

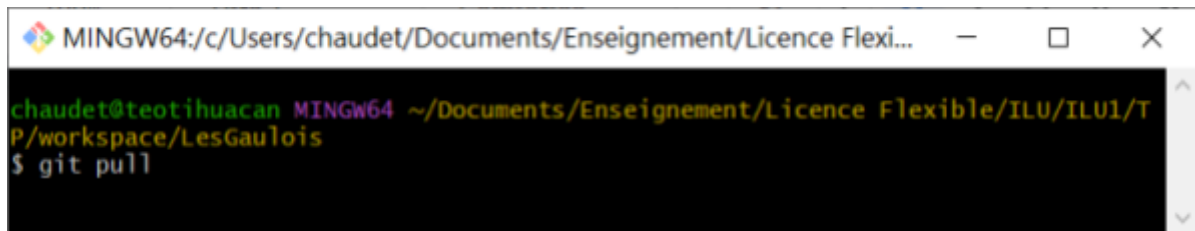
Commencer à coder en Java

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'`.



- 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*

Historisation périodique de votre projet (Rappel)

Lorsque vous avez fini une question du sujet de TP (ce qui vous a dû vous amener à créer ou modifier plusieurs classes), vous devez procéder à l'historisation de votre dépôt local et la mise à jour du dépôt distant.

- Avec **un explorateur de fichier** retrouver sous votre projet LesGaulois (vous devez y trouver le fichier `.gitignore`) puis cliquer droit (sans rien sélectionner) et sélectionner 'Git Bash Here'.
- Sous Git Bash : utiliser les commandes git :
 - `git add .`
 - `git commit -m <Intitulé des modifications>`
exemple: `git commit -m "TP1 méthode toString"`
 - `git push`
- Une fenêtre peut s'ouvrir en vous demandant de vous identifier.
Sélectionner "Sign in with your browser", sur les ordinateurs de la fac

copier l'URL sur chrome (edge ne permet pas d'appuyer sur les boutons), si vous êtes déjà identifié sur git sur un autre onglet l'identification se fera instantanément, sinon une autre fenêtre s'ouvre, sélectionner sur "Authorize GitCredentialManager".

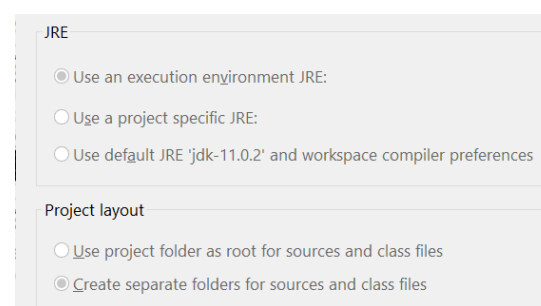
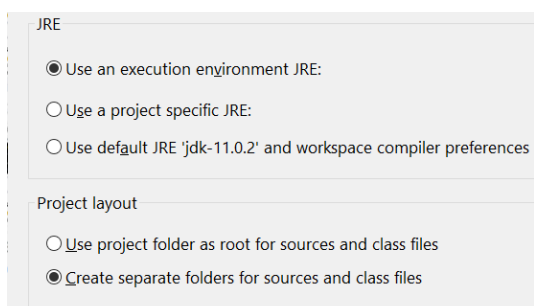
Vous pouvez reprendre les diapos du TP1 de GL si besoin.

ATTENTION Si vous n'avez pas fini le sujet à la fin de la séance, vous devrez le terminer chez vous.

Pour cela vous devrez :

- installer les outils sur votre ordinateur, document "Installation des outils - Git - Eclipse" sous Moodle (TP1 de GL),
- Ajouter le plugin sonarQube, diapo 5&6 du document "2.1 Préparation environnement Eclipse" sous Moodle (TP1 de GL),
- Cloner le projet "LesGaulois" sous un répertoire que vous nommerez workspace,
- Ouvrir eclipse sous le workspace où vous avez cloner le projet.
- Créer le projet "LesGaulois" sous Eclipse. **Veiller à utiliser le même nom de projet que le nom du dépôt. Toutes les options deviennent grisées. Si ce n'est pas le cas, vous n'avez probablement pas ouvert depuis le répertoire workspace où vous avez cloné votre projet (figure ci-dessous),**

Historiser régulièrement votre projet, mais surtout vérifier à la fin du TP que vous avez bien poussé toutes les modifications sur github, afin de récupérer votre travail à la prochaine séance.



