Laboratoire 3

DURÉE PRÉVUE : 8H00

OBJECTIFS

Au terme de ce laboratoire, l'étudiant sera capable de :

- → Créer un projet Java sous *Eclipse* et organiser ses éléments (répartir les fonctions dans différentes classes et les classes dans différents package).
- → Compiler, exécuter et déboguer un programme sous *Eclipse*.
- → Déclarer une fonction statique.
- → Tester une fonction statique à l'aide de *JUnit*.
- → Documenter les classes et fonctions à l'aide de la *javadoc*.

AVANT-PROPOS

Dans un premier temps, ce laboratoire va vous aider à faire vos premiers pas avec le logiciel ***Eclipse***. *Notepad++* ne sera dès lors plus utilisé.

Eclipse est un **environnement de développement intégré** (EDI ou, en anglais, *IDE*) puisqu'il comporte un ensemble d'outils nécessaires à la programmation, en particulier : un éditeur de texte, un compilateur et un débogueur.

AVERTISSEMENT

Vous devez au préalable avoir réalisé dans son entièreté la procédure d'installation d'*Eclipse* détaillée dans le document disponible sur l'espace de cours.

Avant de créer vos premiers programmes dans *Eclipse*, vous allez vous familiariser avec son fonctionnement.

EXERCICE 1 : DÉBUTER AVEC ECLIPSE

CRÉER UN PROJET ET IMPORTER LES PROGRAMMES RÉALISÉS PRÉCÉDEMMENT

Avec *Eclipse*, les programmes doivent être intégrés à un projet pour pouvoir être compilés et exécutés.

Nous allons d'abord mettre en place l'organisation que nous utiliserons dorénavant pour nos programmes :

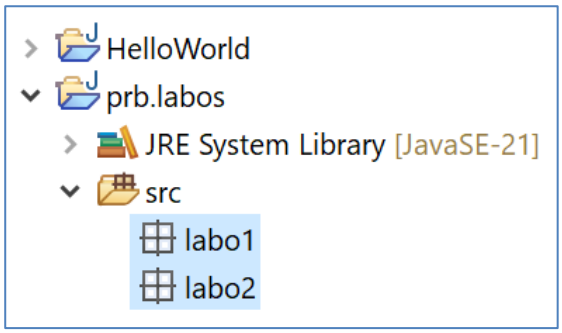
1. Créez un projet Java nommé **`prb.labos`**.

   IMPORTANT

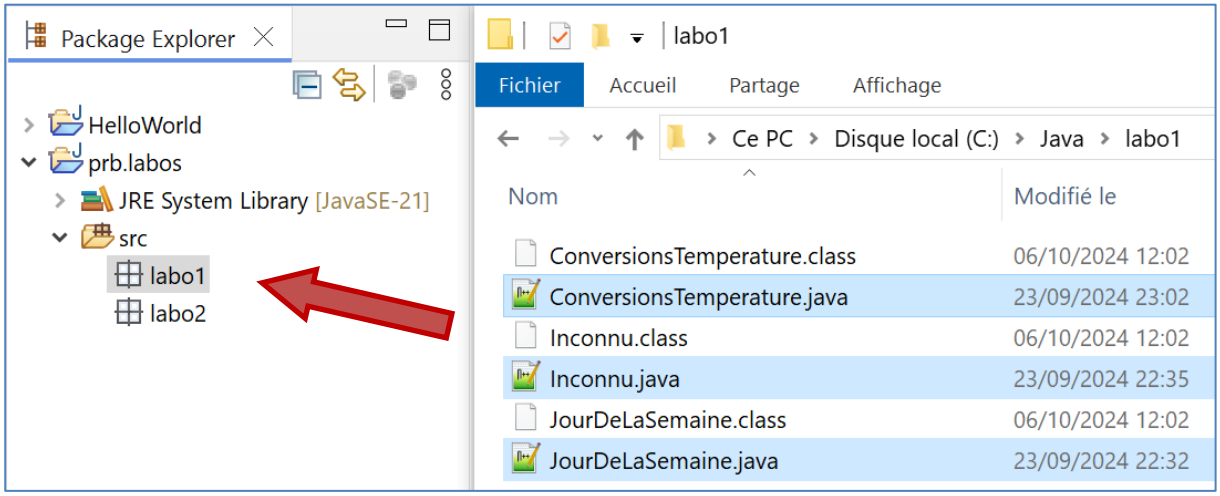
   Décochez la case ***Create module-info.java file***.

2. Afin de récupérer les programmes réalisés dans le cadre des laboratoires précédents, ajoutez deux packages nommés **`labo1`** et **`labo2`** dans le dossier **`src`** destiné aux fichiers sources.

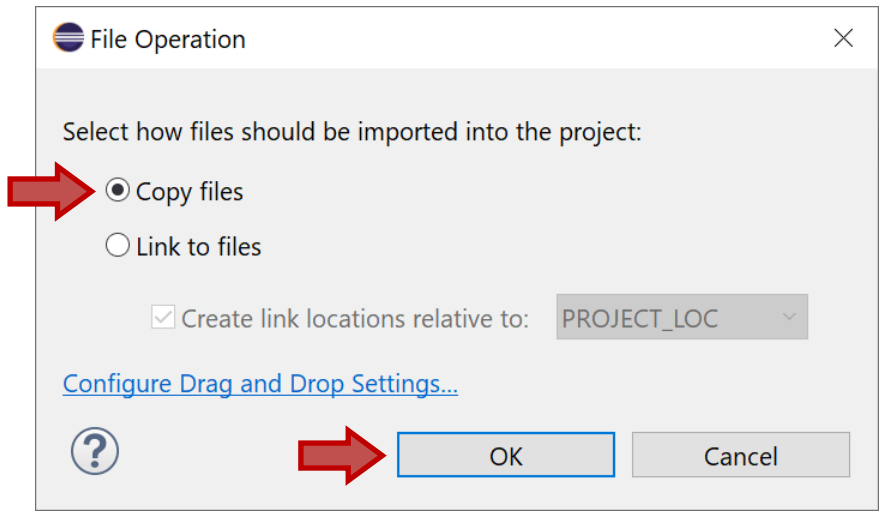
   Pour ce faire, faites un clic droit sur le dossier **`src`** pour faire apparaître le menu contextuel, puis sélectionnez ***New*** > ***Package***. Effectuez cette opération deux fois.



3. Depuis l'*Explorateur de fichiers*, effectuez un glisser-déposer des fichiers « .java » enregistrés dans votre dossier de travail (autrement dit, les fichiers **`C:\Java\labo1\*.java`** et **`C:\Java\labo2\*.java`**) vers les *packages* appropriés du ***Package Explorer*** d'*Eclipse*.



4. Lorsque la fenêtre *File Operation* apparaît, sélectionnez l'option ***Copy Files***, puis cliquez sur le bouton ***OK*** pour créer une copie des fichiers sélectionnés dans le dossier de destination.

