Correction de la série des exercices N°1

Correction

1. Pourquoi Java est appelé un langage Independent de la plateforme (*Platform-Independent*) ?
   Java est indépendant de la plateforme, par ce que après la compilation d'un code Java on obtient le Bytecode qui n'est pas un code machine natif, ainsi il est portable et ne dépend ni du système ni de l'architecture du processeur sur lequel la compilation du code source a eu lieu. Ce Bytecode ne s'exécute pas directement sur le système d'exploitation, mais il est interprété par une machine virtuelle propre au système sur lequel le programme est exécuté. (Chaque système possède sa propre machine virtuelle Java) ; ainsi les programmes Java sont *Platform-Independent* mais la JVM est *Platform-dependent*
2. Est-ce que Java est un langage 100 % orienté objet ? Pourquoi ?
   Pour être un langage 100% objet il faut respecter les principes de la POO ci-dessous :
   1) Encapsulation
   2) Héritage
   3) Polymorphisme
   4) Abstraction
   5) Tous les types prédéfinis sont des objets
   6) Tous les types définis par l'utilisateur sont des objets
   7) Toutes les opérations effectuées sur des objets ne doivent être effectuées que par des méthodes exposées sur les objets.

   Pour Java les points 5) et 7) ne sont pas vérifiés, en effet, les types primitifs ne sont pas des objets et les méthodes déclarées avec le mot clé *static* peuvent être invoquées sans passer par un objet.
3. Qu'elle est la différence entre Double et double?
   double est un type primitif qui permet de définir une variable de type réel, Double est une classe enveloppe qui permet de présenter les réels.
4. Donner un code qui permet de convertir un "int" en "char" ?
   Par un simple cast . Par exemple :
   ```
   int i = 44;
   char c;
   c = (char)i;
   ```
5. Donner un code qui permet de convertir un caractère majuscule en un caractère minuscule.
   Si maj contient un caractère majuscule alors le code suivant permet d'avoir son minuscule :
   ```
   char min = (char) ( 'a'+ ( maj-'A') ) ;
   ```
6. Quels sont les types primitifs et quelles sont leurs valeurs par défaut?

boolean	false
char	Caractère du code nul '\0'
byte, short, int, long	0
float, double	0.0

7. Expliquer comment un programme Java libère la mémoire non utilisée. comment est appelé ce mécanisme ?
   En java la mémoire se libère par la JVM via le mécanisme appelé Garbage Collector. (en francais : ramasse-miettes), tous les objets non référencés dans le programme seront automatiquement supprimés de la mémoire.
8. Quel est le résultat de l'opération suivante en JAVA : 9/2 + 9.0/2 ?

9/2 donne 4 et 9.0/2 donnera 4.5 donc le résultat est 4+4.5 = 8.5

9. Donner la valeur des expressions Java ci-dessous :
   a) 5.0 / 4 - 4 / 5
   b) 7 < 9 - 5 && 3 % 0 == 3
   c) "B" + 8 + 4

   d) 1.25
   e) false
   f) B84

10. Est-ce qu'il est possible d'affecter null à une variable de type double ?
    Non car null présente une référence non initialisée et qui ne référence aucun objet. Par contre une variable de type double n'est pas un objet.
11. Est-ce qu'il est possible d'affecter null à une variable de type Double ?
    Oui, car Double est une classe.
12. Quelle est l'instruction incorrecte dans le code suivant :
    int i = 32;
    float f = 45.0;
    double d = 45.0;

    float f = 45.0; car 45.0 est de type double et ne peut pas être affectée à float la bonne instruction est
    float f = 45.0f;
13. Soit c une variable de type char, L'expression *c++* est-elle équivalente à *c = c + 1* ? sinon pourquoi ?
    Non, c=c+1 est une instruction incorrecte en effet c+1 est de type int ; et on ne peut pas affecter un int à un char.
    c++ est équivalent à c = (char) (c+1);
14. Quelles sont les erreurs contenues dans le code ci-dessous ?

```
int n ;
short p ;
char c ='1';
byte b1, b2 ;
n = b1 * b2+ c ;
p = b1 * b2 ;
```

l'erreur est dans la ligne p = b1 * b2; car b1 * b2 est de type int et ne peut pas être affectée à short

15. Qu'affiche le programme suivant :

```
public static void main(String[] args) {
    int i = 10, j = 10, ii = 10, jj = 10;
    boolean bool1, bool2;
    bool1 = true | (i++ < 0);
    bool2 = true || (j++ < 0);
    System.out.println("i:" + i + " j:" + j);
    bool1 = false & (ii++ < 0);
    bool2 = false && (j++ < 0);
    System.out.println("ii:" + ii + " jj:" + jj);
    bool1 = bool1 ^ true;
    System.out.println(bool1);
    bool1 = bool1 ^ true;
    System.out.println(bool1);
}
```

