

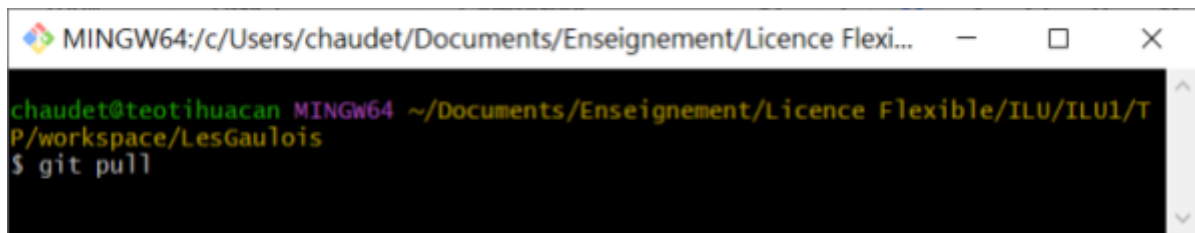
Introduction au développement Java en ILU1

1 ➡ Environnement GitHub

Mettre à jour votre dépôt local avec le contenu du dépôt distant.

Sous l'environnement windows en salle de TPs :

- Avec **un explorateur de fichier** se placer dans le répertoire du projet 'LesGaulois' qui se trouve sous votre workspace (vous devez y trouver le fichier `.gitignore`), puis cliquer droit (sans rien sélectionner) et cliquer sur 'Git Bash Here'.
- Dans Git Bash (ouvert directement dans le répertoire du projet 'LesGaulois'), tapez la commande `'git pull'`.



```
MINGW64:/c/Users/chaudet/Documents/Enseignement/Licence Flexi...
chaudet@teotihuacan MINGW64 ~/Documents/Enseignement/Licence Flexible/ILU/ILU1/TP/workspace/LesGaulois
$ git pull
```

- Ouvrir Eclipse au niveau du workspace et non du projet LesGaulois.
Exemple : Z:/ILU1/TPs/workspace
- Une fois l'IDE ouvert, au niveau de du Package Explorer, cliquer droit sur le projet 'LesGaulois' puis sélectionnez *Refresh*

➡ Les tableaux - première approche

Sous le même workspace contenant le projet "LesGaulois", créer un nouveau projet "BacAsable" (non relié à git). Créer le package "tableaux" contenant la classe "TestTableau". Ajouter un main dans cette classe.

2.1 Déclaration de tableau

Déclarer un tableau d'entier : `int[] tableauEntiers;`

De même déclarer un tableau `tableauChaines` contenant des chaînes de caractères.

Essayer d'afficher le premier tableau : `System.out.println(tableauEntiers);`

Un message d'erreur apparaît : le tableau a été déclaré mais n'a pas été créé !

2.2 Création du tableau

Vous avez deux façon de créer un tableau : en donnant directement les éléments à insérer ou en donnant le nombre de cases que contient le tableau.

```
int[] tableauEntiers = { 5, 7, 2, 5, 3 };
```

```
String[] tableauChaines = new String[5];
```

Dans le premier tableau les cases sont déjà remplies. Dans le deuxième tableau les cases sont vides.

Essayer d'afficher le premier tableau : `System.out.println(tableauEntiers);`

Plus de message d'erreur, mais le résultat n'est pas forcément celui que vous attendiez.

L'affichage par défaut d'un tableau est :

- `[I` → indique que c'est un tableau (`[]` d'entiers (`I` pour `int`)).
- `@24d46ca6` → sorte d'identifiant unique basé sur l'adresse mémoire de l'objet¹.

En gros l'affichage d'un tableau est son type et son identifiant comme la méthode `toString()` de la classe `Objet` vu dans le TP précédent.

2.3 Affichage d'un tableau

Il y a plusieurs façons de parcourir un tableau suivant que le tableau soit plein, rempli successivement à partir de la case d'indice 0 ou rempli sans ordre spécifique.

a - Parcours d'un tableau plein

Parcourons le premier tableau : `tableauEntiers`.

¹ C'est le **hashcode** en hexadécimal

On peut le faire de deux manières différentes.