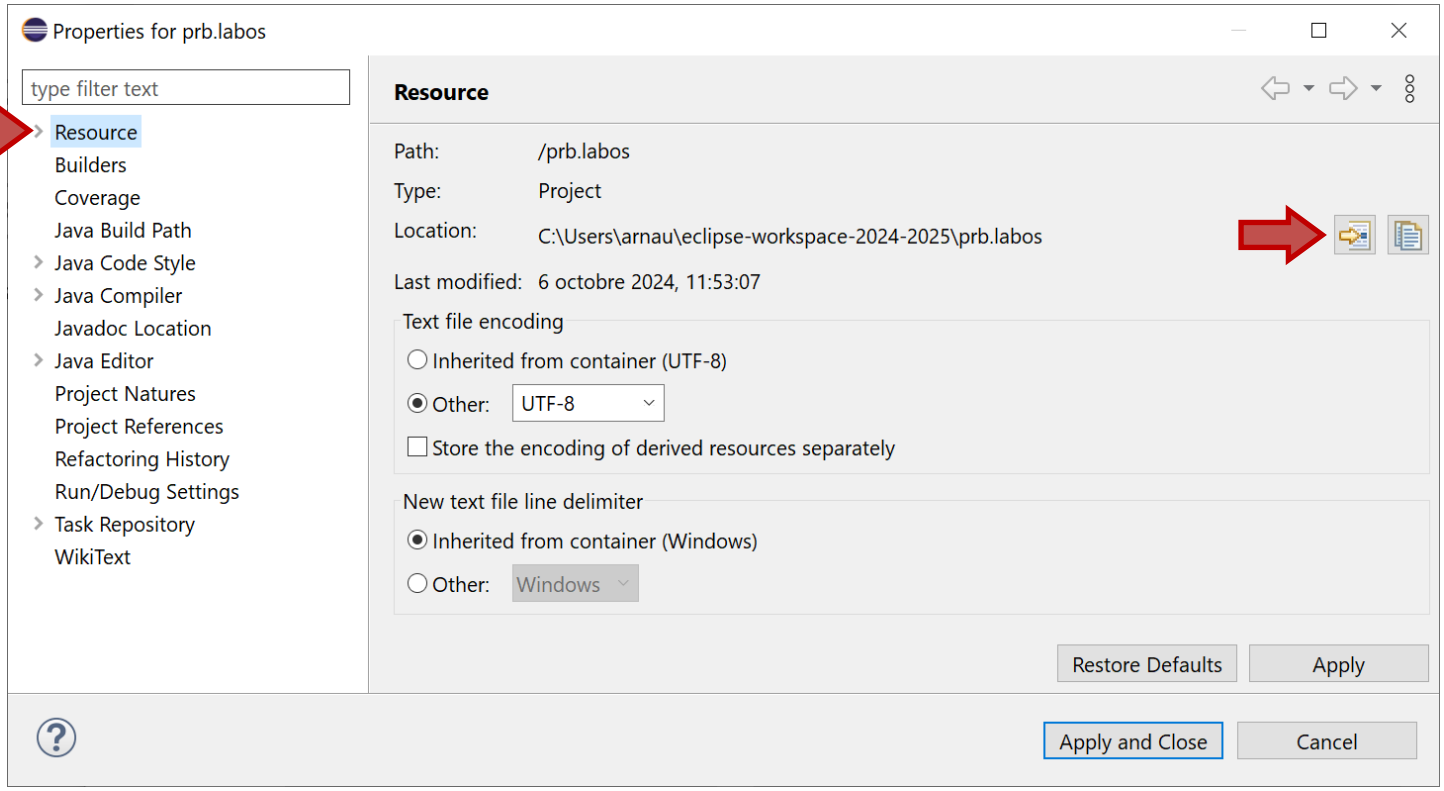


5. Vous pouvez tester l'exécution de l'un de vos anciens programmes…

QU'EST-CE QU'UN PROJET ECLIPSE ?

1. Dans le volet *Package Explorer*, faites un clic droit sur le projet **prb.labos**, puis sélectionnez ***Properties*** dans le menu contextuel.

2. Dans la fenêtre *Properties* qui vient de s'ouvrir, la page ***Resource*** est affichée par défaut. Celle-ci affiche des informations relatives à notre projet. L'emplacement (***Location***) du projet est également indiqué.



3. Cliquez sur le bouton ***Show In System Explorer*** pour vous rendre à l'emplacement du projet. Plusieurs éléments ont été générés dans le dossier **prb.labos**, parmi lesquels :

   → Le fichier « **.project** » contient des informations générales relatives au projet, parmi lesquelles son nom.

   → Le dossier « **src** » est l'emplacement dans lequel tous les fichiers sources (**.java**) sont enregistrés.

   → Le dossier « **bin** » est l'emplacement dans lequel tous les fichiers compilés (**.class**) sont enregistrés.

DÉCOUVRIR LES FONCTIONNALITÉS DE BASE D'ECLIPSE

1. Ajoutez un *package* nommé **`io`** au dossier **`src`**, puis glissez-déposez la classe **`Console`** dans ce package. Modifiez le package de la classe en conséquence.

   > **REMARQUE**
   >
   > Dorénavant, vous ne serez plus obligé de placer la classe **`Console`** dans chaque package. Il vous suffira d'ajouter l'instruction « **`import io.Console;`** » au début de vos programmes.

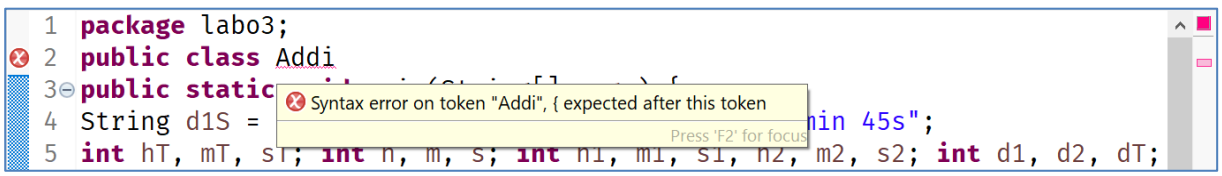
2. Ajoutez un package nommé **`labo3`** au dossier **`src`**.

3. Téléchargez le fichier **`Addi.java`**, puis placez-le dans le package **`labo3`**. Vous pourrez constater que le programmeur qui a réalisé ce programme n'a pas considéré utile de corriger et de soigner son code.

4. Tout d'abord, à la manière des correcteurs présents dans les traitements de texte, l'éditeur d'*Eclipse* nous signale la présence de 3 erreurs dans le code.

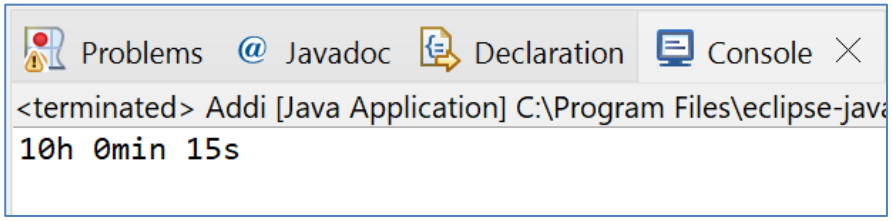
   Chaque erreur est indiquée grâce à un soulignement rouge et une icône affichée dans la marge de gauche.

   Placez le curseur de votre souris sur l'un de ces indicateurs, puis ne le bougez plus afin d'afficher une infobulle décrivant l'erreur correspondante :



   A l'aide de ces descriptions, corrigez les trois erreurs.

5. Compilez et exécutez le programme en cliquant sur le bouton  situé dans la barre d'outils. Le résultat suivant s'affiche dans la console d'*Eclipse* :



6. Pour rendre ce programme plus compréhensible, vous devrez améliorer :

   a. la **mise en forme** du code, appelée usuellement l'*indentation* (une instruction par ligne, chacune décalée d'un certain nombre de tabulations) ;

   b. les **identifiants** de la classe et des variables (noms explicites) ;

   c. la **structure** du code (l'organisation de ses instructions par groupements cohérents) ;

   d. la **documentation** du code (*javadoc* et commentaires expliquant le code).

7. Améliorez la mise en forme du code : via le menu ***Source*** > ***Format*** ou via le raccourci **Ctrl** + **Maj** + **F**.

8. Renommez les identifiants : cliquez d'abord sur l'identifiant à renommer, puis via le menu ***Refactor*** > ***Rename…*** ou via le raccourci **Alt** + **Maj** + **R**.

   Par exemple, renommez :

   a. La classe `Addi` en `AdditionDurees` (le nom du fichier est également renommé par la même occasion).

   b. La variable `d1S` en `duree1EnStr` (dans le même temps, toutes les occurrences de cette variable sont également renommées).

9. Organisez les instructions en effectuant des groupements cohérents. Pour ce faire, ajoutez des lignes vides afin de séparer ces différents groupements.

   Par exemple, créez un groupement pour : les déclarations de variables ; les acquisitions ; chaque étape du traitement ; les instructions relatives à l'affichage.

10. Ajoutez des commentaires en ligne `//` pour décrire chaque groupement.

11. Ajoutez un commentaire *javadoc* à la classe.

    Afin de générer la structure du commentaire, procédez de la manière suivante :

    a. Faites précéder l'intitulé de la classe d'une ligne vide.

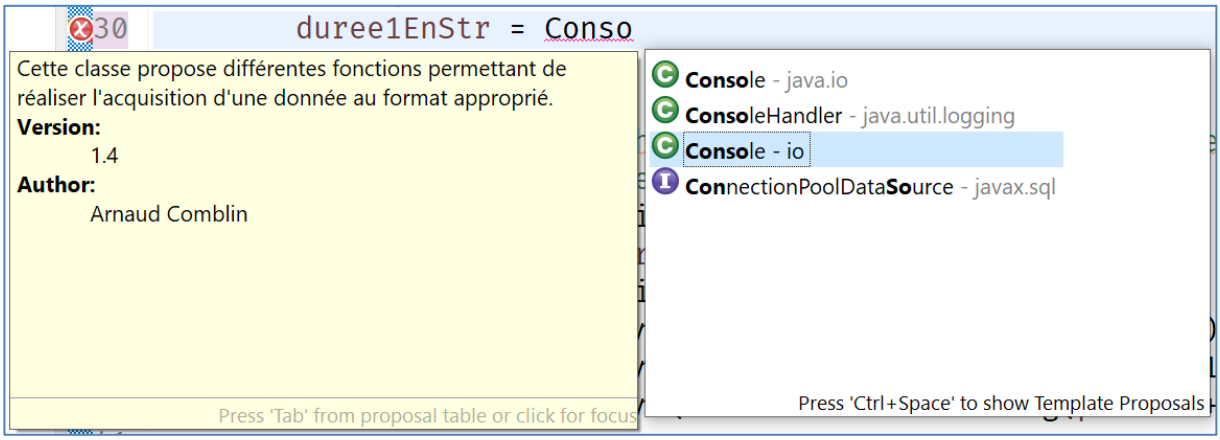
    b. Au début de cette ligne vide, saisissez les trois caractères `/**`, puis appuyez sur la touche ***Entrée***.

    c. Dans la zone du commentaire généré, saisissez une courte description de la tâche réalisée par le programme.

12. Enfin, améliorez le programme en permettant l'acquisition des deux durées à additionner sous la forme de chaînes de caractères.

