

Objectif du TP

Cet exercice doit être développé dans le projet **approche-objet**

Dans ces exercices vous devrez respecter les principes de la POO :

- Encapsulation des données (variables privées avec getters et setters).
 - Attention : Les setters ne seront pas toujours recommandés, notamment dans le cas de classes contenant des variables liées.
- Association
- Héritage

Dans les exercices on ne précisera pas toujours les types des attributs d'instances. Ce sera à vous de choisir les types appropriés.

Exercice Ampoule

- Dans le package **fr.diginamic.entites**, créez une classe Ampoule
 - cette classe a un seul attribut d'instance : son état de type booléen.
 - Si état=false alors l'ampoule est éteinte
 - Si état=true alors l'ampoule est allumée
 - Créez un constructeur sans paramètre.
 - Par défaut ce constructeur positionne l'état de l'ampoule à false (éteint)
 - Créez une méthode clic() qui change l'état de l'ampoule. La méthode « allume » l'ampoule si elle est éteinte, et éteint l'ampoule si elle est allumée.
 - Redéfinissez la méthode toString() de manière à ce que cette dernière retourne l'état de l'ampoule sous la forme d'une chaîne de caractères
 - Si état=false alors la méthode retourne « Eteinte »
 - Si état=true alors la méthode retourne « Allumée »
- Dans le package **fr.diginamic.essais**, créez une classe TestAmpoule exécutable
 - Instanciez une Ampoule
 - Affichez son état
 - Invoquez la méthode clic()
 - Affichez à nouveau son état

Exercice Cercle

- Dans le package **fr.diginamic.entites**, créez une classe **Cercle**
 - cette classe a un seul attribut d'instance : son rayon de type double
 - Créez un constructeur pour cette classe avec le rayon en paramètre.
 - Créez une méthode qui retourne le périmètre du cercle
 - Créez une méthode qui retourne la surface du cercle.
- Dans le package **fr.diginamic.essais**, créez une classe **TestCercle**
 - Instanciez 2 cercles différents et affichez les résultats des méthodes de calcul de périmètre et de calcul de surface

Exercice Facture

- Dans le package **fr.diginamic.entites**, créez une classe **Facture**
 - cette classe a 3 attributs d'instance :
 - nbKwh : qui représente le nombre de kwh d'électricité consommé
 - montantTTC : qui représente le prix TTC de la facture ($=nbKwh * 0.15$)
 - montantHT : qui représente le prix HT de la facture ($=80\%$ du prix TTC)
 - Créez un constructeur avec uniquement le nombre de kwh consommé initial et calcule les montants TTC et HT initiaux.
 - Créez une méthode **addKwh(int nb)** qui permet **d'ajouter** un nombre de kwh consommé à la facture et qui recalcule les montants HT et TTC.
 - Redéfinissez la méthode toString() de manière à ce que cette dernière retourne les différentes informations de la facture sous forme de chaîne de caractères.
- Dans le package **fr.diginamic.essais**, créez une classe **TestFacture** exécutable
 - Instanciez une Facture avec une valeur initiale
 - Affichez les informations associées à la facture
 - Ajoutez un nb de kwh
 - Affichez les informations associées à la facture et vérifiez que les montants ont correctement évolué.

