Intro 1: Classes et programmes Java

➙ Nous allons utiliser l'evironnement de développement intégré Éclipse. Malheureusement, celui-ci n'est plus géré sur le réseau de la FdS. Il faut donc l'installer dans votre espace personnel ou le mettre à jour (chaque année !).
Si un répertoire *.p2* (répertoire caché) existe dans votre home, le supprimer. Ensuite, la marche à suivre est disponible à l'adresse suivante :
`https://moodle.umontpellier.fr/mod/page/view.php?id=131074`

➙ Ouvrir Éclipse (attention, il faut choisir celui que vous venez d'installer), et créer un nouveau projet (*File - New - Java Project*), décocher la création de module et nommer votre projet **tp0**.

➙ Ensuite dans ce projet, créer une classe par un clic droit sur le projet, puis *New* et *Class*. Nommer cette classe **Demo** et cocher dans la fenêtre de création l'option *public static void main(String[] args)*. Remarquer que cette classe a été automatiquement placée dans un package nommé *tp0*.

➙ Si besoin, regarder un tutoriel sur Eclipse. Il y en a plein sur internet. Les deux suivant, slides et vidéo, sont bien :
`http://www-igm.univ-mlv.fr/~forax/ens/java-avance/cours/pdf/old/Eclipse%20pour%20les%20null.pdf`
`https://www.youtube.com/watch?v=8xfwuSHUdAM`

Exercice 1 *Coordonnées*

➙ Dans votre projet tp0, créer un nouveau package (*clic droit - New - Package*) que vous appellerez **Exo1**. Dans ce package, créer ensuite deux classes **Cartesienne** et **Polaire**. Aucune de ces classes ne doit contenir une méthode main(), puisqu'une seule méthode main() est autorisée par projet !

➙ Insérer `import Exo1.*;` dans la première ligne de votre classe *Demo*, puis implémenter dans les classes *Cartesienne* et *Polaire* ce qui vous est demandé dans la question suivante. Tester chaque nouvelle méthode dans la classe *Demo*.

Question 1. Modélisez et implémentez en Java 2 classes de coordonnées 2D : une classe pour les coordonnées cartésiennes, et une classe pour les coordonnées polaires. Chacune des classes possèdera :

— une méthode toString qui retourne une phrase contenant les coordonnées du type : les coordonnées cartésiennes de ce point sont : x, y.

— une méthode qui retourne les coordonnées polaires du point pour la classe des coordonnées cartésiennes, et une méthode qui retourne les coordonnées cartésiennes du point pour la classe des coordonnées polaires (on utilisera les fonctions de la classe `Math`, par exemple, on peut faire appel à `Math.sqrt()` ou `Math.atan()` ou `Math.PI`.
— un constructeur adéquate.

On rajoutera une méthode translation dans la classe de coordonnées cartésiennes qui translate le point courant d'un vecteur (x, y) passé en paramètre.

On rappelle que si P a pour coordonnées cartésiennes (x, y), alors les coordonnées polaires de P notées (ρ, θ) sont telles que $\rho = \sqrt{x^2 + y^2}$ et
— si $x > 0$, $\theta = \arctan y/x$
— si $x < 0$, $\theta = \arctan y/x + \pi$
— si x=0, $\theta = 0$

Réciproquement, P a pour coordonnées polaires (ρ, θ), alors les coordonnées cartésiennes de P notées (x, y) sont telles que $x = \rho \cos\theta$ et $y = \rho \sin\theta$.

➙ **À tester :** Jeter un oeil à la JavaDoc. Faire un constructeur sans paramètre, le tester. Afficher un objet directement un objet, sans `toString()`. Faire générer automatiquement une méthode `toString()` (menu *Source*). Essayer d'utiliser dans la classe *Demo* une variable privée de la classe *Cartesienne...*

Exercice 2 *Représentation des cartes dans le jeu "Gang of four"*

➙ Créer un nouveau package **gangOfFour**.

Question 2. Modéliser et implémenter une classe `Carte` qui représente une carte d'un jeu de *Gang of Four*. Cette classe aura pour attributs : sa hauteur et sa couleur. On représentera la hauteur par un entier de 1 à 12 (11 pour les *phoenix* et 12 pour le *dragon*).
Pour la couleur, créer une énumération **Coul**, c'est-à-dire une classe de type `Enum` (sur le package, *clic droit - New - Enum*). La remplir très simplement avec les valeurs suivantes : `vert, jaune, rouge;`.
Chaque carte de valeur au plus 10 existe dans les trois couleurs possibles. Les phœnix ne sont que vert ou jaune, et le dragon n'existe qu'en rouge. De plus, les couleurs sont ordonnées selon l'ordre : `vert` < `jaune` < `rouge`.