2 Programmation java Les Gaulois

Rappel du TP GL précédent.

Dans le paquetage "personnages" vous avez déjà créé deux classes : Gaulois et Romain

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

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

    public String getNom() {
        return nom;
    }

    public void parler(String texte) {
        System.out.println(prendreParole() + "\"" + texte + "\"");
    }

    private String prendreParole() {
        return "Le gaulois " + nom + " : ";
    }
}

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

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

    public String getNom() {
        return nom;
    }

    public void parler(String texte) {
        System.out.println(prendreParole() + "\"" + texte + "\"");
    }

    private String prendreParole() {
        return "Le romain " + nom + " : ";
    }
}
```

1. Les objets / instances

- 1.1 Créer le paquetage « test_fonctionnel » et dans ce paquetage créer la classe « TestGaulois ».

Dans cette classe, générer la méthode main : écrire le nom de la méthode « main » à l'intérieur des accolades de la classe *TestGaulois*.

Dans le main, créer l'instance *asterix* qui a pour nom "Astérix" et pour force 8. Si vous utilisez bien la génération c'est à dire G suivi de CTRL + Espace pour générer le type **G** Gaulois, alors Eclipse fera directement l'import de la classe Gaulois.

Dans le main, créer l'instance *obelix* qui a pour nom "Obélix" et pour force 16.

- 1.2 Continuer le main en faisant parler les personnages, pour cela utiliser la méthode parler. La sortie attendue est celle ci-dessous.

```
Le Gaulois Astérix : "Bonjour Obélix."
```

```
Le Gaulois Obélix : "Bonjour Astérix. Ca te dirais d'aller chasser des sangliers ?)"
```

```
Le Gaulois Astérix : "Oui très bonne idée."
```

2. La méthode toString

- 2.1 Dans la classe Gaulois, créer un main. Les "main" étant indépendants, vous devez re-créez l'instance *asterix* qui a pour nom "Astérix" et pour force 8.

Afficher l'objet *asterix*. Il s'agit de l'objet entier et non de son nom. Son affichage est de la forme nom du paquetage "." nom de la classe "@" un nombre en hexadécimal basé sur son adresse mémoire.

Exemple : `personnages.Gaulois@24d46ca6`

- 2.2 Toujours dans la classe *Gaulois* générer la méthode toString : Source > Generate toString... Cette méthode lorsqu'elle est générée en cochant tous les champs (attributs de la classe) permet de connaître l'état de l'objet.

Relancer le main de la classe *Gaulois*. Ce n'est plus le même affichage que précédemment. On obtient la chaîne suivante :

