1: Ist die Factory Methode



Logger-Level:

Die verschiedenen Level im Java-Logging sind:

1. **SEVERE**: Schwerwiegende Fehler, die die Anwendung möglicherweise zum Absturz bringen.
2. **WARNING**: Potenziell gefährliche Situationen.
3. **INFO**: Allgemeine Informationsnachrichten.
4. **CONFIG**: Konfigurationsmeldungen.
5. **FINE**: Detaillierte Debugging-Informationen.
6. **FINER**: Noch detailliertere Debugging-Informationen.
7. **FINEST**: Extrem detaillierte Debugging-Informationen.
8. **OFF**: Deaktiviert das Logging.
9. **ALL**: Aktiviert alle Logging-Nachrichten.

Beispiel:

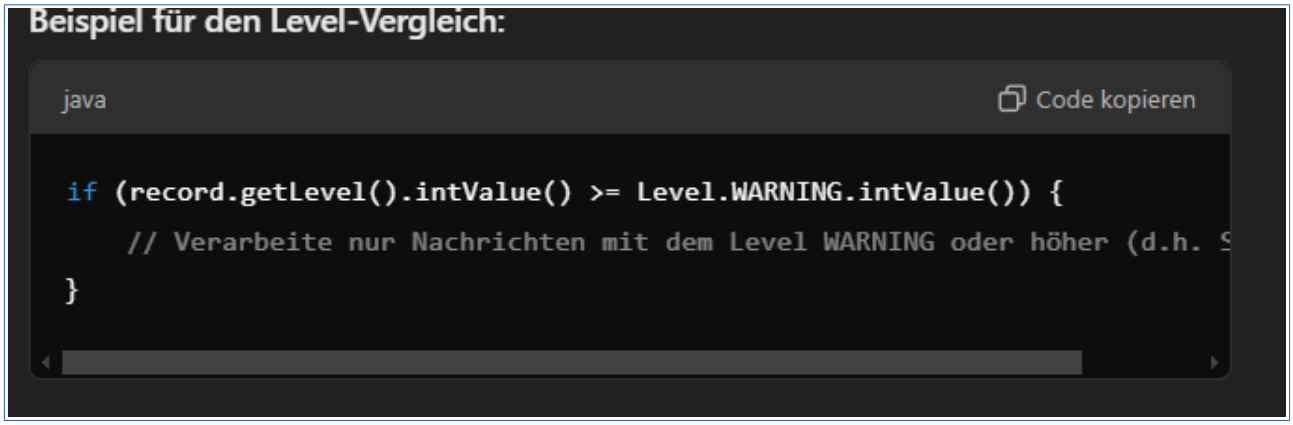
```java
import java.util.logging.Logger;
import java.util.logging.Level;

public class LevelLoggerExample {
    private static final Logger logger = Logger.getLogger(LevelLoggerExample.class.

    public static void main(String[] args) {
        logger.setLevel(Level.ALL);  // Logger-Level setzen

        logger.severe("Schwere Fehlermeldung");
        logger.warning("Warnung");
        logger.info("Info-Nachricht");
        logger.config("Konfigurationsnachricht");
        logger.fine("Feine Debugging-Nachricht");
        logger.finer("Noch detailliertere Debugging-Nachricht");
        logger.finest("Extrem detaillierte Debugging-Nachricht");
    }
}
```

Übersicht der Log-Level und ihre zugewiesenen Zahlenwerte:

- **SEVERE**: 1000
  - Schwerwiegende Fehler, die die Stabilität der Anwendung beeinträchtigen können.
- **WARNING**: 900
  - Warnungen vor potenziell gefährlichen Situationen.
- **INFO**: 800
  - Allgemeine Informationsmeldungen, die den normalen Ablauf der Anwendung betreffen.
- **CONFIG**: 700
  - Meldungen, die Konfigurationsinformationen enthalten.
- **FINE**: 500
  - Detaillierte Debugging-Informationen.
- **FINER**: 400
  - Noch detailliertere Debugging-Informationen.

- **FINEST**: 300
    - Extrem detaillierte Debugging-Informationen.
- **OFF**: Integer.MAX_VALUE (2147483647)
    - Deaktiviert das Logging vollständig.
- **ALL**: Integer.MIN_VALUE (-2147483648)
    - Aktiviert alle Logging-Nachrichten, unabhängig vom Level.

Beispiel für Verwendung:



Handler:

Logger haben standardmäßig einen Handler, der die **Nachrichten** an die **Konsole ausgibt**, es sei denn, andere Handler werden hinzugefügt.

Funktion: Ein Handler ist für das eigentliche Ausgabemedium der Protokollnachrichten verantwortlich. Während der Logger die Entscheidung trifft, ob eine Nachricht protokolliert werden soll, übernimmt der Handler die Aufgabe, die Nachricht an das vorgesehene Ziel zu senden.

- **Typische Handler**:
    - **ConsoleHandler**: Gibt die Protokollnachrichten in die Konsole (System.out oder System.err) aus.
    - **FileHandler**: Schreibt die Protokollnachrichten in eine Datei.
    - **SocketHandler**: Sendet die Protokollnachrichten über das Netzwerk an eine andere Maschine.
    - **MemoryHandler**: Speichert Protokollnachrichten im Speicher und leitet sie bei Bedarf weiter.