    Le code affichera :
    i:11 j:10
    ii:11 jj:10
    true
    false

16. Qu'affiche le programme suivant :

```
public class GiTest1 {
  public static void main(String[] args) {
    byte choix =0;
    choix= choix+1;
    switch(choix){
    case 1 : System.out.println("ok");
    case 2 : System.out.println("ok");break;
    case 3 : System.out.println("NO");
    default : System.out.println("KO");
    }
  }
}
```

    Erreur de compilation dans la ligne choix= choix+1
    Si on déclare choix de type int le programme affichera
    ok
    ok

17. Soient ces déclarations :

byte b ; **short** p ; **char** c ; **int** n ; **float** x ;

Parmi les expressions suivantes, lesquelles sont incorrectes et pourquoi ?

```
c = c + 2;
c--;
c += 1;
b -= c;
p -= b;
p = p – 2* b;
n -= x;
n = n + (x++);
x++;
```

c = c + 2; on ne peut pas convertir de int à char
p = p – 2* b; on ne peut pas convertir de int à short
n = n + (x++); on ne peut pas convertir de float à int

Remarque : Comme les affectations usuelles, les affectations élargies impliquent une conversion dans le type de leur (unique) opérande. Mais elles acceptent une conversion ne respectant pas les hiérarchies légales.
b+=x est équivalent à b = (type de b) (b+x) ;
Attention ceci peut conduire à des erreurs. Par exemple, le programme ci-dessous

```
public static void main (String args[])
{
        byte b=10 ;
        int n = 10000 ;
        b+=n ;
        System.out.println ("b = " + b) ;
}
```

affiche b = 26. En effet b+n est de type int et = 10010 à la conversion en byte on perd une partie de la donnée à cause de dépassement de capacité d'un byte.

18. Soient ces déclarations :

byte b ; **short** p ; **int** n ; **long** q ;
final int N=10 ;
float x ; **double** y ;

Parmi les expressions suivantes, lesquelles sont incorrectes et pourquoi ?

```
b = n
b = 25
b = 500
x = 2*q
y = b*b
p = b*b
b = b+5
p = 5*N-3
```

Les instructions contenant les erreurs sont :
b = n ;
b = 500 ; car 500 bien qu'elle est une constante entière mais elle dépasse la capacité de byte, il n'y aura pas d'erreur si nous remplaçons par b = 127 par exemple;
p = b*b ;
b = b+5 ;

19. Soit le programme :

```
public class FirstProgram {
	public static void main(String args[]) {
		byte lbyte1 = 50, lbyte2 = 100;
		int n;
		n = lbyte1 * lbyte2;
		System.out.println(lbyte1 + "*" + lbyte2 + " = " + n);
		int n1 = 100000, n2 = 200000;
		long p;
		p = n1 * n2;
		System.out.println(n1 + "*" + n2 + " = " + p);
	}
}
```

Le résultat de l'exécution de ce programme est :
50*100 = 5000
100000*200000 = -1474836480
Expliquer pourquoi ?

Dans l'instruction p = n1 * n2; le produit calculé en int ainsi le résultat est un int ce qui conduit à un dépassement de capacité, Java ne conserve que les bits les moins significatifs d'où la dégradation de la valeur le résultat déjà dégradé est affecté à un long.

20. Soient ces déclarations :

```
char c = 60, ce = 'e', cg = 'g' ;
byte b = 10 ;
```

Donner le type et la valeur des expressions suivantes :

- c + 1
- 2 * c
- cg - ce
- b * c

- Conversion de type promotion numérique de c en int le résultat est de type int : c + 1 = 61
- Conversion de type promotion numérique de c en int le résultat est de type int : 2 * c = 120
- cg - ce = 2 En effet l'opérateur - soumet ici ses deux opérandes à la promotion numérique de char en int. On obtient un résultat de type int qui représente l'écart entre les codes des caractères g et e (dans le code Unicode, les lettres consécutives d'une même casse ont des codes consécutifs).
- Conversion par promotion numérique de b et c en int le résultat est de type int b * c = 600

21. Soit le code suivant :

```
class MyProg {
	public static void main(String[] args) {
		if (args.length == 1 | args[1].equals("test")) {
			System.out.println("test case");
		} else {
			System.out.println("production " + args[0]);
		}
	}
}
```

Si nous exécutions la commande : **java MyProg live2** que sera le résultat d'affichage ?

Ce programme va engendrer une erreur à l'exécution dans args[1].equals("test") car le tableau args ne contient qu'un seul élément, et donc 1 est un indice hors des limites. L'exception qui sera remonté par la JVM est ArrayIndexOutOfBoundsException

22. Soit le code suivant :

```
class MyProg {
	public static void main(String[] args) {
		if (args.length == 1 | | args[1].equals("test")) {
```

```
                    System.out.println("test case");
                } else {
                    System.out.println("production " + args[0]);
                }
            }
        }
```

Si nous exécutions la commande : **java MyProg live2** que sera le résultat d'affichage ?

