Td 1- IO

Exercice 1

Donner le corps d'une fonction qui permet d'afficher sur la console tous les noms des fichiers ayants une extension donnée (exemple : l'extension .jpg) et qui se trouvent dans un répertoire donné (exemple : répertoire courant (*System.getProperty("user.dir")*).

La casse des caractères de l'extension des fichiers est ignorée.

public void filtreFichier(String cheminRepertoire, String extensionFichier).

Exercice 2

Construire une classe **Histogramme** permettant de sauvegarder l'histogramme d'un texte dans un fichier texte nommé « fileName». Sachant que l'histogramme d'un texte est la liste des mots présents dans ce texte et pour chacun d'eux, leur nombre d'occurrences.

Exemple :

Le texte est : **aaa bb cc aaa cc aaa**

L'histogramme du texte est donné par :

aaa 3

bb 1

cc 2

Le squelette de la classe Histogramme est donné par:

public class **Histogramme** implements Serializable {

private TreeMap<String, Integer> arbre_Histogram;	Ce champ permet de stocker les mots avec leur occurrence. Chaque enregistrement dans ce champ à pour clé un mot existant dans le texte et la valeur correspond au nombre d'occurrence associé à ce mot. Ce champ est initialement vide.
static private final long serialVersionUID = 1L;	
public Histogramme( )	Constructeur de la classe Histogramme
public void setArbre_Histogram (TreeMap<String, Integer> arbreHistogramNew)	Permet de modifier les infos du champ arbre_Histogram de l'objet courant en fonction des infos du paramètre arbreHistogramNew
public TreeMap<String, Integer> getArbre_Histogram()	Le getteur du champ arbre_Histogram
public void add(String texte)	Permet de décomposer le texte en fonction du séparateur « espace ou plusieurs espaces » puis pour chaque mot de ce texte, on compte le nombre d'occurrence associé puis on ajoute le mot avec son occurrence dans le champ arbre_Histogram.
public String toString( )	Permet d'afficher tous les enregistrements se trouvant dans le champ arbre_Histogram. Chaque

	enregistrement dans une ligne avec le format de chaque ligne est: **Mot :: nbOccurenceMot**
public void enregistrer(String fileName)	Permet d'enregistrer l'objet courant dans le fichier fileName
public Histogramme lire(String fileName)	Permet de lire le fichier fileName dans lequel on a enregistré un objet de type Histogramme puis cette méthode retourne l'objet lu. S'il y a un problème la méthode return un objet null.

}

Problème

Partie 1

On veut écrire un programme permettant de manipuler les joueurs d'un jeu de rôles en ligne où le nombre de joueurs est important.
Pour chaque joueur, on dispose de données diverses dont au moins

- ✓ Un pseudo de type String qui est un identifiant unique dans le jeu;
- ✓ un nom de type String;
- ✓ un niveau du personnage de type String qui représente un entier.
- ✓ la liste des pseudos de ses amis qui existent dans le jeu.

Écrire une classe **Player** contenant un champ de nom infosJoueur dont le type est HashMap<String,Object> et qui associe aux chaînes "pseudo", "name", "level", "friends" les données correspondantes pour le joueur. Les méthodes de cette classe sont données par :

a. Un constructeur sans arguments, qui permet d'initialiser le champ infosJoueur avec des valeurs nulles pour les trois premières données du joueur et d'initialiser la quatrième donnée avec une liste vide de type TreeSet<String>
b. Écrire le getteur et le setteur de cette classe.
c. Comme les objets de cette classe doivent être comparables alors on redéfini la méthode compareTo. La comparaison doit être fait en fonction de l'identifiant
d. Redéfinir la méthode equals qui permet de vérifier l'égalité de deux Player, ceci en fonction de leur identifiant
e. Redéfinir la méthode toString. L'affichage du Player sera :
   - {friends=[zip, pa, mac, dp], level=5, pseudo=mpb, name=beal}

