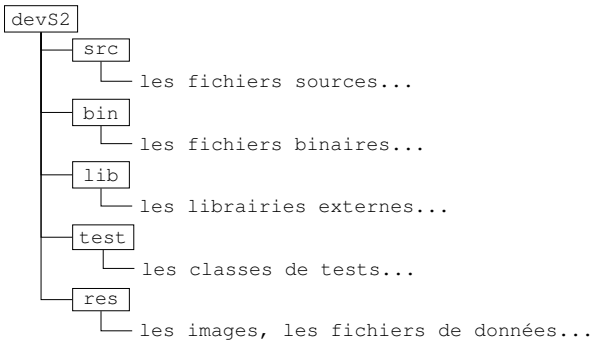
Exercice 1 : Prise en main de l'environnement de développement

Le but de cet exercice est de prendre en main votre outil de développement. Nous conseillons fortement *VSCodium* et de compiler dans le terminal intégré durant au moins les 4 premières semaines.

Q1. Lancez votre outil de développement et définissez un répertoire `devS2` pour votre projet java. Tous les développements futurs devront être dans ce même répertoire.

Q2. Votre projet doit respecter l'arborescence suivante :



Q3. Créez un répertoire `tp01` dans `src` et ajoutez-y une classe `Book` qui doit contenir **exactement** le code suivant :

```java
class Book {
  // class attributes
  String author;
  String title;
  int year;
  // constructor
  Book(String author, String title, int year) {
    this.author = author;
    this.title = title;
    this.year = year;
  }
  // methods
  String getAuthor() {
    return this.author;
  }
  String getTitle() {
    return this.title;
  }
  String print() {
    return author + "\t" + title + "\t" + year;
  }
```

Q4. À partir du répertoire `tp01`, compilez ce programme au moyen de la commande:

```
javac Book.java
```

Normalement aucune erreur de compilation n'est observée

Quand vient le temps de tester le code que l'on vient d'écrire, plusieurs solutions s'offre à nous. La plus directe est d'ajouter une méthode `main` dans la classe que l'on souhaite tester. On rappelle qu'une méthode principale doit nécessairement respecter la signature suivante :

```java
public static void main(String[] args)
```

Q5. Ajoutez à votre classe `Book` une méthode principale qui :

— crée un livre complètement spécifiée (où l'on fournit tous les champs) ;

— affiche les informations par accès direct aux attributs (pas d'invocation de méthode) pour obtenir un message comme :

```
Edwin A. Abbott a écrit Flatland en 1884.
```

L'autre solution est d'externaliser le code du scénario d'utilisation en écrivant une classe spécifiquement dédiée à cela (qui ne contiendrait dans l'absolu qu'un programme principal).

Q6. Créez cette fois-ci une classe `UseBook`, contenant uniquement une méthode `main` qui :
— crée une bibliothèque `biblio` (représentée sous forme d'un tableau défini en extension) ;
— remplit ce tableau avec un minimum de trois ouvrages (au choix) ;
— affiche de manière intelligible le contenu de la bibliothèque.

Exercice 2 : Programme principal paramétrique 👍

Q1. Créez une classe contenant le code suivant :

```
class Parameter {
  public static void main(String[] args) {
    if(args.length==0) {
      System.out.println("No parameter");
    }
    for(int i=0; i<args.length; i++) {
      System.out.println("(" + (i+1) + ") " + args[i]);
    }
  }
}
```

Q2. Compilez et exécutez ce code. Quel affichage obtenez-vous ?

Q3. Ré-exécutez ce code en passant cette fois en paramètre au programme principal `main` les trois valeurs `Java C Php` ?

Q4. Écrivez une classe `RandomSequence` munie d'un programme principal `main` qui peut prendre deux ou trois paramètres :
— le nombre d'éléments à générer
— la valeur maximale pouvant être générée
— éventuellement, le type de valeur étant générée (`INTEGER` ou `REAL`).

