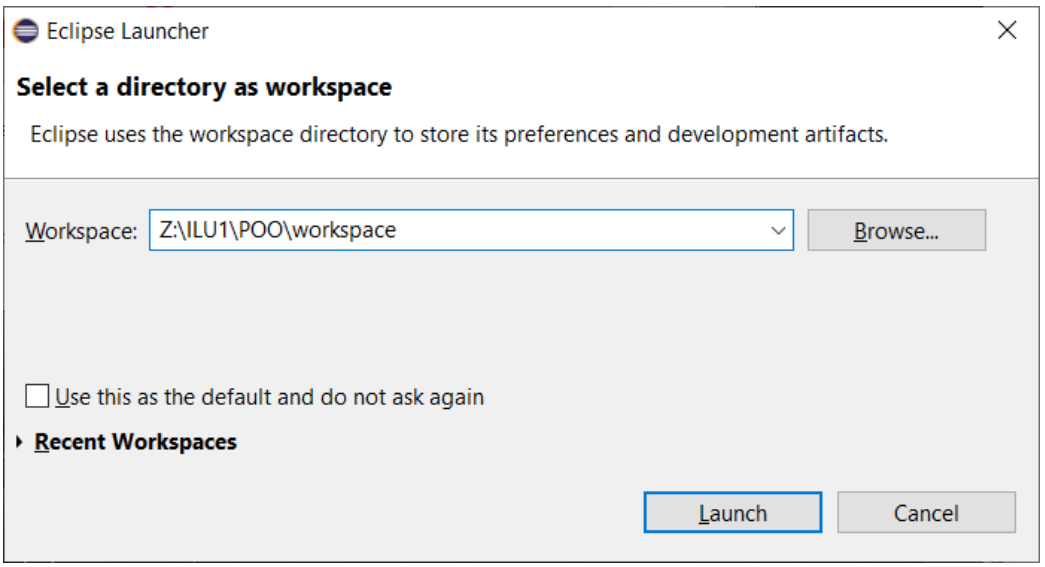
Introduction au développement Java en ILU1

Préparation de l'environnement Eclipse

Ouverture De l'IDE Eclipse

Nous allons utiliser l'IDE Eclipse, vous devrez apprendre à l'utiliser : cela fait partie des compétences que vous devez acquérir.

Aller sous l'environnement Windows et choisir la version **Eclipse 2022-03**



SI la fenêtre ci-dessus ne s'ouvre pas faire File>Switch Workspace>Other...

A l'aide du bouton browse placez-vous sous votre **espace personnel** :

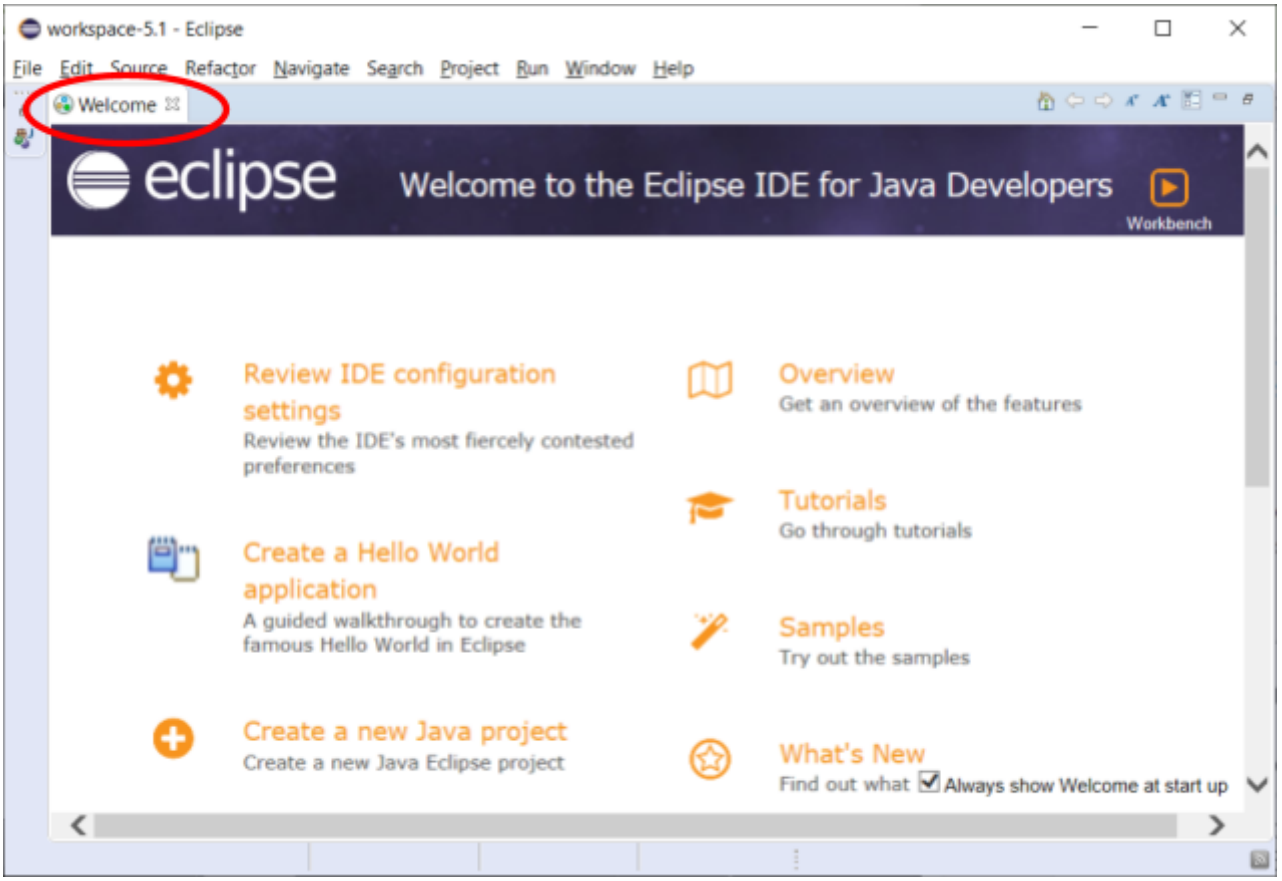
- Sélectionner "Ce PC" puis choisir le lecteur "Z:"
- Créer des répertoires afin d'organiser votre espace de travail (clic droit puis Nouveau\Dossier
  Exemple d'organisation : Z:\ILU1\POO

