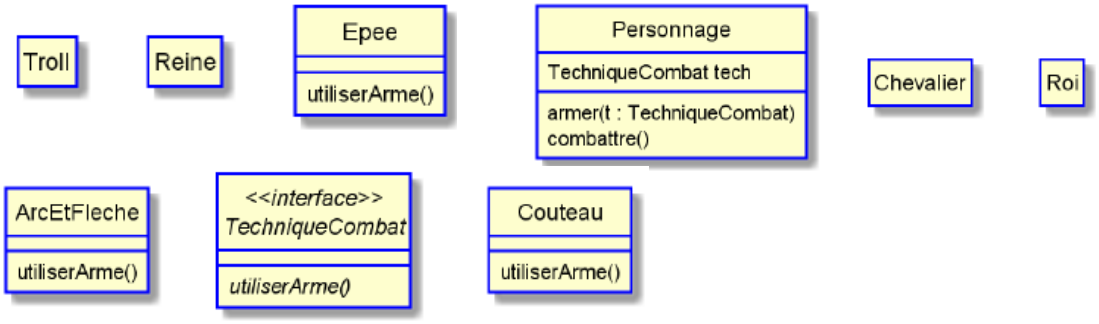
TD/TP3

PREMIERS PATRONS DE CONCEPTION

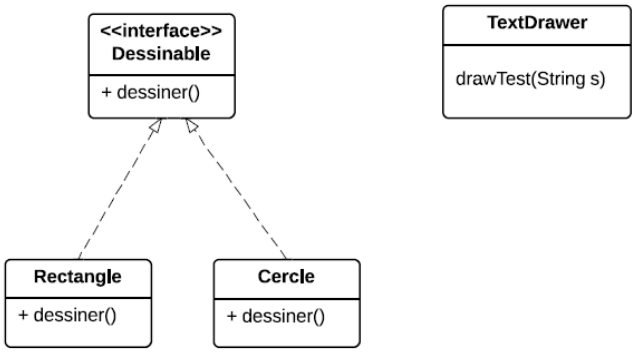
1. ORGANISATION DE CLASSES - STRATEGIE

Replacer les classes suivantes dans un schéma UML conforme au **patron Stratégie**. Coder les méthodes de **Personnage**.



2. INTEGRATION DE CLASSES - ADAPTATEUR

On dispose d'une classe **TextDrawer** capable de dessiner un texte en mode graphique grâce à une méthode **`void drawText(String)`**.



On veut développer un logiciel graphique capable de dessiner différentes formes (classes Rectangle, Cercle) à condition d'implémenter l'interface **Dessinable** qui contient la méthode **`dessine()`**. On souhaite aussi dessiner un texte en utilisant cette interface et la classe **TextDrawer**.

2.1. Compléter le diagramme de classe ci-dessus afin de proposer une solution au problème énoncé.

2.2. Donner les instructions permettant de déclarer un tableau de type **Dessinable** contenant un objet Rectangle, un objet Cercle et un objet qui dessine un texte. Puis, dessiner chaque élément de ce tableau dans une itération

3. PARCOURS - ITERATEUR

L'objectif de cet exercice est de proposer, en utilisant le patron de conception **Itérateur**, des parcours différents d'un tableau à deux dimensions.

3.1. Ecrire une classe **TableauEntier** qui encapsule un tableau à deux dimensions de valeurs entières avec l'interface suivante :

```
public TableauEntier(int[][] t);  / / encapsule le tableau passé en paramètre.

int valeurA(int l, int c) ;  // retourne l'élément ligne l et colonne c.

int getLargeur() ;     // retourne le nombre de colonnes du tableau.

int getHauteur() ;   // retourne le nombre de lignes du tableau
```

3.2. Ecrire une classe abstraite **Parcours** qui implémente l'interface **Iterator<Integer>** (sauf la méthode remove). La méthode **`next()`** utilisera une méthode abstraite **`suivant().`** La classe Parcours aura comme attributs un élément de la classe TableauEntier, les indices courants permettant le parcours du tableau (`ligneCour`, `colonneCour`) ainsi que le nombre courant d'éléments parcourus (`nbParcourus`).

3.3. Ecrire une classe **ParcoursLigne** qui hérite de **Parcours** et permet un parcours par ligne :

----------→
----------→
----------→
----------→

Faire le diagramme des classes de ce programme. **Le faire valider par votre enseignant.**

3.4. Ajouter une méthode **`iterateurLigne()`** à la classe **TableauEntier** qui retourne un objet de type **ParcoursLigne**. Faire une classe de test avec JUnit pour tester cette méthode (pour cela, on ajoutera les éléments parcourus dans une liste et on comparera cette liste à une liste contenant les valeurs attendues).

3.5. Faire un diagramme de séquence correspondant à l'appel de cette méthode dans une classe exécutable. **Le faire valider par votre enseignant.**

3.6. Ecrire une classe **ParcoursZigzag** qui hérite de **Parcours** et permet un parcours en zigzag :

----------→
----------←
----------→
----------←

3.7. Ajouter une méthode **`iterateurZigzag()`** à la classe **TableauEntier** qui retourne un objet de type **ParcoursZigzag**. Faire une classe de test avec JUnit pour tester cette méthode.

3.8. Écrire puis tester une classe **ParcoursColonne** pour un parcours en colonne.

3.9. Modifier la classe **`TableauEntier`** pour la faire implémenter l'interface **`Iterable<Integer>`**. Cette interface nécessite d'implémenter la méthode **`Iterator<Integer> iterator()`** (Retournez un itérateur effectuant un parcours en zigzag).

3.10. Les classes implémentant **`Iterable`** peuvent être utilisées avec la syntaxe des itérations simplifiées. Écrire un test permettant de vérifier que les itérations simplifiées donnent le même résultat que lors de l'utilisation de l'itérateur choisi.