Info – Bloc 1 – UE09 : Programmation Orientée Objet

Laboratoire 1

Durée prévue : 4 heures

Acquis d'apprentissage visés

À la fin du laboratoire, les étudiants seront capables de :

- Créer des objets avec l'opérateur `new`
- Appeler des méthodes d'objet
- Déclarer des fonctions manipulant des objets
- Déterminer si deux objets sont équivalents avec la méthode `equals`
- Comparer deux objets avec la méthode `compareTo`

Plus généralement, ils seront capables de créer des objets et de les manipuler en Java.

Outils

Nous supposons que tu travailles avec des versions d'Eclipse et du JDK compatibles avec les machines de l'école.

Eclipse : 2024-09 **JDK :** 21 **Junit** : 5.

Préalable

Durée estimée : 05 minutes.

Avant de réaliser les exercices, tu dois créer un projet dans un espace de travail, qui est un conteneur de projets. Nous allons créer un espace de travail dédié aux labos de POO.

Créer l'espace de travail

Pour créer un espace de travail, ouvre Eclipse. Par défaut, le lanceur d'Eclipse te propose de choisir un répertoire en tant qu'espace de travail (Figure 1)**. Choisis un chemin que tu retrouveras** facilement à l'aide d'un explorateur de fichiers.

Choisis par exemple :

- `c:\Users\`*`username`*`\Documents\Ecole\Pri\Poo` sous Windows.
- `/Users/`*`username`*`/Documents/Ecole/Pri/Poo` sous Mac.
- `/home/`*`username`*`/Documents/Ecole/Pri/Poo` sous Linux.

username est à remplacer par ton nom d'utilisateur. Évite les chemins comptant des espaces et des caractères accentués.

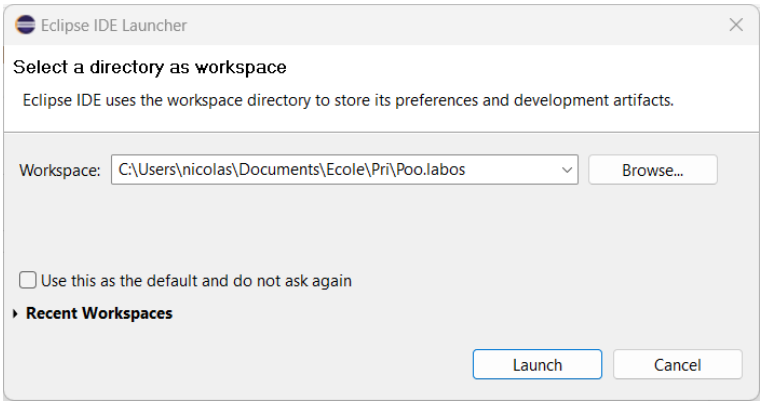


Figure 1 Lanceur Eclipse

Si le lanceur Eclipse n'apparait pas au lancement, tu peux créer un espace de travail depuis le menu *`File>Switch Workspace>Other`*... (Figure 2)

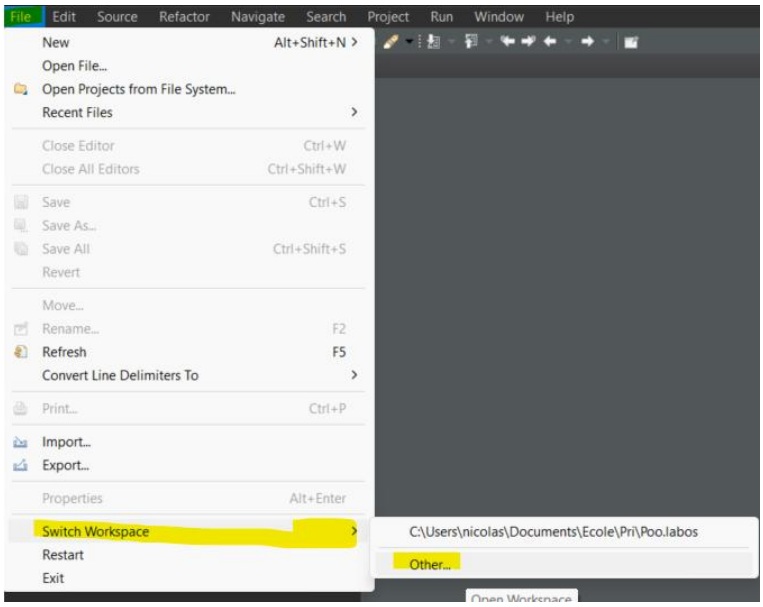


Figure 2 Changement d'espace de travail

Une fois l'espace de travail défini, tu pourras y ajouter tes projets.