DANS TOUS LES CAS que vous soyez sur votre ordinateur personnel ou sur un ordinateur de l'université **terminer votre chemin par le dossier *workspace***

Exemple de chemin complet :

- à l'université : Z:\ILU1\POO\workspace
- sur votre ordinateur personnel :
  C:\Users\chaudet\Documents\ILU1\POO\workspace

Vous arrivez sur une fenêtre Welcome : il faut la fermer avec la croix au niveau de l'onglet



Retrouver cette partie en vidéo sur Moodle : 1-1-ouverture Eclipse.mp4

Installation des plugins Eclipse

Nous allons utiliser deux plugins durant votre formation en ILUx. On va les utiliser de façon complémentaire même si un seul des deux pourrait suffire dans certains cas.

sonarLint

Pour rappeler ce qu'on a vu en cours, un "*Linter*" est un outil qui assiste le développeur en analysant statiquement le code pour en contrôler et en améliorer la qualité.

checkStyle

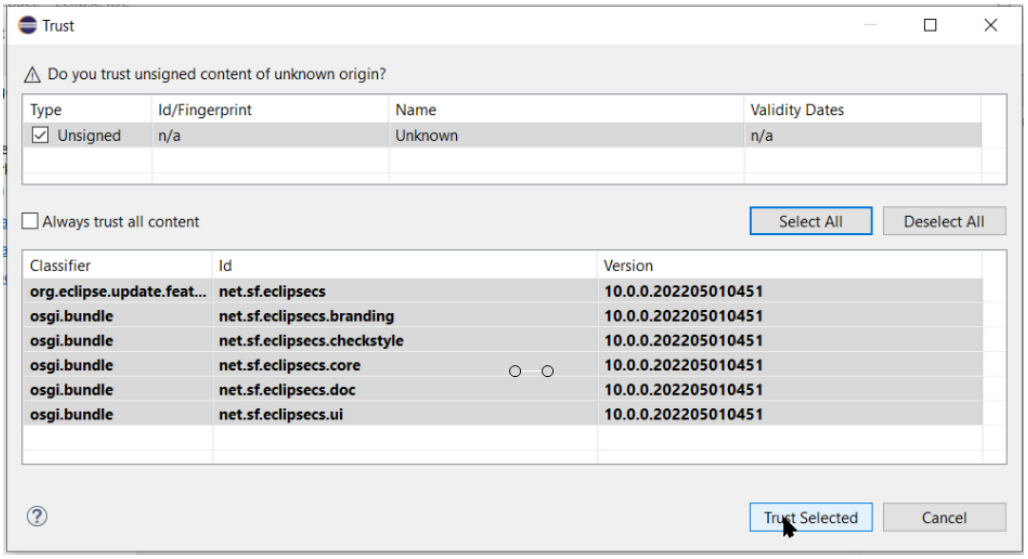
Nous avons également vu en cours que CheckStyle est aussi un outil de *métrologie*[1] qui réalise des métriques sur le code pour en évaluer la qualité.

[1] Métrologie : science de la mesure (https://fr.wikipedia.org/wiki/Métrologie)

Installation

Pour ajouter des plugins, aller sur l'onglet "Help" puis sélectionner "Eclipse Marketplace..."
Dans la barre de recherche "Search" taper le nom du plugin à ajouter : "sonarLint" puis appuyer sur le bouton "Go".
Sélectionner le plugin de "sonarLint" et cliquer sur le bouton "Install".
Si besoin, accepter le contrat puis cliquer sur le bouton "Finish".
Ne pas redémarrer Eclipse de suite : nous allons d'abord installer le deuxième plugin.
Donc recommencer : "Help", "Eclipse Marketplace...", dans la barre de recherche "Search" taper "checkstyle" puis appuyer sur le bouton "Go".
Sélectionner le plugin de "Checkstyle Plug-in 10.0.0" (ou version supérieure) et cliquer sur le bouton "Install".
Si besoin, accepter le contrat puis cliquer sur le bouton "Finish".

Accepter de faire confiance au contenu :



Bouton "Select All" puis cliquer sur "Trust Selected"
Enfin accepter de redémarrer Eclipse afin de terminer l'installation des deux plugins.

Retrouver cette partie en vidéo sur Moodle : 1-2-installationPlugin.mp4

Préparation de l'environnement Git

Au fur et à mesure des TP, nous adopterons de plus en plus de pratiques professionnelles qui amélioreront la productivité et la qualité de développement. Pour commencer, nous allons utiliser l'outil de gestion de version git et le site d'hébergement GitHub. Ceci se décompose en 5 étapes :

- la création de votre compte sur GitHub
- la création de votre dépôt distant pour ce sujet de TP
- le clonage local de votre projet
- l'historisation périodique de votre projet
- la récupération de votre dépôt distant