    Procédez précisément de la manière suivante :

    a. Saisissez les cinq lettres « **Conso** », puis utilisez le raccourci **Ctrl** + **Espace** pour activer l'**auto-complétion** sous la forme d'une liste déroulante proposant les suggestions les plus probables pour terminer la saisie :



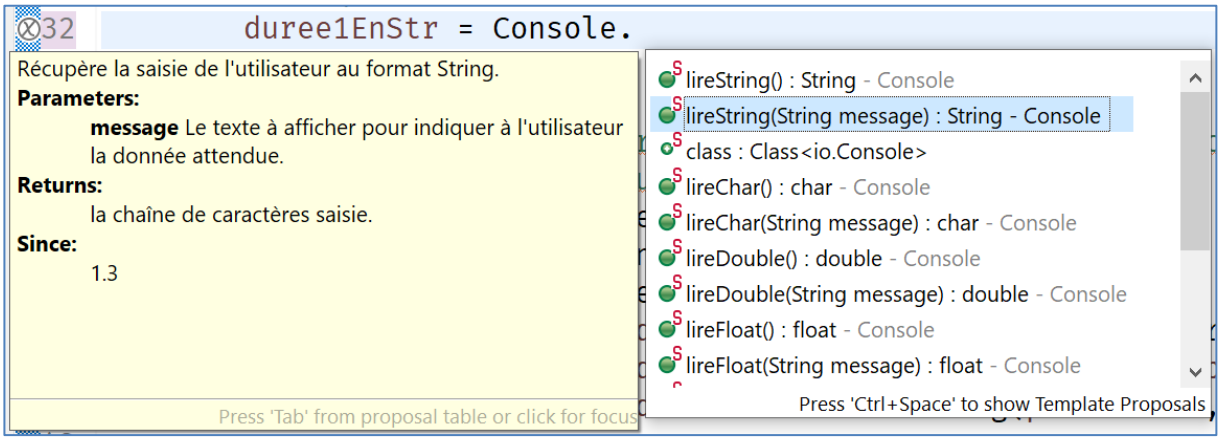
    b. Effectuez d'abord un simple clic gauche pour sélectionner l'élément **Console** - io (ou utilisez les touches représentant les flèches haut et bas pour parcourir les éléments de la liste).

       L'infobulle située à gauche de la liste déroulante s'actualise pour afficher la *javadoc* de la classe ainsi sélectionnée.

    c. Double-cliquez maintenant sur l'élément **Console** - io pour le valider (ou appuyez sur la touche ***Entrée***).

       Dans ce cas, l'effet est double : le nom de la classe se complète et l'instruction `import` correspondante est automatiquement ajoutée en début de fichier.

    d. Poursuivez la saisie avec le caractère `.` (point) pour voir s'afficher une nouvelle liste de suggestions :



    e. Sélectionnez et validez la fonction lireString(String message) : String - Console

f. Terminez de coder les acquisitions.

13. Exécutez et testez le programme.

EXERCICE 2 : PREMIÈRES FONCTIONS

DESCRIPTION DU PROGRAMME

Le programme à réaliser doit être capable de calculer la **température théorique** à la surface des huit planètes de notre système solaire. La température à la surface d'une planète dépend essentiellement de trois facteurs :

- La distance qui sépare la planète du Soleil, car plus la planète est éloignée du Soleil, plus la température à sa surface diminue.
- L'albédo de la surface de la planète, c'est-à-dire sa capacité à réfléchir ou à absorber le rayonnement solaire. Sa valeur est exprimée entre 0 (pas de diffusion) et 1 (pas d'absorption).
- L'effet de serre.

REMARQUE

Pour ceux que cela intéresse, vous trouverez davantage d'explications sur le sujet en consultant la page suivante :

https://fr.wikibooks.org/wiki/Plan%C3%A9tologie/La_temp%C3%A9rature_de_surface_des_plan%C3%A8tes

Pour mesurer les distances dans le système solaire, c'est l'unité astronomique (en abrégé **au**, pour *astronomical unit*) qui est utilisée. 1 au correspond à la distance moyenne séparant la Terre du Soleil. Cela équivaut à environ 150 millions de km (sa valeur exacte est de 149 597 870,700 km).

Voici les données pour les planètes du système solaire :

Planète	Distance moyenne au Soleil *(arrondie au centième près)*	Albédo planétaire
Mercure	0,38 au	0,068
Vénus	0,72 au	0,770
Terre	1,00 au	0,294
Mars	1,52 au	0,250
Jupiter	5,21 au	0,343
Saturne	9,54 au	0,342
Uranus	19,18 au	0,300
Neptune	30,11 au	0,290

La température théorique à la surface d'une planète, exprimée en **degrés Kelvin**, peut être approchée à l'aide de la formule suivante :

$$T_K = 280 \times \sqrt[4]{\frac{1 - albédo}{distance^2}}$$

Ou encore,

$$T_K = 280 \times \left(\frac{1 - albédo}{distance^2}\right)^{1/4}$$

où la distance est exprimée en au.

Ensuite pour exprimer cette température en degrés Celsius, il suffit d'utiliser la formule suivante :

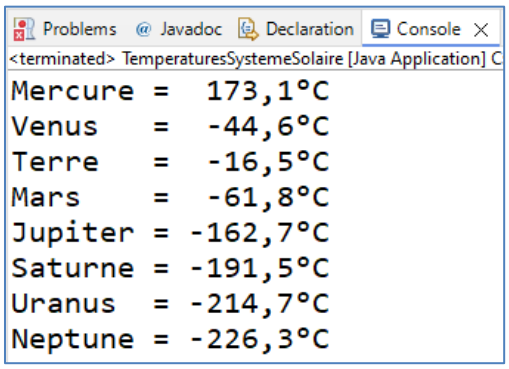
$$T_C = T_K - 273{,}15$$

REMARQUE

La température obtenue avec cette formule ne correspond pas tout à fait à la réalité, car elle ne tient pas compte de certains paramètres, parmi lesquels l'effet de serre. Par exemple, la température moyenne obtenue pour la Terre est de -16,5°C alors qu'elle est en réalité de 15°C.

EXEMPLE D'EXECUTION

Veillez à noter qu'il n'y a aucune acquisition à réaliser dans le cadre de ce programme. Ce dernier se contente d'afficher les températures calculées pour chaque planète :



CONSIGNES

1. Dans le package **`labo3`** créé précédemment, ajoutez une classe nommée **`TemperaturesSystemeSolaire`**.

   ASTUCE

   Cette classe doit contenir la fonction `main`. Pour ce faire, dans la fenêtre *New Java Class*, cochez la case « **`public static void main(String[] args)`** ».

2. Afin de vous faciliter la tâche, vous pouvez copier-coller les constantes ci-dessous dans votre fonction **main**.

CODE À UTILISER

```
final double DIST_MERCURE = 0.38, ALBEDO_MERCURE = 0.068;
final double DIST_VENUS = 0.72, ALBEDO_VENUS = 0.770;
final double DIST_TERRE = 1.00, ALBEDO_TERRE = 0.294;
final double DIST_MARS = 1.52, ALBEDO_MARS = 0.250;
final double DIST_JUPITER = 5.21, ALBEDO_JUPITER = 0.343;
final double DIST_SATURNE = 9.54, ALBEDO_SATURNE = 0.342;
final double DIST_URANUS = 19.18, ALBEDO_URANUS = 0.300;
final double DIST_NEPTUNE = 30.11, ALBEDO_NEPTUNE = 0.290;
```

Dans un premier temps, codez le programme intégralement dans la fonction **main**. Si vous trouvez la tâche trop fastidieuse, vous pouvez vous limiter à 4 planètes au lieu de 8.

Pour calculer la racine 4ième, aidez-vous de la fonction **Math.pow** en spécifiant « 1 / 4 » comme exposant.

ASTUCE

Pour codez plus rapidement un appel de la fonction **println**, saisissez la séquence de caractères « **sysout** », puis activez l'auto-complétion avec le raccourci **Ctrl** + **Espace**.

3. Lorsque votre programme est terminé, exécutez-le et vérifiez que les résultats correspondent à ceux de l'exemple d'exécution.

IMPORTANT

A ce stade, il est probable que vous obteniez des résultats identiques pour toutes les planètes, à savoir **6,9°C**.

Si c'est le cas :

Passez directement à la section « **UTILISER LE DEBOGUEUR D'ECLIPSE** » sans lire la suite de cet encadré !

Si les résultats affichés sont corrects :

Sélectionnez la phrase cachée délimitée par les symboles `/*  …  */`, copiez-la, puis collez-la dans l'éditeur d'*Eclipse*.

```
/*                                                    */
```

Exécutez à nouveau votre programme pour voir les résultats maintenant obtenus, puis passez à la section « **UTILISER LE DEBOGUEUR D'ECLIPSE** ».

