TP - J3 & J4 – AUTO-FORMATION

Objectif du TP

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

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 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 /
- Creez un package **fr.diginamic.essais**

- Dans ce package, creez 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.
    - 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 chaine 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 chaine de caractères

3) Utilisez la méthode **indexOf(char c)** pour afficher l'index du premier « ; » contenu dans la chaine de caractères.

4) La méthode **substring(int start, int end)** permet d'extraire une portion de chaine 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 chaine 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

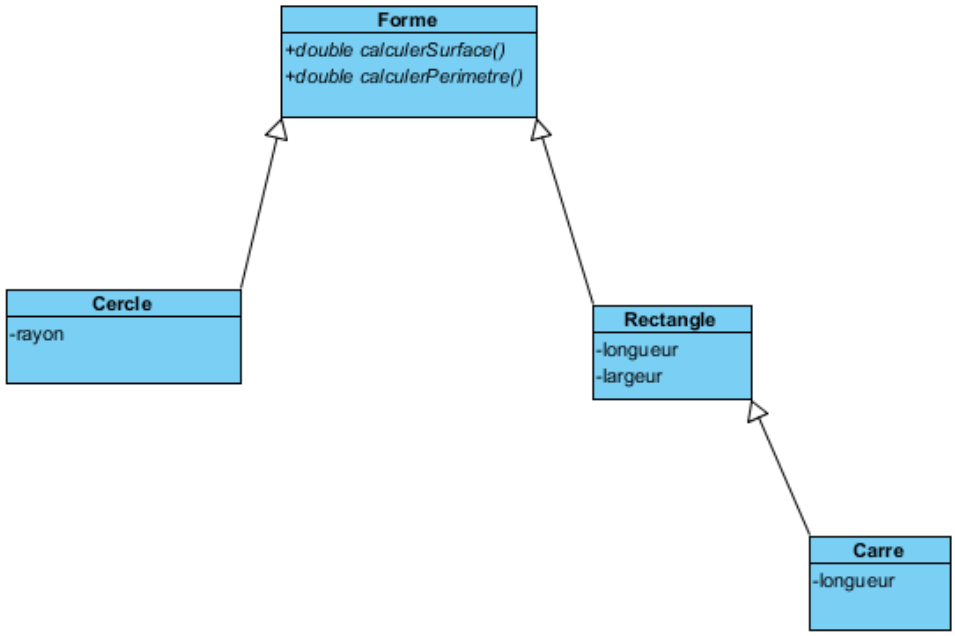
9) A partir des 3 morceaux de chaine de caractères précédents, créez une instance de Salarie :

    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 chaine de caractères en double, utilisez la méthode static 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** qui a un paramètre de type **Forme**. Cette méthode doit afficher le périmètre et la surface de la forme passée en paramètre.
- 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

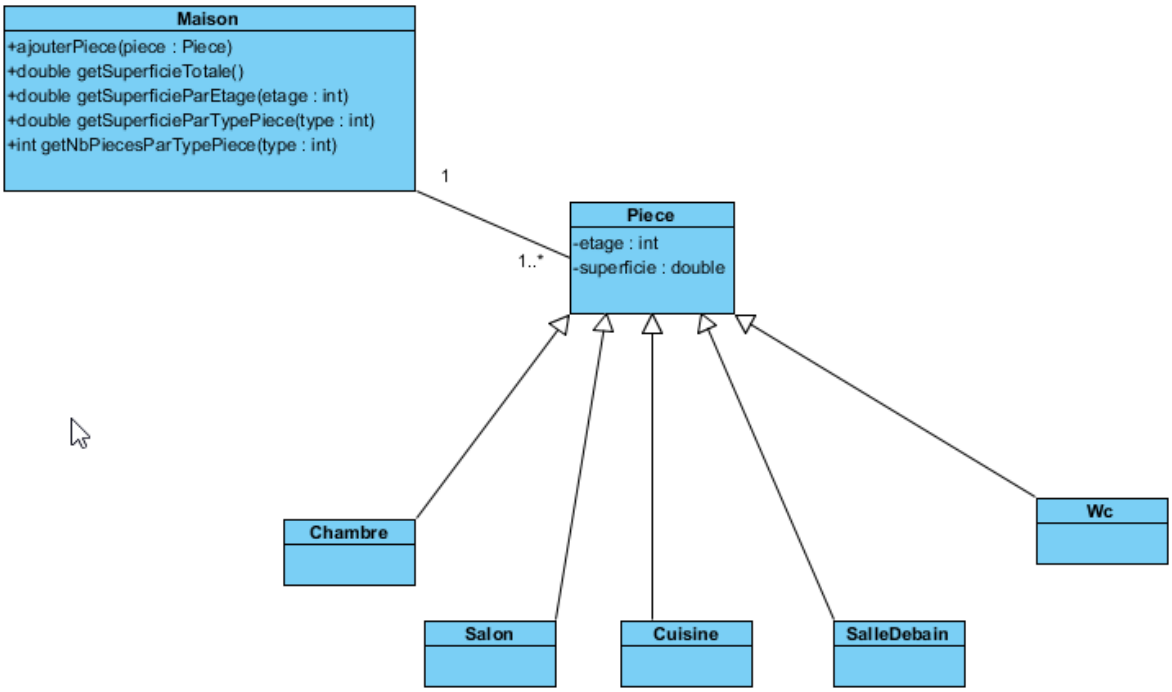
- La classe **Intervenant** a une nouvelle méthode:
  - Cette méthode s'appelle **afficherDonnees** et affiche toutes les données concernant un intervenant :
    - son nom,

- 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 1er 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 à 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.

(Diificile) 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 parmis 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 dégats égale à la différence entre les 2 valeurs d'attaque calculées précédemment.
        - Cette quantité de dégats 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 gobelin 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