Corrigés des exercices sur la mémoire 2

Exercice 10.1 Qu'affiche ce programme ?

Question 1

```
1  public class Mem2_2{
2     public static void plusDix(int[] tab, int x){
3        x = x+10;
4        for (int num=0; num<tab.length; num=num+1){
5           tab[num] = tab[num] + 10;
6        }
7     }
8     public static void main(String[] args){
9        int[] nombres = {1, 2, 3, 4, 5};
10       int nb = 3;
11       plusDix(nombres,nb);
12       System.out.println(nombres[2]);
13       System.out.println(nb);
14    }
15 }
```

```
> java Mem2_2
13
3
```

Les affectations qui changent les cases du tableau (la ligne 5 faite en boucle) modifient un espace mémoire dans le tas. Cette modification concerne donc également la tableau `nombres` de `main`. L'affectation de la ligne 3 modifie un espace de la mémoire privée de `plusDix` dans la pile. Cela n'a aucun effet sur les variables du main.

question[]

```
1  public class Mem2_1{
2     public static void exchange(int[] tab1, int[] tab2){
3        int[] tamp = tab1;
4        tab1 = tab2;
5        tab2 = tamp;
6     }
7     public static void main(String[] args){
```

```
8         int[] un = {1, 2, 3, 4, 5};
9         int[] deux = {10, 20};
10        exchange(un, deux);
11        System.out.println(un.length);
12    }
13 }
```

```
> java Mem2_1
5
```

La méthode `exchange` échange les deux tableaux passés en paramètres, mais les deux affectations qui donnent les nouvelles valeurs (lignes 4 et 5) sont des affectations aux paramètres de la méthode exchange. Cela modifie les espaces mémoires de ces paramètres dans la pile et cela n'a donc aucun effet sur les variables du `main`. On voit au premier coup d'oeil que les affectations sont dans la pile car il n'y a pas de crochets à gauche du signe =. C'est pour cela que la taille du tableau `un` est inchangée par rapport à sa valeur initiale : c'est toujours un tableau de 5 cases.

Question 2

```
1  public class Mem2_3{
2     public static void exchange(int[][] tab1, int[][] tab2){
3        int[] tamp = tab1[0];
4        tab1[0] = tab2[0];
5        tab2[0] = tamp;
6     }
7     public static void main(String[] args){
8        int[][] un = {{1, 2, 3, 4, 5}};
9        int[][] deux = {{10, 20}};
10       exchange(un, deux);
11       System.out.println(un[0].length);
12    }
13 }
```

```
> java Mem2_3
2
```

Le code est quasi-identique à celui de la question précédente. Mais cette fois les affectations des lignes 4 et 5 concernent des espaces situés dans le tas. Cela se voit à ce qu'il y a des crochets à gauche du = dans ces affectations. Ces modifications intervenant dans le tas affectent les valeurs de `un` et `deux` dans le `main`. La case `un[0]` contient un tableau à deux cases, celui qui a 10 dans la première et 20 dans la seconde.

Exercice 10.2 Dessins

Question 1

```java
public class Mem2_4{
   public static int meth2(int x, int y){
      int res;
      x = x + 1;
      y = y *2;
      // faire le dessin à ce point de l'exécution
      return x + y;
   }
   public static int meth1(int x, int y){
      int res = 12;
      x = meth2(y,x);
      res = x -y;
      return res;
   }
   public static void main(String[] args){
      int x = 3;
      int y = 2;
      System.out.println(meth1(x,y));
   }
}
```

1. Dessinez l'état de la pile au moment marqué par un commentaire dans le code. Notez qu'il n'y a rien d'intéressant dans le tas dans ce programme, aussi n'est-il pas nécessaire de le représenter dans le dessin.
2. Qu'affiche ce programme ?

Pile		
Mémoire privée de meth2		
x	:	3
y	:	6
res	:	?
Mémoire privée de meth1		
x	:	3
y	:	2
res	:	12
Mémoire privée de main		
args	:	adresse-1
x	:	3
y	:	2

