Héritage et polymorphisme

Exercice 1

1- Pour manipuler des chaînes de caractères, on construit la classe ManipString qui a un champ mot et un constructeur qui prend un String comme argument.
2- Définir la classe ManipString
3- Rajouter les fonctions suivantes :
   a. Une fonction qui permet de comparer l'objet courant avec un deuxième objet de cette classe
   b. Une fonction qui retourne le nombre de caractères constituant le champ mot et de les afficher :
   c. Une fonction qui permet de retourner une partie du champ mot, a partir d'une position et se terminant a une position fixe, attention aux arguments de la fonction
   d. Une fonction qui permet de retourner la forme minuscule du champ mot
   e. Une fonction qui permet de retourner la forme majuscule du champ mot
   f. Une fonction permettant d'enlever l'espace, s'il existe, au début et à la fin du champ mot
   g. Une fonction qui permet de concaténer l'objet courant et un deuxième objet de cette classe, elle retourne un nouveau objet dans le champ mot est la concaténation du champ mot de l'objet courant et celui de l'argument de la fonction.
   h. Tester ses fonctions dans la fonction main ( ) ;

Exercice 2

L'objectif est d'écrire la classe Phrase ; dont le but est un Compteur de mots dans une chaîne lue au clavier. Puis mise en majuscule de l'initiale de chaque mot. Cette classe Possédant, plus son constructeur par défaut, les 2 méthodes suivantes :

1- nbMots(String ch) qui retourne le nombre de mots contenus dans la chaîne passée en paramètre. Un mot est séparé du suivant par un ou plusieurs espaces.
2- initialesEnMajuscules(String ch) retourne une nouvelle chaîne qui correspond à la chaîne passée en paramètre avec l'initiale de chaque mot en majuscule. Utiliser la classe StringBuffer pour construire de manière efficace la nouvelle chaîne.

Exercice 3

Soient les classes A_1 et B_1 qui sont données par :

```
class A_1 {

        void m(A_1 a){    System.out.println("m de A_1");   }

        void m(B_1 b){  System.out.println("m de A_1 version 2"); }

        void n(A_1 a){   System.out.println("n de A_1");   }
```

```
}

class B_1 extends A_1 {

        void m(A_1 a){   System.out.println("m de B_1");  }

        void n(B_1 b){    System.out.println("n de B_1");    }

}

class Test{

        public static void main(String[] argv){

                A_1   a = new B_1 ( );

                B_1    b = new B_1 ( );

                A_1   a1 = new A_1 ( );

                a.m(b);

                a.n(b);

                b.m(b);

                b.n(b);

                a.m(a1);

                a.n(a1);

                a1.m(b);

                a1.n(b);

        }

}
```

Question : Déterminez les résultats en appliquant le mécanisme de liaison dynamique.

Exercice 4

Qu'affiche le programme suivant ?

```
class A {
        public String f(D obj) { return ("A et D");}
        public String f(A obj) { return ("A et A");}
}
class B extends A {
        public String f(B obj) { return ("B et B");}
        public String f(A obj) { return ("B et A");}
}
```

```
class C extends B{ ……}
class D extends B{…….}
class test {
        public static void main (String [] args){
                A a1 = new A( );
                A a2 = new B( );
                B b = new B( );
                C c = new C( );
                D d = new D( );
                System.out.println(a1.f(b));
                System.out.println(a1.f(c));
                System.out.println(a1.f(d));
                System.out.println(a2.f(b));
                System.out.println(a2.f(c));
                System.out.println(a2.f(d));
                System.out.println(b.f(b));
                System.out.println(b.f(c));
                System.out.println(b.f(d));
        }
}
```

Exercice 5

Un éleveur de volaille reçoit d'un fournisseur de jeunes canards et de jeunes poulets qu'il élève jusqu'à ce qu'ils aient la taille nécessaire à leur commercialisation.
Une volaille est caractérisée par son poids et un numéro d'identification reporté sur une bague qu'elle porte a sa petite patte. Les volailles arrivent à l'élevage à l'âge de trois semaines. Elles sont baguées et enregistrées dans le système informatique.
Il y a deux sortes de volailles : des canards et des poulets. Le prix du canard et celui du poulet sont deux prix différents, exprimés en euros par kilo. En revanche, le prix est le même pour tous les individus de la même espèce. Ce prix varie chaque jour. Le poids auquel on abat les bêtes est différents pour les canards et les poulets, mais c'est le même pour tous les poulets (respectivement, tous les canards).
Écrivez une classe des volailles avec deux sous-classes des poulets et des canards. Il faut pouvoir enregistrer les prix du jour, les poids d'abatage, le poids d'une volaille donnée.
Écrivez une classe permettant de représenter l'ensemble des animaux de l'élevage au moyen d'un tableau.
Des méthodes doivent permettre de trier les animaux à abattre et d'évaluer le prix obtenu pour ces animaux. Il faut également pouvoir enregistrer les jeunes animaux qui arrivent.

