TP — Tests boîte blanche (JUnit, JaCoCo & EclEmma)

Matière : Management de la Qualité

Prof : D. Essabbar

Java 17 | Maven 3.9+ | Eclipse

Objectif. Mettre en pratique la **méthode de test boîte blanche** (instructions, branches, bornes) sur deux méthodes Java : `StringTools.safeSlice` et `DateUtils.isLeapYear`. Les tests seront écrits avec **JUnit 5**, et la couverture mesurée via **JaCoCo** (Maven) et **EclEmma** (Eclipse).

Contexte et ressources

Le projet Maven `tp-junit-jacoco` contient les méthodes suivantes :

1. StringTools.safeSlice

```java
public String safeSlice(String s, int start, int end) {
    if (s == null) throw new IllegalArgumentException("null s");
    int n = s.length();
    if (start < 0) start = 0;
    if (end > n) end = n;
    if (start >= end) return "";
    return s.substring(start, end);
}
```

2. DateUtils.isLeapYear

```java
public static boolean isLeapYear(int year) {
    if (year % 4 != 0) return false;
    if (year % 100 != 0) return true;
    return year % 400 == 0;
}
```

Partie A — Graphe de contrôle et chemins

1. Pour la méthode `StringTools.safeSlice` :

a) Dessiner le **graphe de contrôle** (CFG) en identifiant les blocs de base et les arêtes (vraies/-faux).

b) Donner l'**expression des chemins de contrôle** sous forme de séquences de nœuds ou d'arêtes (exemple : $N_0 \to N_1(T) \to N_2 \to \dots$).

c) Calculer la **complexité cyclomatique** $V(G)$.

d) En déduire le **nombre minimal de chemins indépendants** à tester.

2. Pour la méthode `DateUtils.isLeapYear` :

a) Dessiner le **graphe de contrôle** en identifiant les décisions : `year % 4`, `year % 100`, `year % 400`.

b) Donner les **chemins de contrôle possibles** correspondant aux quatre cas :

- non divisible par 4 ;
- divisible par 4 mais pas par 100 ;
- divisible par 100 mais pas par 400 ;
- divisible par 400.

c) Calculer $V(G)$ et le nombre de chemins indépendants à tester.

Partie B — Implémentation, installation et couverture

1. Installation d'EclEmma (plugin Eclipse)

1. Ouvrir Eclipse.
2. Menu `Help → Eclipse Marketplace`.
3. Dans la barre de recherche, taper **EclEmma**.
4. Cliquer sur `Install → Confirm → Restart Eclipse`.
5. Une fois installé, un menu `Coverage As` apparaîtra dans les options d'exécution.

Utilisation :

— Ouvrir la classe de test ou le projet complet.
— Clic droit → `Coverage As → JUnit Test`.
— Les lignes de code s'affichent colorées :
  — Vert : lignes exécutées.
  — Rouge : lignes non exécutées.
  — Jaune : branches partiellement couvertes.
— Ouvrir la vue `Coverage` pour visualiser les pourcentages de couverture par classe et méthode.

2. Installation et utilisation de JaCoCo (Maven)

1. JaCoCo est déjà configuré dans le fichier `pom.xml`.
2. Pour exécuter les tests et générer le rapport :

   ```
   mvn clean verify
   ```

3. Le rapport est généré automatiquement dans :

   ```
   target/site/jacoco/index.html
   ```

4. Ouvrir ce fichier dans un navigateur pour consulter :
   — la couverture par classes, méthodes, lignes et branches ;
   — la liste des instructions non exécutées ;
   — la navigation directe dans le code source coloré.

3. Comparaison des résultats

Comparer les résultats des deux outils :

— **EclEmma** : couverture en temps réel dans Eclipse.
— **JaCoCo** : rapport complet et sauvegardé via Maven.

Identifier les éventuelles différences de pourcentage et expliquer les causes possibles (fichiers exclus, différences de contexte d'exécution...).

Livrables attendus

— Schémas des graphes de contrôle (CFG) pour les deux méthodes.
— Expressions des chemins de contrôle et calculs de $V(G)$.
— Tableaux chemins → cas de test.
— Projet complet contenant les tests JUnit.
— Captures d'écran :
  — Vue `Coverage` dans Eclipse (EclEmma).
  — Rapport HTML JaCoCo dans `target/site/jacoco`.