UTILISER LE DÉBOGUEUR D'ECLIPSE

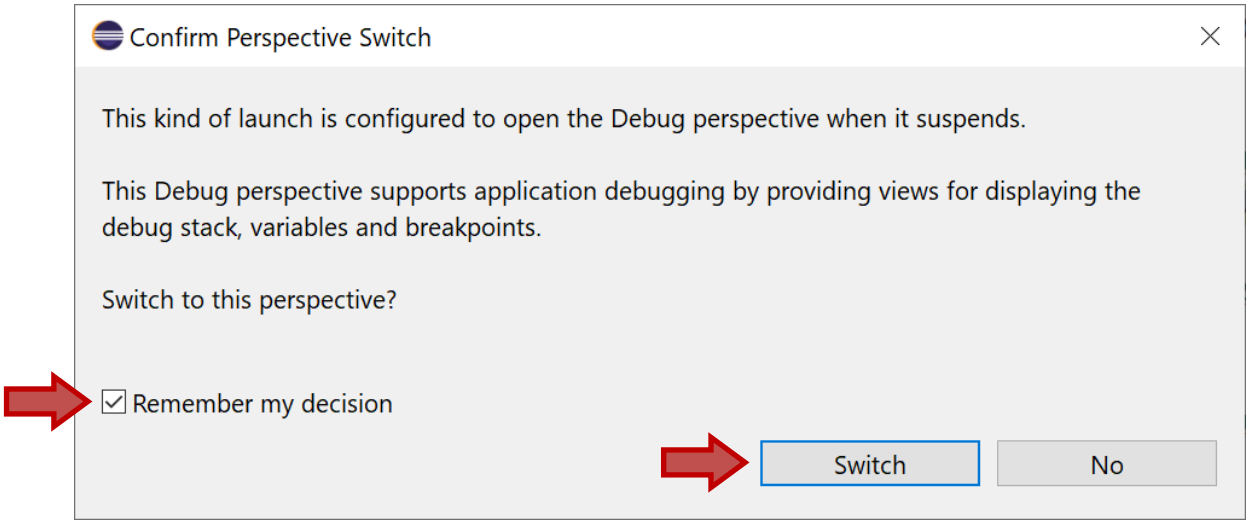
Pour trouver rapidement la cause d'un résultat erroné, la maîtrise de l'outil de débogage est indispensable. L'approche proposée ici est la plus simple. Elle vous permettra de vous débrouiller dans la plupart des cas. Sachez toutefois que le débogueur propose de nombreuses possibilités.

1. Commencez par placez un **point d'arrêt** à la ligne du calcul de la puissance 4ième. Pour ce faire, effectuez un double-clic à gauche du numéro de la ligne correspondante :

```
19  // Traitement
20  // 1. Calcul de la température moyenne pour Mercure
21  tempMercureEnC = 280 * Math.pow((1 - ALBEDO_MERCURE) / Math.pow(DIST_MERCURE, 2), 1 / 4) - 273.15;
22
```

2. Exécutez maintenant le programme cliquant sur le bouton afin d'activer le **mode Débogage** (qui prend en compte les points d'arrêts).

   Si c'est la première fois que vous exécutez un programme dans ce mode, *Eclipse* vous propose de changer de **Perspective** :



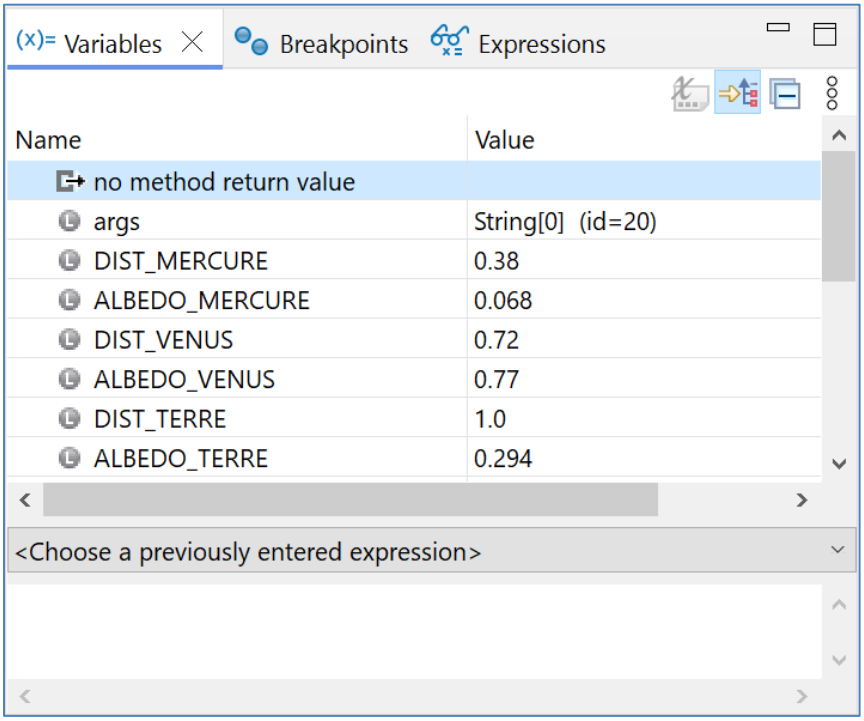
   Cochez la case « ***Remember my decision*** », puis cliquez sur ***Switch***.

   Une Perspective est un agencement de sous-fenêtres que le développeur peut configurer selon ses besoins pour la réalisation de tâches spécifiques. Dans le cadre de ce cours, deux perspectives seront utiles : **Java** pour développer en Java, **Debug** pour effectuer du débogage.

   Le choix d'une Perspective peut se faire via le menu situé en haut à droite de la fenêtre d'*Eclipse* :

   Vous pourrez ainsi basculer de l'une à l'autre d'un simple clic.

3. Après ce changement de Perspective, de nouvelles sous-fenêtres font leur apparition, en particulier : **Variables**, **Breakpoints**, **Expressions**.



C'est la sous-fenêtre Variables qui va particulièrement nous intéresser dans cet exercice. Celle-ci renseigne actuellement les noms et valeurs des constantes et des variables (ces dernières vont apparaître au fur et à mesure de l'exécution du programme) de la fonction **`main`**.

4. Dans l'éditeur d'*Eclipse*, la ligne à laquelle le point d'arrêt a été placé est affichée en surbrillance :

```
19  // Traitement
20  // 1. Calcul de la température moyenne pour Mercure
21  tempMercureEnC = 280 * Math.pow((1 - ALBEDO_MERCURE) / Math.pow(DIST_MERCURE, 2), 1 / 4) - 273.15;
22
```

Cela signifie que toutes les instructions précédentes ont déjà été exécutées, tandis que l'instruction actuelle sera la prochaine à être exécutée.

5. A partir de ce point, il est possible de diriger l'exécution du programme à l'aide des boutons ci-dessous :

Exécutez l'instruction actuelle : via le bouton **Step Over** ou via le raccourci **F6**.

L'instruction suivante s'affiche en surbrillance et la variable apparaît dorénavant dans la sous-fenêtre Variables :

> **REMARQUE**
>
> Notez l'approximation de la valeur enregistrée qui est due à la représentation binaire utilisée : la **virgule flottante**, ou *floating-point*.

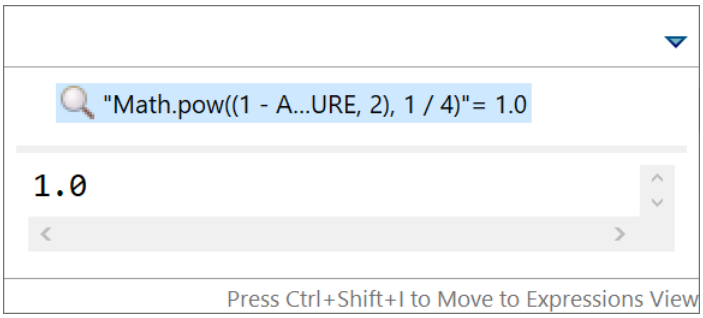
6. Sans surprise, la valeur observée est identique à celle affichée lors de l'exécution précédente. Nous allons cette fois essayez d'en apprendre plus sur le calcul effectué qui produit ce résultat erroné.

   Dans un premier temps, sélectionnez l'expression qui doit calculer la racine 4ième :

```
19  // Traitement
20  // 1. Calcul de la température moyenne pour Mercure
21  tempMercureEnC = 280 * Math.pow((1 - ALBEDO_MERCURE) / Math.pow(DIST_MERCURE, 2), 1 / 4) - 273.15;
22
```

   Faites évaluer le résultat de cette expression : via ***Run > Inspect*** ou via le raccourci **Ctrl** + **Maj** + **I**.

   La fenêtre ci-dessous s'affiche avec le résultat correspondant :



7. Procédez ensuite de la même manière pour connaître les valeurs des deux arguments transmis à la fonction `Math.pow`, à savoir :

   a. la base (ici, « `(1 - ALBEDO_MERCURE) / Math.pow(DIST_MERCURE, 2)` ») ;

   b. l'exposant (ici, « `1 / 4` »).

   Vous devriez ainsi comprendre d'où provient le problème. Appuyez sur le bouton d'**arrêt de l'exécution** .

8. Sélectionnez la Perspective **Java**, puis corrigez votre code.

9. Exécutez en **mode Normal** le programme afin de vérifier si les résultats affichés sont dorénavant tous corrects.

CREER ET DOCUMENTER DES FONCTIONS

Lors de la réalisation de ce programme, vous aurez évidemment constaté la redondance des traitements à effectuer. Cette redondance peut être réduite à l'aide de fonctions. Sur base des consignes suivantes, vous devrez créer deux fonctions, puis les utiliser dans votre programme principal.