Exercice Date

- Dans le package **fr.diginamic.entites**,
 - créez une classe **Date** qui contient 3 attributs : jour, mois et année de type int
 - Créez un **constructeur** qui permet de créer une Date avec les 3 attributs.
 - Créez une méthode **format(int type)** qui prend un type en paramètre et retourne une chaîne de caractères :
 - Si type est égal à 1 alors la méthode format retourne une chaîne de caractères avec la date formatée de manière textuelle.
 - Exemple : 12 février 2022 avec le nom du mois au lieu de son numéro
 - Si type est égal à 2 alors la méthode format retourne une chaîne de caractères avec la date au format dd/MM/yyyy
 - Exemple : 12/02/2022
 - Si type est égal à 3 alors la méthode format retourne une chaîne de caractères avec la date au format yyyy/MM/dd
 - Exemple : 2022/02/12
 - Créez une méthode **estAvant(Date date)** qui prend une date en paramètre et qui retourne :
 - true si la date passée en paramètre est avant la date courante
 - false si la date passée en paramètre est après la date courante ou égale à la date courante
 - Créez une méthode **estApres(Date date)** qui prend une date en paramètre
 - true si la date passée en paramètre est après la date courante
 - false si la date passée en paramètre est avant la date courante ou égale à la date courante
 - Créez une méthode **estEgale(Date date)** qui prend une date en paramètre et qui retourne :
 - true si la date passée en paramètre est égale la date courante
 - false dans le cas contraire
- Dans le package **fr.diginamic.essais**, créez une classe TestDates exécutable
 - Testez les 3 méthodes estAvant, estApres et estEgale avec des dates différentes afin de vérifier que ces méthodes retournent des résultats pertinents.
 - Testez la méthode format avec des valeurs différentes de type.

Exercice CercleFactory

- Créez un package **fr.diginamic.utils**
- Dans ce package créez une classe **CercleFactory**
 - cette classe a une méthode **de classe** (static) qui prend en paramètre un double et retourne un Cercle
- dans la classe **TestCercle**, faites appel à la méthode static de **CercleFactory** pour créer vos cercles.
- **PRECISION** : en POO, on appelle **Factory** une classe qui en construit une autre. Il existe aussi le concept de Builder.

Exercice Operations

- Créez un package **fr.diginamic.operations**
- Dans ce package créez une classe **Operations**
 - cette classe a une méthode **de classe calcul** qui prend en paramètre 2 double a et b et un opérateur qui est de type char.
 - Si l'opérateur vaut '+' alors la méthode **calcul** retourne a+b
 - Si l'opérateur vaut '-' alors la méthode **calcul** retourne a-b
 - Faites la même chose pour les opérateurs *et /
- Créez un package **fr.diginamic.essais**
- Dans ce package, créez une classe **TestOperations** qui permet de tester les 4 opérations.

Exercice CalculMoyenne

- Dans le package **fr.diginamic.operations** créez une classe **CalculMoyenne**
 - Cette classe a un **attribut d'instance** de type tableau de double.
 - Le constructeur de la classe **CalculMoyenne** n'a pas de paramètre
 - Cette classe a également une **méthode ajout** qui permet d'ajouter un double au tableau. Vous aurez besoin de gérer l'agrandissement du tableau.
 - Enfin cette classe a une méthode **calcul** qui ne prend pas de paramètre et retourne la moyenne des éléments du tableau.
- Dans le package **fr.diginamic.essais**, créez une classe **TestMoyenne**
 - Vérifiez que votre classe **CalculMoyenne** correctement en effectuant au moins 2 tests différents.
- **PRECISION** : c'est une bonne pratique d'avoir le réflexe de faire des classes utilitaires de ce type.

Exercice Theatre

- Dans le package **fr.diginamic.entites**, créez une classe **Theatre**
 - cette classe a 4 attributs d'instance :
 - son nom
 - sa capacité max (en nb de personnes)
 - le total de clients inscrits
 - la recette totale de l'établissement
 - Dans la classe **Theatre**, créez une méthode **inscrire** qui prend en paramètres :
 - le nombre de clients
 - le prix de la place
 - Cette méthode effectue le traitement suivant :
 - Si la capacité max du théâtre n'est pas atteinte, elle met à jour le nombre total de clients inscrits ainsi que la recette totale de l'établissement
 - Si la capacité max est atteinte, elle affiche un message d'erreur.
- Dans le package **fr.diginamic.essais**, créez une classe **TestTheatre**
 - Créez une instance de Theatre et appelez plusieurs fois la méthode jusqu'à obtention du message d'erreur
 - Affichez le total de clients inscrits
 - Affichez la recette totale de l'établissement

Exercice ManipulationChaine

Dans ce TP nous allons apprendre à manipuler des chaines de caractères, ce qui est très utile dans de nombreux cas de figures :

- Créez un package **fr.diginamic.chaines**.
- Dans ce package, créez une classe exécutable **ManipulationChaine**
- Dans cette classe, déclarez la chaine de caractères suivante :

```
String chaine = "Durand;Marcel;2 523.5";
```

- Comme vous le constatez, cette chaine stocke des informations séparées par le caractère « ; ». C'est par exemple le format utilisé dans les **fichiers CSV** pour stocker des données.