A la différence de la question précédente nous avant un « ou logique avec court-circuit », ainsi, l'expression args[1].equals("test") ne sera pas exécutée. Et le programme affichera dans ce cas test case.

23. Qu'affiche le code ci-dessous ?

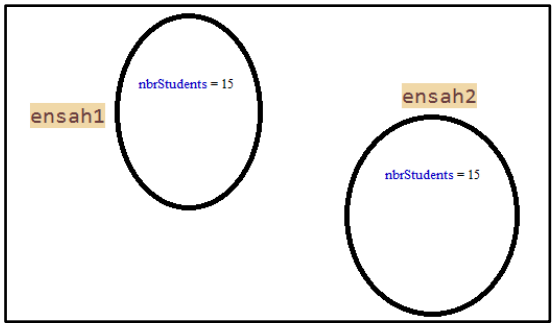
```
class ProgENSAH {
    public static void main(String[] args) {
    Integer i = 42;
    String s = (i < 40) ? "al-hoceima" : (i > 50) ? "Imzouren" : "Beni Bouayach";
    System.out.println(s);
    }
}
```

On a :
(i < 40) ? "al-hoceima" : (i > 50) ? "Imzouren" : "Beni Bouayach"
➔ *(i > 50) ? "Imzouren" : "Beni Bouayach"*
➔ *"Beni Bouayach"*
Donc le programme affiche : Beni Bouayach

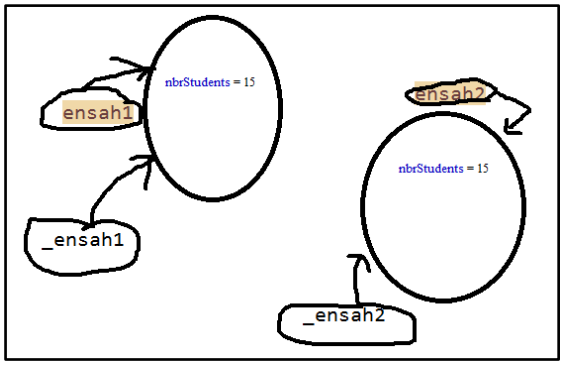
24. Qu'affiche le code ci-dessous ?

```
class Ensah {
    int nbrStudents = 15;
    public static void main(String[] args) {
        final Ensah ensah1 = new Ensah();
        Ensah ensah2 = new Ensah();
        Ensah ensah3 = EnsahMethod(ensah1, ensah2);
        System.out.println((ensah1 == ensah3) + " " + (ensah1.nbrStudents == ensah3.nbrStudents));
    }
    static Ensah EnsahMethod(Ensah pEnsah1, Ensah pEnsah2) {
        final Ensah ensah = pEnsah1;
        ensah.nbrStudents = 16;
        return ensah;
    }
}
```

Après les deux instructions
final Ensah ensah1 = new Ensah();
Ensah ensah2 = new Ensah();
La situation en mémoire pourra être schématisée par :



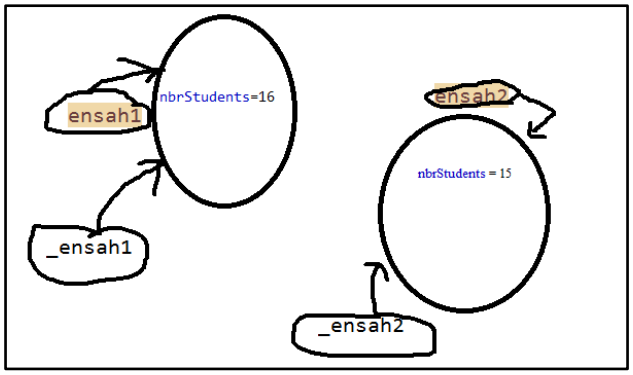
L'appel EnsahMethod(ensah1, ensah2) va copier les références vers les paramètres fictifs ensah1 et ensah2. Pour éviter la confusion on changera les noms des paramètres fictifs ainsi : _ensah1, _ensah2



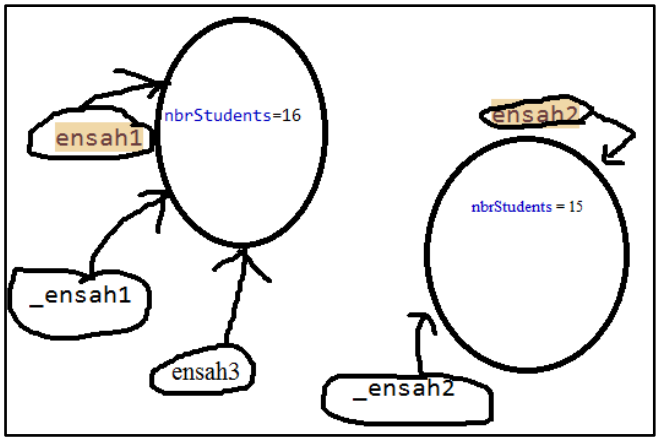
Le corps de la méthode est donc peut être vu ainsi :

```
final Ensah ensah = _ensah1;
_ensah1.nbrStudents = 16;
return _ensah1;
```

