	Conception Orientée Objet *M1 – Info* S. *Caffiau - année 2015/2016* Pré-conditions, Contrats et exceptions	

Pré-conditions, exceptions, documentations et autres mécanismes de JAVA pour améliorer la qualité du code

Exercice 1 : Programmation par contrat

Pour prévenir les déroulements non-normaux de votre code, il faut identifier les conditions qui pourraient empêcher le déroulement normal. Les conditions à vérifier sur les paramètres passés aux méthodes peuvent être vérifiées en appliquant une programmation par contrat. Le principe est de mettre en place des mécanismes qui imposent que les valeurs d'un objet sont toujours valident.

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));
}
}
```

Question 2 : Identifiez quel pourrait être les données qui pourraient créer un dysfonctionnement. Au besoin, créez des instances et utilisez les méthodes pour tester avec des valeurs différentes.

Question 3 : Modifiez votre code pour que les instances d'Author soient toujours valides (indication : regardez dans les méthodes proposées dans Objects - http://docs.oracle.com/javase/7/docs/api/java/util/Objects.html -)

Question 4 : Reprenez toutes les classes que vous avez créées la semaine dernière et répétez la procédure pour qu'il n'y ait plus possibilité de créer des instances d'objets invalides.

Exercice 2 : Documentations

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	Docuemtne 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 1 : Reprenez Author et ajoutez les commentaires pour générer la Javadoc de la classe.

Question 2 : Générez la javadoc dans un répertoire Doc (javadoc Author.java -d Doc).

Question 3 : Reprenez toutes les classes que vous avez créées la semaine dernière et répétez la procédure pour que toutes les doc existent.

Exercice 3 : Exceptions

Le mécanisme des exceptions permet de sortir du déroulement normal des méthodes (pour peu qu'on l'ait prévu). Il repose sur 2 étapes successives :

- la levée d'exception (on sort du déroulement normal car une anormalité est arrivée)
- la gestion d'exception (une fois sortie on traite l'exception)

Lorsqu'un type particulier d'exception va être traité on utilise les mots clés : **try** et **catch.** **Try** permet de définir un bloc d'instruction qui est "sensibilisé" aux exceptions, à la fin du bloc, l'instruction **catch()** permet d'identifier l'exception à capturer et le traitement à lui appliquer. Le mot clé **Throws** permet de lister l'ensemble des exceptions qui peuvent être levées.

Question 1 : Dans le code suivant, identifiez quel(s) problème(s) pourrait parvenir.

```
public PlaqueImmatriculation getPlaque(Voiture v) {
        return v.plaque;
}
```

Question 2 : Que pouvez vous faire pour être averti du problème (avec un message)

- avec throws ?
- avec try/catch ?

Question 3 : Traitez toutes les methodes des classes que vous avez réalisées jusqu'à maintenant et mettez à jour les tags de la Javadoc

Exercice 4 : Assertions

Pour vérifier des conditions dans un code (vérifier des valeurs de donnée), des assertions peuvent être ajoutées dans le code (et exécutées que pendant le développement). En java, une assertion est déclarée en utilisant le mot clé **assert**.
assert i==j; vérifie que i et j ont la même valeur

assert i==j: “i et j sont différents”; vérifie que i et j ont la même valeur et si ce n’est pas le cas affiche le message d’erreur “i et j sont différents”.

Question 1 : Ecrivez le code de la classe IntStack.

```
public class IntStack {
private final int[] array;
private int top;
/**
* Put the value on top of the stack.
* @param value the value to push on the stack.
* @throws IllegalStateException if the stack is full
*/
public void push(int value) {
   if (array.length == top) {
   throw new IllegalStateException("stack is full");
}
array[top++] = value;
}
}
```

Question 2 : Ajoutez une assertion pour s’assurer que la valeur “value” est bien à la place top-1 dans le tableau (c’est une vérification de la post-condition).

Attention les assert ne sont vérifiés que lorsque le programme est lancé avec java -ea.

Question 3 : Ajoutez une assertion pour s’assurer que la valeur de top est comprise entre 0 et la taille du tableau (vérification d’invariant).