Création de votre compte GitHub

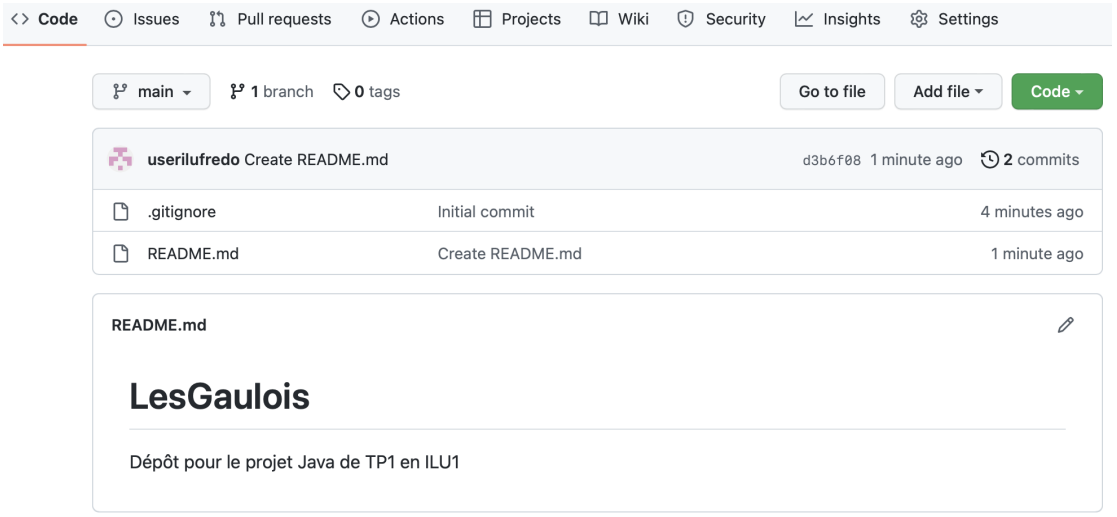
Connectez vous sur le site https://github.com et créez un nouveau compte si vous n'en avez pas déjà un (bouton `Sign Up`) avec une adresse mail personnelle ou de l'université, un mot de passe solide, un pseudo d'utilisateur et vos réponses aux questions suivantes. Après réception d'un code par mail et son enregistrement sur la plateforme, vous êtes prêt pour les étapes suivantes.

Création de votre dépôt distant pour ce sujet de TP

Connectez-vous et placez-vous sur la page de vos dépôts (*repositories*). Cliquez sur le bouton New pour démarrer la création du dépôt. Renseignez :

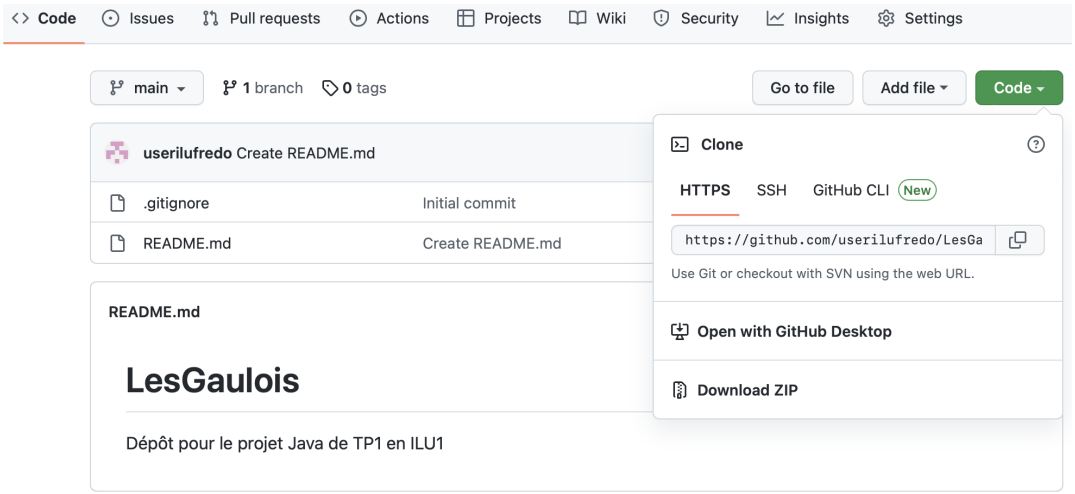
- le nom du dépôt (pourquoi pas "LesGaulois" ?),
- la description ("Dépôt pour le projet Java de TP1 en ILU1" vous conviendrait ?),
- la visibilité `'Public'` (au moins pour celui-ci),
- l'ajout d'un fichier `'README'` initial (qui définit ce qu'il faut lire en premier sur le dépôt),
- l'ajout d'un fichier .gitignore en utilisant le template java (pour spécifier les fichiers qui ne seront pas automatiquement rajoutés car considérés comme inutiles sur le dépôt)
- pas de précision de licence pour le moment.

Une fois la création confirmée, vous devriez obtenir une première version de votre dépôt distant comparable à l'image ci-dessous.



Clonage local initial de votre projet

Cette procédure peut être réalisée soit par l'intermédiaire de l'outil graphique GitHub Desktop (https://desktop.github.com) soit grâce à des commandes en ligne (via l'outil `git` installé par défaut sur les machines de TP). Pour cela, tout démarre du dépôt distant sur GitHub.



Le bouton `'Code'` guide pour les deux solutions :

- En restant sur le mode HTTPS, on peut copier l'URL du dépôt présentée dans la pop-up au moyen du bouton à droite du champ de texte (`'https://github.com/userilufredo/LesGaulois.git'` pour ce qui concerne cet exemple). Vous pouvez alors utiliser la commande git `'git clone'`+ `l'URL copiée` dans le répertoire `'workspace'` (celui créé pour Eclipse ci-dessus) pour votre workspace (voir comment créer le projet `'LesGaulois'` plus loin dans la partie `'Créer un projet Java'`).
- En cliquant sur 'Open with GitHub Desktop' (qui vous proposera de l'installer si vous ne l'avez jamais fait). L'outil graphique crée le dépôt local à partir des mêmes paramètres.

Historisation périodique de votre projet

Lorsque vous avez fini une question du sujet de TP (ce qui vous a amené à créer ou modifier plusieurs classes), vous devez procéder à l'historisation de votre dépôt local et la mise à jour du dépôt distant. En ligne de commande, il faut réaliser trois commandes successives :

- Enregistrement des modifications à historiser dans l'index au moyen de la commande `'git add .'` (n'oubliez pas le point)
- Enregistrement dans le dépôt local des fichiers mis à l'index avec la commande `'git commit –m <Intitulé des modifications>'`
- Mise à jour du dépôt distant avec la commande `'git push'`[2]

- Avec l'outil graphique, la procédure est simplifiée à deux actions :
  - Dans l'onglet `'Changes'`, la liste des fichiers créés ou modifiés apparaît avec une présélection de tous les fichiers. Il ne reste plus qu'à écrire l'intitulé du commit (et la description éventuelle en-dessous) et cliquer sur le bouton `'Commit to main'` en bas de fenêtre.
  - La fenêtre principale se met à jour et présélectionne le bouton `'Push origin'`. Il n'y a plus qu'à cliquer dessus.

