Classes Internes - Exceptions

Préparation de l'environnement

Dans les TPs nous utiliserons obligatoirement :

- l'IDE Eclipse avec le plugin SonarLint,
- l'historisation du développement avec Git.

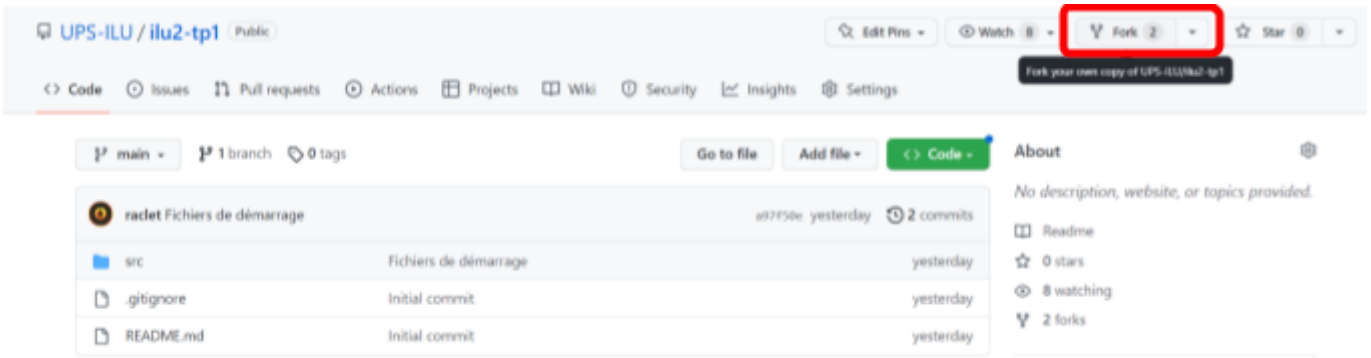
Assurez-vous d'avoir les outils nécessaires (IDE Eclipse, terminal pour l'utilisation de GIT, un compte GitHub), sinon vous trouverez les documents d'installation sur Moodle.

