Corso di Laurea in Ingegneria Informatica e Automatica
A. A. 2018/2019

# Progettazione del Software

Giuseppe De Giacomo, Paolo Liberatore,
Massimo Mecella

# Logging

## Approccio tradizionale alle stampe per il debugging

- L'approccio tradizionale alle stampe
  - inserire nel codice istruzioni di stampa; es: System.out.println()
  - per visualizzare sullo schermo durante l'esecuzione il valore delle variabili, dei parametri e l'evoluzione del flusso di controllo
  - le istruzioni vengono commentate dopo aver trovato l'errore ed eventualmente ripristinate per altri errori

## Ma …

- Il vantaggio di questo approccio
  - è un approccio semplice e immediato
- Gli svantaggi di questo approccio sono però numerosi:
  - è necessario modificare il codice per commentare – decommentare le stampe di debug (introduce potenzialmente errori)
  - le stampe vengono prodotte tutte sullo standard output e possono facilmente produrre lo scorrimento veloce dello schermo pregiudicandone la leggibilità, d'altro canto produrre le stampe in un flusso diverso (es: file) complicherebbe decisamente le istruzioni di stampa



## Logging

- Sistema di logging
  - sistema che permette di includere nel codice stampe di debug, che descrivono il funzionamento dell'applicazione e di produrle solo su richiesta dell'utente (durante le sessioni di correzione)
  - oltre che in formati e su supporti diversi
  - escludendole nelle versioni in produzione

## Logging

- Sistema di logging
  - libreria di classi per la registrazione di informazioni sull'esecuzione del codice
- Caratteristiche fondamentali
  - consente registrazioni a diversi livelli di "gravità"
  - consente di registrare su dispositivi diversi
  - consente di registrare in formati diversi
- Il sistema di logging standard di Java è il package `java.util.logging`
- Un sistema di logging molto diffuso è Log4j (`org.apache.log4j`)



## Concetti principali

- **Messaggio di logging ("log record")**
  - stringa registrata durante l'esecuzione di un metodo
- **Logger**
  - componente responsabile della creazione dei messaggi di logging
  - è possibile utilizzare più di un Logger all'interno della stessa applicazione; es: un logger per package o per classe
  - ogni logger ha un nome
  - i logger sono organizzati in una gerarchia, con un logger principale, detto "root" logger

## Concetti principali

- **Handler (o Appender)**
  - componente responsabile di registrare un messaggio di logging prodotto da un logger su un flusso o dispositivo con un formato specificato
  - esempio: console, file, dbms, socket, ecc.
  - a ciascuna categoria di logger può essere associato uno o più handler
- **Livello di logging**
  - descrive la gravità e l'urgenza di un messaggio
  - Alcuni livelli tipici, in ordine decrescente
    - SEVERE: problema molto grave
    - WARNING: avvertimento potenz. grave
    - INFO: messaggio informativo sul funzionam.
    - FINE: messaggio di "tracing"
    - FINEST: messaggio di "tracing" dettagliato

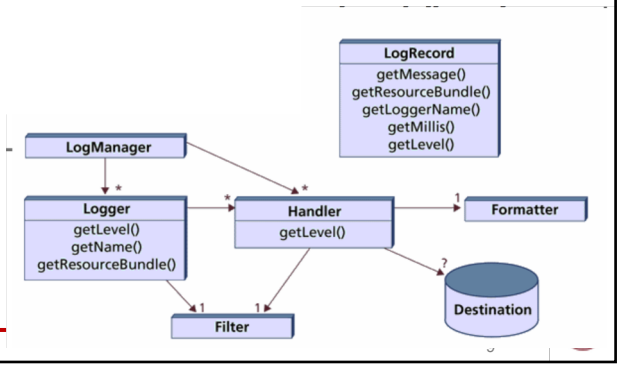


## Sistema di logging

- Una caratteristica fondamentale:
  - è possibile associare un livello a ciascun logger, in modo da abilitare o disabilitare i messaggi prodotti dal logger
  - il livello standard è tipicamente INFO, ma è configurabile attraverso un file di configurazione, che consente di controllare quali messaggi vengono effettivamente prodotti, eventualmente disabilitandoli completamente
  - le istruzioni di creazione dei messaggi vengono introdotte e restano nel codice (non c'è bisogno di commentarle); a seconda del livello abilitato, le registrazioni vengono effettuate o meno senza dover modificare il codice