Récupération du dépôt distant

A l'issu d'un TP sur une machine de l'université, vous aurez donc modifié le dépôt local de la machine de TP et le dépôt distant. Si vous voulez travailler sur une machine personnelle en dehors de la séance, il vous faut mettre à jour votre dépôt local avec le contenu du dépôt distant (sauf s'il s'agit de créer le projet sur cette machine personnelle ce qui nécessite de reprendre la procédure décrite ci-dessus depuis `'Clonage local initial de votre projet'`).
Pour cela, toujours deux solutions :

- En ligne de commande, en étant dans le répertoire du projet `'LesGaulois'`, tapez la commande `'git pull'`.
- Avec l'outil graphique, le bouton `'Fetch origin'` permet de mettre à jour la fenêtre principale pour connaître les modifications à mettre à jour localement. Il suffit ensuite de cliquer sur le bouton `'Pull origin'`.

Il faut bien noter que cette récupération du dépôt distant sera également à réaliser quand vous reviendrez en TP.

[2] A la première utilisation de la commande, une authentification par le navigateur peut être demandée. Complétez cette identification dans votre navigateur et revenez sur votre terminal continuer votre processus Git.

Prise en main d'un IDE

L'avantage d'utiliser un IDE (Integrated Development Environment) c'est qu'il intègre plusieurs fonctionnalités et outils qui améliorent la qualité du développement. Une fois certaines habitudes prises, il permet de faciliter la productivité du développeur en écrivant certaines parties de code pour lui ou en automatisant certaines tâches. Parmi les fonctionnalités que nous utiliserons beaucoup, il y a :

- la création d'un projet Java
- la création de squelettes de code
- la coloration syntaxique
- la complétion des symboles (de variables ou méthodes accessibles)

Nous allons reprendre l'écriture des classes vues en TD en expliquant comment le faire rapidement. Donc si vous avez déjà réécrit le TD, on vous propose de le recommencer !

Créer les différents éléments

Retrouver cette partie en vidéo sur Moodle : 3-1-creationElements.mp4

Créer un projet Java

Attention, en cas de clonage d'un dépôt distant avec Git. Il faut cloner le dépôt avant de créer le projet avec Eclipse et veiller à utiliser le même nom de projet que le nom du dépôt.

File > New > Java Project
Donner un nom représentant le sujet de votre projet (pas le numéro de séance de TP) par exemple "LesGaulois"
Dans la partie "Module" décocher l'option : "Create module-info.java file".
Cliquer sur le bouton "Finish"

Développer l'arbre :

Créer un paquetage (package)

Cliquer droit sur le dossier "src" puis New > Package
Donner un nom représentant ce que contiendra le package, ici "personnages".
Convention d'écriture : le nom d'un package doit être entièrement écrit en minuscule.

Créer une classe Java

Cliquer droit sur le package "personnages" puis New > Class
Donner le nom de la classe que vous voulez créer, ici "Gaulois".
Convention d'écriture : le nom d'une classe doit :
- commencer par une majuscule,
- être au singulier,
- ne pas comporter d'accents.

S'il est composé de plusieurs mots ils seront accolés et chaque début de mot commencera par une majuscule (*CamelCase*, ex : SousMarin).

Créer les attributs d'une classe

Gaulois.java

```
1  package personnages;
2
3  public class Gaulois {
4      private String nom;
5      private int force;
6      private int effetPotion = 1;
7  }
```

Créer les attributs comme ci-contre.

Convention d'écriture : le nom d'un attribut doit :
- commencer par une minuscule,
- ne pas comporter d'accents.

S'il est composé de plusieurs mots ils seront accolés et chaque début de mot commencera par une majuscule (ex : sousMarin).

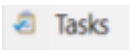
Générer le code standard

Un IDE permet de générer certaines parties du code. Selon la configuration de votre IDE Eclipse, celui-ci ajoute un commentaire :

// TODO Auto-generated method stub

Cela indique qu'il s'agit d'un code généré et qu'il faut le personnaliser.
Une fois que vous avez vérifié qu'il ne faut rien modifier ou au contraire que vous avez modifié ou complété le code alors vous devez supprimer ce commentaire.

Vous pouvez vous même ajouter des commentaires // TODO afin de noter les tâches à réaliser. Par exemple pour vous rappeler qu'une méthode n'est pas terminée et indiquer les différents points que vous devez encore réaliser.

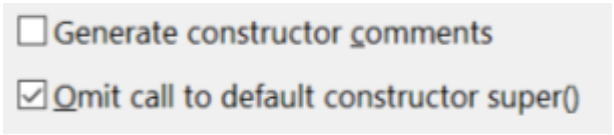
Pour voir l'ensemble des tâches que vous devez terminer sélectionner :
Window > Show View > Task 

Retrouver cette partie en vidéo sur Moodle : 3-2-genererCodeStandard.mp4

Générer le constructeur

Source > Generate Constructor using Fields...
Sélectionner les attributs qui seront affectés par les paramètres d'entrée du constructeur comme dans l'illustration ci-dessous.