```
Gaulois [nom=Astérix, force=8]
```

- 2.3 Vous pouvez aussi personnaliser cette méthode. Par exemple, modifier la méthode afin d'afficher le nom du gaulois. Relancer le même main vous devez obtenir :

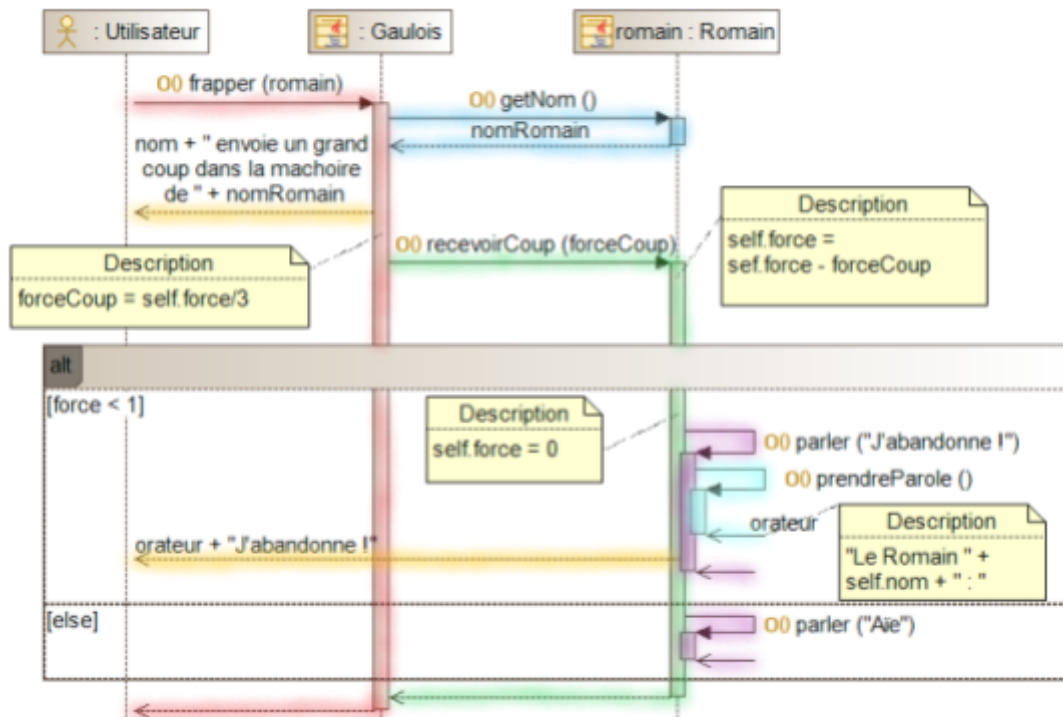
```
Astérix
```

Donc, par la suite, si vous voyez à l'affichage une adresse comme `personnages.Gaulois@24d46ca6` alors que vous souhaitez la valeur d'un attribut comme `Astérix` c'est sûrement que vous avez écrit `System.out.println(asterix);` au lieu de `System.out.println(asterix.getNom());`, et qu'il n'y a pas eu réécriture du toString dans la classe.

3. Interaction entre les objets

Les gaulois peuvent frapper les romains, autrement dit les romains peuvent recevoir des coups ! Pour comprendre l'interaction entre deux objets nous allons utiliser le diagramme de séquence.

Le diagramme ci-dessous décrit deux méthodes. La méthode *frapper* (repérée en rouge), la méthode *getNom* (repérée en bleu) et la méthode *recevoirCoup* (repérée en vert).



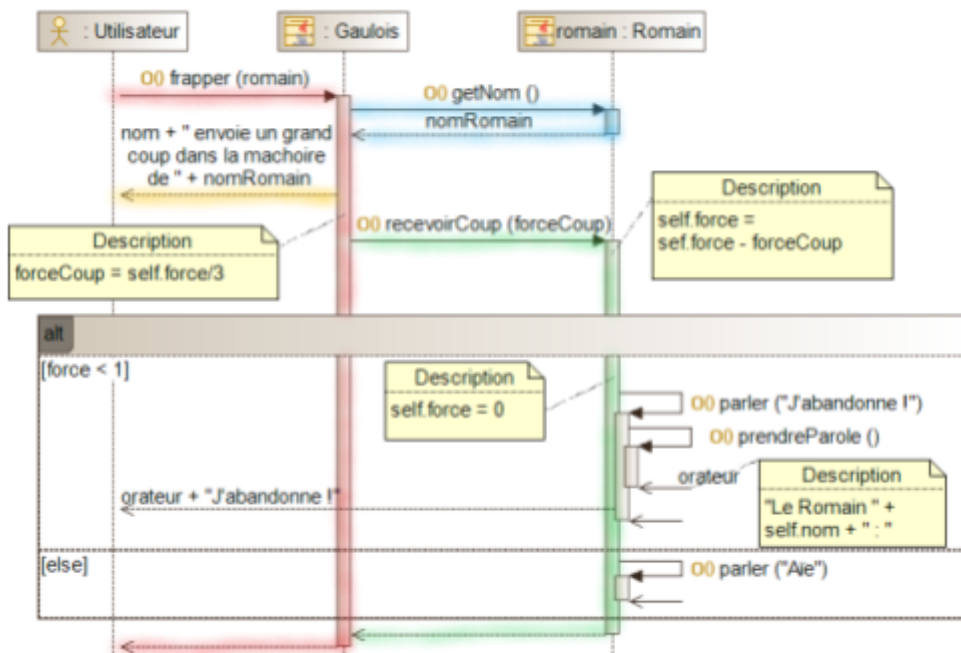
Les méthodes sont implémentées dans la classe correspondante à l'objet pointé par la flèche :

- La flèche représentant l'appel à la méthode *frapper* pointe sur l'objet de type *Gaulois*, donc la méthode *frapper* devra être implémentée dans la classe *Gaulois*.
- La flèche représentant l'appel à la méthode *getNom* pointe sur l'objet *romain* de type *Romain*, donc la méthode *getNom* devra être implémentée dans la classe *Romain*. Pareil pour la méthode *recevoirCoup*.

Pour implémenter une méthode il faut identifier le paramètre d'entrée et le paramètre de sortie.

- Pour la méthode *frapper* le paramètre d'entrée est de type *Romain* et n'a pas de paramètre de retour : rien n'est noté sur la flèche en pointillé à la fin de la période d'activité.
- Pour la méthode *getNom* elle ne possède pas de paramètre d'entrée mais possède un paramètre de retour : la flèche en pointillé à la fin de la période d'activité retourne le nom du romain.

3.1 Implémenter la méthode *frapper*



Pour implémenter une méthode nous allons prendre en considération toutes les flèches qui sortent de sa période d'activité.

La méthode *frapper* commence par appeler la méthode *getNom* dans l'objet de type *Romain*, est récupère son nom dans la variable *nomRomain*:

```
public void frapper(Romain romain) {
    String nomRomain = romain.getNom();
}
```

La flèche sortante suivante est repérée en jaune : elle va d'un objet à l'acteur Utilisateur : il s'agit d'un affichage. L'acteur c'est vous et l'affichage correspond à ce que vous lirez sur la console. La variable *nom* est l'attribut *nom* de la classe *Romain*. La variable *nomRomain* est celle que vous venez de récupérer.

```
public void frapper(Romain romain) {
    String nomRomain = romain.getNom();
    System.out.println(nom + " envoie un grand coup dans la mâchoire de " + nomRomain);
}
```

Une note UML précise que la force du coup est la force du gaulois / 3. Le mot clef *self* en UML correspond au *this* en Java. *forceCoup* = *force* / 3;

Enfin la dernière flèche sortante, repérée en vert, appelle la méthode *recevoirCoup* dans l'objet de type *Romain*. Il n'y a aucune variable de retour (la flèche en pointillé verte est vide).

```
public void frapper(Romain romain) {
    String nomRomain = romain.getNom();
    System.out.println(nom + " envoie un grand coup dans la mâchoire de " + nomRomain);
    int forceCoup = force / 3;
    romain.recevoirCoup(forceCoup);
}
```

Analysons le code obtenu en lecture directe du diagramme de séquence et procédons si besoin à du refactoring.

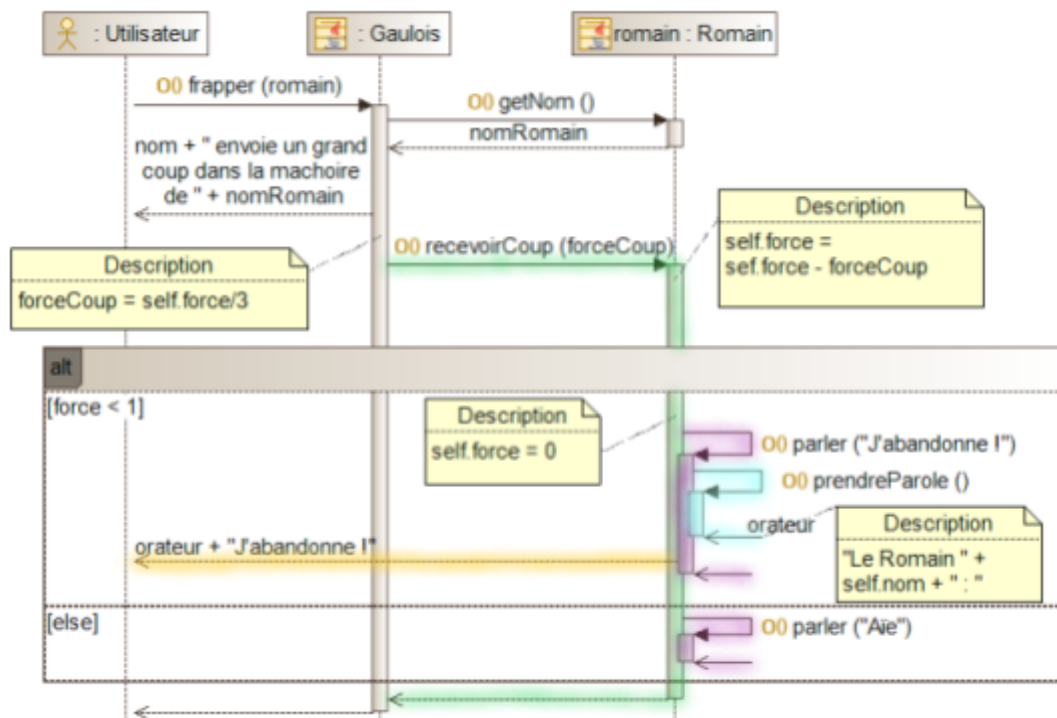
- *nomRomain* est utilisé uniquement dans l'affichage : la variable n'est pas utile,
- *forceCoup* est utilisé uniquement dans l'appel à la méthode *recevoirCoup* : la variable n'est pas utile.