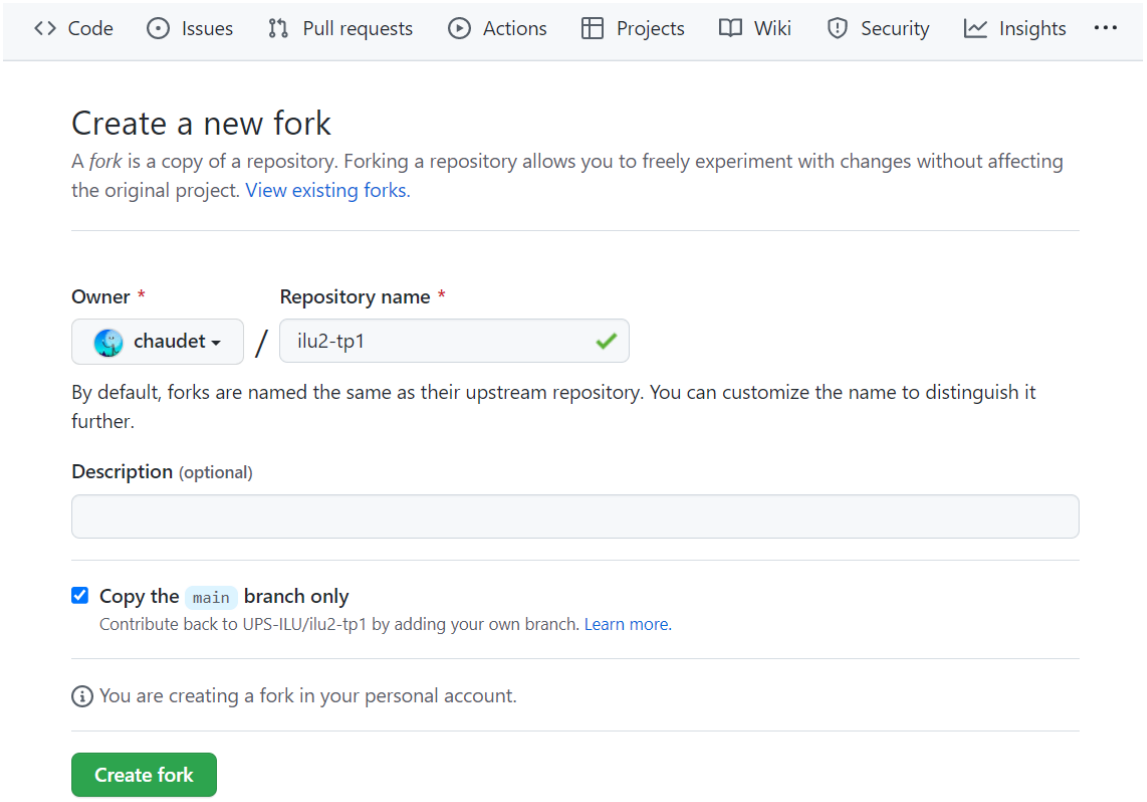
Récupérer un projet existant sur Github

1 - Connectez-vous à votre compte GitHub  chaudet <- votre pseudo

2 - Allez à l'adresse suivante : https://github.com/UPS-ILU/ilu2-tp1

3 - Faire un Fork du projet :





Cliquez sur Create fork

Sur votre compte github vous avez maintenant le projet ilu2-tp1

4 - Reprendre les étapes vues en ILU1 pour faire une copie local de votre projet distant (git clone). Si vous ne vous souvenez plus des étapes suivre le document "Utilisation de github - rappel ILU1" sous Moodle

3 Les Gaulois

Le dossier source du projet ilu2-tp1 contient :

- le paquetage personnages avec les classes :
  - Personnage,
  - Gaulois,
  - Chef,
  - Druide
- le paquetage villagegaulois avec les classes :
  - Village
  - Etal
- le paquetage histoire avec la classe Scenario.

Dans ce TP nous allons travailler avec la classe Village.

```
public class Village {
  private String nom;
  private Chef chef;
  private Gaulois[] villageois;
  private int nbVillageois = 0;

  public Village(String nom, int nbVillageoisMaximum) {
    this.nom = nom;
    villageois = new Gaulois[nbVillageoisMaximum];
  }

  public String getNom() {
    return nom;
  }

  public void setChef(Chef chef) {
    this.chef = chef;
  }

  public void ajouterHabitant(Gaulois gaulois) {
    if (nbVillageois < villageois.length) {
      villageois[nbVillageois] = gaulois;
      nbVillageois++;
    }
  }
```

```
  public Gaulois trouverHabitant(String nomGaulois) {
    if (nomGaulois.equals(chef.getNom())) {
      return chef;
    }
    for (int i = 0; i < nbVillageois; i++) {
      Gaulois gaulois = villageois[i];
      if (gaulois.getNom().equals(nomGaulois)) {
          return gaulois;
      }
    }
    return null;
  }

  public String afficherVillageois() {
    StringBuilder chaine = new StringBuilder();
    if (nbVillageois < 1) {
      chaine.append("Il n'y a encore aucun habitant au village du chef "
                + chef.getNom() + ".\n");
    } else {
      chaine.append("Au village du chef " + chef.getNom()
                + " vivent les légendaires gaulois :\n");
      for (int i = 0; i < nbVillageois; i++) {
          chaine.append("- " + villageois[i].getNom() + "\n");
      }
    }
    return chaine.toString();
  }
}
```

Le Marché

Cette partie est réalisable durant la séance de TP. Si vous ne l'avez pas terminée à la fin des deux heures vous devrez la terminer chez vous avant la séance suivante.

Environnement - classe Etal

Le marché se trouve au centre du village, et possède plusieurs étals qui pourront être prêtés au villageois le temps qu'ils réalisent leur vente. Le nombre d'étal est défini à la création du village.

Les attributs d'un étal sont donc un booléen permettant de savoir si l'étal est occupé ou pas, le vendeur, le produit et la quantité de produit à vendre (en début de marché) ainsi que la quantité restante à vendre (mis à jour après chaque vente).

Un villageois (prenant le rôle d'un marchand) peut :
- occuper un étal `occuperEtal` pour vendre un ou plusieurs produits,
- libérer un étal une fois ses affaires terminées `libereEtal`.

Un villageois (prenant le rôle d'un client) peut :
- voir le contenu d'un étal `afficherEtal`,
- acheter un ou plusieurs produits à un étal d'un marchand `acheterProduit`.

D'autres opérations permettront de gérer ses différentes fonctionnalités :
- `isEtalOccupe`,
- `getVendeur`,
- `contientProduit`.