Instruction _ensah1.nbrStudents = 16 va changer la valeur de nbrStudents dans l'instance référencée par _ensah1 et ensah1:



L'instruction EnsahMethod(ensah1, ensah2) retourne la référence _ensah1, ainsi ensah3 et _ensah1 référencent la même instance



Ainsi, à partir du dernier schéma on peut conclure que la valeur de la référence stockée dans les deux variables ensah1 et ensah3 est la même (car référencent même instance) ainsi ensah1 == ensah3 est une expression vraie. Donc ensah1.nbrStudents == ensah3.nbrStudents est vraie aussi.
Finalement le programme affichera : true true

25. Que pouvez-vous dire de l'affectation de la valeur null lors de la déclaration de maVariable dans la classe ci-dessous :

```
public Class A{
  private String maVariable = null;
….
}
```

Ce n'est pas nécessaire, null est la valeur par défaut des objets. Il serait alors redondant d'affecter null à l'attribut maVariable.

26. Un attribut, n'est ni public, ni protected, ni private, quelle est donc sa portée ?

Porté package. Il ne sera visible que dans le même package où la classe est définie.

27. Y'a-t-il une erreur dans la classe suivante ? Si oui laquelle ?

```
public class Toto {
 private String monAttrribut;
 public static String maMethode(){
  if ( monAttribut == null ){
  monAttribut = "Mon attribut";
  }
  return monAttrribut;
 }
```

Oui il y aura une erreur de compilation, car l'attribut doit être static ou bien, la méthode ne doit pas être static.
(Les méthodes de classes ne peuvent pas manipuler les attributs d'instance)

28. Soit maChaine une variable String. De manière générale, vaut-il mieux utiliser : maChaine.equals("abc") ou bien "abc".equals(maChaine) ? pourquoi ?

En utilisant la seconde expression, on évite toute possibilité de subir une erreur *NullPointerException* (car avec la première on n'est pas certain que maChaine soit non null)

29. Quel est le résultat de l'expression suivante : (obj1==obj2) où obj1 et obj2 sont deux objets.

Ici on teste l'identité et non pas l'égalité, en effet les objets en Java sont représentés dans les programmes via leurs références. Par conséquent cette expression ne serait vraie que si obj1 et obj2 référencent le même objet.

30. A quoi sert le mot clé final en java ? donner une réponse détaillée en prenant en compte toutes ses possibilités d'utilisation.

Final empêche la modification, ainsi :
Si final est placé sur une classe : empêche l'héritage de la classe
Si final est placé sur une méthode : empêche la redéfinition de la méthode
Si final est placé sur une variable : empêche la modification de la variable
Si final est placé devant un paramètre fictif de la méthode : empêche la modification de la valeur de ce paramètre dans le corps de la méthode.

31. Qu'est-ce qu'un Design Pattern ?

Patron/Modèle de développement permettant de résoudre une problématique récurrente en programmation orientée objet. Il est accompagné des indications sur "quand" et "comment" appliquer ces solutions à de nouveaux contextes.

32. Qu'est-ce qu'un Design Pattern Singleton ? Comment peut-on le coder en Java ?

L'objectif du patron de conception singleton est de garantir qu'une classe ne possède qu'une seule instance et de fournir un point d'accès global à celle-ci.

```
public class MySingleton {
    private static MySingleton INSTANCE;

    private MySingleton() {
       // ... code du constructeur
    }
    public static MySingleton getInstance() {
       if (INSTANCE == null)
           INSTANCE = new MySingleton();
       return INSTANCE;
    }
    // ... Reste de la classe
}
```

N.B : pour le cas de multithreading la solution présentée n'est pas adaptée. Il existe plusieurs solutions possibles pour le cas du multithreading, une de ces méthodes a été présentée dans le cours, une autre façon (simpliste) est de synchroniser la méthode getInstance :

```
public static synchronized MySingleton getInstance(){
                   if (INSTANCE == null)
                       INSTANCE = new MySingleton();
                   return INSTANCE;
}
```

D'autres solutions plus performantes seront vues dans le cours sur les Threads.

33. On considère la suite d'instructions suivantes :

```
A x,u,v ;
x=new A();
A y=x;
A f = x;
A z=new A();
```

Combien d'instances de la classe A sont créés ? Pourquoi ?

Deux instances (crées à ligne 2 et la ligne 5).

