Prérequis — séquence d'environnement

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-31

Table des matières

1 Systèmes d'exploitation cibles 1
2 Installer le JDK 2
2.1 La voie classique : Temurin, `JAVA_HOME` et `PATH` 2
2.2 SDKMan : gérer plusieurs versions 3
3 Espace de travail et projet type 3
3.1 La structure 3
3.2 La classe d'exemple 3
3.3 Compiler et exécuter 5
4 Empaqueter : l'archive jar 5
4.1 Créer et utiliser une archive 5
4.2 Rendre l'archive exécutable directement 6
5 Les outils du cours : Git et bash 7
6 Installer Git sur votre machine personnelle 7
7 Et pour les commentaires de documentation ? 7
8 Liste de contrôle 7

Prérequis auto-guidé, hors chemin noté. Ce document prépare votre environnement Java sur machine personnelle ; le cours lui-même se déroule dans l'image du cours, qui ne demande aucune installation.

Les leçons du cours renvoient ici pour tout ce qui concerne la machine. Parcourez ce document dans l'ordre ; la liste de contrôle finale vérifie que vous pouvez enchaîner sur les séances.

1 Systèmes d'exploitation cibles

Les instructions qui suivent s'adressent aux systèmes de la famille Unix (Linux, macOS) et, sous Windows, au sous-système WSL (Windows Subsystem for Linux), dont Microsoft documente l'installation. Le shell de référence est **bash** : il est présent

dans l'image du cours ; sur votre machine, les sections qui suivent vous mettent dans un équivalent.

2 Installer le JDK

2.1 La voie classique : Temurin, `JAVA_HOME` et `PATH`

Pour développer en Java, un JDK (kit de développement) suffit. Le cours recommande la distribution Temurin, retenue pour ses conditions de licence — c'est aussi celle de l'image du cours, qui fournit le JDK 25.0.4 à la date d'octobre 2026. Téléchargez-le, décompressez-le dans le répertoire de votre choix, puis déclarez-le à votre shell :

```
export JAVA_HOME=/usr/local/jdk-25                # le chemin dépend de votre installation
export PATH="$JAVA_HOME/bin:$PATH"
```

Ajoutez ces deux lignes à votre `.bashrc` (ou `.zshrc`) pour qu'elles s'appliquent à chaque ouverture de session. La consigne de version du cours est le **JDK 25 (LTS)** à la date de 2026-10 : une installation plus ancienne ne vous empêchera pas de suivre les séances, mais les observations de la leçon de mécanique (versions de bytecode, propriétés de la machine) sont calées sur l'image du cours.

Vérifiez d'abord que les variables sont définies (dans une nouvelle session, ou après `source ~/.bashrc`) :

```
echo "JAVA_HOME = $JAVA_HOME"
echo "PATH = $PATH"
```

puis que les commandes attendues répondent bien :

```
java --version
javac --version
```

La sortie attendue annonce la version installée et sa distribution ; sur l'image du cours, `openjdk 25.0.4`, édité par Eclipse Adoptium (Temurin).

> **Consigne**
>
> Installez sur votre machine un JDK conforme à la baseline du cours (25 à la date de 2026-10), par la voie classique ou par le gestionnaire de versions ci-dessous.

2.2 SDKMan : gérer plusieurs versions

sdkman.io installe et versionne un JDK (et d'autres outils) sans que vous touchiez aux chemins système à la main :

```
curl -s "https://get.sdkman.io" | bash
sdk install java             # la version courante de SDKMan, pas nécessairement
la baseline du cours
sdk list java | grep installed  # les versions présentes sur votre machine
```

`sdk list java` énumère les distributions et versions disponibles ; c'est dans cette liste que vous puisez l'identifiant d'une version précise, plutôt que de le recopier d'un document qui vieillit. Pour la baseline du cours, choisissez-y un identifiant `25.*` et installez-le par `sdk install java <identifiant>`.