Tâches à réaliser :

La première tâche est corrigée ci-après afin d'avoir un exemple d'utilisation d'une méthode de la classe String.

- 1) Utilisez la méthode **charAt(int index)** pour afficher le premier caractère de la chaîne de caractères.

Correctif :

```
char premierCaractere = chaine.charAt(0);  
System.out.println("Premier caractère: " + premierCaractere);
```

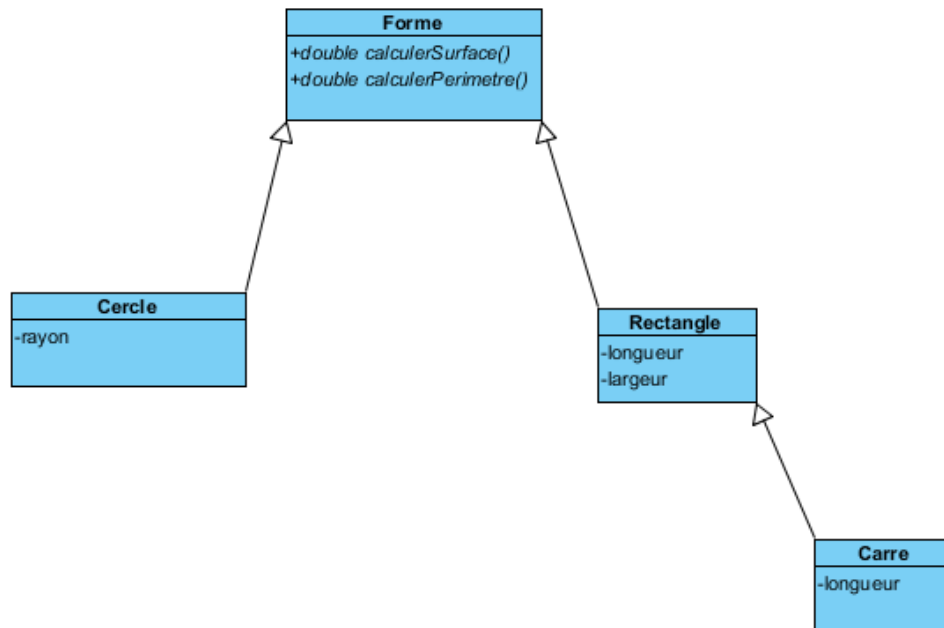
- 2) Utilisez la méthode **length()** pour afficher la longueur de la chaîne de caractères
- 3) Utilisez la méthode **indexOf(char c)** pour afficher l'index du premier « ; » contenu dans la chaîne de caractères.
- 4) La méthode **substring(int start, int end)** permet d'extraire une portion de chaîne de caractères comprise entre un index de début (inclus) et un index de fin (exclu).

Combinez la méthode **substring** et **indexOf** pour extraire le nom de famille de la personne.

- 5) Utilisez la méthode **toUpperCase()** pour afficher le nom de famille en majuscules.
- 6) Utilisez la méthode **toLowerCase()** pour afficher le nom de famille en minuscules.
- 7) Utilisez la méthode **split** pour découper la chaîne de caractères en morceaux. La méthode Le séparateur est le caractère « ; ». Cette méthode retourne un tableau. Affichez le tableau ainsi obtenu.
- 8) Créez une classe **Salarie** dans le package **entites** avec 3 attributs
 - a. nom : String
 - b. prenom : String
 - c. salaire : double
- 9) A partir des 3 morceaux de chaîne de caractères précédents, créez une instance de Salarie :
 - a. Consigne : le nombre « 2 523.5 » contient un espace qu'il ne faut pas supprimer manuellement mais de manière informatique en utilisant la méthode `replace(String, String)` de la classe String
 - b. Astuce : pour transformer une chaîne de caractères en double, utilisez la méthode statique de la classe Double : `Double.parseDouble(String)`