Dans la classe **`TemperaturesSystemeSolaire`** :

1. Déclarez une fonction nommée **`enCelsius`** capable de convertir en degrés Celsius une température exprimée en degrés Kelvin :

   ```
   public static double enCelsius(double tKelvin)
   ```

   Le paramètre `tKelvin` indique la température exprimée en degrés Kelvin à convertir.

   La fonction retourne la température spécifiée convertie en degrés Celsius.

2. Déclarez une fonction nommée **`temperatureTheorique`** capable de calculer la température théorique de la surface d'une planète du système solaire :

   ```
   public static double temperatureTheorique(double distanceAU, double albedo)
   ```

   Le paramètre `distanceAU` indique la distance moyenne qui sépare la planète du Soleil exprimée en au.

   La paramètre `albedo` indique la valeur d'albédo de la surface de la planète.

   La fonction retourne la température théorique de la surface exprimée en degrés Kelvin.

3. Ajoutez une *javadoc* aux deux fonctions précédentes.

   Afin de générer la structure du commentaire, procédez de la manière suivante :

   a. Faites précéder l'en-tête de la fonction d'une ligne vide.

   b. Au début de cette ligne vide, saisissez les trois caractères **`/**`**, puis appuyez sur la touche ***Entrée***.

   c. Dans la zone du commentaire généré, saisissez une courte description de la tâche réalisée par le programme, des paramètres et de la valeur retournée.

      Pour ce faire, inspirez-vous des descriptions données aux points précédents. Par exemple :

```
/**
 * Convertit en degrés Celsius une température exprimée en degrés Kelvin.
 *
 * @param tKelvin La température exprimée en degrés Kelvin à convertir.
 * @return la température spécifiée convertie en degrés Celsius.
 */
public static double enCelsius(double tKelvin) {
```

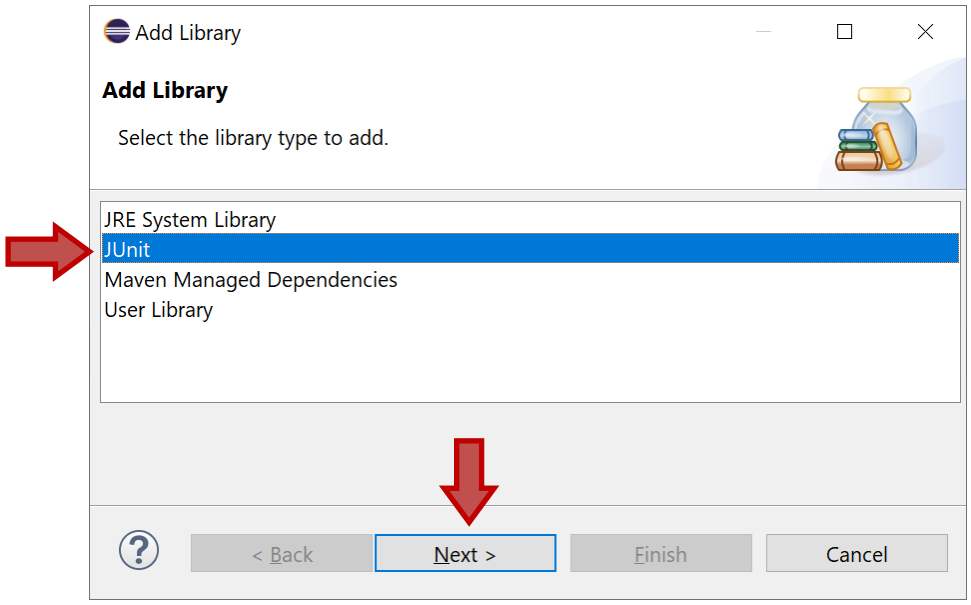
TESTER DES FONCTIONS AVEC JUNIT

Dorénavant, nous prendrons l'habitude de tester systématiquement nos fonctions à l'aide de l'outil ***JUnit*** avant de les utiliser dans le cadre du programme principal, et ce, afin de se prémunir de bugs plus complexes à déceler ultérieurement.

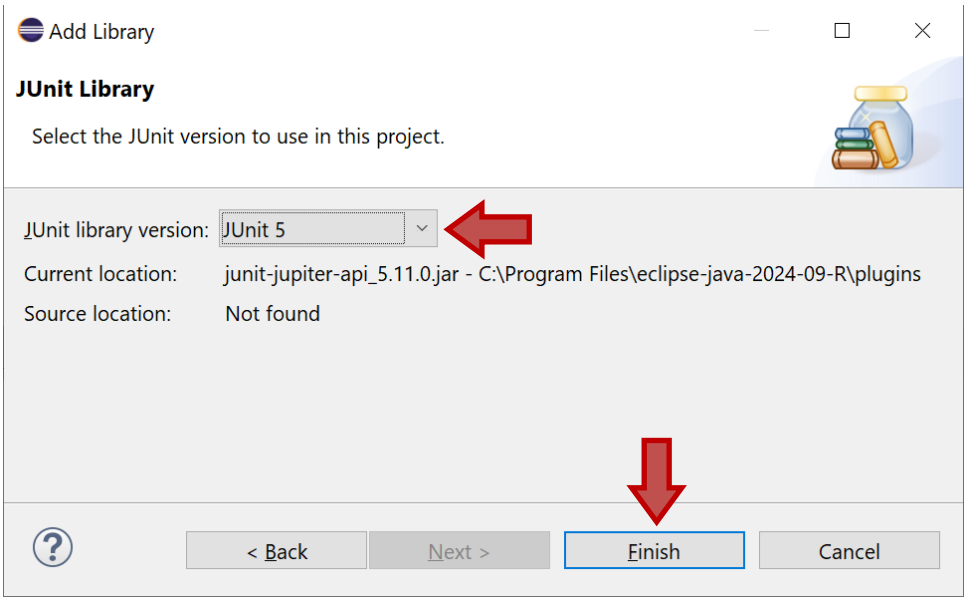
Dans le ***Package Explorer*** :

1. Avant tout, il faut intégrer la bibliothèque regroupant les fonctionnalités de *JUnit*.

   Faites un clic droit sur le projet **`prb.labos`** pour faire apparaître le menu contextuel, puis sélectionnez ***Build Path*** > ***Add Libraries...***.



   Dans cette première fenêtre, sélectionnez ***Junit***, puis cliquez sur ***Next >***.



   Dans la fenêtre suivante, sélectionnez ***JUnit 5*** dans la liste déroulante, puis cliquez sur ***Finish***.

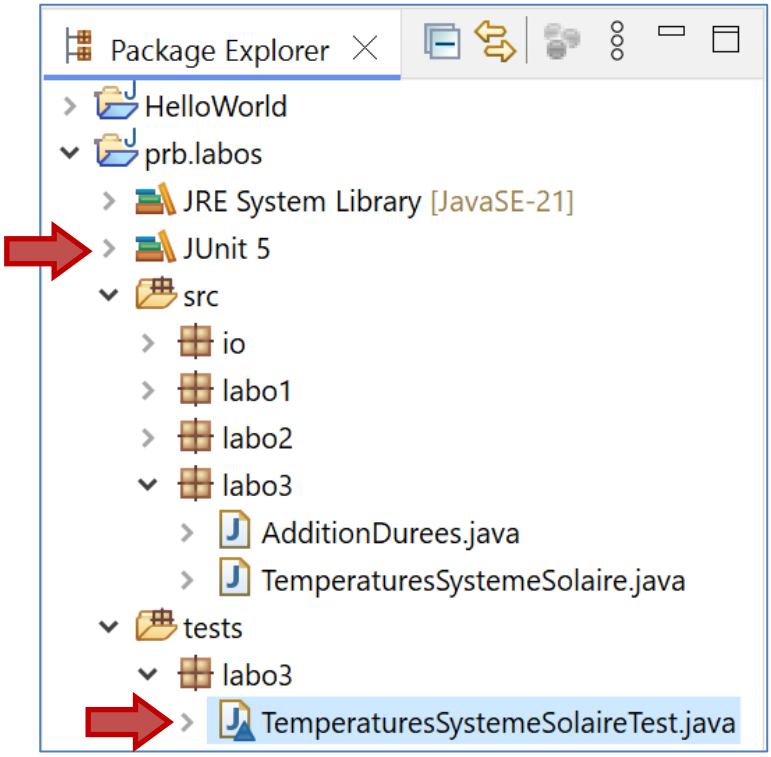
2. Ensuite, nous devons organiser notre projet afin d'y placer nos tests. En effet, les fichiers de test seront placés séparément des fichiers de nos programmes.

   Dans le ***Package Explorer***, faites un clic droit sur le projet **`prb.labos`** pour faire apparaître le menu contextuel, puis ajoutez un nouveau dossier nommé **`tests`** : via ***New > Source Folder***.

   Faites ensuite un clic droit sur le dossier **`tests`** que vous venez de créer, puis ajoutez un package nommé **`labo3`** : via ***New > Package***.

   Téléchargez la classe **`TemperaturesSystemeSolaireTest`**, puis placez-la dans le package **`labo3`** du dossier **`tests`**.

