Réponses - Questionnaire 03

1. Pourquoi est-il dangereux dans un programme de comparer deux réels avec l'opérateur == ?

> Car les réels ne sont que des approximations (pas extrêmement précis)
>
> ```
> double val = 0.33333333333333;
> if (val == 1.0/3) {    // ON N'ENTRE PAS DANS LE IF !!!!!!!!!
>     System.out.println("La valeur vaut un tiers");
> }
> ```

2. Comment compare-t-on deux réels entre eux (sans l'opérateur ==) ?

> On les compare en tenant compte d'une tolérance admissible de l'imprécision.
>
> ```
> if (abs(valeurCible - valeurReel)<=toleranceAdmissible) {...}
> ```

3. Quelle est la question ou les questions à se poser pour choisir le bon type de boucle ?

> "Connaît-on le nombre de répétition à faire ?"
> - Si oui => boucle `for`
> - Sinon :
>
> "Faut-il faire le traitement au moins une fois ?"
> - Si oui => boucle `do...while`
> - Sinon => boucle `while`

4. Qu'est ce qui différencie la boucle `do...while` de la boucle `while` ?

> La boucle `do...while` a sa condition à la fin de la boucle (donc se fait au moins une fois),
> La boucle `while` a la condition au début de la boucle (donc ne se fait peut-être pas)

5. La boucle `for` nécessite trois instructions (dans ses parenthèses) pour pouvoir fonctionner. Quelles sont-elles ?

```
for (...; ...; ...) {
    // instructions à répéter
}
```

> - La première consiste à déclarer une variable de boucle est de lui donner une valeur de départ.
> - La seconde correspond à la condition pour laquelle on reste dans la boucle.
> - La dernière sert à indiquer comment la variable de boucle change à chaque itération

```
for (int compteur=1; compteur<=10; compteur++) {
    // instructions à répéter
}
```

6. A quoi doit-on faire attention lors de l'utilisation d'une boucle `do...while` ou d'une boucle `while` ?

> Il faut absolument déclarer toutes les variables nécessaires à l'écriture de la condition AVANT la boucle.
>
> Exemple d'une boucle qui ne fonctionne pas car la variable `ok` n'est pas déclarée avant la boucle.
>
> ```
> do {                // PIEGE !!!
>     boolean ok;
>     ok = ...;
> } while (ok);
> ```
>
> voici comment procéder :
>
> ```
> boolean ok;         // BOUCLE FONCTIONNELLE
> do {
>     ok = ...;
> } while (ok);
> ```
>
> Ceci est dû au fait qu'une variable n'existe que dans le bloc "{ }" dans laquelle elle est déclarée !

7.) Soit la boucle suivante:

```
while (aEncoreUneVie || nbPoints>=1000) {   //     || signifie OU ;-)
    // continue la partie
}
```

Quelle(s) condition(s) doit/doivent être réunie(s) pour qu'on puisse quitter la boucle ?

> Il faut que la variable `aEncoreUneVie` soit à `false`
>
> ET (😉)
>
> que `nbPoints` soit inférieur à 1000

8.) Quelle est la différence entre les instructions `System.out.print(...)` et `System.out.println(...)` :

```
System.out.print("Bonjour, ");
System.out.println("les amis");
```

- `System.out.print` Affiche ce qu'il y a dans les parenthèses dans la console (sans passer à la ligne)
- `System.out.println` Affiche ce qu'il y a dans les parenthèses dans la console (puis, passe à la ligne)

9.) Quelle est la différence entre ces deux boucles for :

```
// 1ère boucle
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```
// 2ème boucle
int i;
for (i = 0; i < 10; i++) {
    System.out.println(i);
}
```

Dans la première boucle, la variable de boucle `i` meurt à la fin de la boucle.
Dans la seconde boucle, la variable de boucle `i` est toujours accessible après la boucle

```
// 1ère boucle
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
System.out.println(i); // PAS POSSIBLE CAR LA VARIABLE i N'EXISTE PLUS
```

```
// 2ème boucle
int i;
for (i = 0; i < 10; i++) {
    System.out.println(i);
}
System.out.println(i); // POSSIBLE, CAR LA VARIABLE i EXISTE ENCORE
```

10.) A quoi sert le "debugger" ?

Le debugger permet d'exécuter le programme en mode "instruction par instruction" (F8) et de voir l'effet de chacune d'elle sur les variables du programme.