Cocher la case "Omit call to default constructor super()"



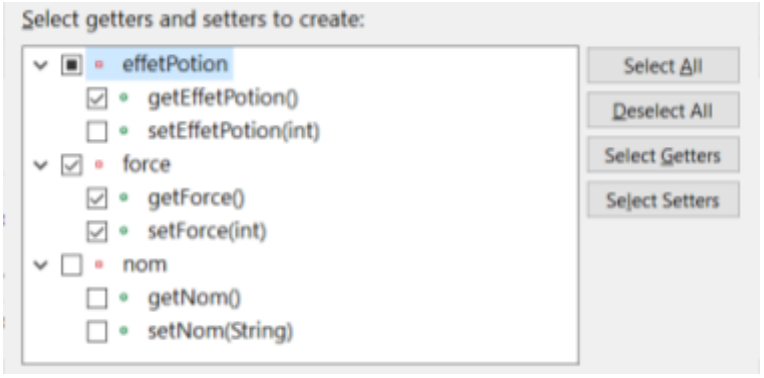
Cliquer sur "generate"

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

Générer les getters / setters

1. Pour générer plusieurs getteur et setteur :

Source > Generate Getters and Setters...
Développer l'arbre afin de choisir exactement les getteurs et setteurs dont vous avez besoin.



Cliquer sur "generate"
Remarque : un getteur sur un attribut de type boolean s'écrira is + le nom de l'attribut. Par exemple :

```
private boolean ouvert;

public boolean isOuvert(){
        return ouvert;
}
```

Supprimer tous les getteur et setteur qui ont été générés (dans ce programme nous n'avons besoin que du getteur sur le nom).

2. Pour générer un getteur ou un setteur en particulier

Ecrire directement à l'endroit où vous souhaitez générer :

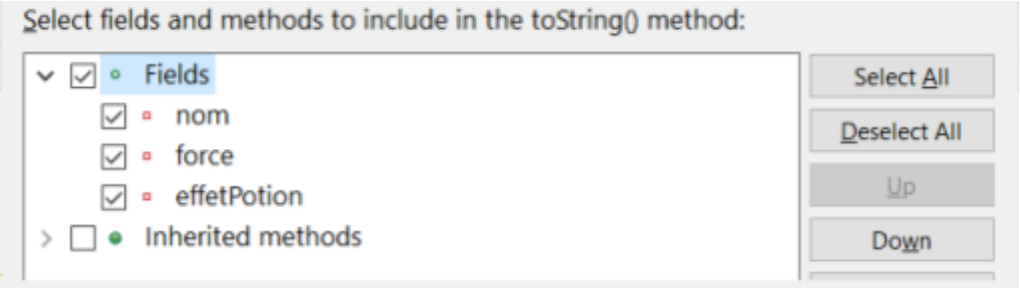
- un getteur : get puis appuyer sur les touches Ctrl + la barre d'espace
  ou is puis appuyer sur les touches Ctrl + la barre d'espace
- un setteur : set puis appuyer sur les touches Ctrl + la barre d'espace

Générer le getteur sur le nom.

Générer la méthode toString

La méthode toString() est une méthode qui est automatiquement appelée lorsque l'on souhaite obtenir une représentation textuelle (String) de l'objet, comme c'est le cas pour un affichage par exemple. Vous pouvez la personnaliser ou utiliser celle générée par Eclipse en choisissant les champs que vous souhaitez voir apparaître.

Source > Generate toString()...



Cliquer sur "generate"

```
@Override
public String toString() {
    return "Gaulois [nom=" + nom + ", force=" + force + ", effetPotion=" + effetPotion + "]";
}
```

Générer la méthode main

Il s'agit de la méthode qui sera exécutée lorsque vous ferez un "run" sur cette classe.

Ecrire directement à l'endroit où vous souhaitez générer la méthode main :
main puis appuyer sur les touches Ctrl + la barre d'espace.