# Logging (di default) in Java

- Il sistema di logging standard di Java è il package `java.util.logging`
- la classe `LogManager`
  - metodo `getLogManager` si ottiene il manager
  - con `addLogger` si aggiungono loggers per poi reperirli mediante il metodo getLogger




# Configurazione

- I parametri di configurazione di default usati dal LogManager vengono letti dal file di properties che si trova nella directory lib del JRE (logging.properties). Questi settings possono essere cambiati a runtime utilizzando dei metodi appropriati o modificando il file di properties; la classe LogManager mette a disposizione dei metodi che consentono di rileggere il file di configurazione.
- La configurazione di default prevede due Handler globali, uno su file nella home directory utente e un altro su schermo (Console). Il livello di default di questi logger è INFO.
- Per modificare queste impostazioni a runtime si può intervenire in vari modi:
  - Creando nuovi Handler come ad esempio FileHandlers, MemoryHandlers, e PrintHandlers
  - Creando nuovi Logger da associare a specifici Handler
  - Utilizzando il metodo setLevel di Logger o LogManager per cambiare il livello dei Loggers

```java
package it.uniroma1.diag;

import java.util.logging.*;

public class Loggata {

    // ottiene un logger
    private static Logger logger = Logger.getLogger("it.uniroma1.diag.Loggata");

    private static void metodoEsempio() {
        System.out.println("... sto eseguendo un'operazione complessa ...");
    }

    public static void main(String args[]) {

        try{
            Loggata.metodoEsempio();
        } catch (Exception ex) {
            // Log the exception
            logger.log(Level.WARNING, "problemi nel metodo", ex);
        }

        // Log un messaggio di tracing INFO
        // provare ad usare Level.FINE e vedere cosa succede
        logger.log(Level.INFO, "fatto");
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> Loggata [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home/bin/java (Dec 10, 2
... sto eseguendo un'operazione complessa ...
Dec 10, 2016 11:06:29 AM it.uniroma1.diag.Loggata main
INFO: fatto
```

```java
package it.uniroma1.diag;

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

public class Loggata_2 {

    // ottiene un logger
    private static Logger logger = Logger.getLogger("it.uniroma1.diag.Loggata_2");

    private static void metodoEsempio() {
        System.out.println("... sto eseguendo un'operazione complessa ...");
    }

    public static void main(String args[]) throws SecurityException, IOException {

        // richiede che ogni dettaglio venga loggato (Level.ALL)
        logger.setLevel(Level.ALL);

        // manda il log su file
        FileHandler fh = new FileHandler("mylog.txt");
        logger.addHandler(fh);

        logger.log(Level.FINE, "tutto bene");

        try{
            Loggata_2.metodoEsempio();
        } catch (Exception ex) {
            // Log the exception
            logger.log(Level.WARNING, "problemi nel metodo", ex);
        }

        logger.log(Level.FINE, "fatto");
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> Loggata_2 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_65.jdk/Contents/Home/bin/java (D
... sto eseguendo un'operazione complessa ...
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2016-12-10T11:24:44</date>
  <millis>1481365484378</millis>
  <sequence>0</sequence>
  <logger>it.uniroma1.diag.Loggata_2</logger>
  <level>FINE</level>
  <class>it.uniroma1.diag.Loggata_2</class>
  <method>main</method>
  <thread>1</thread>
  <message>tutto bene</message>
</record>
<record>
  <date>2016-12-10T11:24:44</date>
  <millis>1481365484404</millis>
  <sequence>1</sequence>
  <logger>it.uniroma1.diag.Loggata_2</logger>
  <level>FINE</level>
  <class>it.uniroma1.diag.Loggata_2</class>
  <method>main</method>
  <thread>1</thread>
  <message>fatto</message>
</record>
</log>
```