Le code final en Java sera donc celui ci-dessous. En rouge les caractères que vous devez écrire, le reste est généré (cf TP1 de GL).

```
public void frapper(Romain romain) {
    System.out.println(nom + " envoie un grand coup dans la mâchoire
    de " + romain.getNom());
    romain.recevoirCoup(force / 3);
}
```

3.2 Implémenter la méthode *recevoirCoup*

Procéder de manière identique pour implémenter la méthode *recevoirCoup*.



Dans le diagramme on peut voir un fragment combiné (combined fragment) alt.

Rappel : les combined fragment correspondent à des comportements complexes.

En ILUx nous verrons :

- opt : représentant soit une structure si alors (if)
- alt : représentant soit une structure si alors sinon (if else) soit une structure cas où (switch).
- loop : représentant une boucle, prendre celle qui est le plus adapté entre for, do while et while.

3.3 Test fonctionnel des méthodes précédentes : Baston entre Gaulois et Romain

Dans la classe *TestGaulois* (créé en 1.1) compléter le main en ajoutant :

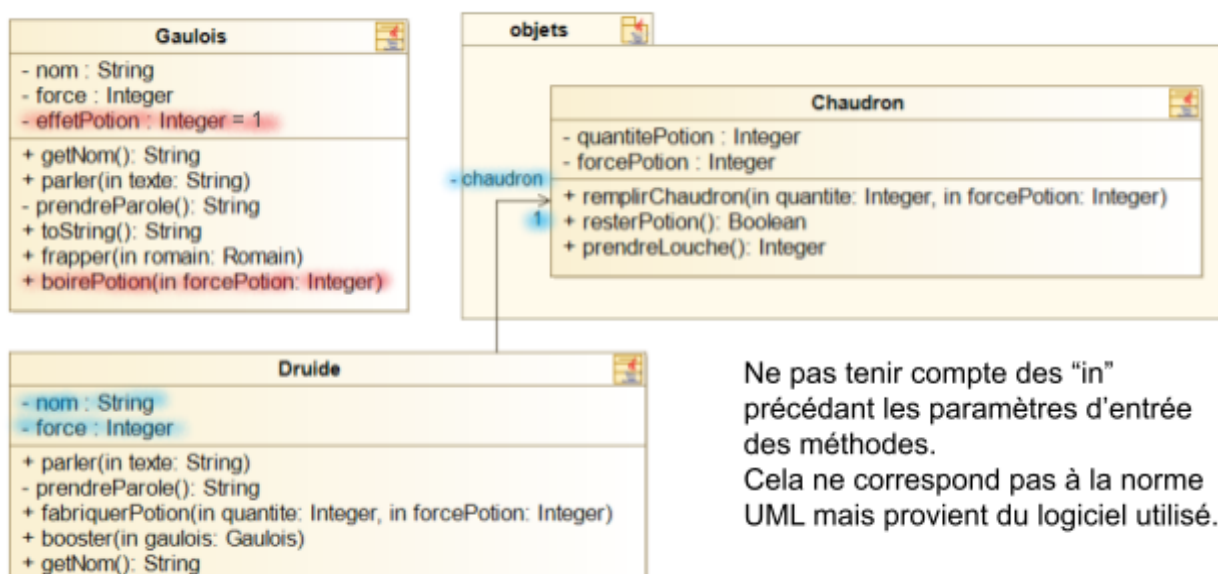
- l'instance minus de type *Romain* qui a une force de 6,
- l'affichage de la phrase suivante : "Dans la forêt Astérix et Obélix tombent nez à nez sur le romain Minus". Dans cette phrase les noms des personnages doivent être récupérés grâce à la méthode *toString* ou au getteur sur le nom selon le type des différentes instances,
- en utilisant une boucle, Astérix frappe 3 fois le pauvre Minus.

Sortie console :