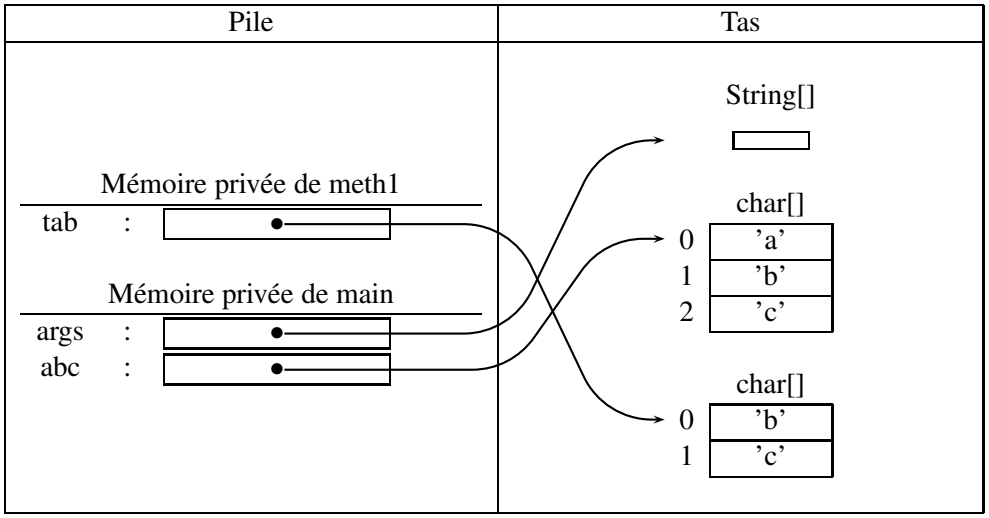
Affichage de l'exéuction :

```
> java Mem2_4
5
```

Question 2

```
1  public class Mem2_5{
2     public static char[] meth2(char[] tab){
3        char[] res;
4        res = new char[tab.length-1];
5        for (int i=0; i< res.length; i = i + 1){
6           res[i] = tab[i+1];
7        }
8        return res;
9     }
10    public static void meth1(char[] tab){
11       tab = meth2(tab);
12       // faire le dessin à ce point de l'exécution
13       tab[tab.length-1] = tab[0];
14    }
15    public static void main(String[] args){
16       char[] abc = {'a', 'b', 'c'};
17       meth1(abc);
18       System.out.println(abc);
19    }
20 }
```

1. Dessinez l'état de la mémoire (pile et tas) au moment marqué par un commentaire dans le code.
2. Qu'affiche ce programme ?



Affichage de l'exécution :

```
> java Mem2_5
abc
```

Ici encore, les affectations qui enregistrent l'adresse du nouveau tableau (lignes 4 et 11) mettent cette adresse dans la pile et cela se voit à ce qu'il n'y a pas de crochets à gauche du = dans ces deux affectations. Le tableau `abc` de `main` est inchangé.

Exercice 10.3 écureuil et noisettes

On va représenter un damier où des noisettes sont posées sur des cases (chaque noisette est comme le pion d'un jeu, il ne peut y en avoir qu'une par case). Un écureuil est placé au centre du damier. Puis il se met en mouvement de façon aléatoire et quand il arrive sur une case avec une noisette, il la mange.

La représentation du damier sera un tableau de caractères à deux dimensions avec un espace pour les cases vides, un ô pour une noisette et un 2 pour l'écureuil (voyez-vous le panache de sa queue ?). Le damier aura 11x11 cases. Le centre sera donc en ligne 5, colonne 5 (en commençant à compter à 0, comme en Java).

Question 1 *création du damier*

Le but est de poser les noisettes au hasard, sur des cases différentes à chaque fois. Pour vous aider, on vous fournit le méthode suivante qui tire au sort un nombre compris entre 0 et 10.

```
public static int auSortOnze(){
     return (int) (Math.random() *11);
}
```

Écrivez une méthode qui renvoie un damier avec un écureuil au centre et 25 noisettes disséminées au hasard. Suggestion : tirez au sort les deux coordonnées d'une case et y placer une noisette si la case n'est pas déjà occupée par une noisette ou l'écureuil.

```
public static char[][] creeDamier(){
   char[][] res = new char[11][11];
   int noisAPlacer = 25;
   int lig, col;
   for (int i = 0; i<11; i=i+1){
      for (int j=0; j<11; j=j+1){
         res[i][j]='␣';
      }
   }
   res[5][5]='2';
   while (noisAPlacer > 0){
      lig = auSortOnze();
      col = auSortOnze();
      if (res[col][lig] == '␣'){
         res[col][lig] = 'ô';
         noisAPlacer = noisAPlacer -1;
      }
   }
```

```
        return res;
    }
```

Il faut d'abord créer le damier et l'initialiser avec des espaces dans toutes les cases. Ensuite, placer l'écureuil. Ensuite, la boucle while tente de placer une noisette à chaque tour de boucle, mais elle n'y parvient pas toujours. Une case est tirée au sort, mais on ne peut y placer de noisette que si elle est vide (ni écureuil, ni noisette).

