Tinylog explanation in Java

give me the deep and detailed explaination of Tinylog in java so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explaination

Below is a **very deep, structured, and beginner-to-advanced explanation** of **Tinylog** in Java. I'll explain **what it is, why it exists, how it works internally, how to configure it, how it compares to other loggers**, and **when you should (and should not) use it** — all in **simple language**, step by step, so that **no doubt remains**.

🔷 What is Tinylog in Java?

Tinylog is a **lightweight logging framework** for **Java**.

In simple words:

> **Tinylog is a tool that helps your Java program write log messages (info, debug, error, etc.) in a very fast and simple way, with almost zero configuration.**

It is designed for:

- ⚡ **High performance**
- 🧠 **Very simple API**
- 🧩 **Minimal dependencies**
- 🚀 **Easy setup**

🔷 Why Do We Need Logging at All?

Before understanding Tinylog, you must understand **why logging exists**.

Logging helps you:

- See **what your application is doing**
- Debug problems
- Track errors in production
- Understand application flow
- Store historical information

Example without logging:

```
java

System.out.println("User logged in");
```

Problems:
❌ Cannot control log level
❌ Cannot disable logs easily
❌ Cannot redirect to files
❌ Not suitable for production

👉 Logging frameworks (like Tinylog) solve this.

🔷 Why Tinylog Exists (The Core Philosophy)

Most logging frameworks (Log4j2, Logback, etc.) are:

- Powerful ✔️
- Feature-rich ✔️
- But **complex** ❌
- Heavy ❌
- Configuration-heavy ❌

Tinylog was created to solve these pain points.

Tinylog philosophy:

> **"Logging should be simple, fast, and invisible to developers."**

🔷 Key Characteristics of Tinylog

Feature	Explanation
Lightweight	Very small JAR
Simple API	Very few classes
No XML required	Simple properties file
High performance	Minimal overhead
No external dependencies	Clean design
Supports lambda logging	Lazy evaluation
Thread-safe	Safe for multi-threaded apps

🔷 Tinylog Architecture (VERY IMPORTANT)

Tinylog is built on **three main concepts**:

```nginx
Logger → Writer → Output
```

1️⃣ Logger

- Entry point for developers
- Used to log messages

2️⃣ Writer

- Decides **how logs are written**
- Formatting, filtering, async writing

3️⃣ Output (Destination)

- Console
- File
- Rolling file
- Custom output

🔷 Tinylog Versions

Tinylog has **two major generations**:

Version	Status
Tinylog 1.x	Legacy
Tinylog 2.x	Modern, recommended ✅

👉 Always use **Tinylog 2.x**

🔷 Adding Tinylog to a Java Project

Maven Dependency

```xml
<dependency>
    <groupId>org.tinylog</groupId>
    <artifactId>tinylog-api</artifactId>
    <version>2.7.0</version>
</dependency>

<dependency>
    <groupId>org.tinylog</groupId>
    <artifactId>tinylog-impl</artifactId>
    <version>2.7.0</version>
</dependency>
```

Why two dependencies?

- **`tinylog-api`** → logging interface
- **`tinylog-impl`** → actual logging engine

This separation allows flexibility.

🔷 Using Tinylog (Basic Usage)

```
java

import org.tinylog.Logger;

public class Main {
    public static void main(String[] args) {
        Logger.info("Application started");
        Logger.debug("Debug message");
        Logger.error("Something went wrong");
    }
}
```

✔️ No configuration
✔️ No setup
✔️ Works instantly

🔷 Log Levels in Tinylog

Tinylog supports standard log levels:

Level	Meaning
TRACE	Very detailed internal info
DEBUG	Debugging details
INFO	General information
WARN	Something unexpected
ERROR	Serious problem

Example:

```
java

Logger.warn("Disk space is low");
```

🔷 Lazy Logging (One of Tinylog's BEST Features)

Tinylog supports **lazy evaluation using lambdas**.

❌ Traditional logging (problem)

```java
Logger.debug("User data: " + expensiveMethod());
```

Even if DEBUG is disabled:
👉 **`expensiveMethod()`** still executes ❌

✅ Tinylog Lazy Logging

```java
Logger.debug(() -> "User data: " + expensiveMethod());
```

✔️ **`expensiveMethod()`** runs **ONLY if DEBUG is enabled**
✔️ Huge performance benefit

🔷 Exception Logging

```java
try {
    int x = 10 / 0;
} catch (Exception e) {
    Logger.error(e, "Calculation failed");
}
```

This logs:

- Message
- Exception type
- Stack trace

🔷 Configuration in Tinylog

Tinylog uses **tinylog.properties** file.

📁 Location:

```css
src/main/resources/tinylog.properties
```

🔷 Basic Configuration Example

```properties
writer = console
level = info
```

✔️ Logs go to console
✔️ INFO and above shown

🔷 Log to File

```properties
writer = file
writer.file = app.log
level = debug
```

🔷 Log Format (Pattern)

```properties
writer.format = {date} [{thread}] {level} {class}.{method}() - {message}
```

Example output:

```css
2025-12-18 [main] INFO Main.main() - Application started
```

🔷 Multiple Writers (Console + File)

```properties
writers = console, file

writer.console = console
writer.console.level = info

writer.file = file
writer.file.file = app.log
writer.file.level = debug
```

✔️ Console → INFO
✔️ File → DEBUG

🔷 Asynchronous Logging

```properties
writer = async
writer.async = console
```

Benefits:

- Logging doesn't block main thread
- Better performance

🔷 Rolling File Logging

```properties
writer = rolling file
writer.file = app-{count}.log
writer.policies = size:10MB
```

✔️ Creates new file after 10MB
✔️ Prevents huge log files

🔷 Class-Level Log Filtering

```properties
level@com.myapp.service = debug
level@com.myapp.repository = warn
```

✔️ Fine-grained control
✔️ Cleaner logs

🔷 Tinylog vs SLF4J (IMPORTANT)

SLF4J is a **facade**, not a logger.

Feature	Tinylog	SLF4J
Is a logger	✅	❌
Needs backend	❌	✅
Configuration	Very simple	Depends on backend
Performance	Very high	Depends
API size	Very small	Small

👉 Tinylog can **replace both SLF4J + backend** in small/medium apps.

🔷 When Should You Use Tinylog?

✅ Use Tinylog if:

- You want **simplicity**
- You hate XML configs
- You need **high performance**
- You are building:
  - Microservices
  - CLI tools
  - Small to medium apps
  - Libraries

❌ Avoid Tinylog if:

- You need extremely complex logging pipelines
- You require enterprise-level integrations
- You already standardized on Log4j2/Logback

🔷 Internal Working (High-Level)

1. Logger receives message
2. Checks log level
3. Applies filtering
4. Passes to writer
5. Writer formats message
6. Output writes to destination

All steps are **optimized for speed**.

🔷 Common Mistakes (IMPORTANT)

❌ Using string concatenation
❌ Logging sensitive data
❌ Logging at DEBUG in production
❌ No rolling file policy

🔷 Best Practices

✔️ Use lambda logging
✔️ Use proper log levels
✔️ Enable DEBUG only when needed
✔️ Use async writer for production
✔️ Store logs externally

🔷 Final Summary (One-Line)

> **Tinylog is a fast, lightweight, zero-complexity logging framework that makes logging in Java simple, clean, and highly efficient.**

If you want, next I can:

- Compare **Tinylog vs Log4j2 vs Logback**
- Explain **Tinylog configuration line by line**
- Show **real-world production setup**
- Explain **Tinylog internals with diagrams**

Just tell me 👍