```
Le Gaulois Astérix : "Bonjour Obélix."
Le Gaulois Obélix : "Bonjour Astérix. Ca te dirais d'aller chasser des sangliers ?);"
Le Gaulois Astérix : "Oui très bonne idée."
Dans la forêt Astérix et Obélix tombent nez à nez sur le romain Minus.
Astérix envoie un grand coup dans la mâchoire de Minus
Le Romain Minus : "Aïe"
Astérix envoie un grand coup dans la mâchoire de Minus
Le Romain Minus : "Aïe"
Astérix envoie un grand coup dans la mâchoire de Minus
Le Romain Minus : "J'abandonne !"
```

4. Diagramme de classes et consolidation des différents points précédents

4.1 Implémenter les classes *Druide* et *Chaudron* en suivant le diagramme de classe ci-dessous. Pour vous aider les attributs de la classe *Druide* sont en bleu, et en rouge les modifications à effectuer dans la classe *Gaulois*. Ne remplissez pas le corps des méthodes de suite, deux diagrammes de séquence vous seront fournis.

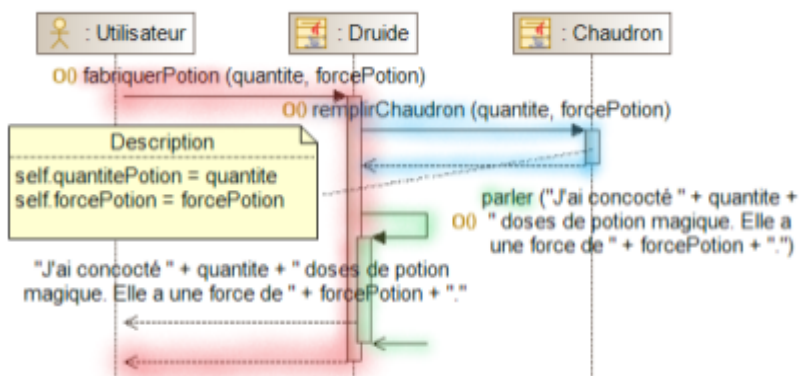


4.2 Compléter les méthodes simples

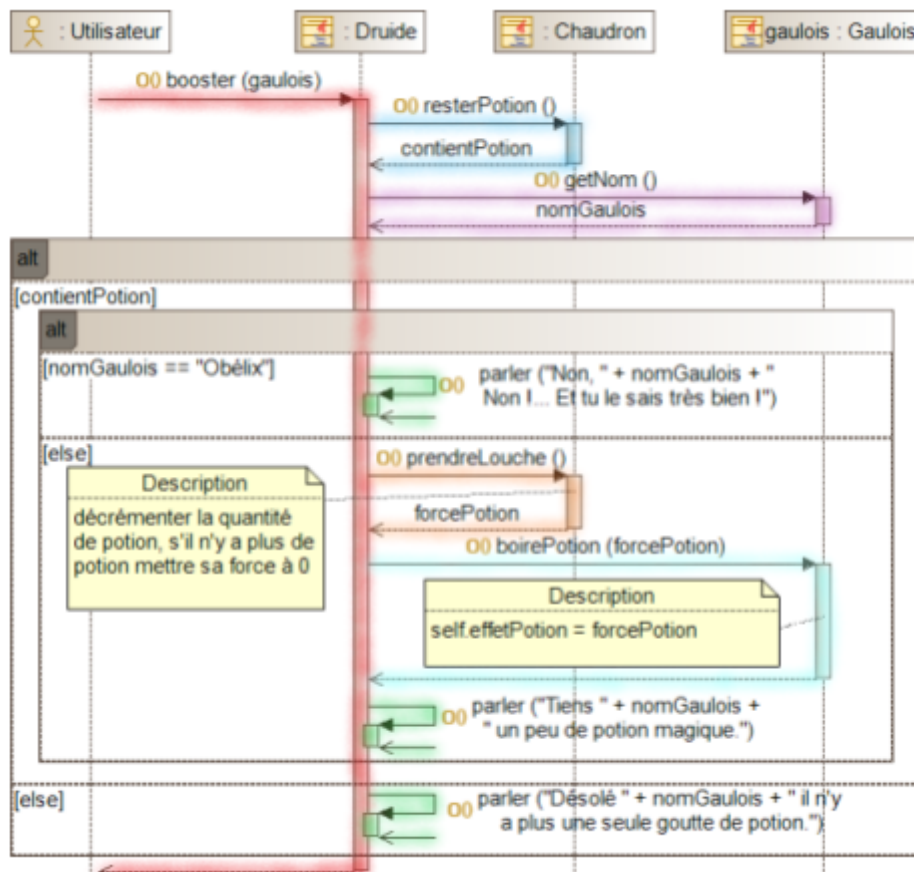
Dans la classe *Druide* compléter les méthodes *getNom*, *parler* et *prendreParole*. Cette dernière méthode est un peu modifiée par rapport à celle dans la classe *Gaulois*, elle affiche "Le Druide" et son nom.

Dans la classe *Chaudron* compléter la méthode *resterPotion* qui retourne **true** ou **false** suivant que la quantité de potion est égale à 0.

4.3 Compléter la méthode *fabriquerPotion* et *remplirChaudron*