Cibler Java 21

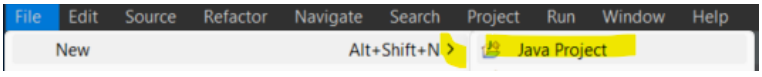
Nous te demandons de coder pour la version 21 de Java. Pour éviter tout problème de compatibilité, tu dois indiquer à Eclipse que ton espace de travail cible le JDK 21.

Pour ce faire, exécute l'item *`Window > Preferences`*. Une boîte de dialogue s'affiche. Ouvre l'onglet *`Java`* et clique sur *`Compiler`*. Dans la boite de choix *Compiler compliance level*, choisis 21.

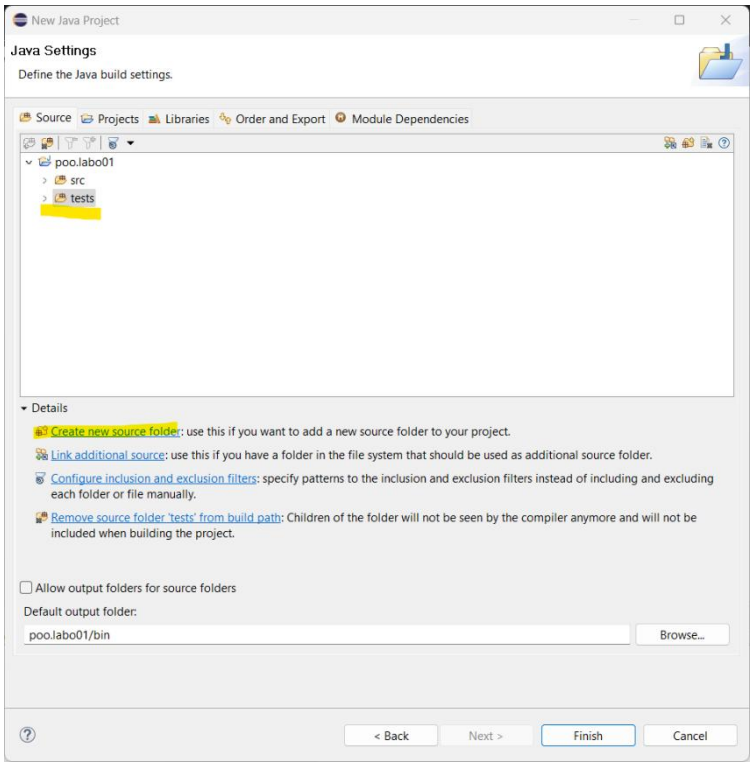
Applique les changements et ferme la boite de dialogue.

Créer un nouveau projet

Exécute l'item *`File>New>Java Project`*.



Une boite de dialogue apparait. Encode `poo.1abo01` comme nom de projet et clique sur *Next*. Crée un second répertoire de sources appelé tests (Figure 5) et clique sur *Finish*.



Ton projet est configuré. Eclipse l'affiche dans l'explorateur de paquetages (*Package Explorer*) de l'espace de travail.

Si tu fermes l'explorateur de package ou si Eclipse ne l'affiche pas, tu peux l'ouvrir en cliquant sur *Window* > *Show View* > *Package Explorer*.

Exercices

Le laboratoire propose des séries d'exercices de difficulté croissante. Après chaque série, le responsable propose une solution et répond à vos questions.

Certains exercices dépendent des exercices précédents : fais-les dans l'ordre. D'autres exercices donneront l'occasion de rappeler certains éléments essentiels pour aborder sereinement la POO.

Exercice 1

Objectifs :

- Créer des objets
- Appeler des méthodes d'objet de la classe
- Ecrire des tests unitaires

Durée estimée : 15 minutes.

Dans le répertoire des sources `src`, crée le paquetage `poo.labo01.utils`. Ajoutes-y une classe publique `StringUtils`. Dans cette classe, déclare la fonction `public static String min(String s1, String s2)`.

`min(String s1, String s2)` retourne le plus petit String entre s1 et s2 quand on les compare selon l'ordre lexicographique (celui du dictionnaire). Cette fonction appellera la méthode d'objet String.compareTo(String).

Les paramètres peuvent recevoir `null` en argument. Tout argument `null` sera remplacé par le texte vide, "".

Pour valider ta fonction, crée une classe `poo.labo01.utils.StringUtilsTest` dans le répertoire de source `tests`. Écris un test unitaire par ligne du tableau.