3. Vous obtenez alors la situation suivante :



REMARQUE

Si des erreurs sont renseignées dans la classe **`TemperaturesSystemeSolaireTest`**, cela provient très probablement de l'utilisation d'identifiants différents de ceux attendus :

- nom du package **`labo3`** ;
- nom de la classe **`TemperaturesSystemeSolaire`** ;
- noms des fonctions **`enCelsius`** et **`temperatureTheorique`**.

Le cas échéant, corrigez ces erreurs à l'aide de la fonctionnalité de renommage d'*Eclipse*.

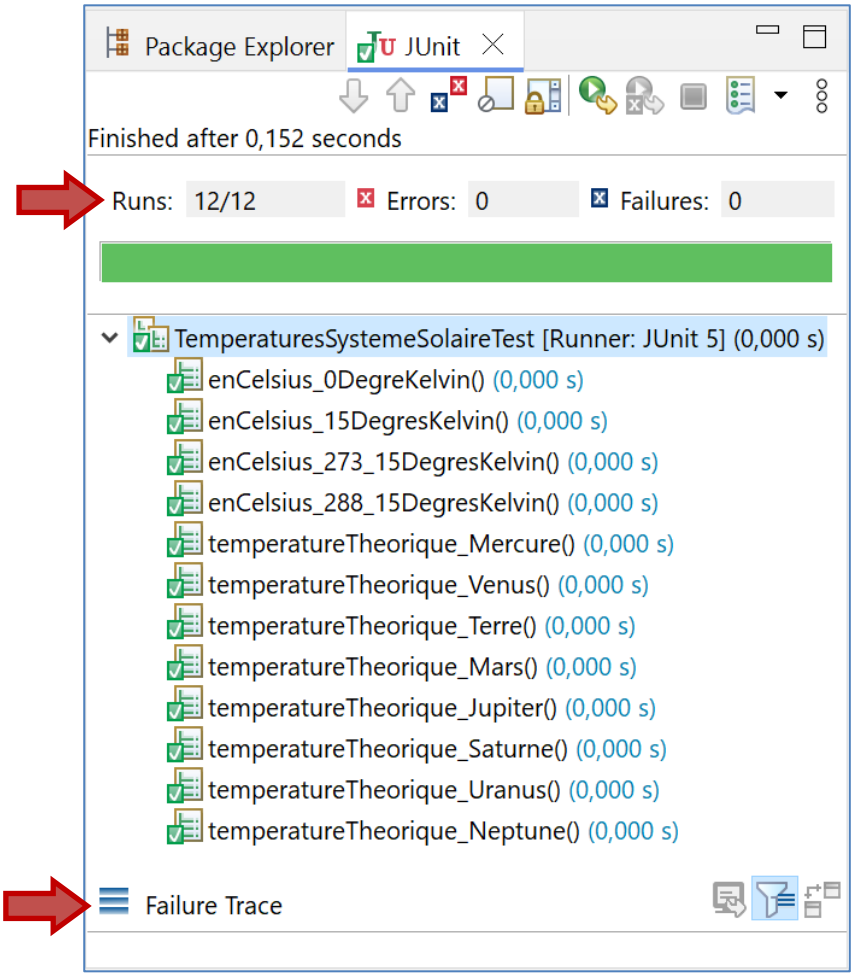
4. Il ne nous reste plus qu'à exécuter les tests pour vérifier la validité des deux fonctions.

   Voici deux méthodes possibles :

   a. Ouvrez en édition le fichier **`TemperaturesSystemeSolaireTest.java`**, puis exécutez les tests en cliquant sur le bouton ***Run***.

   b. Faites un clic droit sur le fichier **`TemperaturesSystemeSolaireTest.java`**, puis sélectionnez ***Run As > JUnit Test***.

   La sous-fenêtre suivante s'affiche avec les résultats :



   Voici comment comprendre les différentes informations affichées :

   - ***Runs*** indique le nombre de fonctions de test qui ont été exécutées (une fonction de test est une fonction précédée par l'annotation `@Test`).
   - ***Errors*** indique le nombre de fonctions de test qui ont déclenché une erreur (plus précisément, une **exception**) lors de leur exécution.
   - ***Failures*** indique le nombre de fonctions de test dont le résultat obtenu diffère de celui attendu.

IMPORTANT

En cas de test échoué, pensez à consultez systématiquement les informations plus détaillées qui sont affichées dans la partie inférieure ***Failure Trace*** !

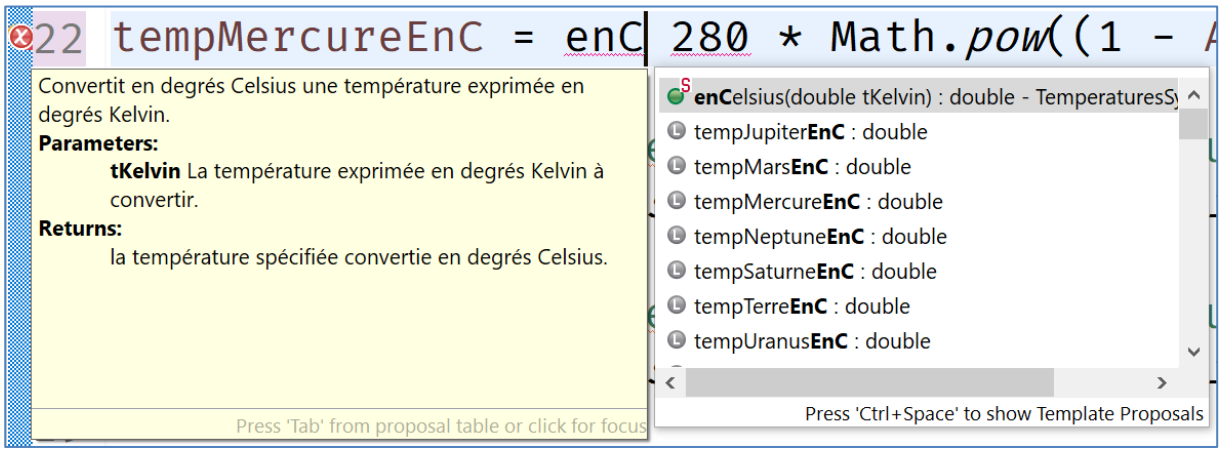
Sélectionnez au préalable le test échoué dans la partie supérieure pour obtenir les informations correspondantes.

UTILISER DES FONCTIONS

Dans la fonction `main` :

1. Effectuez des appels de la fonction `enCelsius` au lieu de soustraire 273,15.

   Profitez-en pour tester l'auto-complétion après avoir saisi les 3 caractères « `enC` » (pour que cela fonctionne, un espace doit être présent à leur suite).



   Exécutez votre programme afin de vérifier que les résultats n'ont pas changé.

2. Effectuez des appels de la fonction `temperatureTheorique` en lieu et place des calculs restants.

   Exécutez votre programme afin de vérifier que les résultats n'ont pas changé.

ASTUCE

Un moyen très pratique de naviguer dans le code consiste à **maintenir la touche Ctrl pressée** pendant que vous survolez avec le curseur de la souris les éléments du programme. Ces éléments (variables, constantes, classes, fonctions…) deviennent aussitôt cliquables.

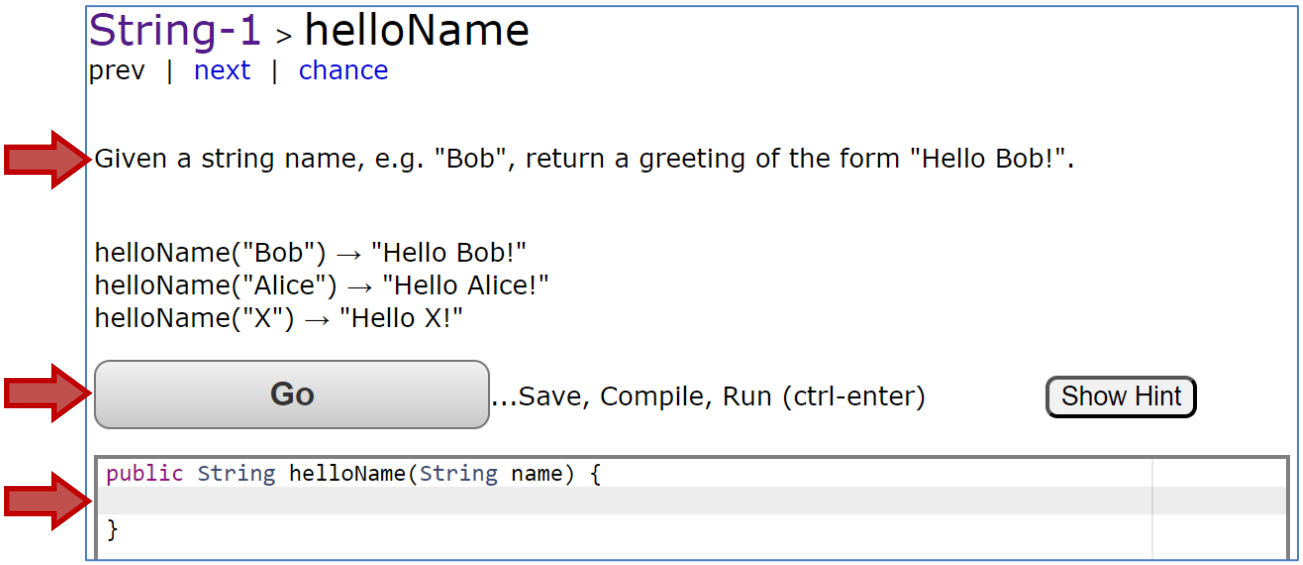
Essayez par exemple cette technique sur le nom de la fonction `enCelsius` dans le cas d'un appel de cette fonction dans le programme principal.