4.4 Compléter la méthode *boosterGaulois* et toutes les méthodes qui lui sont nécessaires



4.4 Modification de la méthode *frapper* dans la classe Gaulois.

Lorsqu'un gaulois frappe un romain la force du coup est multipliée par l'effet de la potion avant d'être divisée par 3.

Une fois qu'il a frappé un romain l'effet de la potion est décrémentée. L'effet ne descendra jamais au-dessous de 1, ce qui signifie qu'il a retrouvé sa force initiale.

4.5 Test fonctionnel des méthodes précédentes : les gaulois sont sous influence

Dans la classe *TestGaulois* (créé en 1.1) compléter le main en ajoutant :

- l'instance brutus de type *Romain* qui a une force de 14,
- l'instance panoramix de type *Druide* qui a une force de 2,
- le druide prépare 4 doses de potion de force 3,
- le druide booste Obélix,
- le druide booste Astérix,
- en utilisant une boucle, Astérix frappe 3 fois Brutus.

Sortie console de la partie ci-dessus :

```
Le Druides Panoramix : "J'ai concocté 4 doses de potion magique. Elle a une force de 3."
Le Druides Panoramix : "Non, Obélix Non !... Et tu le sais très bien !"
Le Druides Panoramix : "Tiens Astérix un peu de potion magique."
Astérix envoie un grand coup dans la mâchoire de Brutus
Le Romain Brutus : "Aïe"
Astérix envoie un grand coup dans la mâchoire de Brutus
Le Romain Brutus : "Aïe"
Astérix envoie un grand coup dans la mâchoire de Brutus
Le Romain Brutus : "J'abandonne !"
```

Finir ce sujet avant le prochain TP et penser à pousser vos modifications sur github !