3 Espace de travail et projet type

3.1 La structure

Rassemblez vos projets sous un unique répertoire de travail — noté `$HOME/work` dans la suite ; adaptez-le à vos habitudes. Chaque projet y occupe un répertoire propre, organisé en cinq sous-répertoires standards : `src` pour les sources, `bin` pour le byte-code compilé, `doc` pour la documentation, `dist` pour les archives livrables et `lib` pour les dépendances externes. Le premier projet du cours, `java/TP1/premierprogramme`, se met en place ainsi :

```
# (re)met à zéro le projet, sans toucher au reste de l'espace de travail
rm -rf "$HOME/work/java/TP1/premierprogramme"
mkdir -p "$HOME/work/java/TP1/premierprogramme"/{src,bin,doc,dist,lib}
```

3.2 La classe d'exemple

Le fichier source de référence tient en une page ; vous l'écrirez directement dans votre arborescence, ligne `package` déjà en place — sans copie depuis un dépôt. Un paquetage est un groupement nommé de classes dont le nom se traduit en arborescence de répertoires sous `src` (JLS, chapitre 7 « Packages ») ; d'où la coïncidence obligatoire entre le chemin du fichier et sa ligne `package`.

Les commandes ci-dessous utilisent `$LOGNAME`, la variable d'environnement qui porte votre login de session ; vérifiez que `echo "$LOGNAME"` affiche bien votre login, sinon affectez-le (`LOGNAME=<votre_login>`). Si votre institution n'est pas `univtln`, transposez aussi ce préfixe ; le gabarit est `fr.<domaine>.<login>.tp.tp1`.

```
PROJET="$HOME/work/java/TP1/premierprogramme"
PAQUETAGE="fr/univtln/$LOGNAME/tp/tp1"        # version chemin du paquetage
mkdir -p "$PROJET/src/$PAQUETAGE"
cat > "$PROJET/src/$PAQUETAGE/HelloWorld.java" <<EOF
package fr.univtln.${LOGNAME}.tp.tp1;

/**
 * A very simple class to say Hello
 * @author Emmanuel Bruno
 *
 */
public class HelloWorld {

    /**
     * A dummy constructor
     */
    public HelloWorld() { }

    /**
     * The entry point of the class
     * @param args The first parameter is a firstname
     */
    public static void main(String[] args) {
        if (args.length==1)
            System.out.println("Bonjour "+args[0]);
        else
            System.out.println("Usage : PremierProgramme [prenom]");
    }

}
EOF
tree "$PROJET"                              # à défaut de tree : ls -R "$PROJET"
```

La dernière commande affiche l'arborescence obtenue : les cinq répertoires standards, et `HelloWorld.java` au fond du chemin de votre paquetage. Si votre login contient des caractères interdits dans un identifiant Java (point, tiret), remplacez-les par des tirets bas avant d'affecter `LOGNAME`. Note de rejouabilité : `$PROJET` et `$PAQUETAGE` vivent le temps d'un shell ; si vous reprenez plus tard, recommencez par leurs affectations (cellule ci-dessus).

Consigne

Reproducez cette structure manuellement — sans les commandes ci-dessus, dans l'éditeur ou le terminal de votre choix — et vérifiez que l'instruction `package` du fichier écrit porte bien votre login.

3.3 Compiler et exécuter

Depuis le répertoire du projet, la compilation indique où chercher les sources (`-sourcepath`), où déposer le bytecode (`-d`) et quel fichier compiler :

```
cd "$HOME/work/java/TP1/premierprogramme"
javac -sourcepath src -d bin "src/$PAQUETAGE/HelloWorld.java"
```

L'exécution charge la classe depuis `bin` via le classpath et appelle sa méthode `main` :

```
java -classpath bin "fr.univtln.${LOGNAME}.tp.tp1.HelloWorld" Pierre
```

Avec un prénom en argument, la sortie attendue est `Bonjour Pierre` ; sans argument, le programme affiche sa ligne d'usage. Pourquoi ces trois objets (source, bytecode, JVM) obéissent à ces commandes est précisément le sujet de la leçon de mécanique ; installez ici le geste.

Consigne

Compilez et exécutez la classe `HelloWorld`, avec et sans argument.

4 Empaqueter : l'archive jar

4.1 Créer et utiliser une archive

`jar` regroupe un ensemble de classes compilées dans un fichier unique, une archive Java. Depuis le répertoire `bin`, on emballe le paquetage entier dans `dist` :

```
cd "$HOME/work/java/TP1/premierprogramme/bin"
jar cvf ../dist/TP01EX01.jar fr
```

Une archive se consomme comme un répertoire de classes : elle s'ajoute au classpath, la classe à exécuter restant désignée par son nom complet :

```
cd "$HOME/work/java/TP1/premierprogramme"
java -cp dist/TP01EX01.jar "fr.univtln.${LOGNAME}.tp.tp1.HelloWorld" Max
```

Consigne

Générez l'archive de votre projet et exécutez-la.

