FICHE · JAVA · LOGGERS 1/2

Journaliser avec java.util.logging

Remplacer System.out.println par le logger fourni avec le JDK.

Frédéric Guyomarch · novembre 2026

Pourquoi ne pas utiliser System.out ?

```
void payer(Commande c) {
  System.out.println("debut payer");
  System.out.println("c = " + c);
  ...
  System.out.println("on passe ici");
}

// En production :
//  - ces lignes s'affichent toujours,
//  - on ne sait ni quand, ni où, ni qui,
//  - pour les retirer, il faut recompiler.
```

Pas de niveau

Impossible de distinguer une trace de mise au point d'une vraie erreur.

Pas de contexte

Ni date, ni thread, ni classe : on ne sait plus d'où vient la ligne.

Pas de réglage

Tout ou rien, toujours vers la console. Un logger se configure sans recompiler.

Le trajet d'un message de log




```
2026-11-13 11:10:20 INFO    fr.iut.panier.Panier - Commande 42 validée
```

Logger

Un objet par classe, nommé d'après elle. Il filtre selon le niveau.

Handler

La destination : ConsoleHandler, FileHandler, SocketHandler...

Formatter

La mise en forme : SimpleFormatter (texte) ou XMLFormatter.

Les niveaux de gravité

Niveau	Valeur	Quand l'utiliser
SEVERE	1000	Une opération a échoué : il faut agir
WARNING	900	Situation anormale mais gérée
INFO	800	Événement métier important : démarrage, commande validée
CONFIG	700	Paramètres de configuration au démarrage
FINE	500	Détails utiles au développeur
FINER, FINEST	400, 300	Suivi très fin : entrée et sortie de méthodes

Règle : un logger réglé sur INFO affiche INFO et tout ce qui est plus grave, et ignore le reste. Level.ALL et Level.OFF ouvrent ou ferment tout.

Déclarer et utiliser un logger

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

public class Panier {
  private static final Logger LOG =
      Logger.getLogger(Panier.class.getName());

  void ajouter(Article a) {
    LOG.fine(() -> "Ajout de " + a);
    if (a.prix() < 0)
      LOG.warning("Prix négatif : " + a.id());
  }
}
```

Un logger par classe

static final, nommé d'après la classe : fr.iut.panier.Panier.

Une hiérarchie

Les noms forment un arbre : fr.iut.panier hérite des réglages de fr.iut.

Le message à la demande

Avec une lambda, la chaîne n'est construite que si FINE est actif.

Configurer avec logging.properties

```
# logging.properties
handlers = java.util.logging.ConsoleHandler,\
           java.util.logging.FileHandler
.level = INFO
fr.iut.panier.level = FINE
java.util.logging.ConsoleHandler.level = FINE

java.util.logging.FileHandler.pattern = app%u.log
java.util.logging.FileHandler.formatter =\
           java.util.logging.SimpleFormatter
java.util.logging.SimpleFormatter.format =\
           %1$tF %1$tT %4$-7s %3$s - %5$s%6$s%n

$ java -Djava.util.logging.config.file=\
       logging.properties fr.iut.panier.App
```

Deux filtres

Le logger et le handler ont chacun un niveau. Le ConsoleHandler s'arrête à INFO par défaut.

Handler

La destination : ConsoleHandler, FileHandler (en XML si on ne précise rien).

Sans recompiler

Le fichier est lu au lancement : on change les niveaux sans toucher au code.

Journaliser une exception

```
// À éviter
catch (NumberFormatException e) {
  e.printStackTrace();                // sur System.err
  LOG.severe("Erreur : " + e);        // trace perdue
}
```

```
// Recommandé : l'exception en argument
catch (NumberFormatException e) {
  LOG.log(Level.SEVERE, "Quantité illisible : " + saisie, e);
}

// Ou, message construit à la demande
LOG.log(Level.WARNING, e, () -> "Saisie : " + saisie);
```

Toute la pile

Passer l'objet exception, pas son toString : la trace complète est écrite.

Du contexte

Quelle valeur, quelle commande, quel utilisateur : de quoi reproduire le problème.

Une seule fois

Journaliser ou relancer, pas les deux, sinon la même erreur apparaît plusieurs fois.

Les bonnes pratiques avec JUL

```
// À éviter
LOG.fine("Panier : " + panier);
LOG.info("Connexion de " + login + " / " + motDePasse);
```

```
// Recommandé
LOG.fine(() -> "Panier : " + panier);
LOG.log(Level.INFO, "Connexion réussie pour {0}", login);
if (LOG.isLoggable(Level.FINE)) {
  // plusieurs lignes de détail…
}
```

Le bon niveau

INFO pour les événements métier, FINE pour les détails, SEVERE seulement si quelqu'un doit agir.

Pas de calcul inutile

Une lambda ou des paramètres {0} ; isLoggable pour un bloc entier.

Jamais de secrets

Ni mot de passe, ni jeton, ni donnée personnelle inutile : les logs sont lus, copiés, archivés.

À RETENIR

java.util.logging en bref

1. Un logger static final par classe, plutôt que System.out.println.
2. Logger, Handler, Formatter : qui filtre, vers où, sous quelle forme.
3. Deux seuils : celui du logger et celui du handler.
4. logging.properties règle tout au lancement, sans recompiler.
5. L'exception en argument, un message construit à la demande, jamais de secrets.

Pour toute question, un mail !

frederic.guyomarch@univ-lille.fr