Premier argument de min		Second argument de min	Résultat attendu
`"abc"`		`"def"`	`"abc"`
`"Abc"`		`"Abc"`	`"Abc"`
`"def"`		`"Def"`	`"Def"`

Ajoute un quatrième test validant les arguments null.

Premier argument de min	Second argument de min	Résultat attendu
`null`	`"abc"`	`""`
`"def"`	`null`	`""`
`null`	`null`	`""`

Exercice 2

Objectifs :

- Créer des objets
- Appeler des méthodes d'objet de la classe
- Ecrire des tests unitaires

Durée estimée : 10 minutes.

Déclare la fonction `public static String max(String s1, String s2)` dans la classe `StringUtils`.

`max(String s1, String s2)` retourne le plus grand String entre s1 et s2 quand on les compare selon l'ordre lexicographique. Cette fonction appellera la méthode d'objet String.compareTo(String).

Les paramètres peuvent recevoir la valeur `null` en argument. Tout argument `null` sera remplacé par le texte vide, "".

Pour valider ta fonction, déclare un test unitaire pour chaque ligne du tableau dans la classe `StringUtilsTest`.

Premier argument de max	Second argument de max	Résultat attendu
`"abc"`	`"def"`	`"def"`
`"Abc"`	`"Abc"`	`"Abc"`
`"def"`	`"Def"`	`"def"`

Ajoute un quatrième test validant les arguments null.

Premier argument de max	Second argument de max	Résultat attendu
`null`	`"abc"`	`"abc"`
`"def"`	`null`	`"def"`
`null`	`null`	`""`

Quand tes fonctions `min(String, String)` et `max(String, String)` réussissent leurs tests, remanie ton code pour réduire les répétitions. Déclare une fonction `private static String checkNotNullOrReturnEmpty(String s)` qui retourne `s` si ce dernier n'est pas `null` et un string vide sinon. Appelle `checkNotNullOrReturnEmpty(String)` dans les fonctions `min(String, String)` et `max(String, String)`.

Exercice 3

Objectifs :

- Appeler des méthodes d'objet de la classe String
- Valider une fonction à l'aide de tests unitaires

Durée estimée : 20 minutes.

Déclare la fonction `public static String wordToTitleCase(String w)` dans la classe `StringUtils`.

`wordToTitleCase(String w)` retourne `w` transformé de la façon suivante :

- Si `w` est null ou blanc, retourner un texte vide.
- Sinon retourner la concaténation du premier caractère de w, converti en lettre majuscule, avec le reste de `w`, converti en lettres minuscules.

Dans cet exercice, nous supposons que les arguments sont des mots ne contenant ni séparateurs, comme un '-' ou '_', ni espaces entre deux lettres. Si l'argument commence ou se termine par des symboles d'espacement comme ' ' ou '\n', la fonction les enlèvera avant de procéder à la transformation.

Au minimum, ta fonction appellera les méthodes d'objet `isBlank()`, `charAt(int)`, `trim()` et `toLowerCase()`. Par ailleurs, la classe `Character` propose une fonction toTitleCase(char) pour la conversion d'un caractère en lettre majuscule.

Le paramètre peut recevoir la valeur `null` en argument. Tout argument `null` sera remplacé par le texte vide, "".

Pour valider ta fonction, déclare un test unitaire pour chaque ligne du tableau dans la classe `StringUtilsTest`.

Argument de wordToTitleCase(String)	**Résultat attendu**
`"abc"`	`"Abc"`
`"aBC"`	`"Abc"`
`"DEF"`	`"Def"`
`" \tDEF"`	`"Def"`
`"deF \n"`	`"Def"`
`"\t abc \t"`	`"Abc"`

Ajoute un dernier test validant les arguments null et les textes blancs.

Argument de wordToTitleCase(String)	**Résultat attendu**
null	`""`
`" \t "`	`""`
`""`	`""`

Correction avec le responsable

Durée estimée : 30 minutes.

Avant de poursuivre, ton responsable présente les corrections pour ces exercices. Les points d'attention suivants seront passés en revue :

- Clarifier le résultat de la méthode `compareTo(String)`. `compareTo(String)` ne retourne pas que -1,0 ou 1 (voir documentation officielle).
- Distinguer la valeur `null` du `String` vide, de longueur 0. Le `String` vide accepte les appels de méthode.
- Mettre en évidence le rôle joué par les tests pendant le remaniement : les tests signalent les régressions du code.
- Insister sur l'influence de l'objet associé aux appels sur le résultat. Les résultats retournés par les méthodes `isBlank()`, `charAt(int)`, `trim()` et `toLowerCase()` **dépendent des objets qui reçoivent les appels.**

Exercice 4 (intégration)

Objectifs : Écrire des appels de fonctions et des structures de contrôle

Durée estimée : 30 minutes

Déclare une classe `SmallestAndGreatestWords` dans le package `poo.labo01`. Déclare une méthode `public static void main(String[] args)` dans cette classe.