Question 2 *affichage*

Écrivez une méthode qui affiche le damier. Pour cela, elle doit parcourir toutes les cases du tableau dans lebon ordre (ligne par ligne).

```
    public static void afficher(char[][] tab){
        for (int i=0; i<13; i++){
            System.out.print("#");
        }
        System.out.println();
        for (int l = 0; l<11; l=l+1){
            System.out.print("#");
            for (int c=0; c<11; c = c+1){
                System.out.print(tab[c][l]);
            }
            System.out.print("#");
            System.out.println();
        }
        for (int i=0; i<13; i++){
            System.out.print("#");
        }
        System.out.println();
    }
```

Le damier est entouré d'un cadre de signes dièzes qui facilitent la perception des limite du damier.

Question 3 *déplacement au hasard de l'écureuil*

Écrivez une méthode qui prend en paramètre un damier et change la place de l'écureuil pour une des 4 cases voisines (1 pas horizontalement ou verticalement). Pour cela, vous pourrez utiliser la méthode suivante qui tire au sort un nombre entre 0 et 3.

```
public static int auSortQuatre(){
     return (int) (Math.random() *4);
}
```

La méthode doit-elle renvoyer le damier modifié ?

Attention : il faut faire attention à ce que l'écureuil reste bien sur le damier. S'il est sur un bord, une direction est impossible et s'il est dans un coin, il y a deux directions impossibles.

```
    public static void deplacer(char[][] tab){
        int posX, posY, direction, newX, newY;
```

```
        posX = 0;
        posY = 0;
        // recherche de la position de l'ecureuil
        while (tab[posX][posY] != '2'){
            if (posX==10){
                posX=0;
                posY=posY+1;
            }else{
                posX=posX+1;
            }
        }
        // essai de changement
        do{
            direction = auSortQuatre();
            if (direction == 0){
                newX = posX;
                newY = Math.max(0,posY-1);
            }else if (direction == 1){
                newX = Math.min(10,posX+1);
                newY = posY;
            }else if (direction == 2){
                newX = posX;
                newY = Math.min(10,posY+1);
            }else{
                newX = Math.max(0,posX-1);
                newY = posY;
            }
        }while (posX == newX && posY == newY);
        // on deplace pour de vrai
        tab[posX][posY] = ' ';
        tab[newX][newY] = '2';
    }
```

Cette méthode un peu compliquée tire au sort un nombre entre 0 et 3 qui donnent une direction (0 : vers le haut, 1 vers la droite, 2 vers le bas, 3 vers la gauche). A chaque fois, elle calcule les nouvelles coordonnées à partir des anciennes, mais avec un Math.min ou un Math.max pour empêcher d'avoir des coordonnées supérieures à 10 ou inférieures à 0. Donc, au bout du compte, si le déplacement tiré au sort est impossible, les nouvelles coordonnées sont égales aux anciennes. Dans ce cas, on reprend en boucle le tirage au sort de la direction.

Question 4 *méthode main*

Écrivez une méthode main qui alterne un déplacement de l'écureuil et une pause d'une seconde. Pour cette pause, vous utiliserez la méthode `Thread.sleep` qui prend en paramètre une durée exprimée en millisecondes (il faut 000 millisecondes pour faire une seconde). Pour pouvoir utiliser cette méthode, il faut donner comme entête à la méthode main :

```
public static void main(String[] args) throws InterruptedException
```

```
public static void main(String[] args) throws InterruptedException{
   char[][] damier = creeDamier();
   afficher(damier);
   while(true){
      Thread.sleep(1000);
      deplacer(damier);
      afficher(damier);
   }
}
```

Ce programme boucle éternellement à cause de la boucle `while(true)`. Pour l'arrêter, il faut l'interrompre brutalement (controle-c ou controle-z ou bouton reset de Drjava).

Question 5 *méthode main (version 2)*

Modifiez le programme pour qu'il s'arrête quand il n'y a plus de noisette sur le damier.

```
public static boolean fini(char[][] tab){
   int nbNois = 0;
   for (int col=0; col<11; col=col+1){
      for (int lig=0; lig<11; lig=lig+1){
         if (tab[col][lig] == 'ô'){
            nbNois = nbNois + 1;
         }
      }
   }
   if (nbNois == 0){
      return true;
   }else{
      return false;
   }
}
public static void main(String[] args) throws InterruptedException{
   char[][] damier = creeDamier();
   afficher(damier);
   while(!fini(damier)){
      Thread.sleep(1000);
      deplacer(damier);
      afficher(damier);
   }
}
```