	Conception Orientée Objet *M1 – Info* S. *Caffiau - année 2017/2018* Implémentation de diagramme	

Implémentation de modèles objets et production de documents

Exercice 1 : Documentations

Question 1 : Créez une classe Author, dont le code est le suivant et compilez le.

```
public class Author {

	private final String firstName;
	private final String lastName;

	public Author(String firstName, String lastName){
		this.firstName=firstName;
		this.lastName=lastName;
	}

	public boolean equals(Object o){
		if(!(o instanceof Author)){
			return false;
		}
		Author author = (Author)o;
		return (author.firstName.equals(this.firstName) && (author.lastName.equals(this.lastName)));
	}
}
```

Il existe deux types de commentaires en JAVA :

- les commentaires pour utiliser une méthode
- les commentaires pour expliciter quelque chose qui n'est pas classique dans le code

Intéressons nous aux premiers.

Chaque méthode publique doit être commentée pour indiquer :

- ce qu'elle fait
- quels sont les arguments attendus
- quels sont les valeurs de retour possible
- quels sont les exceptions qui sont levées et pourquoi sont-elles levées (ce dernier point sera traité dans l'exercice suivant)

Pour que toutes ces informations soient toujours à jour en demandant le minimum d'efforts aux développeurs de la classe, JAVA dispose d'un mécanisme de génération de documentation directement dans le langage : Javadoc.

Des balises permettent de donner les informations à documenter :

Tag	**Description**
@author	Nom du développeur
@deprecated	Marque la méthode comme dépréciée (méthode obsolète).
@exception	Documente une exception lancée par une méthode.
@param	Définit un paramètre de méthode. Requis pour chaque paramètre
@return	Document la valeur de retour
@see	Documente une association à une autre méthode ou classe
@since	Précise à quelle version du SDK/JDK une méthode a été ajoutée à la classe
@throws	Documente une exception lancée par une méthode (synonyme pour @exception)
@version	Donne la version d'une classe ou d'une méthode

Dans le code, le bloc de documentation de la Javadoc se situe avant chaque méthode publique et toutes les classes public et se présente comme ceci (exemple pour une méthode "boolean estUnDeplacementValide (int p1, int p2, inte p3, int p4)") :

```
/**
* Valide un mouvement de jeu d'Echecs.
* @param p1 File de la pièce à déplacer
* @param p2 Rangée de la pièce à déplacer
* @param p3 File de la case de destination
* @param p4 Rangée de la case de destination
* @return vrai(true) si le mouvement d'échec est valide ou faux(false) si invalide
*/
```

Question 2 : Reprenez Author et ajoutez les commentaires pour générer la Javadoc de la classe. En particulier, apportez des connaissances pour l'utilisation de equals (objet de type Object et non de type Author).

Question 3 : Générez la javadoc dans un répertoire Doc (javadoc Author.java -d Doc).

NB : à partir de maintenant, toutes vos classes doivent disposer d'une javadoc.

Exercice 2 : L'entreprise de transport

Question 1 : Implémentez le diagramme UML que vous avez réalisé sur l'entreprise de transport en vous appuyant sur les tableaux suivants qui présente la "traduction" des éléments d'un modèle en code et les conventions de nommage que vous devez suivre en JAVA (complétez votre modèle en cas de besoin).

Modélisation	**Implémentation**
association (permanente) entre 2 classes	une des classes sert de **type** à un attribut de l'autre classe - si unidirectionnelle de C1 vers C2 alors C2 ne connait pas C1 (attribut dans C1 de type C2) - si multiplicité l'attribut est un tableau
dépendance (association affaiblie)	C1 peut recevoir un paramètre de type C2
agregation	même chose que pour l'association pour les attributs constructeur de C1 avec un objet C2 en paramètre
composition	même chose que pour l'association pour les attributs constructeur de C1 qui crée son objet C2
héritage	C2 extends H1
héritage multiple	une interface I pour une des classes mère puis C2 implements I, extends H1 ou deux interfaces (pour les 2 classes mères) puis C2 implements I1, I2

Element	**Convention**
Classe	première lettre = majuscule
Méthodes	reflète une action -> verbe première lettre = minuscule

Exercice 3 : Les smartphones

Question 1 : Réalisez le code correspondant au diagramme UML que vous avez réalisé la semaine dernière sur les smartphones (exercice 4) en vous appuyant sur les tableaux ci-dessus.