EXERCICE 3 [EN PARTIE À DOMICILE] : CODINGBAT

Dans cet exercice, vous devrez utilisez un nouvel outil appelé ***CodingBat*** (Remarque : cet outil n'est pas la propriété de *HELMo*). Il est disponible sur le site web : *https://codingbat.com/java*.

CodingBat vous propose de nombreux défis afin d'améliorer vos compétences dans différents registres de la programmation. Nous nous intéresserons dans un premier temps à la réalisation de fonctions manipulant des chaînes de caractères (**`String`**).

Un défi prend la forme d'une page web constituée d'un bref énoncé en anglais, d'un éditeur léger et d'un bouton ***Go*** permettant d'exécuter des tests sur le code réalisé :



IMPORTANT

Dans chaque défi, le code doit être réalisé uniquement dans le bloc d'instructions **`{ ... }`** de la fonction partiellement déclarée dans l'éditeur.

Il ne faut surtout pas ajouter une fonction **`main`** ou d'appel de la fonction présente ! Ne modifiez pas non plus l'en-tête de la fonction !

C'est *CodingBat* qui se charge d'exécuter la fonction lorsque vous cliquez sur le bouton ***Go***. Différents arguments sont alors transmis à la fonction afin de tester si les résultats retournés correspondent à ceux attendus, comme c'est le cas avec *JUnit*.

Les plus attentifs parmi vous remarquerez l'absence du qualificatif **`static`** dans l'en-tête de la fonction. Ne vous en souciez pas. Cela n'affectera en rien le code que vous devez réaliser.

PREMIER EXERCICE

Rendez-vous sur la page *http://codingbat.com/java/String-1*.

Cliquez ensuite sur le premier exercice nommé **helloName**.

Voici une traduction de l'énoncé : *« Étant donné un nom, par exemple "Bob", retournez un message d'accueil de la forme "Hello Bob!" »*.

La déclaration de fonction à compléter est :

```
public String helloName(String name) {

}
```

Vous devez ajouter les instructions permettant de créer une nouvelle chaîne de caractères à partir de celle reçue en paramètre, tout en respectant **précisément** les consignes, puis à la retourner.

Une fois votre code prêt, cliquez sur le bouton ***Go*** pour le tester. Répétez l'opération jusqu'à ce que tous les tests soient au vert !

IMPORTANT

Une fois l'outil ***CodingBat*** maîtrisé, il est recommandé de réaliser les exercices ci-dessous à domicile.

EXERCICES SUIVANTS

Réalisez au moins quatre exercices parmi ceux-ci :

1. **makeAbba** : *« Étant données deux chaînes, a et b, retournez une chaîne où a et b sont assemblées dans l'ordre abba, par exemple "Hi" et "Bye" produisent la chaîne "HiByeByeHi" ».*

2. **makeTags** : *« Étant données deux chaînes, tag et word, retournez une chaîne dans laquelle word est délimitée par des balises tag, par exemple "i" et "Yay" produisent la chaîne "Yay" ».*

3. **makeOutWord** : *« Étant données deux chaînes, out (une chaîne constituée de 4 caractères, tels que "<<>>") et word, retournez une chaîne dans laquelle word apparaît au milieu de out, par exemple "<<>>" et "Yay" produisent la chaîne "<<Yay>>" ».*

4. **extraEnd** : *« Étant donnée une chaîne str constituée d'au moins deux caractères, retournez une chaîne dans laquelle les deux derniers caractères de str sont recopiés trois fois, par exemple "Hello" produit la chaîne "lololo" ».*

5. **firstTwo** : *« Étant donnée une chaîne str, retournez une chaîne dans laquelle seuls les deux premiers caractères apparaissent, par exemple "Hello" produit la chaîne "He". Si str contient moins de deux caractères, retournez str telle quelle, par exemple "X" produit "X" et la chaîne "" produit "" (Astuce : utilisez la fonction `Math.min`) ».*

6. **firstHalf** : *« Étant donnée une chaîne str constituée d'un nombre pair de caractères, retournez la première moitié de la chaîne, par exemple "WooHoo" produit la chaîne "Woo" ».*

7. **withoutEnd** : *« Étant donnée une chaîne str constituée d'au moins deux caractères, retournez une chaîne identique à l'exception du premier et du dernier caractères qui ont été retirés, par exemple "Hello" produit la chaîne "ell" ».*

EXERCICE 4 : EMPRUNTS À TAUX D'INTÉRÊT FIXE

DESCRIPTION DU PROGRAMME

Un conseiller en matière d'emprunts hypothécaires souhaite mettre à la disposition de ses clients un programme informatique leur permettant de réaliser des simulations pour leurs emprunts à taux d'intérêt fixe.

Une personne contractant un emprunt de ce type doit payer des mensualités constantes pour rembourser le capital emprunté sur une durée déterminée.

Un taux d'intérêt est fixé au début de l'emprunt.

Pour calculer le montant des mensualités, il faut d'abord établir le **taux mensuel t_m** sur base du taux annuel t_a :

$$t_m = (1 + t_a)^{1/12} - 1$$

Le **montant des mensualités M** s'obtient alors sur base du capital emprunté C, du nombre de mensualités N et du taux d'intérêt mensuel t_m :

$$M = C \times \frac{t_m}{1 - (1 + t_m)^{-N}}$$

Par exemple, imaginons qu'un client souhaite emprunter 25.000,00€ à rembourser sur une durée de 20 ans avec un taux d'intérêt de 3,04% :

$$t_m = (1 + 0{,}0304)^{1/12} - 1 \approx 0{,}0024987063$$

Ceci équivaut donc à 0,24987%.

Puisque N vaut dans ce cas 240 (20 années fois 12 mois),

$$M = 25000 \times \frac{0{,}0024987063}{1 - (1 + 0{,}0024987063)^{-240}} \approx 138{,}63€$$

REMARQUE

Pour tester d'autres simulations pour un emprunt à taux d'intérêt fixe, rendez-vous sur le site : *http://www.guide-epargne.be/epargner/simulation-creditlogement.html*.

CLASSES EMPRUNT ET EMPRUNTTEST

1. Dans le package **`labo3`**, créez une classe nommée **`Emprunt`**.

> **REMARQUE**
>
> La classe **`Emprunt`** ne doit pas posséder de fonction **`main`**.

2. Dans la classe **`Emprunt`**, vous allez devoir déclarer deux fonctions nommées **`calculerTauxMensuel`** et **`calculerMensualite`** qui mettent en œuvre les formules présentées précédemment.

   Cependant, avant de coder ces fonctions, vous devez réaliser au préalable une phase préparatoire pour chacune d'elles afin de compléter les tableaux ci-dessous.

   Fonction **`calculerTauxMensuel`** :

Argument : *tauxAnnuel*	**Résultat attendu**
0,0304	0,0024987063…

   Fonction **`calculerMensualite`** :

Argument n°1 : *capital*	**Argument n°2 :** *nbMois*	**Argument n°3 :** *tauxMensuel*	**Résultat attendu**
25000	240	0,0024987063…	138,62997…

3. Avant de poursuivre, testez ces fonctions à l'aide de ***JUnit***.

   Pour ce faire, téléchargez le fichier **`EmpruntTest`**, puis placez-le dans le package **`labo3`** du dossier **`tests`**.

   Ce fichier comporte 9 fonctions de test, mais incomplètes. A ce stade, vous ne devez travailler qu'avec les **6 premières fonctions de test**. Les 3 autres fonctions de test sont prévues pour l'exercice 6.

   Renommez de manière explicite chacune de ces fonctions, décommentez leurs instructions, puis complétez ces dernières à l'aide des tableaux précédents.

4. Exécutez les tests et, le cas échéant, aidez-vous du débogueur pour trouver les causes des erreurs détectées.

ASTUCE DE DÉBOGAGE

Lorsqu'un test échoue, il est possible d'isoler ce dernier des autres pour effectuer le débogage.

Pour ce faire, placez un **point d'arrêt dans la fonction testée**.

Cliquez ensuite sur le nom de la fonction de test correspondante :

```
19    @Test
20    @Order(1)
21    void calculerTauxMensuel_test1() {
22 //      assertEquals(?, Emprunt.calculerTauxMensuel(?), MARGE_EF
23    }
```

Enfin, exécutez cette fonction de test en mode Débogage : via le bouton  ou via un clic droit sur son nom, puis ***Run As*** > ***JUnit Test***.

