Enoncé question 2-5

Que pensez vous de cette réponse pour les classes Pile, Pile2, Pile3 et Pile4 ?

```
 public boolean equals(Object o) {
  if (o instanceof PileI) {
   PileI p = (PileI) o;
   return this.capacite() == p.capacite()
      && this.hashCode() == p.hashCode();
  } else
   return false;
 }
```

A- *Est-elle correcte ?, quelle est la règle à retenir ? (à toutes fins utiles la javadoc de java.lang.Object)*

B- *Cette autre réponse, élégante, n'est pas correcte*

```
public boolean equals(Object o) {
  return this.toString().equals(o.toString());
 }
```

pourquoi ?

Réponse sur question 2-5

A- Oui le code mentionné en A est correcte.

Pour que la methode *.equals()* en Java retourne True, en comparant o à p, il faut que:

1-Reflexivite : l'objet doit être égal à lui-même. *p.equals(p) == true;*

2-Aucun des 2 objets ne soient null. if (o == null) return false;

3-Looper autour de o et p et vérifier que les éléments de o au même indexe sont les mêmes que p.

4-o et p sont de même taille et capacité.

En plus il faut que *.equals* soit :

5-Symmetrique: *si (p.equals(o)==true) then o.equals(p) == true*.

6-Transitive: *si o.equals(p) and p.equals(s); then o.equals(s)*

7-Consistent: *si p.equals(o)==true* et aucune valeur est modifiée, alors elle retourne *True* à chaque appel.

8-A chaque objet p non-null, *p.equals(null)==false*

this.hashCode() == p.hashCode();

si retourne false, plus besoin d'exécuter la methode equals.

Mais si retourne true, il faut executer la methode equals pour s'assurer que l'objet actuel (o) est identique à p.

<u>la règle à retenir (à toutes fins utiles la javadoc de java.lang.Object) :</u>

Si *o.equals(p) retourne True* alors *o. hashCode()=p.hashCode()* .

B- *.equals()* teste l'equivalence logique entre *this.toString* et *o.toString* (compare leurs valeures-leur contenu- si égales) mais non pas leurs références,d'où l'utilité du *hashCode().*