La première, et la plus simple, est d'utiliser le foreach (ctrl + espace -> sélectionner foreach). Utiliser la tabulation pour choisir le tableau à parcourir.

```
for (int entier : tableauEntiers) {  
    System.out.print(entier + " ");  
}
```

Affichage : 5 7 2 5 3

La deuxième est d'utiliser "for - use index on array with temporary variable", cela permet de gérer selon l'indice en cours. Toutes les cases seront affichées : `tableauEntiers.length`

```
for (int i = 0; i < tableauEntiers.length; i++) {  
    int entier = tableauEntiers[i];  
    System.out.print(entier);  
    if (i < tableauEntiers.length - 1) {  
        System.out.print(", ");  
    }  
}
```

Affichage : 5, 7, 2, 5, 3

Essayer d'afficher le deuxième tableau de la même manière.

```
for (String chaine : tableauChaines) {  
    System.out.print(chaine + " ");  
}
```

Affichage : null null null null null

C'est normal le tableau `tableauChaines` a été déclaré (en tant que tableau contenant des chaînes), il a été créé (comme un tableau contenant 5 cases), mais n'a pas été rempli.

b - Parcours d'un tableau rempli successivement à partir de la case d'indice 0

Lorsque l'on remplit un tableau, il faut lui associer un entier permettant de connaître quelles sont les cases qui ont été remplies.

```
String[] tableauChaines = new String[5];  
int nbChaines = 0;
```

Ajouter les chaînes "Hello" "World"

```
tableauChaines[nbChaines] = "Hello";  
nbChaines++;  
tableauChaines[nbChaines] = "World";  
nbChaines++;
```

L'affichage peut se faire avec "for - use index on array with temporary variable", comme précédemment, mais on peut se passer de la variable temporaire. Donc utiliser "for - use index on array". Dans les deux boucles cela permet de s'arrêter au nombre de cases qui ont été réellement remplies `nbChaines`.

```
for (int i = 0; i < nbChaines; i++) {  
    System.out.print(tableauChaines[i] + " ");  
}
```

Affichage: Hello World

c - Parcours d'un tableau rempli sans ordre spécifique

Dans ce type de parcours, la variable `nbChaines` ne sera pas utilisée étant donné que l'on ne connaît pas où sont les objets dans le tableau.

Ajouter la chaîne " !" dans la 4ème case du tableau `tableauChaines`.

```
tableauChaines[3] = " !";
```

Le parcours peut se faire avec le `foreach` ou avec le "for - use index on array". Ce qui est important c'est d'ajouter une condition permettant de vérifier que l'objet dans la case est différent de `null`.

```
for (String chaine : tableauChaines) {  
    if (chaine != null) {  
        System.out.print(chaine + " ");  
    }  
}
```

Affichage: Hello World !

Fermer le projet "BacAsable": clique droit sur le projet puis `close project`.

Résumé

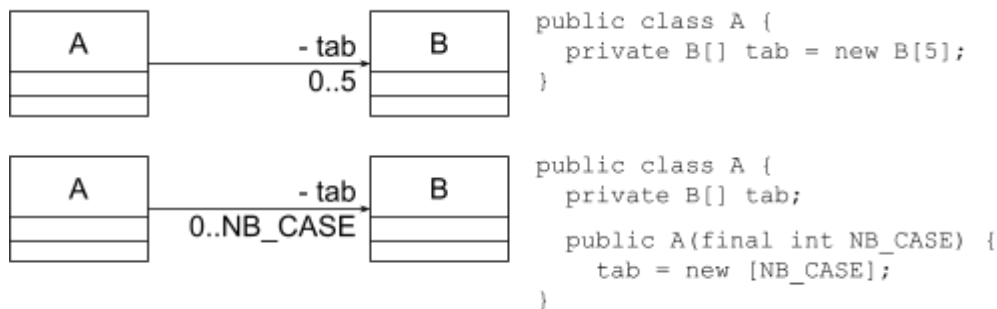
1. **Déclarer** : donner le type des éléments contenus dans le tableau.
2. **Créer** : donner le nombre de cases pour créer le tableau.
3. **Remplir** : soit entièrement dès sa création en donnant les éléments entre accolades ou avec une boucle ; soit de manière incrémentale (on utilisera une variable permettant de connaître la dernière case vide du tableau) ; soit sans ordre spécifique.

3 Le village des Gaulois

Reprenons le projet `LesGaulois` afin de mettre en pratique ce que l'on vient de voir sur les tableaux.