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

}
```

Mettre en forme le code selon les bonnes pratiques

Source > Format (Ctrl + Shift + F)

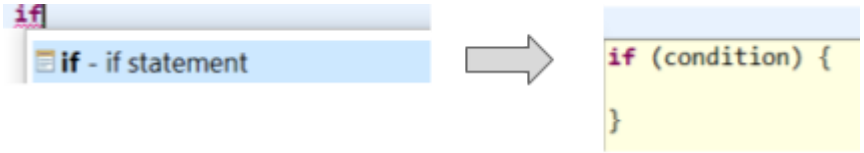
Coder rapidement

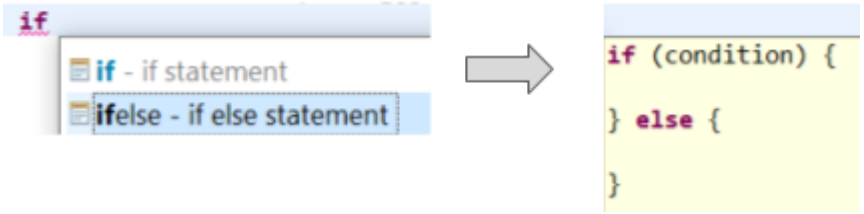
Les structures algorithmiques

Se placer dans la méthode main.

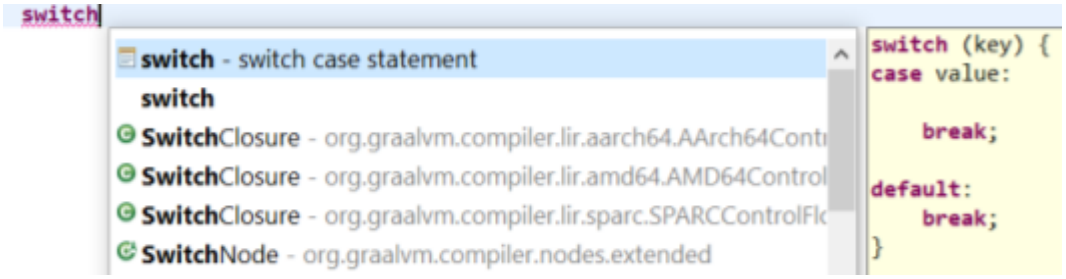
En tapant quelques lettres Eclipse génère un squelette à compléter. Pour passer d'un champ au suivant taper sur la touche tabulation.

Taper if puis appuyer sur les touches Ctrl + la barre d'espace vous permet de choisir entre le if ou le ifelse

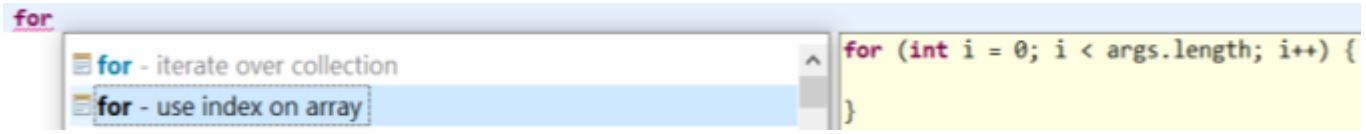




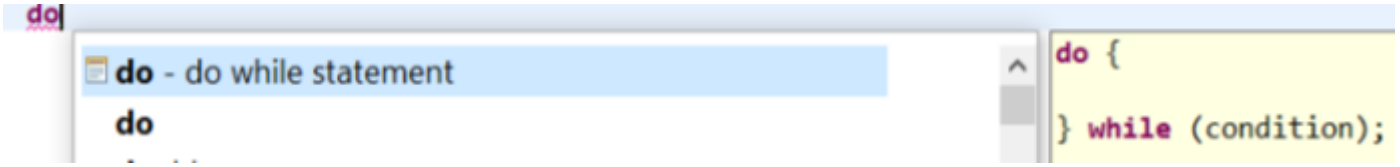
Taper switch puis appuyer sur les touches Ctrl + la barre d'espace



Taper for puis appuyer sur les touches Ctrl + la barre d'espace

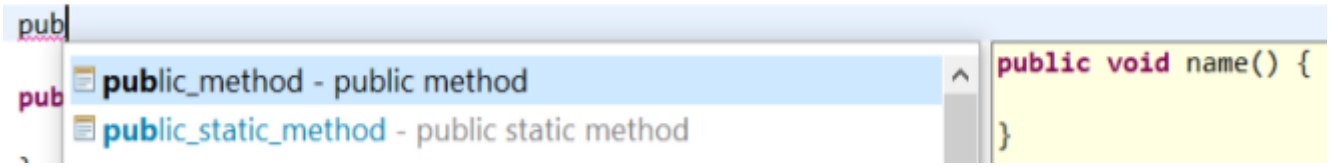


Taper do puis appuyer sur les touches Ctrl + la barre d'espace



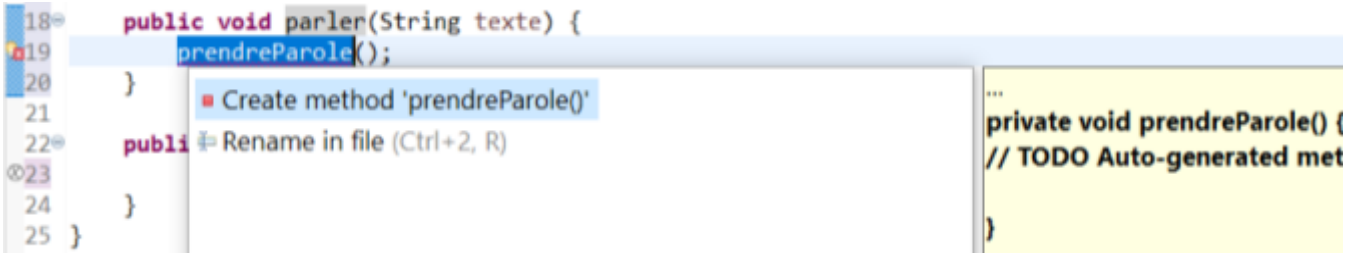
Les méthodes

Taper pub puis appuyer sur les touches Ctrl + la barre d'espace

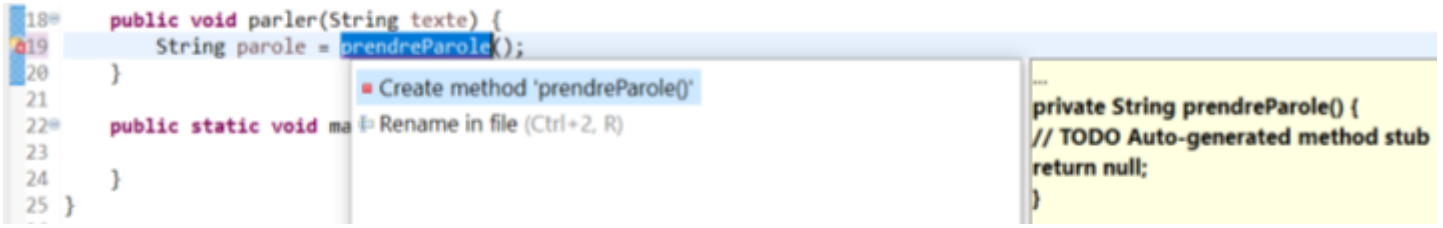


Une méthode peut être créée lorsque l'on en a besoin dans une autre méthode. Par exemple, j'ai besoin d'une méthode *prendreParole* que je n'ai pas encore écrite. Appuyer sur l'ampoule à gauche, puis appuyer sur les touches Ctrl + la barre d'espace, la méthode est ainsi générée.
Eclipse détermine si la méthode à générer possède un paramètre de retour ou non.

Méthode sans paramètre de retour :



Méthode avec paramètre de retour :



Les instructions

Ctrl + espace -> liste des paramètres, des variables locales, des attributs puis des méthodes que vous pouvez utiliser.
Afin de limiter le choix, taper la première lettre de ce que vous recherchez.

syso Ctrl + espace -> System.out.println();

Autres...
Sauvegarder l'ensemble des fichiers : Ctrl + Shift + S

Ecrire le code des classes vu en TD

En utilisant les différentes techniques ci-dessus, écrire les différentes classes vues en TD. Une démonstration vous est proposée dans la vidéo 3-3-coderRapidement.mp4 disponible sous Moodle.

Les classes à implémenter sont données ci-dessous :

```
public class Gaulois {
   private String nom;
   private int force;
   private int effetPotion = 1;