CLASSE SIMULATIONEMPRUNT

1. Dans le package **`labo3`**, créez une classe nommée **`SimulationEmprunt`** comportant une fonction **`main`**.

2. Dans la fonction **`main`**, réalisez un programme permettant de simuler un emprunt s'étalant sur deux durées différentes (voir l'exemple d'exécution ci-dessous). Pour ce faire, utilisez les fonctions de la classe **`Emprunt`**.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu.

```
Capital ? 25000
Taux annuel en % ? 3.04
Durée 1 en années ? 15
Durée 2 en années ? 20

Capital à emprunter    = 25000,00 EUR
Taux d'intérêt mensuel = 0,24987%

Sur 15 ans :
Nombre de paiements    = 180
Remboursement mensuel =    172,63 EUR
Total remboursé       =  31072,81 EUR
Intérêts prêt         =   6072,81 EUR
```

```
Sur 20 ans :
Nombre de paiements   = 240
Remboursement mensuel =    138,63 EUR
Total remboursé       =  33271,19 EUR
Intérêts prêt         =   8271,19 EUR
```

AMÉLIORATION POSSIBLE

L'affichage des deux simulations mène à coder des instructions identiques à l'exception des cinq valeurs affichées : la durée en années, le nombre de paiements, le montant des mensualités, le total remboursé et le montant des intérêts.

1. Dans la classe **`SimulationEmprunt`**, déclarez une fonction nommée **`afficherSimulationEmprunt`** qui permet d'afficher les informations d'une simulation d'emprunt.

   Afin de limiter l'usage de cette fonction à cette classe, utilisez le mot-clé **`private`**.

2. Dans la fonction **`main`**, remplacez l'affichage des deux simulations par deux appels de la fonction **`afficherSimulationEmprunt`**.

EXERCICE 5 : AMÉLIORATION D'UN PROGRAMME DU LABO 2

HEURES PRESTÉES

Le programme **HeuresPrestees** réalisé dans le cadre du labo 2 comporte de base de nombreuses répétitions dans son traitement. Maintenant que vous êtes familier avec les fonctions, vous pouvez les utiliser à bon escient afin d'améliorer cet ancien programme.

CONSIGNES

1. Copiez le fichier **HeuresPrestees.java** du labo 2 dans le package du labo 3.
2. Cette fois, créez par vous-même une classe nommée **HeuresPresteesTest** destinée aux tests unitaires des fonctions que vous réaliserez dans le cadre de cet exercice.

   Faites un clic droit sur la package **labo3** du dossier **tests**, puis ajoutez une classe de tests unitaires : via ***New > JUnit Test Case***.

   > **REMARQUE**
   >
   > Si vous ne trouvez pas cette option, utilisez cet autre chemin :
   >
   > ***New > Other… > Java > JUnit > JUnit Test Case***.

3. A chaque fois que vous créez une fonction, réalisez au moins 3 fonctions de test pour cette dernière dans la classe **HeuresPresteesTest**.

VERSION 1

1. Dans la classe **HeuresPrestees**, déclarez une fonction nommée **lireHeures** capable d'extraire d'une chaîne de caractères la valeur des heures sous la forme d'un entier :

   ```
   public static int lireHeures(String hhmm)
   ```

   Par exemple, si `hhmm` correspond à `"7:15"`, alors la fonction retourne l'entier 7.

2. Dans la classe **HeuresPrestees**, déclarez une fonction nommée **lireMinutes** capable d'extraire d'une chaîne de caractères la valeur des minutes sous la forme d'un entier :

   ```
   public static int lireMinutes(String hhmm)
   ```

   Par exemple, si `hhmm` correspond à `"7:15"`, alors la fonction retourne l'entier 15.

3. Testez vos fonctions à l'aide de ***JUnit***.
4. Lorsque les tests réussissent, remplacez tous les traitements correspondants présents dans la fonction **main** par des appels de ces nouvelles fonctions, puis vérifiez que le programme affiche toujours les résultats adéquats.

VERSION 2

1. Toujours dans la classe **`HeuresPrestees`**, déclarez une fonction nommée **`convertirEnMinutes`** capable d'exprimer la durée spécifiée en paramètre sous la forme d'un entier exprimant cette même durée en minutes :

   ```
   public static int enMinutes(String hhmm)
   ```

   Dans cette fonction, effectuez un traitement permettant.

   Par exemple, si `hhmm` correspond à `"7:15"`, alors la fonction retourne l'entier `435` (puisque 7 × 60 + 15 donne 435).

   > **IMPORTANT**
   >
   > Pour effectuer ce traitement, utilisez les fonctions **`lireHeures`** et **`lireMinutes`** précédemment déclarées.

2. Testez votre fonction à l'aide de ***JUnit***.

3. Lorsque les tests réussissent, remplacez tous les traitements correspondants présents dans la fonction **`main`** par des appels de cette nouvelle fonction, puis vérifiez que le programme affiche toujours les résultats adéquats.

EXERCICE 6 [FACULTATIF] : DÉTAIL D'UN EMPRUNT

DESCRIPTION DU PROGRAMME

Le programme de l'exercice 4 (classes **`Emprunt`** et **`SimulationEmprunt`**) doit maintenant permettre aux clients de consulter le détail de la formule d'emprunt qui les intéresse.

Ce détail reprend les mêmes informations que précédemment (dans un ordre légèrement remanié) et ajoute ensuite les bilans des trois premières années (voir l'exemple d'exécution de la page suivante).

Chacun de ces bilans indique :

1. le solde du capital emprunté
2. la part du capital emprunté qui a été remboursée sur l'année
3. la part des intérêts qui a été remboursée sur l'année

Pour y parvenir, il nous faut connaître le solde du capital, autrement dit la part du capital emprunté qui est encore à rembourser.

En effet, à chaque mensualité versée, une part est dédiée au remboursement du capital emprunté et l'autre part au remboursement des intérêts. Cette dernière part décroît de mois en mois au profit du remboursement du capital.

Le **solde du capital** S peut s'obtenir sur base du montant des mensualités M, du nombre de mensualités restantes N et du taux d'intérêt mensuel t_m :

$$S = M \times \frac{1 - (1 + t_m)^{-N}}{t_m}$$

Par exemple, pour un emprunt de 25.000,00€ à rembourser sur une durée de 20 ans avec un taux d'intérêt de 3,04%, le solde après un an est :

$$S = 138{,}63 \times \frac{1 - (1 + 0{,}0024987063)^{-228}}{0{,}0024987063} \approx 24073{,}39€$$

Étant donné que 1663,56€ (autrement dit, 12 fois 138,63€) sont versés durant l'année, il est facile de calculer ensuite les parts du capital et des intérêts remboursées.

CLASSES EMPRUNT ET EMPRUNTTEST

1. Dans la classe **`Emprunt`**, déclarez une fonction nommée **`calculerSoldeCapital`** qui met en œuvre la formule donnée précédemment.

   Cependant, avant de coder cette fonction, vous devez réaliser au préalable une phase préparatoire afin de compléter le tableau ci-dessous.

   Fonction **`calculerSoldeCapital`** :

Argument n°1 : *mensualite*	**Argument n°2 :** *nbMoisRestants*	**Argument n°3 :** *tauxMensuel*	**Résultat attendu**
138,62997…	228	0,0024987063…	24073,3865288…

2. Via les 3 dernières fonctions de tests encore non utilisées de la classe **`EmpruntTest`**, testez la fonction **`calculerSoldeCapital`**.

   Renommez de manière explicite chacune de ces fonctions, décommentez leurs instructions, puis complétez ces dernières à l'aide du tableau précédent.

CLASSE DETAILEMPRUNT

1. Dans le package **`labo3`**, créez une classe nommée **`DetailEmprunt`** comportant une fonction **`main`**.

2. Dans la fonction **`main`**, réalisez un programme permettant d'obtenir le détail pour une formule d'emprunt. Pour ce faire, utilisez les fonctions de la classe **`Emprunt`**.

EXEMPLE D'EXÉCUTION

Vous devez respecter la mise en forme proposée dans l'exemple ci-dessous. Les **données saisies par l'utilisateur** sont indiquées en bleu.

```
Capital ? 25000
Taux annuel en % ? 3.04
Durée en années ? 20

Taux d'intérêt mensuel = 0,24987%
Nombre de paiements    = 240
Remboursement mensuel  = 138,63 EUR

Capital à emprunter =  25000,00 EUR
Total remboursé     =  33271,19 EUR
```

```
Intérêts prêt       =   8271,19 EUR

Année 1 :
Solde du capital    =  24073,39 EUR
Capital remboursé   =    926,61 EUR
Intérêts remboursés =    736,95 EUR

Année 2 :
Solde du capital    =  23118,60 EUR
Capital remboursé   =    954,78 EUR
Intérêts remboursés =    708,78 EUR

Année 3 :
Solde du capital    =  22134,80 EUR
Capital remboursé   =    983,81 EUR
Intérêts remboursés =    679,75 EUR
```

AMÉLIORATION

Comme dans l'exercice précédent, l'affichage des trois bilans annuels mène à coder des instructions identiques.

1. Dans la classe **`DetailEmprunt`**, déclarez une fonction nommée **`afficherBilanAnnee`** qui permet d'afficher les informations d'un bilan annuel.

   Afin de limiter l'usage de cette fonction à cette classe, utilisez le mot-clé **`private`**.

2. Dans la fonction **`main`**, remplacez l'affichage des trois bilans annuels par trois appels de la fonction **`afficherBilanAnnee`**.