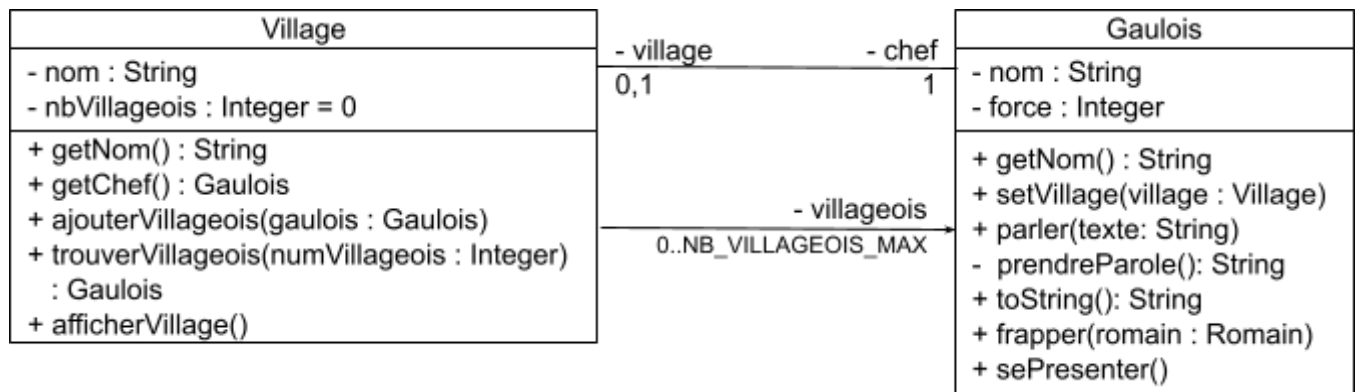
1. Les tableaux - Le village des gaulois

Rappel : UML et les tableaux



Les tableaux

- a. Dans le paquetage personnages ajouter la classe `Village` en suivant le diagramme UML ci-dessous.



Focus sur la multiplicité 1 : le diagramme de classe spécifie que le village a 1 chef de type Gaulois. Le fait que la multiplicité n'est pas 0,1 mais 1 signifie que :

- le chef doit être initialisé dès la création de l'objet,
- le chef ne doit jamais être à null durant toute la vie de l'objet.

La propriété "le village a exactement un chef" :

- est un invariant du système qui se traduit par un invariant de la classe village,
- s'exprime en UML par une multiplicité de 1.

Attention : dans ce TP le chef ne fait pas partie des villageois.

- b. Écrire la méthode `ajouterVillageois` qui prend en paramètre d'entrée un gaulois et le place dans le tableau `villageois`. N'oubliez pas de mettre à jour le village du gaulois.
- c. Écrire la méthode `trouverVillageois` qui prend en paramètre d'entrée le numéro du villageois dont on veut récupérer la référence. Attention le gaulois numéro 1 se trouve dans case d'indice 0 du tableau. La méthode vérifie que le numéro entré correspond bien à une case occupée par un gaulois avant de le retourner. Si le numéro n'est pas cohérent, il faudra afficher ("Il n'y a pas autant d'habitants dans notre village !") et retournera `null`.

d. Un main dans la classe Village :

- Créer le gaulois Abraracourcix qui a une force de 6 et l'ajouter comme chef du village.
- écrire un main dans lequel vous créez un Village `village` dont le nom est "Village des Irréductibles", le chef Abraracourcix et qui possède au maximum 30 habitants.
- Vérifier qu'aucun message de levée d'exception n'apparaît si vous écrivez l'instruction :

```
Gaulois gaulois = village.trouverVillageois(30);
```

- Créer le gaulois Astérix qui a une force de 8, et l'ajouter au village.
- Ecrire les instructions :

```
Gaulois gaulois = village.trouverVillageois(1);
```

```
System.out.println(gaulois);
```

```
gaulois = village.trouverVillageois(2);
```

```
System.out.println(gaulois);
```

Exemple de sortie attendue :

```
Astérix
```

```
Il n'y a pas autant d'habitants dans notre village !
```

```
null
```

e. Toujours dans la classe Village :

- Écrire la méthode `afficherVillageois` qui affiche le chef puis les noms de tous les villageois en évitant les messages d'erreur !

Exemple de sortie attendues :

```
Dans le village "Village des Irréductibles" du chef Abraracourcix
vivent les légendaires gaulois :
```

```
- Astérix
```

f. Dans la classe Gaulois écrire la méthode `sePresenter` qui affiche, par exemple, selon le cas :

- s'il s'agit du chef du village :

Le Gaulois Abraracourcix : "Bonjour, je m'appelle Abraracourcix. Je suis le chef le village Village des Irreductibles."