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

   @Override
   public String toString() {
     return "Gaulois [nom=" + nom + ", force=" + force + ", effetPotion="
          + effetPotion + "]";
   }

   public static void main(String[] args) {
     //TODO créer un main permettant de tester la classe Gaulois
   }
}
```

```
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 + " : ";
    }

    public void recevoirCoup(int forceCoup) {
        force -= forceCoup;
        if (force > 0) {
            parler("Aïe");
        } else {
            parler("J'abandonne...");
        }
    }
}

public class Druide {
    private String nom;
    private int effetPotionMin;
    private int effetPotionMax;

    public Druide(String nom, int effetPotionMin, int effetPotionMax) {
        this.nom = nom;
        this.effetPotionMin = effetPotionMin;
        this.effetPotionMax = effetPotionMax;
        parler("Bonjour, je suis le druide " + nom
            + " et ma potion peut aller d'une force " + effetPotionMin + " à "
            + effetPotionMax + ".");
    }

    public String getNom() {
        return nom;
    }

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

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

```
public class Village {
    private String nom;

    public Village(String nom) {
        this.nom = nom;
    }

    public String getNom() {
        return nom;
    }
}
```

```
public class Chef {
    private String nom;
    private int force;
    private Village village;

    public Chef(String nom, int force, Village village) {
        this.nom = nom;
        this.force = force;
        this.village = village;
    }

    public String getNom() {
        return nom;
    }

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

    private String prendreParole() {
        return "Le chef " + nom + " du village " + village.getNom() + " : ";
    }

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

Baston chez les gaulois !

Cette partie va nous permettre de créer un scénario dans lequel les gaulois vont se battre contre les romains.

1. *La méthode toString*

Dans la classe Gaulois, mettre en commentaire la méthode toString :
- sélectionner la méthode complète,
- appuyer sur les touches : Ctrl + Shift + /

Dans la méthode main :

a. Créer l'objet asterix de la classe Gaulois correspondant à un gaulois ayant comme nom "Astérix" et de force 8.

b. Ajouter la ligne de code suivante : `System.out.println(asterix);`

Appuyer sur le bouton run 

Le résultat est de la forme :

nomDuPaquage.nomDeLaCLasse@adresseMemoire

par exemple : `personnages.Gaulois@15db9742`

Il s'agit de l'affichage d'un objet entier.

c. Modifier l'instruction `System.out.println(asterix);` afin d'obtenir le nom de l'objet.

d. Décommenter la méthode toString (exactement de la même manière que vous l'avez commentée.

Ajouter la ligne de code suivante : `System.out.println(asterix);`

Appuyer sur le bouton run

Cette fois l'affichage est : `Gaulois [nom=Astérix, force=8]`

Conclusion : La méthode toString permet de transformer l'affichage de l'objet sous sa forme nomDuPaquage.nomDeLaCLasse@adresseMemoire en chaine de caractère correspondant à l'état de l'objet.

2. *Vérification du bon fonctionnement des méthodes*

a. Toujours dans le main de la classe Gaulois, écrire les instructions nécessaires à la vérification du fonctionnement des méthodes :

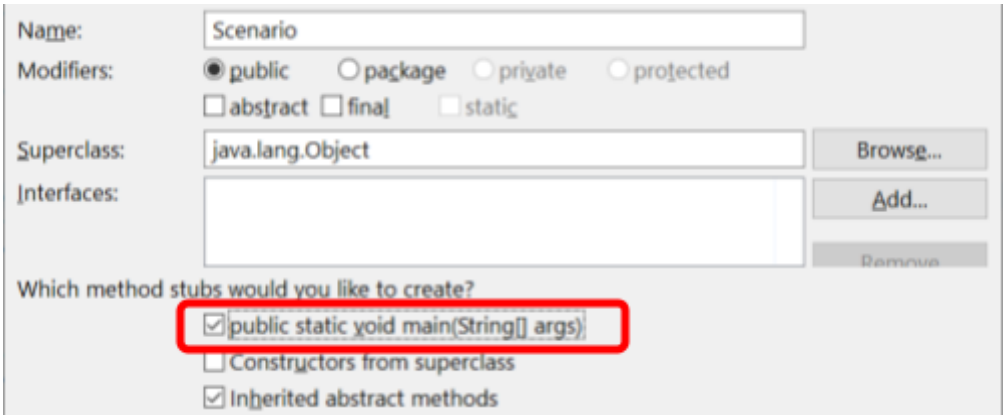
- prendreParole
- parler
- frapper

b. A la fin de la classe Romain, générer la méthode main et vérifier les méthodes :

- prendreParole
- parler
- recevoirCoup

c. Création d'un scénario

- créer un nouveau package "histoire"
- dans ce package, créer une nouvelle classe "Scenario" en cochant la case main.



- Ecrire dans le main les inscriptions permettant d'obtenir le scénario suivant (Astérix à une force de 8 et Minus une force de 6) :