Cette classe aura pour méthodes :
— Un constructeur pour initialiser la hauteur et la couleur de la carte (sans contrôle).
— Une méthode retournant une chaîne descriptive de la carte (par exemple : *dragon*, ou *2 vert*).
— Une méthode permettant d'obtenir la hauteur de la carte.
— Une méthode permettant d'obtenir la couleur de la carte.
— Une méthode prenant une autre carte en paramètre et déterminant si la carte courante est plus grande que la carte prise en paramètre (on comparera d'abord la hauteur des cartes, puis en cas d'égalité la couleur des cartes).
— Une méthode qui vérifie que la carte initialisée est bien une carte valable du jeu.

➙ Lisez le cours suivant sur les énumérations (avec inscription à OpenClassroom) :
`https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/22162-decouvrez`

ou celui-ci (sans inscription) :
`http://blog.paumard.org/cours/java/chap04-structure-classe-enumeration.html`

Étoffer votre énumération `Coul`.

Question 3. Dans la classe `Main`, coder les instructions permettant de :
- Créer un 2 vert ;
- Afficher le 2 vert ;
- Vérifier que le 2 vert est une carte bien formée ;
- Créer un 12 vert et vérifier qu'il est mal formé ;
- Créer un 2 rouge ;
- Afficher si le 2 rouge est plus grand que le 2 vert.

Exercice 3 *Référence et type primitif*

Quand on exécute l'instruction `MaPremiereClasse o1=new MaPremiereClasse()`, on crée un nouvel objet, et on l'affecte à o1. On dit que o1 référence l'objet. Un même objet peut être référencé plusieurs fois, par exemple en faisant :

```
MaPremiereClasse o1=new MaPremiereClasse();
MaPremiereClasse o2=o1;
```

o1 et o2 référencent le même objet.

Tous les objets en Java sont typés par des classes : soit des classe de la bibliothèque (par exemple String pour les chaînes de caractères), soit des classes développées par un programmeur lambda (vous par exemple). Tous les objets sont manipulés via des références vers ces objets. Par contre, il existe en Java des types dits simples ou primitifs qui permettent de manipuler les données les plus courantes, comme par exemple les entiers (int) ou les booléens (boolean). Les noms des types primitifs commencent par une minuscule. Le test d'égalité pour les types primitif est ==. Il y a deux tests d'égalité pour les objets : == pour l'égalité de référence (est-ce que deux références référencent le même objet ?) et la méthode equals pour l'égalité de valeur (est-ce que les objets référencés portent les mêmes valeurs ?).

Question 4. Etudiez les paramètres des méthodes de la classe Carte précédemment écrite.
- Créer dans la classe `Carte` une méthode `equals(Carte.c)` telle que `c1.equals(c2)` renvoie vrai si, et seulement si, `c1` et `c2` ont même hauteur et couleur.
- Essayer de générer automatiquement la méthode `equals` (menu *Source*). Regarder les différentes options possibles.

Question 5. Créer deux cartes `c1` et `c2` ayant les mêmes paramètres. Les comparer avec == et avec `equals`.

Exercice 4 *Associations et collections d'objets*

On va écrire une classe `MainJoueur` qui représente la main d'un joueur, puis `JeuCartes` qui gère les mains des 4 joueurs.

Question 6. Modélisez la relation entre `MainJoueur` et `Carte`.

Question 7. En java, définissez un attribut cartes dans la classe `MainJoueur` de type : arraylist de cartes ($ArrayList < Carte >$). Le constructeur crée cette liste vide.

Question 8. Munir la classe d'un attribut nombre qui représente le nombre de cartes de la main.

Question 9. Ecrire une méthode qui permet d'ajouter à la main une carte prise en paramètre. Cette méthode doit s'assurer que la main contient au maximum 16 cartes.

Question 10. Ecrire une méthode qui enlève une carte de la main.

Question 11. Ecrire une méthode qui retourne la carte la plus forte du jeu.

Question 12. Ecrire une méthode qui enlève et retourne la plus forte carte du jeu.

Question 13. Ecrire une méthode afficher qui affiche la main du joueur.

Question 14. Modélisez et implémentez une classe `JeuCartes` qui gère les mains des 4 joueurs. On y trouvera :

- un tableau de 4 `MainJoueur`
- un tableau de 64 cartes pour le jeu de cartes avant distribution
- une méthode genererNouveauJeu() qui vous est donnée et qui génère toutes les cartes
- une méthode battreLeJeu() qui bat le jeu de 64 cartes. Pour cela, on tirera un double alétoirement, on le multipliera arbitrairement par 100, on le rangera modulo 63 dans un entier. On reproduira l'opération. On aura donc deux entiers i1 et i2 compris entre 0 et 63. On permutera ensuite les cartes positionnées dans ce tableau aux indices i1 et i2. On répètera cette procédure une centaine de fois.
- une méthode distribuer() qui génère un nouveau jeu, et répartit les cartes entre les mains des 4 joueurs.
- une méthode afficherJeu() qui affiche les mains des 4 joueurs.