34. Pour la classe C définie comme suit:

```
class C {
public static int i;
public int j;
public C() {i++; j=i; }
}
```

Qu'affichera le code suivant ?
C x=new C(); C y=new C(); C z= x;
System.out.println(z.i + "et "+ z.j);

Le programme affiche : 2et 1.
En effet :
Le champ de classe C.i est initialisé par défaut à 0
L'instruction C x=new C(); initialise par défaut le champs d'instance j à 0 (x.j=0) puis incrémente la valeur de C.i donc C.i= 1 et affecte i à x.j donc x.j=i donnera x.j=1
L'instruction C y=new C() initialise par défaut le champ d'instance y.j =0 et par la suite incrémente le champs de classe C.i donc C.i = 2 et y.j=2
L'instruction C z=x est une affectation entre référence ainsi z référence la même instance référencée par x
Donc z.i = x.i = 2 et z.j = x.j = 1

Remarque :
Bien qu'il soit possible d'accéder à un champ statique avec un objet comme dans z.i, il est préférable d'utiliser le nom de la classe pour accéder aux champs de classe donc z.i devrait etre écrite C.i

35. Quelle erreur a été commise dans cette définition de classe ?

```
class Math {
    private final float pi;
    private final int v = 10;
    private final int x;
    private final int z = v;
    public ClassA(float pPi) {
        pi = pPi;}}
```

Les attributs de types final *pi et x* ne sont pas initialisés. Ce qui conduit à deux erreurs de compilation
Un attribut final doit être initialisé explicitement à sa déclaration ou dans tous les constructeurs.

36. Quel est le résultat de l'exécution du programme ci-dessous :

```
public class MyMath {
  public static int  add(int i, int j){
        System.out.println(1);
        return i+j;
  }
  public static double  add(double i, int j){
        System.out.println(2);
        return i+j;
  }
  public static double  add(double i, double j){
        System.out.println(3);
        return i+j;
  }
  public static float  add(float i, float j){
        System.out.println(4);
        return i+j;
  }
  public static double  add(double i, float j){
```

```
        System.out.println(5);
        return i+j;
    }
    public static double  add(float i, double j){
        System.out.println(6);
        return i+j;
    }
    public  static void main(String[] args) {
        float f =0.1f;long lLong =1;byte b =1;
        MyMath.add(1, 2); // add(int, int) sera appelée donc affiche 1
        MyMath.add(1, 1.2); // la plus adaptée est add(float, double) donc affiche 6
        MyMath.add(1.1, b); // la plus adaptée est add(double, int) donc affiche 2
        MyMath.add(lLong, 1.2); // la plus adaptée est add(float, double) donc affiche 6
        MyMath.add(1*0.1, 1/2); // add(double, int) sera appelée donc affiche 2
        MyMath.add(1.2, 1.2); // add(double, double) sera appelée donc affiche 3
        MyMath.add(b, 1.2); // la plus adaptée est add(float, double) donc affiche 6
        MyMath.add(f, 1.2); // add(float, double) sera appelée donc affiche 6
        MyMath.add(0.1*2/2,(float) 1); // add(double, float) sera appelée donc affiche 5
        MyMath.add(1, f); // la plus adaptée est add(float, float) donc affiche 4
        MyMath.add(lLong, f);     // la plus adaptée est add(float, float) donc affiche 4
    }
}
```

Le résultat est donné ci-dessous, l'explication est donnée sous forme de commentaires dans le code :

```
1
6
2
6
2
3
6
6
5
4
4
```

37. Quelles erreurs figurent dans la définition de la classe suivante ?

```
class ExempleSurdef {
    public void afficheInt(int n) {
        System.out.println(n);
    }
    public int afficheInt(int p) {
        System.out.println(p);
    }
    public void afficheFloat(final float x) {
        System.out.println(x);
    }
    public void afficheFloat(double x) {
        System.out.println(x);
    }
    public void afficheLong(long n) {
        System.out.println(n);
    }
    public int afficheLong(final long p) {
        System.out.println(p);
    }
}
```

La méthode afficheInt est dupliquée. Le type de retour n'intervient pas dans la surcharge

```
public void afficheInt(int n) {
    System.out.println(n);
```

```
}
public int afficheInt(int p) {
        System.out.println(p);
}
```

De meme la méthode afficheLong est dupliquée.