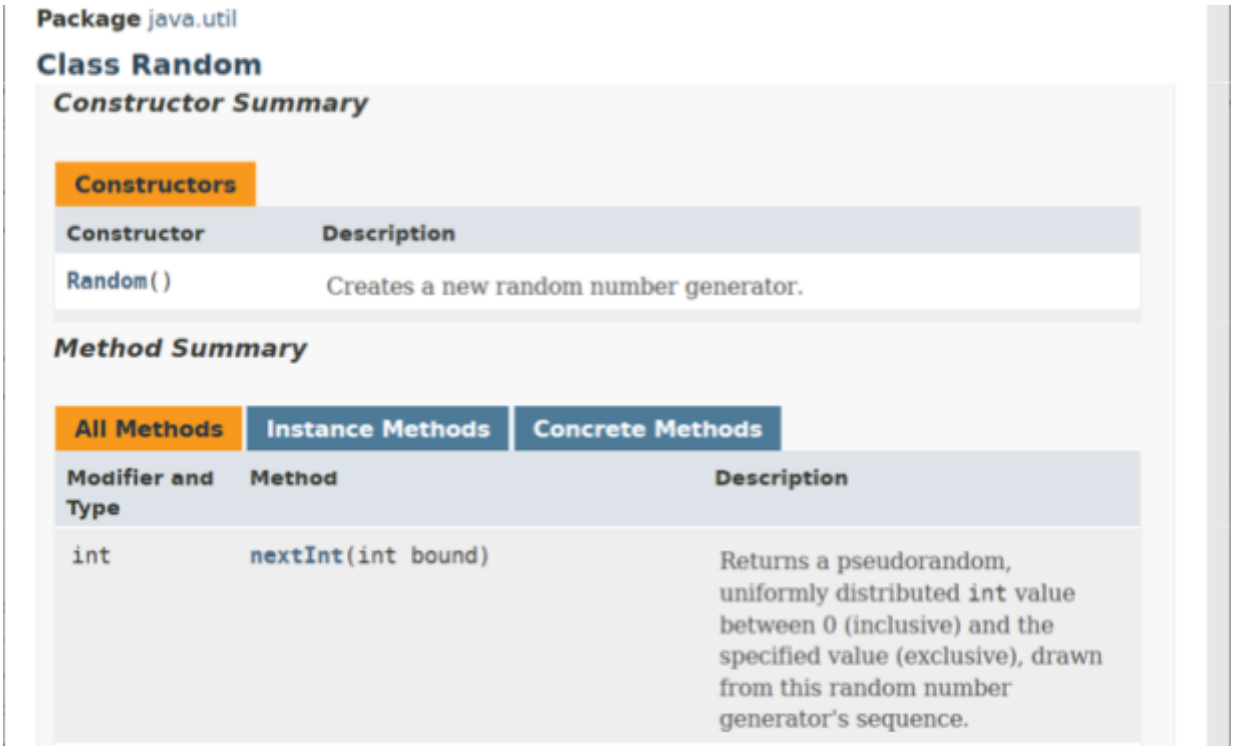
Le gaulois Astérix : « Bonjour à tous»
Le romain Minus : « UN GAU… UN GAUGAU…»
Astérix envoie un grand coup dans la mâchoire de Minus
Le romain Minus : « Aïe»
Astérix envoie un grand coup dans la mâchoire de Minus
Le romain Minus : « Aïe»
Astérix envoie un grand coup dans la mâchoire de Minus
Le romain Minus : « J'abandonne…»

3. *Le druide prépare la potion*

Dans la classe Druide créer :

- l'attribut forcePotion : un entier initialisé à 1
- la méthode *preparerPotion*. Pour cette méthode nous allons utiliser la classe Random de la bibliothèque util de Java.

Extrait de la JavaDoc :



Package java.util

Class Random

Constructor Summary

Constructors

Constructor	Description
`Random()`	Creates a new random number generator.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
`int`	`nextInt(int bound)`	Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html

a. Dans la méthode *preparerPotion :*

- créer un objet random de type Random (attention bien importer la classe Random de « java.util » et non « java.math »). Puis utiliser la méthode *nextInt* avec en paramètre d'entrée l'attribut *effetPotionMax*. Le nombre retourné par la méthode *nextInt* sera la force de la potion préparée par le druide.
- faire parler le druide :
  - si la force de la potion est supérieure à 7 le druide dira : « J'ai préparé une super potion de force » puis donnera la force de la potion.
  - sinon il dira : « Je n'ai pas trouvé tous les ingrédients, ma potion est seulement de force » puis donnera la force de la potion.

b. Dans la classe Druide créer un main :
- Créer le druide Panoramix qui prépare une potion entre 5 et 10.
- Tester la méthode *preparerPotion* : lancer le test plusieurs fois. Vous devez obtenir une force de potion qui varie entre 0 et 9.

c. Nous souhaitons que la force de la potion varie entre les valeurs des attributs *effetPotionMin* et *effetPotionMax* (entre 5 et 10 pour notre druide).
- Modifier la méthode *preparerPotion* afin d'obtenir une force adéquate.
- Tester

4. Le gaulois bois la potion

a. Dans la classe Gaulois :
   - rappel : l'attribut *effetPotion* est initialisé à 1
   - créer la méthode *boirePotion* qui prend en paramètre d'entrée la force de la potion et l'affecte à l'attribut *effetPotion*.
     Le gaulois dit : « Merci Druide, je sens que ma force est N fois décuplée. » si la force de la potion est de N.
   - Modifier la méthode frapper pour prendre en compte l'effet de la potion : la force du coup sera la force du gaulois divisée par 3 et multipliée par l'effet de la potion.

b. Dans le main de la classe Gaulois, tester la méthode *boirePotion*.

5. Le druide booste les guerriers

a. Dans la classe "Druide" créer la méthode *booster* qui prend en paramètre d'entrée un gaulois à qui il fait boire la potion.

b. Modifier la méthode *booster* pour que le gaulois nommé « Obélix » ne puisse pas avoir de potion : le druide dit « Non, Obélix !... Tu n'auras pas de potion magique ! ».

6. Compléter le scénario

Sachant que minus à une force de 6, astérix une force de 8 et Obélix une force de 25, modifier le scénario de la classe Scenario pour obtenir la sortie suivante :

Le druide Panoramix : « Bonjour, je suis le druide Panoramix et ma potion peut aller d'une force 5 à 10.»
Le druide Panoramix : « Je vais aller préparer une petite potion...»
Le druide Panoramix : « Je n'ai pas trouvé tous les ingrédients, ma potion est seulement de force 6.»
Le druide Panoramix : « Non, Obélix !... Tu n'auras pas de potion magique !»
Le gaulois Obélix : « Par Bélénos, ce n'est pas juste !»
Le gaulois Astérix : « Merci Druide, je sens que ma force est 6 fois décuplée.»
Le gaulois Astérix : « Bonjour»
Le romain Minus : « UN GAU... UN GAUGAU...»
Astérix envoie un grand coup dans la mâchoire de Minus

Le romain Minus : « J'abandonne…»