- o s'il s'agit d'un habitant du village :

Le Gaulois Astérix : "Bonjour, je m'appelle Astérix. J'habite le village Village des Irreductibles."

- o s'il s'agit d'un habitant sans village :

Le Gaulois Doublepolémix : "Bonjour, je m'appelle Doublepolémix. Je voyage de villages en villages."

- g. A nouveau dans le main de la classe Village :

- o Créer le gaulois Obélix qui a une force de 25, et l'ajouter au village.
- o Appeler la méthode `afficherVillageois` pour contrôler qu'Obélix fait bien partie des villageois affichés.
- o Créer le gaulois DoublePolémix qui à une force de 4.
- o Appeler la méthode `sePresenter` sur les objets *abraracourcix*, *asterix* et *doublePolemix*.

- h. Sauvegarder votre travail sous GitHub (cf p1 - Historisation périodique de votre projet - exemple de message pour le commit "le village gaulois")

2. Les pré, post conditions et les invariants - La force des romains

Ci-dessous l'extrait de la classe `Romain` écrite dans le TP précédent.

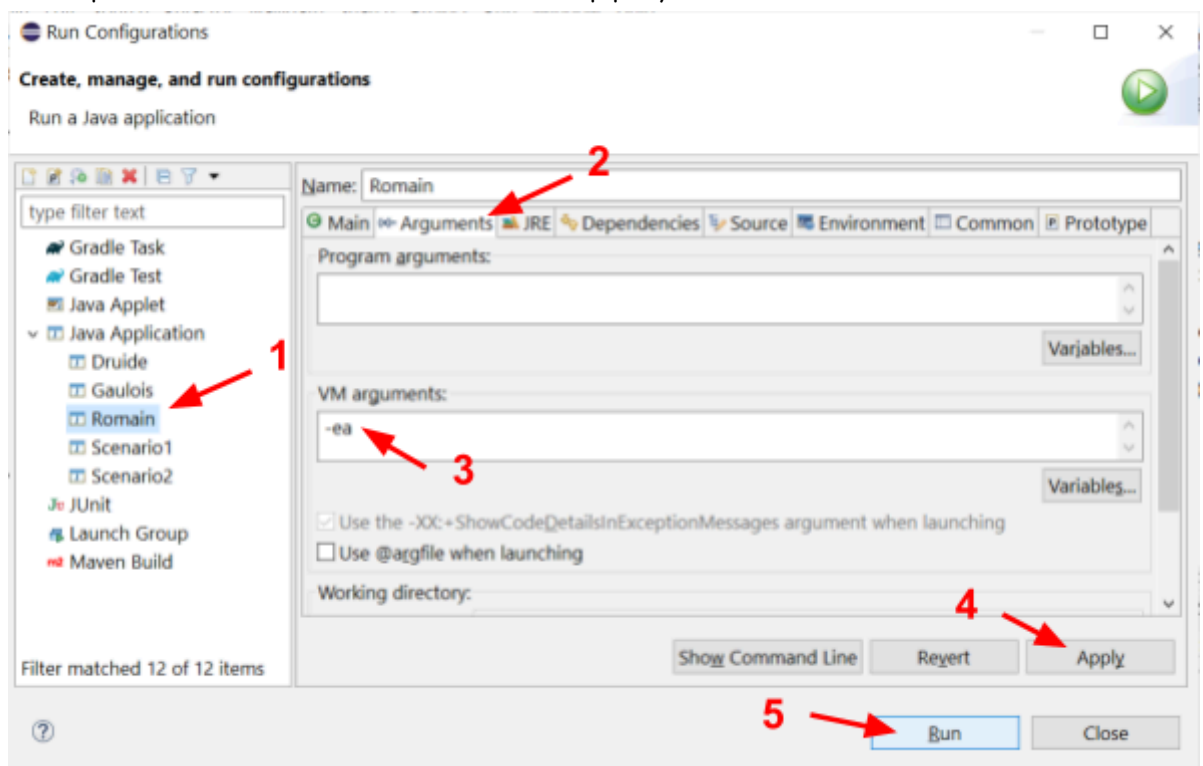
```
public class Romain {
    private String nom;
    private int force;

    public Romain (String nom, int force) {
        this.nom = nom;
        this.force = force;
    }

    public void recevoirCoup(int forceCoup) {
        force -= forceCoup;
        if (force < 1) {
            force = 0;
            parler("J'abandonne !");
        } else {
            parler("Aïe");
        }
    }
}
```

a. Travail sur l'invariant :