Exercice Forme

- Créez un package **fr.diginamic.formes**
- La classe **Forme** va représenter la classe mère de diverses formes géométriques.
 - cette classe est abstraite
 - elle possède une méthode abstraite **calculerSurface**
 - elle possède une méthode abstraite **calculerPerimetre**
- Voici les autres classes à mettre en place avec leurs attributs :



- Implémentez toutes les classes de ce modèle objet dans le package **fr.diginamic.formes**.
- Dans le package **fr.diginamic.essais**, créez une classe **AffichageForme** :
 - cette classe possède une méthode **afficher**. Cette méthode prend en paramètre une variable de type **Forme** et affiche le périmètre de la forme ainsi que sa surface.
- Créer une classe **TestForme** :
 - Créer une variable de type cercle, une de type rectangle et une de type carré et tester la méthode **afficher** avec ces diverses variables.
- **CONCLUSION** : comme vous le constatez la méthode **afficher** peut prendre en paramètre n'importe quelle instance d'une classe qui hérite de **Forme**. C'est l'essence même du **polymorphisme**.

Exercice CalculSalaire

Dans une application de gestion de la paie d'un journal quotidien « La Voix de Saint-Herblain », on a une hiérarchie d'objets suivants :

- **Intervenant** : classe mère, qui désigne une personne travaillant pour la société,
- **Salarie** : classe fille de la classe **Intervenant**, qui désigne un salarié du journal,
- **Pigiste** : classe fille de la classe **Intervenant**, qui désigne un indépendant intervenant au sein du journal pour une courte durée, parfois pour une seule journée. C'est un statut qu'on retrouve beaucoup dans le milieu médiatique.

Etape 1 : implémentation des classes

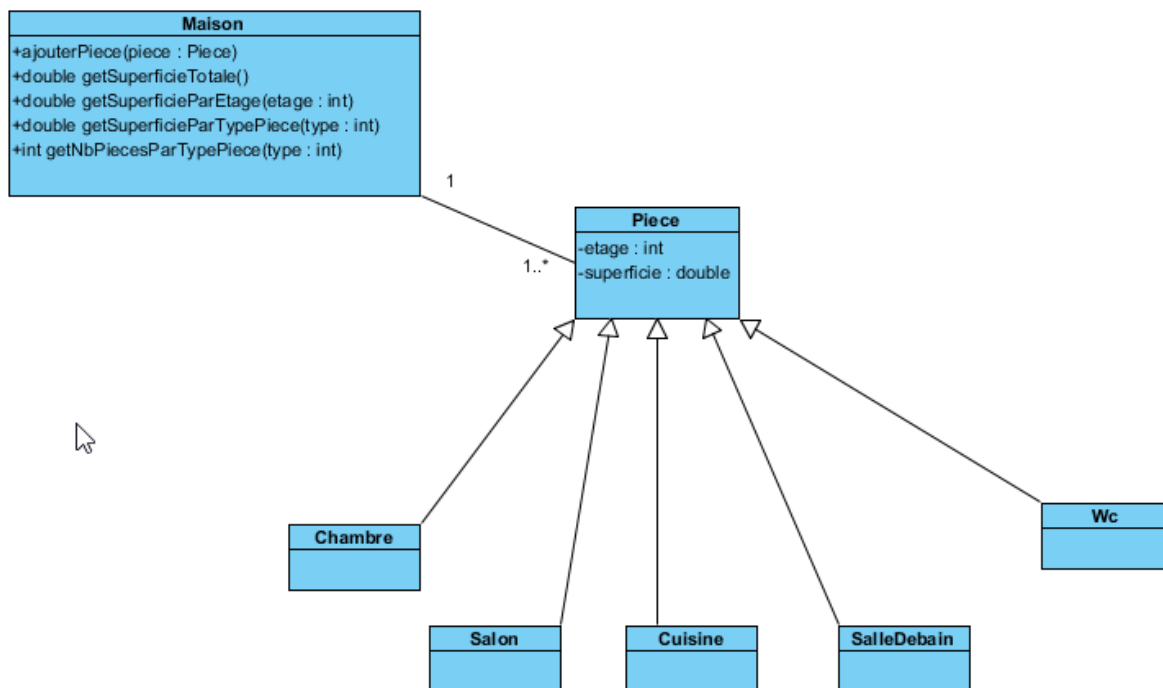
- Créez un package **fr.diginamic.salaire**
- La classe **Intervenant** va représenter la classe mère des diverses personnes travaillant pour le journal
 - cette classe a 2 attributs : **nom** et **prénom**
 - cette classe a une méthode **abstraite getSalaire()**
- La classe **Salarie**:
 - est une classe fille de la classe **Intervenant**.
 - représente un salarié du journal, i.e. ceux qui ont un contrat de travail type CDI ou CDD.
 - a un attribut d'instance : le montant du salaire mensuel
- La classe **Pigiste** représente les personnes payées à la journée. Cette classe a 2 attributs :
 - un attribut qui représente le nombre de jours travaillés pour la société durant le mois
 - un attribut qui représente le montant journalier de rémunération.
- Implémentez la méthode **getSalaire()** pour les 2 classes : **Pigiste** et **Intervenant**.
- Dans le package **fr.diginamic.essais**, développez une classe **TestIntervenant** :
 - Créez une instance de **Salarie** et affichez le résultat retourné par la méthode **getSalaire()**
 - Créez une instance de **Pigiste** et affichez le résultat retourné par la méthode **getSalaire()**

