

Corrigé de l'examen d'informatique

MIAS1 – MASS1

Marianne Durand

22 janvier 2003

1 Exercices

1.1 Exercice 1

Question 1 : Ecrire une méthode qui affiche :

```
1 2 3 4
4 1 2 3
3 4 1 2
2 3 4 1
```

Question 2 : Ecrire une méthode qui affiche le même tableau en taille $n \times n$. C'est-à-dire le tableau dont la première ligne est 1 2 3 ... n , et les suivantes sont décalées d'un cran à chaque fois comme sur l'exemple.

Reponse :

```
public static void question1(){
    System.out.println("1 2 3 4");
    System.out.println("4 1 2 3");
    System.out.println("3 4 1 2");
    System.out.println("2 3 4 1");
}

public static void question2(int n){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            System.out.print((((n+1-i)+(j-1))%n)+1+" ");
        }
        System.out.println();
    }
}
```

1.2 Exercice 2

1. Que renvoie la méthode suivante ?

Cette fonction renvoie la valeur $n!$.

```

static int toto(int n) {
    int k;
    int f = 1;

    for (k = 2; k <= n; k++){
        f = f * k;}
    return f;
}

```

2. Que se passe-t-il lorsque n devient grand ? Et si le type `int` est systématiquement remplacé par le type `long`, ou encore le type `float` ? Indice : on donne les valeurs numériques suivantes : $15! = 1307674368000$ et $2^{31} = 2147483648$.

Lorsque n devient trop grand il y a un dépassement de capacité, et la réponse deviet fausse. Avec le type `long`, ce dépassement arrive plus tard. Si on utilise le type `float`, la réponse deviendra approchée, puis fausse.

3. Que renvoie la méthode suivante ?

```

public static int decompose(int[] facteur, int n) {
    int d = 2;
    int i = 0;

    while (d <= n)
        if (n % d == 0) {
            facteur[i] = d;
            i=i+1;
            n = n/d;
        }
        else
            d = d + 1;
    return i;
}

```

Cette méthode renvoie le nombre de diviseurs premiers de n , en comptant les éventuelles répétitions.

2 Nombres Complexes

Un nombre complexe est défini par une partie réelle et une partie imaginaire. Voici le code d'une classe qui permet de coder les nombres complexes.

```

class Complex{
    float re;
    float im;
}

```

1. Ecrire un constructeur pour la classe `Complex` qui prend en argument la partie réelle et la partie imaginaire.

2. Ecrire un constructeur équivalent au constructeur par défaut.

3. Ecrire une méthode `static Complex conjugué(Complex z)` qui renvoie en résultat le conjugué de z . On rappelle que le conjugué de $x + iy$ est $x - iy$.

4. Ecrire un méthode `static Complex add(Complex z,Complex w)` qui renvoie en résultat la somme de `z` et `w`.
5. Ecrire un méthode `static Complex mult(Complex z,Complex w)` qui renvoie en résultat le produit de `z` et `w`.
6. Ecrire un méthode **non static** qui calcule le module d'un nombre complexe.

Corrigé :

```
class Complex{
    float re;
    float im;

    Complex(float re,float im){
        this.re=re;
        this.im=im;}

    Complex(){}}

    static Complex conjugue(Complex z){
        return new Complex(z.re,-z.im);
    }
    static Complex add(Complex z,Complex w){
        return new Complex(z.re+w.re,z.im+w.im);
    }
    static Complex mult(Complex z,Complex w){
        return new Complex(z.re*w.re-z.im*w.im,z.im*w.re+z.re*w.im);
    }

    double module(){
        return Math.sqrt(this.re*this.re+this.im*this.im);
    }
    public String toString(){
        return(this.re+"i"+this.im);
    }
}
```

3 Bataille Navale

Préliminaire. On joue à la bataille navale sur des grilles 10×10 . Il n'y a qu'un bateau par joueur, et il est de taille 1×1 . Chaque joueur a en mémoire une grille de boolean qui code la position de son bateau (true si la case contient le bateau et false sinon), et une deuxième grille de int, qui mémorise les coups déjà tentés sur la grille de l'adversaire. Les conventions sont que la case vaut 0 s'il ne s'est rien passé, 1 si on a mis un coup dans l'eau et 2 si on a coulé un bateau. Chaque joueur possède aussi deux attributs qui mémorisent le dernier coup tenté.