38. Quel est le résultat de l'exécution du programme ci-dessous :

```
public class MyMath {
  public static double  add(double i, int j){
    System.out.println(2);
    return i+j;
  }
  public static double  add(double i, double j){
    System.out.println(3);
    return i+j;
  }
  public static float  add(float i, float j){
    System.out.println(4);
    return i+j;
  }
  public static double  add(byte i, double j){
    System.out.println(6);
    return i+j;
  }
  public  static void main(String[] args) {
    float f =0.1f;byte b =1;
    MyMath.add(b, f);
  }
}
```

Erreur dans l'instruction MyMath.add(b, f); à cause d'une ambiguïté dans la méthode add à appeler. Cette ambiguïté est entre les deux méthodes :

float add(float i, float j)
et
double add(byte i, double j)

39. Détecter les appelles incorrectes de la méthode add dans la méthode main de la classe suivante :

```
public class Calcul {
  public float add(int n, float x) {
    return n + x;
  }
  public static void main(String[] args) {
    Calcul lCalcul =  new Calcul ();
    int n = 1,m = 1;
    byte b = 1;
    float x= 1.0f;
    double y= 1.2;
    lCalcul.add(n, x); //  (int, float) ➔ ok
    lCalcul.add(b + 1, x); // b + 1 ➔ byte +int ➔ int (promotion numérique)  ➔ (int,float) ➔ok
    lCalcul.add(b, x); //  (byte, float) , byte sera convertit en int ➔ ok
    lCalcul.add(n, y); //  erreur de compilation car double ne peut pas être implicitement convertit en float
    lCalcul.add(n, (float) y); // (int, float) donc OK
    lCalcul.add(n, 21 * x); //  (int, int * float) ➔ (int, float) ➔ ok
    lCalcul.add(n + 9, x + 0.21); //x+0.21 ➔ float+double ; donc de type double ➔ erreur de compilation
    lCalcul.add(0.2, x); //0.2 est de type double donc erreur de compilation
```

```
        lCalcul.add(m+n, x); // OK
        lCalcul.add(n, x*(m+n)); // OK
    }
}
```

La réponse est insérée comme commentaire dans le code

40. Détecter les appelles incorrectes de la méthode add dans la méthode main de la classe suivante :

```
public class MyCalcul {
    public float add(byte b, float x) {
        return b + x;
    }
    public static void main(String[] args) {
        MyCalcul lCalcul;
        int n;
        byte b;
        float x;
        lCalcul.add(b, x);
        lCalcul.add(b, 0.1f);
        lCalcul.add(b, 0.1 * n);   // erreur car 0.1 * n est de type double inconvertible en float
        b = 12;
        lCalcul.add(b, x);
        lCalcul.add(12, x);  // erreur car 12 est de type int inconvertible en byte
    }
}
```

41. Si on transmet un paramètre int à une méthode et on modifie la valeur de ce paramètre dans la méthode, la variable int d'origine qui a été transmise reste inchangée. Expliquer pourquoi.

    Car le passage de paramètre en Java se fait par valeur. C'est les valeurs des variables qui sont copiées vers les paramètres fictifs de la méthode ; la méthode effectue par la suite son traitement sur les paramètres fictifs et non pas directement sur les variables qui restent donc inaffectés.

42. Est-il possible de forcer le Garbage collector en Java ? Si oui comment ?

Dans Java la mémoire est gérée par la JVM, on ne peut pas forcer donc l'appel de GC.

La méthode System.gc () fournit simplement un "indice" à la JVM que la récupération de mémoire doit être exécutée. Ce n'est pas garanti que la JVM répond immédiatement.

43. Qu'affiche le programme :

```
public class Prog {
    private int i;
    public void test() {
            System.out.println(i);
    }
    public static void main(String[] args) {
            Prog p = new Prog();
            p.test();
    }
}
```

Prog p = new Prog() ➔ l'attribut i prend la valeur par défaut qu'est 0

p.test() ➔ affiche donc 0

44. Qu'affiche le programme

```
public class Prog {

    public void test() {
            int i;
            System.out.println(i);
    }
    public static void main(String[] args) {
            Prog p = new Prog();
            p.test();
    }
}
```

Erreur de compilation dans System.out.println(i) car i n'est pas initialisé, rappelé bien donc que l'initialisation par défaut concerne les attributs et non pas les variables locales.

45. Qu'affiche le programme ci-dessous ?

```
class Exemple2{
  public Exemple2() {
            System.out.print("Bonjour ");
  }
  public Exemple2(int i) {
            this();
            System.out.println("ENSAH " + i);
  }
  public Exemple2(double i) {
            this();
            System.out.println("Imzouren " + i + 2 + 1);
  }
  public static void main(String[] args) {
            Exemple2 lObj=new Exemple2(2011.);
  }
}
```

« 2011. » est de type double, donc le constructeur Exemple2(double i) est appelé.

this() ➔ appel du constructeur sans arguments qui affiche Bonjour
System.out.println("Imzouren " + i + 2 + 1); ➔ "Imzouren " + i + 2 + 1 ➔ ((("Imzouren " + i) + 2) + 1) ➔ ((("Imzouren " + 2011.0) + 2) + 1) ➔ ((("Imzouren " + "2011.0") + 2) + 1) ➔ (("Imzouren 2011.0" + 2) + 1) ➔("Imzouren 2011.0" + "2") + 1) ➔("Imzouren 2011.02" + 1) ➔("Imzouren 2011.02" + "1")➔"Imzouren 2011.021"