Etape 2 : la méthode afficherDonnees

- La classe **Intervenant** a une nouvelle méthode:
 - Cette méthode s'appelle **afficherDonnees** et affiche toutes les données concernant un intervenant :
 - son nom,

- son prénom,
- son salaire,
- son statut
- Développez cette méthode dans la classe **Intervenant**. Vous aurez sans doute besoin d'une autre méthode pour que cela fonctionne.
- Développez une classe **TestIntervenant** :
 - Pour chacune des instances de salarié et de pigiste invoquez ces méthodes afin de vérifier qu'elles fonctionnent

Exercice Immobilier



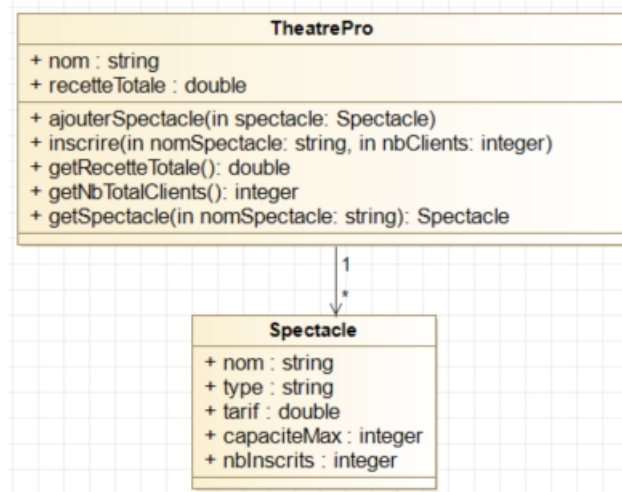
Dans cet exercice nous allons modéliser une **maison** avec ses **diverses pièces** qui peuvent être de types différents : Chambre, Cuisine, Salon, Salle de bain, WC

- Créez un package **fr.diginamic.maison**
- On va commencer par créer une classe abstraite **Piece**, qui a 2 attributs :
 - la superficie
 - le numéro de l'étage. On considèrera par convention que l'étage 0 désigne le RDC, 1 le 1^{er} étage, et ainsi de suite.
- La classe **Piece** a un constructeur avec 2 paramètres permettant d'initialiser les variables d'instance superficie et étage.
- Comme le montre le diagramme de classes, la classe **Piece est la classe mère** de toutes les pièces de la maison. Cette classe mère a **5 classes filles** :
 - Chambre
 - Cuisine