Prenez le temps de regarder les différentes méthodes de la classe `Etal`.

Classe interne - classe Marche

Seule la classe `Village` parmi les classes données en début de TP doit être modifiée durant ce TP.

1) Créer la classe interne `Marche`. Cette classe possède l'attribut `etals` qui est un tableau de type `Etal`.

2) Ajouter un constructeur prenant en paramètre d'entrée le nombre d'étal du marché, et initialiser le tableau d'objet avec autant d'instance d'`Etal` que nécessaire.

3) Ajouter la méthode `void utiliserEtal(int indiceEtal, Gaulois vendeur, String produit, int nbProduit)` permettant à un gaulois de s'installer à un étal.

4) Ajouter la méthode `int trouverEtalLibre()` permettant de trouver un étal non occupé dans le tableau `etals`. S'il n'y a pas d'étal disponible, la méthode retourne -1.

5) Ajouter la méthode `Etal[] trouverEtals(String produit)` qui retourne un tableau contenant tous les étals où l'on vend un produit.

6) Ajouter la méthode `Etal trouverVendeur(Gaulois gaulois)` qui retourne l'étal sur lequel s'est installé le vendeur passé en paramètre d'entrée ou `null` s'il n'y en a pas.

7) Ajouter la méthode `afficherMarche()` qui retourne une chaîne de caractères contenant l'affichage de l'ensemble des étals occupés du marché. S'il reste des étals vide la chaîne de retour se terminera par : "`Il reste " + nbEtalVide + " étals non utilisés dans le marché.\n`".
   Remarque :
   a) la classe Etal contient la méthode afficherEtal
   b) vous ajouterez la méthode afficherMarche() dans la classe interne Marche
   c) Exemple :

```
Assurancetourix vend 5 lyres
Obélix vend 2 menhirs
Panoramix vend 10 fleurs
Il reste 2 étals non utilisés dans le marché.
```

8) Prendre du recul et réfléchir sur les mots clés static / public / private

Classe englobante - classe Village

La classe `Village` possède un marché, le nombre d'étal qu'il contient est défini à la création du village.

1) Modifier le constructeur de la classe Village afin de créer le marché. Dans la classe `Scenario` décommenter les 3 premières lignes du main.
2) Créer les méthodes dans la classe Village qui interagissent avec la classe interne Marche et qui permettent de gérer les affichages (voir les sorties consoles). Vous devez créer vos chaines à retournées avec la classe `StringBuilder` (vous avez un exemple d'utilisation dans la méthode `afficherVillageois`).

   Tester les méthodes au fur et à mesure (classe `Scenario`), n'attendez pas la fin !
   a) `public String installerVendeur(Gaulois vendeur, String produit,int nbProduit).`
   b) `public String rechercherVendeursProduit(String produit),`
   c) `public Etal rechercherEtal(Gaulois vendeur),`
   d) `public String partirVendeur(Gaulois vendeur),`
   e) `public String afficherMarche()`

Sorties console :

```
Il n'y a pas de vendeur qui propose des fleurs au marché.

Bonemine cherche un endroit pour vendre 20 fleurs.
Le vendeur Bonemine vend des fleurs à l'étal n°1.

Seul le vendeur Bonemine propose des fleurs au marché.

Bonemine cherche un endroit pour vendre 20 fleurs.
Le vendeur Bonemine vend des fleurs à l'étal n°1.

Assurancetourix cherche un endroit pour vendre 5 lyres.
Le vendeur Assurancetourix vend des lyres à l'étal n°2.

Obélix cherche un endroit pour vendre 2 menhirs.
Le vendeur Obélix vend des menhirs à l'étal n°3.

Panoramix cherche un endroit pour vendre 10 fleurs.
Le vendeur Panoramix vend des fleurs à l'étal n°4.

Les vendeurs qui proposent des fleurs sont :
- Bonemine
- Panoramix
```

rechercherVendeursProduit

installerVendeur

rechercherVendeursProduit

installerVendeur

rechercherVendeursProduit

```
Abraracourcix veut acheter 10 fleurs à Bonemine. Abraracourcix, est ravi de
tout trouver sur l'étal de Bonemine

Obélix veut acheter 15 fleurs à Bonemine, comme il n'y en a plus que 10,
Obélix vide l'étal de Bonemine.
Assurancetourix veut acheter 15 fleurs à Bonemine, malheureusement il n'y en
a plus !
```