Crée un programme qui lit des mots encodés par l'utilisateur **jusqu'à ce que dernier encode le mot « Fin »**. Le programme affiche alors le plus petit mot et le plus grand mot encodés, **sans tenir compte du mot Fin**.

Quand il lit un mot, le programme lui applique les traitements suivants :

1. Il supprime les caractères d'espacement situés autour du mot ;
2. Il convertit le mot en *Titlecase*. La première lettre sera une majuscule et les suivantes seront des minuscules.
3. Il met à jour les minimum et maximum.

Utilisez les fonctions `min(String, String)`, `max(String, String)` et `wordToTitleCase(String)` que tu as définies précédemment.

Utilise la classe `Console` du cours de Programmation de base.

Exemples d'exécution

Cas d'exécution standard

```
Poo - Labo 1 - Exercice 4
=========================
Encodez un mot (Fin pour terminer) : BONJOUR
Encodez un mot (Fin pour terminer) : TOUT
Encodez un mot (Fin pour terminer) : VA
Encodez un mot (Fin pour terminer) : BIEN
Encodez un mot (Fin pour terminer) : 42
Encodez un mot (Fin pour terminer) : FIN
Plus petit mot : 42
Plus grand mot : Va
Fin d'exécution
```

Si l'utilisateur encode tout de suite "Fin", le programme affiche le message « Aucun mot encodé ».

```
Poo - Labo 1 - Exercice 4
=========================
Encodez un mot (Fin pour terminer) : Fin
Aucun mot encodé
Fin d'exécution
```

Le plus petit string est le string vide tandis qu'un string suffisamment grand est celui commençant par "\uffff".

Correction avec le responsable

Durée estimée : 30 minutes.

Avant de poursuivre, ton responsable présente la correction pour cet exercice et répond aux questions.

Exercice 5

Objectifs :

- Appeler des méthodes d'objet de la classe String
- Manipuler les appels de méthode et les structures de contrôle
- Utiliser des expressions régulières
- Créer des objets de la classe `java.util.StringJoiner` et appeler ses méthodes

Durée estimée : 20 minutes.

Déclare la fonction `public static String sentenceToTitleCase(String s)` dans la classe `StringUtils`.

`sentenceToTitleCase(String s)` retourne la phrase `s` transformée de la façon suivante :

- Si s est `null` ou blanc, la fonction retourne le texte vide.
- Sinon les mots qui composent s sont convertis en *TitleCase*, chaque mot est séparé par un espace.

Dans cet exercice, nous supposons que les arguments reçus sont des phrases où les mots sont des séquences de lettres et de chiffres. Les mots sont séparés par des signes de ponctuation et/ou des espaces. La fonction élimine également les éventuels caractères d'espacement situés avant et après le mot.

Pour séparer les mots de la phrase, appelle la méthode d'objet `String.split(String regex)`. Pour transformer les mots en TitleCase, appelle la fonction définie pendant l'exercice 3. Enfin, pour reconstruire la phrase, utilise un objet StringJoiner. Tu trouveras un exemple d'utilisation de cette classe dans la section 2.4 du syllabus.

Le paramètre peut recevoir la valeur `null` en argument. Tout argument `null` sera remplacé par le texte vide, "".

Pour valider ta fonction, déclare un test unitaire pour chaque ligne du tableau dans la classe `StringUtilsTest`.

Argument de sentenceToTitleCase(String)	**Résultat attendu**
`"bonjour TOUT va BiEN"`	`"Bonjour Tout Va Bien"`
`"BONJOUR, TOUT VA BIEN !"`	`"Bonjour Tout Va Bien"`
`"\u03b1, \u03b2, \u03b3."`	`"\u0391 \u0392 \u0393"`[1]
`" \t \n"`	`""`
`Null`	`""`

[1] \u0391 est un exemple de symbole Unicode. Il correspond à la lettre α.

Exercice 6

Objectifs :

- Copier des fichiers source Java aux bons endroits
- Analyser du code écrit par un tiers
- Déterminer si deux Strings sont équivalents

Durée estimée : 10 minutes.

Deux fichiers Java `WordFrequencies` et `WordFrequenciesTest` sont disponibles en annexe. Télécharge-les et ajoute-les à ton projet.