- Salon
- SalleDeBain
- WC
- La classe **Maison** va représenter une maison avec un unique attribut : un tableau d'objets de type **Piece**.
 - cette classe possède une méthode **ajouterPiece(Piece piece)** qui permet d'ajouter une pièce à la maison.
 - cette classe possède une méthode qui retourne la superficie totale de la maison
 - cette classe a une méthode qui retourne la superficie d'un étage donné.
- Dans le package fr.diginamic.essais, créez une classe **TestMaison** qui permet de tester la création d'une maison. Ajoutez des pièces de diverses natures à différents étages et vérifiez que toutes vos méthodes fonctionnent.
- Que se passe t'il si vous passez **null** en paramètre de la méthode **ajouterPiece(Piece piece)** ? Faites un test.
 - Si vous détectez une erreur, ajoutez un contrôle dans la méthode ajouterPiece pour éviter d'ajouter au tableau quelque chose de **null**.
- Que se passe t'il si la pièce a une superficie ou un étage négatif ?
 - Ajoutez un contrôle pour éviter d'ajouter au tableau une pièce avec des données non cohérentes.
- **Plus difficile** : l'écriture des 2 méthodes suivantes demande un peu de réflexion
 - Dans la classe Maison, écrivez une méthode qui **prend en paramètre un type de pièce donné** et retourne la superficie globale pour ce type de pièce donné : par exemple, la superficie globale des chambres.
 - Dans la classe Maison, écrivez une méthode qui retourne le nombre de pièces d'un type donné : par exemple le nombre de chambres.

[DIFFICILE] Exercice TheatrePro

- Nous allons améliorer notre modèle de gestion de théâtre.



- Dans le package **fr.diginamic.entites**, créez une classe **Spectacle**
- Cette classe **Spectacle** a 5 attributs d'instance :
 - Le nom du spectacle
 - Le type de spectacle (concert, comédie, improvisation ou danse)
 - Le tarif unitaire
 - La capacité max du spectacle
 - Nombre d'inscrits
- Créez la classe **TheatrePro** qui possède les attributs d'instance suivants :
 - Un nom
 - Un **tableau de spectacles**
- La classe **TheatrePro** possède une méthode **ajouterSpectacle** qui permet d'ajouter un nouveau spectacle au théâtre.
- La classe **TheatrePro** possède une méthode **getSpectacle** qui prend en paramètre le nom d'un spectacle et retourne le spectacle correspondant.
- La classe **TheatrePro** possède une méthode **inscrire**:
 - Cette méthode prend en paramètres :
 - un nombre de clients
 - le nom du spectacle.
 - Cette méthode ajoute le nombre de clients passés en paramètre au spectacle dont le nom est passé en paramètre.
 - Attention si le spectacle est complet ou si le nom du spectacle est inconnu, la méthode affiche un message d'erreur.

- La classe **TheatrePro** possède une méthode **getRecetteTotale** qui ne prend pas de paramètre et retourne la recette globale de l'établissement
- Enfin la classe **TheatrePro** possède une méthode **getNbTotalClients** qui ne prend pas de paramètre et retourne le nombre total de clients inscrits aux divers spectacles.
- Dans le package **fr.diginamic.essais**, créez une classe **TestTheatrePro**
 - Créez une instance de **TheatrePro** qui possède au moins 1 spectacle de chaque type avec ses attributs propres (nom, capacité max, tarif unitaire)
 - Inscrivez des clients dans divers spectacles
 - Affichez la recette globale
 - Affichez la recette totale de l'établissement

(Difficile) Exercice Parseur

Dans cet exercice nous allons écrire un parseur capable à partir d'une chaîne de caractères de construire une expression mathématique simple.

Le parseur doit être capable de « parser » une expression mathématique du type :

- $x+3.5$
- $3/x$
- $2.5-x$
- $y+2$
- etc...