Problème (Jeu des petits chevaux)

On pose :

DJ = Depart jaune

DR= Depart rouge

AJ = Arrivée jaune

AR = Arrivée rouge

DJ	1	2	3	4	5	6	7	8	9	**DR**
AR										11
38										12
37										13
36										14
35										15
34										16
33										17
32										18
31										19
30	**AJ**	28	27	26	25	24	23	22	21	20

Les règles du jeu des petits chevaux auquel, on joue ici sont les suivantes :

- Le jeu se joue à deux joueurs, chaque joueur a une couleur différente (rouge ou jaune)
- Chaque joueur possède 2 pions, 1 lent et 1 rapide. Lorsque c'est son tour de jouer, le joueur choisit un pion, et, lance un dè si le pion est lent et 2 dès si le pion est rapide. Le pion avance ensuite de la valeur du ou des dès.
- Le plateau de jeu est comme sur la figure 1. C'est une grille de taille 40 (voir la figure), qui comporte 2 points de départ (DJ et DR) et deux points d'arrivée (AJ et AR).
- Un pion est arrivé lorsque sa position est supérieure à la position de l'arrivée. Il ne peut alors plus rien lui arriver.
- Un joueur a gagné lorsque ses deux pions sont arrivés.

A- Les pions

On va d'abord définir une classe Pion dont le squelette est donné par :

```
class Pion{
        String couleur;
        boolean vitesse ;
        int position ;
        // …… autres codes de la classe
        void avancer (int n){//…}
        static void avancer(int n, Pion p){//…}
        void avancerAvecConflit(int n, Pion p){//…}
        boolean estArrive(){//…}
}
```

1- Ecrire un premier constructeur pour la classe Pion qui prend en argument la couleur, la vitesse et la position de l'objet qu'on veut créer.
2- Ecrire un deuxième constructeur qui prend en argument la couleur et la vitesse, et initialise la position correctement
3- Ecrire la méthode **estArrive** qui renvoi true si le pion est arrivé, et false sinon.
4- La méthode non static **avancer(int n)** doit faire avancer le pion de n cases, sans le faire dépasser la ligne d'arrivée. Ecrire la méthode **avancer**
5- Ecrire la version static de cette méthode.

B- La classe Jeu va utiliser la classe Pion. Il est donc vivement recommandé d'utiliser les méthodes de la classe Pion.

Le squelette de la classe Jeu est donné par :

```
class Jeu{

        Pion jauneLent ;
        Pion jauneRapide ;
        Pion rougeLent ;
```

```
        Pion rougerapide ;


        // …… autres codes de la classe
        void jouer(String couleur){}
       void jouerModifier(String couleur){}
        boolean aGagne(String couleur){}
}
```

Pour l'instant les deux joueurs cohabitent paisiblement sur le plateau de jeu.

1- Ecrire un constructeur pour la classe Jeu. Sachant que la vitesse d'un pion lent est égale à false ;
2- Ecrire la méthode **jouer**, qui décide aléatoirement quel pion avance et tire au hasard le nombre de case dont le pion avance. On rappelle que les pions ont des vitesses différentes
3- Ecrire la méthode **jouerModifier**, qui fait la même chose que la méthode jouer mais comprend en compte le fait de ne pas faire jouer un pion qui est déjà arrivé
4- Ecrire la méthode **aGagne**. Un joueur a gagné si ses deux pions sont arrivés

Les pions rentrent maintenant en conflit. Si un pion arrive sur une case où se trouve un pion adverse, il le renvoi à sa case de départ.

5- Ecrire une méthode **boolean conflit(Pion p)** qui teste si la position du pion p est identique à celle d'un pion adverse et gère les conséquences
6- Ecrire la méthode **void avancerAvecConflit(int n, Pion pio)**, qui fait la même chose que la méthodes **avancer** de la classe Pion et qui prend compte de cette modification (la présence de conflit entre pion courant et le point adverse **pio**).
7- Ecrire une méthode qui fait jouer les joueurs alternativement jusqu'à la victoire d'un des deux joueurs : **void jeuComplet()** ;

Annexe :

*) dans la classes Math, on a la méthode suivante, qui permet de retourner une valeur aléatoire entre 0 et 1.

static double	**random**() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.