Funktionalität:

- **Ein Logger kann mehrere Handler haben**, d. h., eine Nachricht kann gleichzeitig an mehrere Ziele gesendet werden.
- Jeder Handler kann eigene Filter und Formatter haben, um die Nachrichten selektiv weiterzuleiten und zu formatieren.
- Handler können auf bestimmte **Log-Levels** eingestellt werden, sodass sie nur Nachrichten eines bestimmten Levels oder höher verarbeiten.

Beispiel:

```
Logger logger = Logger.getLogger(MyClass.class.getName());
try {
    // Datei-Handler erstellen, der Nachrichten in eine Datei schreibt
    FileHandler fileHandler = new FileHandler("myLogFile.log", true);
    logger.addHandler(fileHandler);
} catch (IOException e) {
    logger.severe("Fehler beim Erstellen des FileHandlers");
}
```

Formatter:

- **Funktion**: Ein Formatter ist für das Formatieren der Protokollnachrichten verantwortlich. Er legt fest, wie eine Nachricht dargestellt wird, bevor sie von einem Handler weitergeleitet wird.
- **Details**:
    - Jede Protokollnachricht besteht aus mehreren Bestandteilen wie dem Zeitpunkt, dem Log-Level, der Quelle der Nachricht (Klasse, Methode), der eigentlichen Nachricht und optionalen Ausnahmen (Exceptions).
    - Der Formatter entscheidet, wie diese Informationen kombiniert und dargestellt werden.
    - Es gibt verschiedene vorgefertigte Formatter, wie:
        - **SimpleFormatter**: Gibt einfache Textausgaben in einer lesbaren Form aus.
        - **XMLFormatter**: Protokolliert Nachrichten im XML-Format, das nützlich ist, wenn Protokolle maschinell verarbeitet werden sollen.

Es ist auch möglich, eigene benutzerdefinierte Formatter zu erstellen, um das Protokollformat an spezielle Anforderungen anzupassen.

```java
import java.util.logging.Formatter;
import java.util.logging.LogRecord;

class CustomFormatter extends Formatter {
    @Override
    public String format(LogRecord record) {
        return record.getLevel() + ": " + record.getMessage() + "\n";
    }
}

Logger logger = Logger.getLogger(MyClass.class.getName());
try {
    FileHandler fileHandler = new FileHandler("myLogFile.log", true);
    fileHandler.setFormatter(new CustomFormatter());
    logger.addHandler(fileHandler);
    logger.info("Das ist eine benutzerdefinierte Formatierung");
} catch (IOException e) {
    logger.severe("Fehler beim Erstellen des FileHandlers");
}
```

Zusammenwirken der Komponenten:

1. **Logger**: Der Logger empfängt eine Protokollierungsanforderung, z. B. `logger.info("Nachricht")`, und entscheidet basierend auf seinem aktuellen Log-Level, ob die Nachricht protokolliert werden soll.
2. **Handler**: Wenn der Logger die Nachricht weiterverarbeitet, wird sie an alle Handler gesendet, die dem Logger zugeordnet sind. Ein Logger kann mehrere Handler haben, sodass eine Nachricht an verschiedene Ziele wie die Konsole und eine Datei gleichzeitig gesendet werden kann.
3. **Formatter**: Jeder Handler verwendet einen Formatter, um die Nachricht in einem bestimmten Format darzustellen. Zum Beispiel könnte ein Formatter eine Protokollnachricht so formatieren, dass sie leicht lesbar oder für maschinelle Verarbeitung geeignet ist.

Beispiel für das Zusammenspiel:

Angenommen, du möchtest, dass deine Anwendung Nachrichten sowohl in der Konsole als auch in einer Datei protokolliert und diese in einem spezifischen Format darstellt. Du würdest Folgendes tun:

1. Erstelle einen `Logger`.
2. Füge einen `ConsoleHandler` hinzu, der die Nachrichten in die Konsole schreibt, und einen `FileHandler`, der die Nachrichten in eine Datei schreibt.
3. Verwende für beide Handler geeignete `Formatter`, um die Nachrichten entsprechend darzustellen (z. B. `SimpleFormatter` für die Konsole und einen benutzerdefinierten Formatter für die Datei).

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

public class LoggingExample {
    private static final Logger logger = Logger.getLogger(LoggingExample.class.getName());

    public static void main(String[] args) {
        try {
            // ConsoleHandler mit SimpleFormatter
            ConsoleHandler consoleHandler = new ConsoleHandler();
            consoleHandler.setFormatter(new SimpleFormatter());

            // FileHandler mit benutzerdefiniertem Formatter
            FileHandler fileHandler = new FileHandler("logfile.log", true);
            fileHandler.setFormatter(new CustomFormatter());

            // Handler hinzufügen
            logger.addHandler(consoleHandler);
            logger.addHandler(fileHandler);

            // Nachrichten loggen
            logger.info("Dies ist eine Info-Nachricht");
            logger.warning("Dies ist eine Warnung");

        } catch (IOException e) {
            logger.severe("Fehler beim Erstellen der Handler: " + e.getMessage());
        }
    }

    // Benutzerdefinierter Formatter
    static class CustomFormatter extends Formatter {
        @Override
        public String format(LogRecord record) {
            return "[" + record.getLevel() + "] " + record.getMessage() + "\n";
        }
    }
}
```