Une fois le parsing effectué, vous devez être capable d'évaluer l'expression pour une valeur donnée de votre variable. Exemple

- l'évaluation de $3/x$ avec la valeur $x=6$ fournit le $3/6=0.5$
- l'évaluation de $y+2$ avec la valeur $y=2$ fournit le $2+2=4$

Spécifications :

- Les développements seront réalisés dans un package **fr.diginamic.parseur** du projet approche-objet.
- Le parseur est capable de parser n'importe quelle chaîne de caractères contenant une variable (x dans l'exemple), un opérateur ($+$, $-$, $*$ ou $/$) et une constante.
- Le **parseur** retourne un objet de type **Expression**, qui a 3 attributs
 - un membre gauche qui peut être une variable (x par exemple) ou une constante (par exemple 3)
 - un opérateur à choisir parmi $+$, $-$, $*$, $/$
 - un membre droit qui est également soit une variable soit une constante
- L'objet **Expression** a une méthode **evaluer** :
 - Cette méthode a 2 paramètres :
 - le nom de la variable, par exemple x

- la valeur de cette variable, par exemple 1
- Pour évaluer la valeur de l'expression il doit être fait appel aux méthodes évaluer des membres.

Exemple de mise en œuvre :

```
String chaine = "x+3";  
Expression expr = Parser.parse(chaine);  
double resultat = expr.evaluer("x", 2.5); // expression évaluée avec x=2.5  
System.out.println(expr.evaluer("x", 2.5)); // affiche 5.5
```

(Facultatif) Exercice JeuDeRole

Pour ce TP nous allons développer un jeu assez simple inspiré des jeux de rôles.

Dans ce jeu il va y avoir une interaction avec l'utilisateur qui sera donc amené à saisir des actions au clavier.

Pour simplifier le jeu, les actions se feront grâce à un système de menu.

Conseils avant de commencer le développement de ce jeu :

- Créez un package **fr.diginamic.jdr** pour placer l'ensemble des classes de ce jeu.
- imaginez les classes dont vous allez avoir besoin pour créer ce petit jeu.

Menu du jeu :

- Créer le **personnage**
 - lorsqu'on choisit cette option, un personnage va être créé avec 3 attributs :
 - force (entre 12 et 18) tirée aléatoirement
 - points de vie (entre 20 et 50) tirés aléatoirement
 - score (à 0)
- Combattre une **créature**
 - Le choix de cette option va permettre au personnage d'engager un combat. Ce combat n'est possible que si votre personnage a un nombre de points de vie > 0, sinon un message est affiché : « Votre personnage est décédé. Il a obtenu le score de X points. Veuillez créer un nouveau personnage ».
 - lorsqu'on choisit cette option, le personnage doit combattre une créature parmi les créatures suivantes :
 - Un **loup** : force (entre 3 et 8) et points de vie (entre 5 et 10)
 - Un **gobelin** : force (entre 5 et 10) et points de vie (entre 10 et 15)
 - Un **troll** : force (entre 10 et 15) et points de vie (entre 20 et 30).
 - le combat dure jusqu'à ce que votre personnage ou la créature soit victorieuse. Le combat se déroule de la manière suivante :
 - A chaque tour, on calcule l'attaque des 2 protagonistes. l'attaque est calculée de la manière suivante : force + nombre aléatoire entre 1 et 10.
 - Celui dont l'attaque est la plus forte remporte le tour
 - Celui qui remporte le tour inflige une quantité de dégâts égale à la différence entre les 2 valeurs d'attaque calculées précédemment.
 - Cette quantité de dégâts se soustrait au nombre de points de vie de celui qui a perdu le tour.

- Si votre personnage perd, la partie est finie et le score du joueur est affiché.
- Si votre personnage gagne le combat, son score augmente de : 1 si c'est un loup, 2 si c'est un goblin et 5 si c'est un troll. Un message affiche alors l'issue du combat avec le nouveau score.
- Vous pouvez engager un nouveau combat tant que votre personnage est encore en vie. L'objectif du jeu étant d'avoir un score maximum.
- Afficher score
 - cette méthode affiche le score.
- Sortir

Commitez vos développements sur GitHub