La fonction `public static Object[][] computeWordsFrequencies(String[] words, String t)` retourne un tableau composé de paires `(String, int)`. Le premier élément de la paire est `words[i]` et le second est le nombre de fois que `words[i]` apparait dans `t`.

Exemple

```
Si words = {"quand", "poules", "la"}
et t =  "quand trois poules vont aux champs, la première va devant.
         la deuxieme suit la premiere.
         la troisieme vient en derniere.
         quand trois poules vont aux champs, la première va devant."

computeWordsFrequencies(words, t) retourne le tableau
{
    {"quand", 2},
    {"poules", 2},
    {"la", 5},
}
```

Un bug s'est glissé dans ce code. Il empêche de compter correctement les fréquences des mots dans le texte. Trouve ce bug et corrige-le.

Pour corriger le problème, rappelle-toi que les objets possèdent une identité et un état. Rappelle-toi également que l'égalité entre deux objets ne correspond pas toujours à l'égalité entre deux références.

Correction avec le responsable

Durée estimée : 20 minutes.

Ton responsable présente les corrections pour ces exercices. Les points d'attention suivants seront notamment passés en revue :

- Revenir sur les expressions régulières et leur rôle dans la méthode `split(String)`
- Insister sur la nécessité de créer un objet avant de pouvoir l'utiliser

- Tester l'égalité entre deux objets se fait avec la méthode `equals(Object)`.

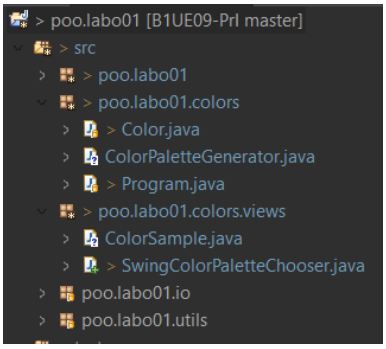
Exercice 7

Objectifs :

- Créer des objets
- Combiner appels de méthode et structures de contrôle
- Créer et garnir des tableaux
- Concevoir et définir une suite de tests unitaires JUnit5

Durée estimée : 60 minutes à terminer pour la séance suivante

Télécharge et décompresse l'archive `poo.labo01.colors.zip` à la racine de ton projet, puis rafraichis-le (touche F5). Le package explorer devrait te présenter la structure suivante.



Tu dois compléter une application générant une palette de 5 couleurs harmonieuses à partir d'une couleur choisie par l'utilisateur. La Figure 3 te donne une idée du résultat attendu.

Figure 3 Exemple de palette générée avec la couleur HELMo

L'application doit générer une palette de 5 couleurs harmonieuses sur base d'une couleur choisie par l'utilisateur. Pour générer la palette, nous appliquerons 5 rotations à la couleur choisie de 0, 72, 144, 216 et 288 degrés.

Tu dois produire une dernière couleur pour l'affichage du texte. Tu peux déterminer cette couleur avec la luminosité de la couleur encodée : si la luminosité est inférieure à 30%, ta couleur est plutôt sombre et tu peux utiliser le blanc, sinon le noir convient mieux.

Modifie le corps de la méthode `ColorPaletteGenerator.updatePalette(int r, int g, int b)` pour qu'elle retourne un tableau composé de 6 couleurs formattées en HTML hex (#RRGGBB).

La méthode reçoit la couleur choisie par l'utilisateur sous la forme de trois entiers représentant les composantes rouge, verte et bleue de la couleur. Les 5 premiers éléments correspondent aux couleurs obtenues par rotation et le dernier élément correspond à la couleur utilisée pour afficher les textes.

Pour accomplir ta tâche, utilise la version augmentée de la classe `Color`[2] proposée dans l'archive. Tu peux notamment appeler les méthodes suivantes :

- `Color rotate(float angleInDegrees)` qui retourne une nouvelle couleur construite en opérant une rotation de angleInDegrees degrés de la teinte de cette couleur.
- `float getLightness()` qui retourne la luminosité de cette couleur sous la forme d'un `float` compris entre `0.0f` et `1.0f` inclus.
- `String toHTMLHex()` qui retourne une représentation de cette couleur au format HTML hex.

Tâche de rendre ton code souple. Idéalement, changer le nombre de couleurs à générer ne doit demander que de modifier une variable : le reste de ta méthode doit s'adapter naturellement.

Écris quelques tests unitaires pour valider ta méthode. Au minimum, vérifie que pour la couleur HELMo (r=217, v=11, b=67), ta méthode te retourne le tableau #D90B43, #D9CB0B, #0BD926, #0B94D9, #790BD9 et #000000.

Une correction du travail sera proposée au début du labo 2.

[2] Cette classe a été utilisée lors des premières séances théoriques.