Finalement le programme affiche : Bonjour Imzouren 2011.021

46. Lesquelles des méthodes ci-dessous respectent la norme JavaBeans ?
    a. addStudent
    b. getSize
    c. deleteUser
    d. isGreen
    e. putStudents

Rappel sur la norme JavaBeans :

Un composant JavaBean est une simple classe Java qui respecte certaines conventions sur le nommage, la construction et le comportement des méthodes. Le respect de ces conventions rend possible l'utilisation, la réutilisation, le remplacement et la connexion de JavaBeans par des outils de développement.

Les conventions à respecter sont les suivantes :

- ✓ la classe doit être « Serializable » pour pouvoir sauvegarder et restaurer l'état d'instances de cette classe
- ✓ la classe doit posséder un constructeur sans paramètre (constructeur par défaut) ;
- ✓ les attributs privés de la classe (variables d'instances) doivent être accessibles publiquement via des méthodes accesseurs et mutateurs. (voir ci-dessous les règles de nommage)
- ✓ la classe ne doit pas être déclarée final.

JavaBean Property Naming Rules

- ✓ *If the property is not a boolean, the getter method's prefix must be get. For example, `getSize()` is a valid JavaBeans getter name for a property named "size." Keep in mind that you do not need to have a variable named size*
- ✓ *If the property is a boolean, the getter method's prefix is either `get` or `is`. For example, `getStopped()` or `isStopped()` are both valid JavaBeans names for a boolean property.*
- ✓ *The setter method's prefix must be set. For example, `setSize()` is the valid JavaBean name for a property named size.*
- ✓ *To complete the name of a getter or setter method, change the first letter of the property name to uppercase, and then append it to the appropriate prefix (`get`, `is`, or `set`).*
- ✓ *Setter method signatures must be marked `public`, with a `void` return type and an argument that represents the property type.*
- ✓ *Getter method signatures must be marked `public`, take no arguments, and have a return type that matches the argument type of the setter method for that property*

POJO:

Beaucoup de programmeurs Java confond JavaBeans avec POJO(plain old Java object)

POJO est un objet Java lié à aucune autre restriction que celles forcées par la spécification du langage Java. En d'autres termes, il est impératif qu'un POJO :

- ✓ n'hérite pas de classes pré-spécifiées
- ✓ n'implémente pas des interfaces pré-spécifiées
- ✓ ne contienne pas des annotations pré-spécifiées

POJO VS JavaBean

Un JavaBean est un POJO qui doit être sérialisable (implémente Serializable), a un constructeur sans arguments, et permet l'accès à des propriétés utilisant des méthodes getter et setter dont les noms sont déterminés par une convention simple.

47. Soit le code ci-dessous :

```
public class Prog {
        public static void main(String[] args) {
                doWork(1);
                doWork(1, 2);
        }
        // ligne 6
```

}

Lesquelles des méthodes ci-dessous peuvent être insérées à la ligne 6 pour que le code compile ?

a. static void doWork (int... doArgs) { }
b. static void doWork (int[] doArgs) { }
c. static void doWork (int doArgs...) { }
d. static void doWork (int... doArgs, int y) { }
e. static void doWork (int x, int... doArgs) { }

a et e

48. Soit deux fichiers :

Fichier 1:

```
package ensah;
 public class TestEnsah {
 int a = 6;
 protected int b = 7;
 public int c = 8;
}
```

Fichier 2:

```
package fsth;
 import ensah.*;
 public class TestFSTH {
 public static void main(String[] args) {
   TestEnsah t = new TestEnsah ();
   System.out.print(" " + t.a);
   System.out.print(" " + t.b);
   System.out.print(" " + t.c);
 }
}
```

Qu'il est le résultat d'exécution ?

a. 6 7 8
b. 6 puis une exception
c. Erreur de compilation à la ligne 7
d. Erreur de compilation à la ligne 8
e. Erreur de compilation à la ligne 9
f. Erreur de compilation à la ligne 10

Erreur de compilation dans TestFSTH dans les lignes

```
  System.out.print(" " + t.a);
  System.out.print(" " + t.b);
```

49. Soit le code ci-dessous :

```
class Prog {
    Prog tester(Prog cb) {
            cb = null;
            return cb;
    }
    public static void main(String[] args) {
            Prog c1 = new Prog();
            Prog c2 = new Prog();
            Prog c3 = c1.tester(c2);
            c1 = null;
```

```
            // do Work
        }
        }
```

Lorsque l'exécution arrive à //do Work, combien d'objets sont éligibles pour GC?

a. 0
b. 1
c. 2
d. Erreur de compilation
e. Impossible à savoir
f. Une erreur sera produite à l'exécution

La réponse est 1 voir l'explication dans la réponse à la question suivante.

