# La Rochelle Université
# Licence Informatique
# Génie Logiciel - Qualité Logicielle et Gestion de Projet

**TP1 - Cas d'étude "*Jeu de rôle*"**
**Application des design patterns `Factory Method`, `Singleton` et `Observer`**

## 1 Comment réaliser ce TP

Pour réaliser ce TP, vous utiliserez l'outil Visual Paradigm disponible sur les VM Linux de l'Université (image ULR - Ubuntu 22.04). Vous pouvez également installer la version Community Edition sur vos machines personnelles. Le développement en Java sera réalisé sur l'IDE de votre choix.

Le dépôt de vos travaux dans un fichier `nom_prenom.zip` sera à réaliser pour **dimanche 9 mars 18h**. Les dépôts effectués par mail ne seront pas pris en compte et une note de 0 sera appliquée. Ce TP est à réaliser individuellement.

```
nom_prenom.zip/
├── nom_prenom.pdf (contenant tous les diagrammes)
└── factory-vs-wild/ (contenant votre projet Maven)
```

## 2 Correction du DM1

### 2.1 Gestion des utilisateurs

Vous développez une application de gestion d'utilisateurs. Vous avez besoin d'une classe `UserManager` qui ne doit être instanciée qu'une seule fois dans tout le programme.

Le code de la classe est donné ci-dessous :

```java
public class UserManager {
    private static UserManager instance;
    private List<String> users;

    private UserManager() {
        this.users = new ArrayList<>();
    }

    public static UserManager getInstance() {
        if (instance == null) {
            instance = new UserManager();
        }
        return instance;
```

```
    }

    public void addUser(String user) {
        users.add(user);
    }

    public void displayUsers() {
        for (String user : users) {
            System.out.println(user);
        }
    }
}
```

**1.** Donnez le diagramme de classes de ce code.

**2.** Écrire le `main` qui devra :

1. Récupérer une instance de `UserManager` que vous stockerez dans un objet nommé `userManager1`.
2. Ajouter un utilisateur sur l'objet `userManager1`.
3. Récupérer une instance de `UserManager` que vous stockerez dans un objet nommé `userManager2`.
4. Ajouter un utilisateur sur l'objet `userManager2`.
5. Afficher les utilisateur de `userManager1`.
6. Afficher les utilisateur de `userManager2`.

**3.** Quel résultat obtenez-vous ? Commentez le.

## 3 Jeu de rôle - Description du cas d'étude

Dans le jeu de rôle *Factory vs. Wild*, les personnages et les monstres interagissent dans un monde dynamique. Chaque entité possède des caractéristiques spécifiques (vie, attaque, défense) et peut être affectée par des événements (prise de dégâts, état de santé, bonus d'équipement).

Les informations sur les personnages et les monstres sont enregistrées dans un fichier texte *characters_and_monsters.txt* dont voici un exemple :

```
# Format : Type ; Nom ; TypePersonnage ; HP ; ATK ; DEF ; Compétence
Personnage ;Arthas ;Guerrier ;150 ;20 ;10 ;Coup de bouclier (Étourdit l'ennemi)
Personnage ;Morgana ;Mage ;100 ;30 ;5 ;Boule de feu (Dégâts en zone)
Personnage ;Legolas ;Archer ;120 ;25 ;8 ;Tir perçant (Ignore l'armure)
Personnage ;Gandalf ;Mage ;130 ;35 ;4 ;Invocation d'esprit (Invoque une créature alliée)

# Format : Type ; Nom ; TypeMonstre ; HP ; ATK ; DEF ; Faiblesse ; Résistance
Monstre ;DragonRouge ;Dragon ;300 ;40 ;20 ;Glace ;Feu
Monstre ;OrcSanguinaire ;Orc ;180 ;25 ;10 ;Magie ;Physique
Monstre ;GolemDePierre ;Golem ;250 ;15 ;30 ;Eau ;Physique
Monstre ;SpectreNoir ;Spectre ;120 ;35 ;5 ;Lumière ;Ténèbres
Monstre ;Zombie ;Mort_Vivant ;100 ;10 ;5 ;Feu ;Poison
```

### 3.1 Mise en place du projet

Vous trouverez sur Moodle une archive `factory-vs-wild.zip` contenant un projet Maven qui vous sera utile pour plusieurs semaines.

Vérifiez que la compilation fonctionne et que l'exécution des tests ne pose pas de problème (testé sur la VM linux le 20 février et tout fonctionnait) :

```
mvn compile
mvn test
```

Pour exécuter le `main` de votre projet qui se trouve dans la classe `App`, exécutez la commande suivante :

```
mvn compile exec:java -Dexec.mainClass="fr.lru.App"
```

### 3.2 Définition et implémentation des classes métiers

En vous basant sur le cahier des charges succinct présenté ci-dessus, concevez un diagramme de classes qui modélise l'ensemble des entités du jeu au sein du package `jeu`, puis implémentez ces classes en Java.

## 4 Mise en place du design pattern `Factory Method`

On souhaite mettre en place le système capable de charger en mémoire les listes des personnages et des monstres par lecture du fichier en utilisant le design pattern *Factory Method*. Ici, la fabrique va créer les instances des personnages et des monstres à partir de chacune des lignes du fichier.

A partir de ce que vous avez vu en CM et en TD :

— Modifiez votre diagramme de classes pour intégrer la fabrique, en créant un package `fabrique`.

— Proposez une implémentation Java de ce pattern (la lecture du fichier doit fonctionner).

## 5 Mise en place du design pattern `Singleton`

Modifiez votre diagramme de classes et votre code pour n'autoriser qu'une seule instanciation de votre fabrique.

## 6 Mise en place du design pattern `Observer`

Nous aborderons le design pattern `Observer` lors du prochain CM. Cet exercice va vous permettre de l'appréhender par vous même.

On souhaite mettre en place un mécanisme qui déclenche des événements (ex : changement de météo, boss invoqué, etc.). Ce mécanisme est implémenté dans la classe `GameEventManager`, qu'il faut compléter. Dès qu'un événement se produit, l'ensemble des entités doivent recevoir une notification. Le pattern le plus proche de ce besoin est le pattern *observer*.

— Réalisez des recherches sur internet pour comprendre la structure de ce pattern ainsi que son fonctionnement.

— Dans notre exemple, quelle classe doit être l'observable et quelle classe doit être l'observer ?

— Ajoutez les éléments de ce design pattern au diagramme de classes en intégrant un nouveau package ;

— Implémentez ce pattern en java.