acheterProduit (classe Etal)

```
Le vendeur Bonemine quitte son étal, il a vendu 20 fleurs parmi les 20 qu'il
voulait vendre.
```

partirVendeur

```
Le marché du village "le village des irréductibles" possède plusieurs étals :
Assurancetourix vend 5 lyres
Obélix vend 2 menhirs
Panoramix vend 10 fleurs
Il reste 2 étals non utilisés dans le marché.
```

afficherMarche

Les exceptions

Dans ce TP, nous pouvons utiliser toutes les opérations dans l'ordre que nous souhaitons, et selon l'ordre des instructions il peut y avoir des incohérences, par exemple :

```
Etal etalFleur = village.rechercherEtal(bonemine);
```

pourrait retourner `null` si Bonemine ne s'est pas installée à un étal.

L'instruction suivante :

```
System.out.println(etalFleur.acheterProduit(10, abraracourcix));
```

lancerait une exception :

```
Exception in thread "main" java.lang.NullPointerException
      at villagegaulois.Village$Marche.trouverVendeur(Village.java:183)
      at villagegaulois.Village.rechercherEtal(Village.java:115)
      at histoire.Scenario.main(Scenario.java:35)
```

Dans le TP suivant nous allons travailler sur l'interface utilisateur et il ne sera plus possible d'acheter un produit à un étal non occupé.

Ces cas ne devant plus se produire nous allons mettre en place une gestion des exceptions.

Classe Etal

Pour cette partie, les modifications sont à effectuer dans la classe Etal (+ d'autres classes si précisé dans le sujet).

1) La méthode `libererEtal`, ne peut fonctionner que si l'étal a précédemment été occupé par un vendeur.

   a) en testant l'appel à la méthode `libererEtal` sur un étal n'ayant pas été occupé, déclencher l'exception que vous souhaitez gérer.
   Par exemple créer une nouvelle classe `ScenarioCasDegrade` dans le paquetage `histoire` et créer le main suivant :

```
public static void main(String[] args) {
      Etal etal = new Etal();
      etal.libererEtal();
      System.out.println("Fin du test");
}
```

   b) vérifier le type de l'exception afin d'être sûr que vous pouvez gérer ce type d'exception. javadoc :
   https://docs.oracle.com/en/java/javase/19/docs/api/index.html.
   Entrer l'exception dans le champ search.

   c) gérer l'exception : trouver le bloc qui devra être interrompu et gérer l'exception au sein de la méthode.

2) Pour la méthode `acheterProduit` il y a 3 conditions : que l'étal soit occupé, que la quantité soit positive et que l'acheteur ne soit pas null.

   a) commençons par "l'acheteur ne doit pas être null"
      i) supprimer la condition (`if(etalOccupe)`)
      ii) identifier l'exception (en modifiant la classe `ScenarioCasDegrade`),
      iii) vérifier son type,
      iv) gérer l'exception au sein de la méthode en affichant la pile d'erreurs sur la sortie d'erreur et en retournant une chaîne vide.

   b) "la quantité doit être positive"
      i) cela signifie que le paramètre d'entrée n'est pas "légal" donc on va lever l'exception `IllegalArgumentException` si la quantité est inférieur à 1,
      ii) gérer l'exception dans la méthode appelante (dans la classe `ScenarioCasDegrade` puis si tout se passe bien dans la classe `Scenario`)

   c) "l'étal doit être occupé".
      i) cela signifie qu'il y a une erreur d'appel à la méthode (on veut acheter à un étal vide !) donc lever l'exception `IllegalStateException` si l'étal n'est pas occupé,

ii) gérer l'exception dans la méthode appelante (dans la classe `ScenarioCasDegrade` puis si tout se passe bien dans la classe `Scenario`)

Classe Village

Le constructeur de la classe Chef a pour paramètre d'entrée le village qu'il dirige, le constructeur de la classe Village ne peut donc avoir lui-même le village en paramètre d'entrée. Pourtant le village ne peut exister sans chef.

1) créer l'exception personnalisée `VillageSansChefException`,
2) reprendre la méthode `afficherVillageois` de la classe `Village` qui lancera l'exception personnalisée s'il n'y a pas de chef et la transmettra à la méthode appelante.
3) Cette méthode appelante devra gérer l'exception.