Partie 2

Écrire une classe **Game** qui permet de stocker l'ensemble des joueurs. La classe Game doit permettre de trouver rapidement les données d'un joueur ayant un certain pseudo. Pour ceci, on demande que la classe Game contienne un champ mapPlayers de type TreeMap<String,Player> qui associe à chaque pseudo le joueur ayant ce pseudo. Le champ mapPlayers doit être toujours ordonné dynamiquement. Cette classe contient les méthodes suivantes :

a. Un constructeur par défaut permettant d'initialiser le champ mapPlayers.
b. le getteur et le setteur de cette classe.
c. Une méthode findPlayer() qui permet de chercher l'existence ou non d'un joueur en fonction de son identifiant. Cette méthode retourne un objet de type Player si le joueur existe dans le jeu si non elle retourne l'objet nul.
d. Une méthode highestLevel() qui permet de retourner le plus haut niveau présent parmi tous les joueurs. Cette méthode retourne -1 si le champ mapPlayers est vide.

e. Une méthode highestLevelSet affichant l'ensemble des joueurs qui ont le plus haut niveau. Cette méthode retourne un objet nul si le champ mapPlayers est vide
f. Une méthode add( ) permettant d'ajouter un nouveau Joueur s'il n'existe pas dans le jeu. Si la valeur du "level" n'est pas un entier valide alors il n'y pas d'ajout de nouveau joueur.
g. Une méthode addFriend() permettant d'ajouter un ami à un joueur identifié par son identifiant, si cet ami existe dans le jeu. Le joueur ami est identifié par son identifiant.
h. Une méthode removeFriend() permettant de supprimer un ami à un joueur, si cet ami existe dans le jeu.
i. Une méthode removePlayer() permettant de supprimer un Player existant dans le jeu en fonction de son identifiant. Attention à la liste des amis des autres joueurs.
j. Une méthode commonFriends() permettant de retourner l'ensemble des joueurs amis communs à deux Players. Cette méthode retourne null si l'un des paramètres de cette méthode et nul ou si l'un des joueur n'appartient pas au jeu.
k. Redéfinir la méthode toString qui donne l'affichage suivant pour le Game
    - {ada={ friends=[ ], level=10, pseudo=ada, name=nane},
    - mac={ friends=[beal, dp, pa], level=10, pseudo=mac, name=crochemore},
    - mou={ friends=[zip], level=8, pseudo=mou, name=nir},
    - mpb={ friends=[dp, mac, pa, zip], level=5, pseudo=mpb, name=beal},
    - nab={ friends=[ ], level=10, pseudo=nab, name=ila},
    - zip={ friends=[beal, dr, mac], level=10, pseudo=zip, name=zipstein}}
    -

f. Un constructeur avec un seul paramètre de type string nomFichier permettant de lire le fichier nomFichier et qui permet de construire toutes les informations du TreeMap afin de construire correctement l'objet courant, ceci via une lecture des information stockées dans le fichier nomFichier.
g. Une méthode pour permettant de stocker toutes les informations se trouvant dans le TreeMap des joueurs de l'objet courant dans le fichier nomFichier. Chaque élement du TreeMap doit être enregistré, dans une seule ligne, dans l'ordre la clé, nom, niveau et la liste de tous les pseudos des amis.

Annexe

Les squelettes des classes sont donnés par :

```
public class Player implements Comparable<Player> {
    private HashMap<String, Object> infosJoueur;
    public Player() {}
    public HashMap<String, Object> getInfosJoueur() {}
    public void setInfosJoueur(HashMap<String, Object> dataPlayer) {}
    public String toString() {}
    public int compareTo(Player Joueur) {}
    public boolean equals(Object Joueur) {}
}
public class Game {
    private TreeMap<String, Player> mapPlayers;
    public Game() {}
    public Game(String fileData) {}
    public TreeMap<String, Player> getMapPlayers() {}
    public void setMapPlayers(TreeMap<String, Player> mapJeu) {}
    public Player findPlayer(String identifiantPseudo) {}
    public int highestLevel() {}
    public TreeSet<Player> highestLevelSet() {}
    public void add() {}
    public void addFriend(String IdentifiantJoueur, String identifiantAmi) {}
    public void removeFriend(Player joueur, String identifiantAmi) {}
    public void removePlayer(String identifiant) {}
    public TreeSet<Player> commonFriends(Player joueur_1, Player joueur_2) {}
    public void saveGame(String fileData) {}
    public String toString() {}
```

```
/**
     * Méthode permettant de vérifier si un string représente par un entier valide positif
     * @param val String
     * @return boolean
     */
    public boolean entierRepresnteeParUnString(String val){
        try{
            int entier = Integer.valueOf(val);
            if(entier < 0) return false;
            }catch(NumberFormatException e){
            return false;
        }
       return true;
    }
}
```