- Vérifier l'invariant "la force d'un Romain est toujours positive" à la fin de la création d'un objet et à la fin de chaque méthode qui modifie sa force. Pour cela vous pouvez utiliser une méthode privée `isInvariantVerified` qui retourne `true` si la force du romain est positive ou égale à 0.
- Créer un main dans la classe Romain en créant Minus avec une force -6.
- Tester. Rien ne se passe ? Rassurez-vous c'est normal : il faut activer la vérification des assertions. Cliquer sur l'onglet Run puis sélectionner Run Configurations... (voir la figure ci-dessous)
 1. sous Java Application sélectionner la classe où se trouve votre main (dans notre exemple la classe Romain).
 2. appuyer sur l'onglet "Arguments"
 3. Dans la zone de texte "VM arguments" écrire -ea (pour enable assertions)
 4. Appuyer sur le bouton "Apply"
 5. Vous pouvez lancer l'exécution en appuyant sur le bouton "Run"



- Vous devez obtenir une `java.lang.AssertionError`
 - Remettez la valeur de la force de Minus à 6.
- b. Travail sur les pré et post conditions
- Vérifier la précondition "la force du coup reçu est positive" au début de la méthode `recevoirCoup`

- Vérifier la postcondition "la force d'un Romain a diminué" à la fin de la méthode `recevoirCoup`. N'hésitez pas à ajouter des variables locales au besoin.
- c. Sauvegarder votre travail sous GitHub (cf p1 - Historisation périodique de votre projet - exemple de message pour le commit "invariant pre post conditions")

3. Les énumérations - Des Romains bien protégés

Les énumérés

- a. Dans le package « objets » créer l'enum `Equipement` possédant deux énumérés : `CASQUE` et `BOUCLIER` (clic droit sur le package puis New > Enum).

Tester votre énumération dans le main de la classe `Romain` en faisant afficher `CASQUE` et `BOUCLIER`.

- b. Modifier l'énumération `Equipement` en lui ajoutant :
 - un attribut `nom`,
 - un constructeur prenant en paramètre d'entrée une chaîne qui permettra d'initialiser la valeur de l'attribut `nom`.
 - Par conséquent, vous devrez ajouter cette chaîne pour chacun des énumérés par exemple `CASQUE("casque")` et `BOUCLIER("bouclier")`.
 - Ajouter la méthode `toString` qui affichera le nom de l'énuméré.
- c. Dans la classe `Romain` créer deux attributs :
 - un tableau `equipements` contenant des objets de type `Equipement` de deux cases,
 - un entier `nbEquipement` initialisé à 0.
- d. Veuillez lire l'ensemble de la question avant de commencer à la coder.

Dans la classe `Romain` créer la méthode `sEquiper` qui prend en paramètre un équipement.

Si le romain a déjà deux équipements la méthode affichera : « Le soldat », son nom et « est déjà bien protégé ! ». Par exemple :

Le soldat Minus est déjà bien protégé !

Si le soldat possède déjà un équipement alors il faut regarder dans la première case du tableau s'il s'agit du même équipement que celui donné en paramètre d'entrée. La méthode affichera : « Le soldat », son nom et « possède déjà », le nom de l'équipement et un point d'exclamation. Par exemple :

Le soldat Minus possède déjà un casque !

Dans les autres cas, l'équipement est ajouté au tableau et le nombre d'équipements est incrémenté. La méthode affichera : « Le soldat », son nom, « s'équipe avec un », le nom de l'équipement et un point. Par exemple :

```
Le soldat Minus s'équipe avec un bouclier.
```

Pour cette méthode :

- vous devrez **OBLIGATOIREMENT** utiliser une structure **switch** sur le nombre d'équipement.
- la méthode ne doit pas avoir de portion de code dupliqué, au besoin créer une méthode privée. Si besoin, sélectionner la portion de code dupliquée, et choisir la commande 'Extract Method...' du menu Refactor.

Tester la méthode dans le main de la classe `Romain`, en ajoutant à Minus deux casques puis un bouclier et enfin un autre casque. Vous devez obtenir l'affichage suivant :

```
Le soldat Minus s'équipe avec un casque.  
Le soldat Minus possède déjà un casque.  
Le soldat Minus s'équipe avec un bouclier.  
Le soldat Minus est déjà bien protégé !
```

- e. Sauvegarder votre travail sous GitHub (cf p1 - Historisation périodique de votre projet - exemple de message pour le commit "les énumérés").