50. Soit le code ci-dessous :

```
class Prog {
        Short test = 200;
        Prog tester(Prog cb) {
                cb = null;
                return cb;
        }
        public static void main(String[] args) {
                Prog c1 = new Prog();
                Prog c2 = new Prog();
                Prog c3 = c1.tester(c2);
                c1 = null;
                // do Work
        }
}
```

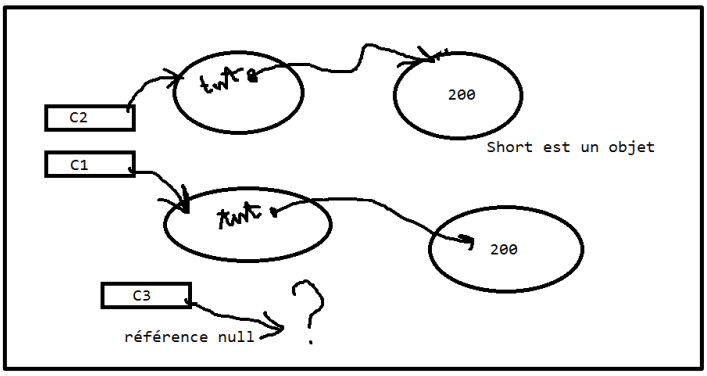
Lorsque l'exécution arrive à //do work, combien d'objets sont éligibles pour GC?

a. 0
b. 1
c. 2
d. Erreur de compilation
e. Impossible à savoir
f. Une erreur sera produite à l'exécution

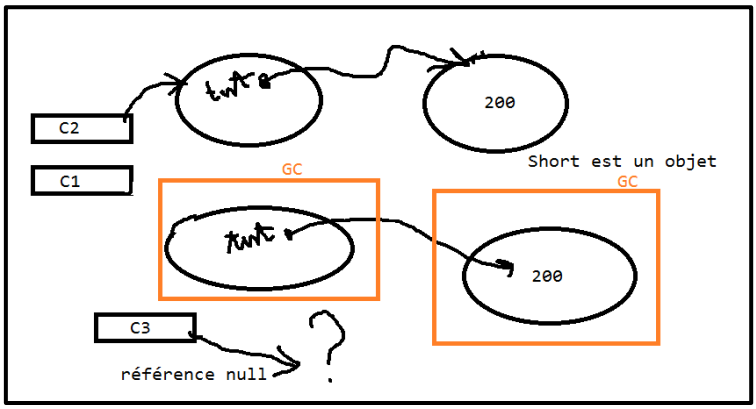
La situation après avoir exécuté la suite d'instructions :

```
Prog c1 = new Prog();
Prog c2 = new Prog();
Prog c3 = c1.tester(c2);
```

est schématisée ci-dessous :



A l'exécution de c1 = null l'une des instances de Prog sera non référencée dans le programme donc elle est candidate au GC, une fois GC supprime cette instance les objets associés ici l'objet de type Short sera également candidat au GC. Donc la réponse est 2



51. Soit le code ci-dessous :

```
class Beta { }
class Alpha {
static Beta b1;
Beta b2;
 }
 public class Tester {
 public static void main(String[] args) {
 Beta b1 = new Beta(); Beta b2 = new Beta();
 Alpha a1 = new Alpha(); Alpha a2 = new Alpha();
 a1.b1 = b1;
 a1.b2 = b1;
 a2.b2 = b2;
 a1 = null; b1 = null; b2 = null;

 // do work
 }
 }
```

Lorsque l'exécution arrive à la ligne // do work , Combien d'objets seront éligible pour GC? Expliquer pourquoi ?

Avec un raisonnement analogue à la question précédente on trouve que la réponse est 1

52. Soit le code ci-dessous :

```
class ProgENSAH {

        ProgENSAH test;
        ProgENSAH() {
        }
        ProgENSAH(ProgENSAH pTest) {
                test = pTest;
        }
        public static void main(String[] args) {
                ProgENSAH p2 = new ProgENSAH();
                ProgENSAH p3 = new ProgENSAH(p2);
```

```
                p3.doWork();
                ProgENSAH p4 = p3.test;
                p4.doWork();
                ProgENSAH p5 = p2.test;
                p5.doWork();
        }
        void doWork() {
                System.out.print("ENSAH ");
        }
}
```

Quel est le résultat de l'exécution ?

a. ENSAH
b. ENSAH ENSAH
c. ENSAH ENSAH ENSAH
d. Erreur de compilation
e. ENSAH, puis exception
f. ENSAH ENSAH, puis exception

La réponse est f . en effet p2 a l'attribut test non initialisé or p5 = p2.test donc p5=null donc p5.doWork() conduit à une exception de type NullPointerException