NFP121 - Examen blanc

Exercice 1 :

1. Le mot @*Override* permet de **redéfinir** une méthode qui remplacera sa précédente définition, à la suite de l'**héritage** d'une classe parente.

   Dans le cas présent, la méthode *toString()* possède ce mot car c'est une méthode héritée de la classe *Object* (dont toutes les classes héritent) et son implémentation initiale renvoie le nom et l'adresse mémoire de l'objet. Afin de lui donner une implémentation qui nous est utile, elle est redéfinie grâce au mot @Override.

   Pour redéfinir la méthode *toString()*, il est nécessaire d'ajouter ce mot pour indiquer au compilateur que c'est une redéfinition.

2. Il est conseillé de ne pas déclarer les attributs en public afin d'éviter que ces attributs puissent être modifiés librement en dehors de la classe qui les définit, pour des soucis de sécurité.

   Généralement, ils sont déclarés en privée (*private*) afin qu'ils ne puissent être changés que dans la classe qui les définit et ainsi, pouvoir contrôler quelles modifications sont disponibles aux autres classes avec des *getters* et des *setters*.

3. Écrire *c1.equals(c2)* peut fonctionner pour comparer deux objets à condition de redéfinir l'implémentation de la méthode *equals()* (et *hashCode()*) car, par défaut, elle vérifie seulement si les deux objets réfèrent au même objet.

   Afin de la faire fonctionner, il faut donc redéfinir son implémentation, à l'aide du mot @*Override*, en vérifiant que l'objet n'est pas null, que les deux objets réfèrent ou non au même objet et ensuite la comparaison de leurs attributs.

Exercice 2 :

1.

```
import static org.junit.Assert.*;
import org.junit.Test;

public class ConfigurationTest {

    Configuration configuration;

    @Test
    public void testConfiguration() {
       Configuration configuration = CLIClassique.configuration("-K
10 -A 0.90 -K 20 -P -K 30 -C".split(" "));
       assertEquals(configuration.toString(), "alpha=0.9,
epsilon=-1.0, indice=30, mode=CREUSE");
    }
}
```

2. Le résultat de `java PageRankClassique -C -K 21 -E .001 -K 100` devrait être : `"alpha=0.85, epsilon=0.001, indice=100, mode=CREUSE"`

3. Le programme ne gère pas l'entrée d'un flottant lorsque l'entrée attendue est un entier. Cela déclenche donc une exception NumberFormatException car *parseInt()* ne fonctionne pas avec un flottant tel que 15.5. On peut donc en déduire que le programme n'est pas robuste d'autant plus que l'erreur est propagé dans "main" ce qui signifie qu'elle n'est pas gérée.

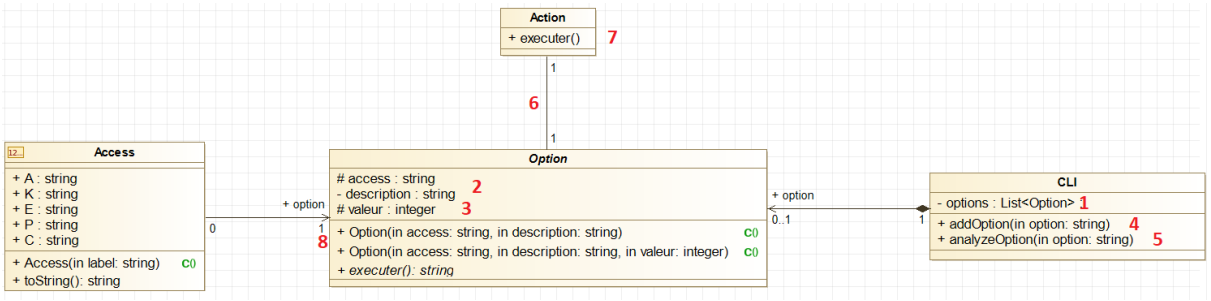
4. En ligne 11, l'appel de fonction `Integer.parseInt(args[++i])` est direct, ce qui implique que nous ne vérifions pas si la valeur est bien un entier dans la chaîne de caractères passée.

6. Plutôt qu'utiliser une boucle while avec un compteur i, il est préférable d'utiliser une boucle for qui intègre par défaut un compteur.

   Il est également préférable de déclarer un constructeur privée qui lève une exception dans la classe CLIClassique pour éviter la possibilité d'utiliser le constructeur public implicite de Java

Exercice 3 :

1.



Exercice 4 :

1. Pour produire la vue il faut créer une fenêtre, dans laquelle sera renseigné un panel permettant d'y placer les boutons, textes et zones de saisie.

2. Afin de faire en sorte que "-C" s'ajoute en bas quand l'utilisateur clique sur "Creuse (C)", il faut ajouter un contrôleur sur le bouton qui sera appelé au moment du clic et permettra d'ajouter l'option à l'affichage.