4.2 Rendre l'archive exécutable directement

Toute archive porte des métadonnées dans l'entrée `META-INF/MANIFEST.MF` ; consultez-la sans dépaqueter (outil `unzip` ; installez-le si votre système ne le fournit pas) :

```
cd "$HOME/work/java/TP1/premierprogramme"
unzip -q -c dist/TP01EX01.jar META-INF/MANIFEST.MF
```

Pour que `java -jar` sache quelle classe lancer, le manifeste doit la déclarer sur une ligne `Main-Class:`. Partez du manifeste déjà présent dans l'archive, retirez ses lignes vides et ajoutez la déclaration :

```
cd "$HOME/work/java/TP1/premierprogramme"
unzip -q -c dist/TP01EX01.jar META-INF/MANIFEST.MF \
  | sed '/^[[:space:]]*$/d' > monManifest.txt
echo "Main-Class: fr.univtln.${LOGNAME}.tp.tp1.HelloWorld" >> monManifest.txt
cat monManifest.txt
```

La sortie attendue montre les lignes d'origine du manifeste suivies de la ligne `Main-Class:` que vous venez d'ajouter — terminez toujours le fichier par un saut de ligne, ce que `echo` fait. Reconstruisez alors l'archive avec ce manifeste (le `m` de `cvfm`) :

```
cd "$HOME/work/java/TP1/premierprogramme/bin"
jar cvfm ../dist/TP01EX01.jar ../monManifest.txt fr
cd ..
java -jar dist/TP01EX01.jar Pierre
```

La dernière commande ne demande plus ni classpath ni nom de classe : le manifeste les fournit. Oracle détaille la fabrication des manifestes dans sa documentation sur les fichiers manifest.

Consigne

Rendez l'archive de votre projet exécutable et lancez-la par `java -jar`.

5 Les outils du cours : Git et bash

Dans l'image du cours, Git et bash sont déjà présents : pour travailler dans le cours, vous n'avez rien à installer. Les séquences consacrées à Git partent de ce constat et travaillent le geste, pas l'installation.

6 Installer Git sur votre machine personnelle

Si vous voulez sortir de l'image et versionner votre travail localement, l'installation est brève :

- **Windows** : Git for Windows fournit Git avec son terminal dédié (Git Bash) ; si vous avez suivi l'option WSL plus haut, Git s'y installe comme sous Linux.
- **macOS** : `brew install git` si vous utilisez Homebrew ; macOS fournit d'office une version système, suffisante pour démarrer.
- **Linux** : `sudo apt install git` sur les distributions de la famille Debian/Ubuntu, ou l'équivalent dans votre gestionnaire de paquets.
- **IDE** : VS Code, IntelliJ et leurs similaires intègrent Git ; aucune installation supplémentaire n'est requise pour commencer.

Quelle que soit la voie, vérifiez par `git --version`. Pour apprendre Git au-delà du geste d'installation, la référence du cours est le livre Pro Git, disponible gratuitement en ligne.

7 Et pour les commentaires de documentation ?

La compétence « écrire et produire une javadoc » est enseignée dans la section « Lire la javadoc (compétence transversale du cours) » de la leçon TP — La mécanique Java, à la main (`02-notebook-java-install/02_java_P_mecanique.qmd`) ; elle n'est pas traitée ici.

8 Liste de contrôle

Vous avez terminé ce prérequis lorsque, sur votre machine :

1. `java --version` et `javac --version` répondent avec un JDK de la baseline du cours (25 à la date de 2026-10), ou un JDK plus récent/ancien, conscient que les observations chiffrées seront décalées ;
2. `git --version` répond ;
3. le projet `premierprogramme` est compilé et exécuté depuis votre espace de travail, avec le paquetage qui porte votre login ;
4. l'archive du projet s'exécute par `java -jar`, sans classpath ni nom de classe.

Ces quatre points acquittés, vous pouvez aborder les séances du cours — en commençant par la leçon de mécanique, qui suppose exactement cet environnement.