Voici la déclaration de la classe Joueur.

```
class Joueur{
    private boolean[][] bateau;
```

```

    int [][] bateauAdverse;
    int x;
    int y;
}

```

Toutes les méthodes de cette classe doivent être **non static**.

1. Ecrire une méthode void `ajouterBateau()` qui place un bateau au hasard sur la grille `bateau` du joueur.

2. Ecrire un constructeur pour la classe `Joueur` qui initialise correctement tous les attributs comme expliqué dans le préliminaire, sans oublier que le joueur doit avoir un bateau posé sur sa grille.

3. Ecrire une méthode void `proposition()` qui trouve une position pas encore proposée sur la grille de l'adversaire, et la stocke dans les variables `x` et `y`.

4. Ecrire une méthode boolean `reponse(int i,int j)` qui répond `true` si le joueur a positionné son bateau à la position `(i,j)` et `false` sinon.

5. Ecrire une méthode void `MiseAJour(boolean b)` qui met à jour le tableau `bateauAdverse` de la façon suivante. Si le boolean vaut `true`, alors la case `(x,y)` doit prendre la valeur 2, et sinon la valeur 1.

Réponse :

```

class Joueur{
    private boolean[][] bato;
    private int [][] batoAdverses;
    int x; //coordonnees du dernier coup joue
    int y;

    public Joueur(){
        bato=new boolean[10][10];
        batoAdverses=new int[10][10];
        this.ajouterBatos();
    }

    public void ajouterBatos(){
        int i=(int)(10*Math.random());
        int j=(int)(10*Math.random());
        this.bato[i][j]=true;
        System.out.println("le bato est pose en "+i+" "+j);
    }

    public void proposition(){
        int i;
        int j;
        do {
            i=(int)(10*Math.random());
            j=(int)(10*Math.random());
        }
        while(this.batoAdverses[i][j]!=0);
        x=i;
        y=j;
    }
}

```

```

public boolean reponse(int i,int j){
    if(this.bato[i][j]==true){return true;}
    else return false;
}

public void miseAJour(boolean b){
    if (b==true){batoAdverses[x][y]=2;}
    else{batoAdverses[x][y]=1;}
}
}

```

On donne maintenant le code de la classe Bataille qui veut faire jouer deux joueurs l'un contre l'autre.

<pre> class Bataille{ Joueur un; Joueur deux; Bataille(){ un=new Joueur(); deux=new Joueur(); } } </pre>	<pre> public int jeu(){ boolean fini=false; boolean t; while(!fini){ un.proposition(); t=deux.reponse(un.x,un.y); un.miseAJour(t); if(t==true){fini=true;return 1;} deux.proposition(); t=un.reponse(un.x,un.y); deux.miseAJour(t); if(t==true){fini=true;return 2;} } return 0; } } </pre>
---	---

6. Ecrire une méthode main qui permet de faire tourner une partie.

```

public static void main(String[] args){
    Bataille B=new Bataille();
    int a=B.jeu();
    System.out.println("le resultat est "+a);
}

```

7. Expliquer quelles modifications apporter aux classes Joueur et Bataille pour pouvoir gérer plusieurs bateaux.

Il faut que dans la classe Joueur il y ait un compteur du nombre de bateaux coulés, et que la classe Bataille ne déclare la victoire d'un joueur que lorsque tous les bateaux adverses sont coulés. La méthode ajouterBatos doit également être modifiée pour ajouter plusieurs bateaux sur la grille.