Si le troisième paramètre n'est pas fourni, le programme doit générer des nombres entiers. Si le nombre ou la valeur des paramètres diffère de ce qui est attendu, le programme doit afficher le message suivant :

`"Correct usage : <nbElt> <maxVal> [INTEGER|REAL]"`

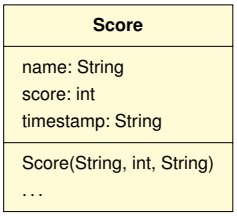
Notons que la transformation d'une chaîne de caractères en entier ou en double se fait via l'instruction suivante :

```
int integerValue = Integer.parseInt("1");
System.out.println(integerValue + 1);
double realValue = Double.parseDouble("0.42");
System.out.println(realValue + 1.0);
```

Exercice 3 : Calcul de Score 👍

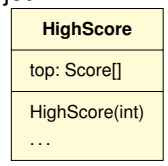
On veut créer un système de gestion des scores, qui ne garde en mémoire qu'un nombre limité des meilleurs scores.

Q1. Créez une classe `Score`, répondant aux spécifications UML suivantes:



La méthode `toString()` doit retourner une score sous la forme suivante: `(<timestamp>) <name> = <score>`.

Q2. Écrivez maintenant une classe `HighScore` et son constructeur. Cette classe est caractérisée par un tableau de `Score` dont la taille est définie à l'instanciation de l'objet.



Q3. Ajoutez une méthode `ajout(...)` qui ajoute un score donné au tableau s'il y a une place de libre, ou si son score est suffisant. Le tableau des scores doit toujours rester trié, le score le plus élevé restant dans la première case du tableau. La méthode retourne `true` si le score a été ajouté, `false` sinon.

```
boolean ajout(Score newScore)
```

Q4. Ajoutez une méthode `toString()` qui met en forme textuelle l'objet à raison d'un score par ligne. Par exemple:

```
TOP SCORE:
(29/01) Bob = 800
(31/01) Alice = 650
(29/01) Alice = 300
```

Q5. Créez une classe `UseHighScore` contenant uniquement une méthode principale où l'on joue le scénario suivant:

— création d'un `HighScore` de taille 3
— ajout de Alice avec un score de 300 le 29/01
— ajout de Bob avec un score de 800 le 29/01
— ajout de Alice avec un score de 42 le 30/01
— ajout de Alice avec un score de 650 le 31/01

Prenez soin d'afficher le tableau des scores entre chaque étape. Vous devriez obtenir le tableau décrit à la question précédente.

Exercice 4 : Structure irrégulière

On souhaite pouvoir disposer d'une structure basée sur un tableau à deux dimensions, mais dont toutes les lignes ne contiennent pas le même nombre d'éléments. Dans l'exemple suivant, les lignes ont successivement des tailles qui varient : la ligne d'indice 0 contient 3 éléments, la ligne d'indice 1 ne porte qu'un seul élément...

3	9	4
6		
2	9	
7	2	1

Ce type de structure est définie sur-mesure et ne doit donc avoir aucune case vide. L'invocation de la méthode `length` donne un résultat différent selon la ligne sur laquelle est invoquée la méthode.

Q1. Écrivez une classe `Irregular` permettant de créer une telle structure, ainsi que son constructeur répondant à la signature suivante. Le constructeur doit donc réserver l'espace mémoire ligne par ligne.

```
Irregular(int[] lineSize)
```

La taille de chacune des lignes est spécifiée dans le paramètre `lineSize`. Un paramètre valant `[1, 3, 2]` mènera à une structure dont la première ligne n'a qu'un élément, la seconde ligne 3 éléments alors qu'il y aura 2 éléments sur la dernière ligne.

Q2. Écrivez une méthode de remplissage aléatoire au sein de cette classe, de signature :

```
void randomFilling()
```

Q3. Écrivez une méthode générant la représentation textuelle d'une telle structure, afin d'en préparer l'affichage, de signature :

```
String display()
```

Q4. Écrivez une classe `UseIrregular` permettant la vérification du code par l'exécution d'un scénario exemple.

Q5. Ajoutez une méthode `isCommon` permettant de déterminer si un élément donné en paramètre est présent dans chacune des lignes la structure :

```
boolean isCommun(int element)
```

Q6. Écrivez une méthode `existCommon` permettant de déterminer l'existence ou non d'un élément commun à toutes les lignes :

```
boolean existCommon()
```

Q7. Modifiez le scénario de la classe `UseIrregular` pour permettre la validation du code.