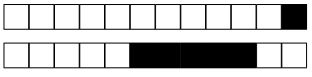

+1/1/60+

QCM Test

31 mai 2017

Une réponse est considérée comme correcte *seulement* quand elle est entièrement correcte. Sinon, elle est incorrecte. Par exemple, si une question demande de cocher toutes les affirmations vraies, qu'il y a trois affirmations vraies sur quatre, et que vous avez coché deux des trois affirmations vraies, la réponse est considérée comme incorrecte.

Une réponse correcte rapporte un point. Une réponse incorrecte coute un demi-point, sauf s'il n'y a que deux réponses possibles (questions de type vrai ou faux), auquel cas une réponse incorrecte coute un point.

Par exemple, si vous avez 5 réponses correctes, 2 réponses incorrectes, et avez laissé 3 questions sans réponse, dans un questionnaire sans questions vrai ou faux, vous obtenez 4/10.

Nom .....

Prénom .....

Question 1 L'implémentation suivante respecte-t-elle bien ce qui est exigé d'une fonction de hachage, relativement à l'implémentation de `equals` fournie ? Voici la javadoc des méthodes invoquées :

- `static boolean equals(Object a, Object b)` : "Returns true if the arguments are equal to each other and false otherwise"
- `static int hash(Object... values)` : "Generates a hash code for a sequence of input values"

```
public class MyClass {
  private String lastName;
  private String firstName;
  public boolean equals(Object o) {
    if (!o instanceof MyClass) return false;
    return Objects.equals(firstName, ((MyClass)o).firstName);
  }
  public int hashcode() {
    return Objects.hash(firstName, lastName);
  }
}
```

☐ Non

☐ Oui

Question 2 Le code suivant compile-t-il ? (On suppose que le code est placé dans des fichiers nommés de façon adéquate et que les imports éventuels sont corrects.)

```
public interface MyInterface {
  public void myFirstMethod() {
    System.out.println("Coucou.");
  }
}
public class MyClass implements MyInterface {
  public void myMethod() {
    System.out.println("Coucou.");
  }
}
```

☐ Oui
☐ Non

Question 3 ♣ SWT est (cocher toutes les affirmations correctes)

☐ une bibliothèque incluse en standard dans Java
☐ un autre nom pour Swing
☐ une bibliothèque non incluse en standard dans Java
☐ une bibliothèque allant souvent de pair avec Swing
☐ une bibliothèque allant souvent de pair avec JFace
☐ *Aucune de ces réponses n'est correcte.*

Question 4 En Java, `"a".equals("b");`

☐ ne compile pas
☐ compile, mais produit une erreur à l'exécution
☐ exécute la méthode equals définie dans la classe String
☐ exécute la méthode equals définie dans la classe Object

Question 5 Soit un projet configuré de façon classique utilisant Maven, structuré en sous-répertoires et en fichiers comme suit :

— Racine, contenant pom.xml
— Racine/src/main/java/rep1, contenant AFile.java et AFile.txt
— Racine/src/test/java, contenant AnotherFile.java

En supposant que le projet compile, quelles classes et ressources se retrouveront dans le classpath lors de l'exécution des tests ? (Une seule réponse.)

☐ La classe AnotherFile.
☐ pom.xml, les classes AFile et AnotherFile, et le fichier ressource AFile.txt.
☐ Les classes AFile et AnotherFile.
☐ Rien.
☐ Les classes AFile et AnotherFile, et le fichier ressource AFile.txt.
☐ Le fichier ressource AFile.txt.

Question 6 Le code suivant compile-t-il ? (On suppose que le code est placé dans des fichiers nommés de façon adéquate et que les imports éventuels sont corrects.)

```
public interface MyInterface<V> {
  public V myFirstMethod();
}

public class MyClass<V> implements MyInterface<Integer> {
  public void myMethod() {
    System.out.println("Coucou.");
  }
  public Integer myFirstMethod() {
    return null;
  }
}
```

- ☐ Non
- ☐ Oui

Question 7 ♣ Cocher toutes les affirmations correctes concernant HEAD dans un dépôt git quelconque.

- ☐ HEAD représente l'équivalent distant de l'historique sauvegardé localement par git, lorsqu'un serveur distant est configuré
- ☐ HEAD pointe toujours vers une branche (directement ou indirectement)
- ☐ HEAD pointe, généralement mais pas toujours, vers une branche (directement ou indirectement)
- ☐ HEAD pointe toujours vers un commit (directement ou indirectement)
- ☐ HEAD représente un sous-ensemble de l'arbre des commits
- ☐ HEAD pointe, généralement mais pas toujours, vers un commit (directement ou indirectement)
- ☐ *Aucune de ces réponses n'est correcte.*

Question 8 ♣ Considérons la classe suivante.

```
public class MyClass {
  public void printOut(Set<String> s) {
    // nothing
  }
  public static void main(String args[]) {
    …
  }
}
```

Quels codes sont corrects, en supposant qu'ils soient insérés dans la méthode `main` ci-dessus à la place des points de suspension ?

- ☐ `HashSet<String> strs = new HashSet<String>(); printOut(strs);`
- ☐ `HashSet<String> strs = new HashSet<>(); printOut(strs);`
- ☐ `Set<String> strs = new HashSet<>(); printOut(strs);`

- ☐ `HashSet<Integer> strs = new HashSet<>(); printOut(strs);`
- ☐ `HashSet<String> strs = new HashSet<Integer>(); printOut(strs);`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 9 Considérer le code Java suivant (supposé constituer une méthode).

```
System.out.println("Coucou1");
Exception e = new IllegalArgumentException();
System.out.println("Coucou2");
throw e;
System.out.println("Coucou3");
```

Cette méthode

- ☐ ne compile pas
- ☐ compile et affiche `Coucou1` puis `Coucou2` puis lance une exception, et n'affiche rien d'autre
- ☐ compile et affiche `Coucou1` puis lance une exception, et n'affiche rien d'autre
- ☐ compile et affiche `Coucou1` puis `Coucou2` puis lance une exception puis affiche `Coucou3` si et seulement si l'appelant de la méthode "attrape" (`catch`) l'exception lancée
- ☐ compile et affiche `Coucou1` puis `Coucou2` puis lance une exception puis affiche `Coucou3` si et seulement si l'appelant de la méthode n'attrape pas l'exception lancée
- ☐ compile et affiche `Coucou1` puis `Coucou2` puis lance une exception puis affiche `Coucou3`

Question 10 ♣ Considérer une méthode d'en-tête suivant, qui retourne $\sqrt{a+b}$.

```
public double sqSum(double a, double b);
```

Le principe de la programmation par contrat requiert (cocher tout ce qui s'applique) de

- ☐ logger les valeurs des paramètres reçus et de la valeur renvoyée
- ☐ documenter l'en-tête des méthodes pour indiquer le fonctionnement détaillé de la méthode de façon suffisamment claire pour que l'utilisateur de la méthode sache comment la méthode procède pour calculer son résultat, sans devoir lire le code source (par exemple en indiquant l'algorithme utilisé pour le calcul, ou les méthodes appelées)
- ☐ documenter l'en-tête des méthodes pour indiquer les contraintes sur les paramètres
- ☐ affecter tout résultat intermédiaire à une variable, dans le corps de la méthode
- ☐ utiliser `assert` pour avertir l'utilisateur final lorsqu'il entre des valeurs incorrectes dans l'interface graphique (si une interface graphique est utilisée)
- ☐ documenter l'en-tête des méthodes pour indiquer les garanties sur les résultats
- ☐ *Aucune de ces réponses n'est correcte.*