

Corrections des exercices du livre "Java pour les enfants, les parents et les grand-parents"

par Laurent Bernabé  jacques_jean

Date de publication : 23 avril 2012

Dernière mise à jour : 4 mai 2011

DÉBUTANT

Le livre "**Programmation Java pour les enfants, les parents et les grand-parents**" propose de nombreux exercices, mais aucune correction n'était disponible. C'est donc le but de cet article.

I - Chapitre 2.....	4
I-A - Afficher son adresse (en modifiant la classe BonjourMonde).....	4
I-B - Afficher "Bonjour" avec des étoiles.....	4
II - Chapitre 3.....	6
II-A - Création de la classe Voiture.....	6
II-A-1 - Création de la classe sans les trois méthodes demandées.....	6
II-A-2 - Petite parenthèse : les commentaires.....	6
II-A-3 - Ajout des méthodes demarrer() et arreter() dans la classe.....	6
II-A-4 - Ajout de la méthode rouler() dans la classe.....	7
II-A-5 - Code complet de la classe Voiture.....	8
II-B - Création de la classe ProprietaireVoiture.....	8
II-C - Création de la classe VoitureJamesBond.....	9
II-D - Synthèse.....	10
III - Chapitre 4.....	11
III-A - La classe ConvertisseurTemperatures.....	11
III-A-1 - La division entière et la division réelle.....	11
III-A-2 - Variable locale et initialisation.....	12
III-A-3 - Code.....	12
III-B - Modification de la classe BulletinAppreciation.....	13
III-C - Exercice pour les petits malins : félicitations aux joueurs.....	13
III-C-1 - Le code original avec le mot-clé continue.....	14
III-C-2 - Le code qu'il nous est demandé d'évaluer.....	14
III-D - Synthèse.....	16
IV - Chapitre 5.....	17
IV-A - Ajout d'un panneau opérations au code Calculatrice.java.....	17
IV-B - Alignement de l'écran de la calculatrice à droite.....	19
IV-C - Exercice pour les petits malins.....	21
IV-D - Synthèse.....	24
V - Chapitre 6.....	25
V-A - On adapte la calculatrice de la correction pour petits malins du chapitre 5.....	25
V-A-1 - Récupération des anciens codes.....	25
V-A-2 - Boucle For each.....	28
V-A-3 - La méthode getComponents() de java.awt.Container.....	29
V-A-4 - Le code final de Calculatrice avant les corrections.....	29
V-B - Division par 0.....	31
V-C - Exercice pour les petits malins.....	34
V-C-1 - Code complet.....	35
V-D - Synthèse.....	37
VI - Chapitre 7.....	38
VI-A - Exercice 1 : nombre de victoires/défaites.....	38
VI-A-1 - Analyses et stratégies.....	38
VI-A-2 - Code complet.....	39
VI-B - Exercice 2 : résolution d'un bogue.....	44
VI-B-1 - Quelques astuces.....	44
VI-B-2 - Méthode actionPerformed corrigée.....	45
VI-B-3 - Code complet.....	45
VI-C - Exercice 3 : ajout d'une méthode main().....	50
VI-C-1 - Code de la méthode construireContainer().....	51
VI-C-2 - Nouvelle version de la méthode init().....	51
VI-C-3 - Code de la méthode main().....	52
VI-C-4 - Code complet.....	53
VI-D - Exercice pour les petits malins.....	59
VI-D-1 - Les tableaux multidimensionnels.....	59
VI-D-1-a - Théorie.....	59
VI-D-1-b - Mise en pratique.....	61
VI-D-2 - Code résultat.....	61
VI-E - Synthèse.....	67
VII - Chapitre 8.....	68

VII-A - Application commande de vélos.....	68
VII-A-1 - Nouvelle version de TropDeVeloException.....	68
VII-A-2 - Correction d'une légère anomalie dans le code original de EcranCommande.....	68
VII-A-3 - EcranCommande sera une fenêtre Swing.....	69
VII-A-4 - Déclaration et construction des composants de la classe EcranCommande.....	70
VII-A-5 - Vérification de la commande et la nouvelle classe ModeleVelo.....	70
VII-A-6 - Code complet de la classe EcranCommande.....	71
VII-B - Exercices pour les petits malins : liste déroulante.....	74
VII-B-1 - Construire l'objet JComboBox.....	74
VII-B-2 - La méthode toString() de tout objet et nouvelle version de la classe ModeleVelo.....	74
VII-B-3 - L'interface ItemListener et la nouvelle version de EcranCommande.....	75
VII-C - Synthèse.....	78
VIII - Chapitre 9.....	79
VIII-A - Exercice 1 : Copier un fichier (mode console).....	79
VIII-B - Exercice pour les petits malins : copier un fichier (mode graphique).....	80
VIII-B-1 - Utilisation du composant Swing JFileChooser.....	80
VIII-B-2 - L'utilisation de classe locale et des constantes.....	81
VIII-B-3 - Dernières considérations.....	82
VIII-B-4 - Le code complet.....	82
VIII-C - Synthèse.....	85
IX - Chapitre 10.....	86
IX-A - Exercice 1 : nouveau constructeur pour la classe Poisson.....	86
IX-B - Exercice 2 : nouveau constructeur pour la classe Score.....	88
IX-C - Exercice pour les petits malins : DemoVector.java.....	90
IX-D - Synthèse.....	91
X - Chapitre 11.....	92
X-A - Exercice 1 : affecter les coordonnées du point.....	92
X-A-1 - Le support de travail.....	92
X-A-2 - Une correction possible.....	94
X-B - Exercice 2 : résolution du bogue.....	97
X-B-1 - Support de travail.....	97
X-B-2 - Correction possible.....	102
X-C - Exercice pour les petits malins 1 : vitesse de la raquette.....	105
X-D - Exercice pour les petits malins 2 : direction du rebond de la balle.....	112
X-E - Synthèse.....	117
XI - Remerciements.....	118

I - Chapitre 2

I-A - Afficher son adresse (en modifiant la classe BonjourMonde)

Ici, il suffit simplement d'effacer le contenu de la méthode main et d'appeler plusieurs fois :

Affichage de texte dans la console

```
System.out.println("Message");
```

On remplace bien sûr "Message" par le contenu voulu pour chaque ligne.

Pour ma part, je suis parti d'une nouvelle classe, tout en ayant coché la checkbox permettant d'ajouter une méthode main.

Bien sûr, il ne faut surtout pas oublier :

- ni les guillemets entourant le message ;
- ni le point-virgule à la fin de chaque ligne.

Voici un exemple possible :

Bonjour Monde (avec adresse)

```
public class BonjourMonde_Adresse {  
  
    public static void main(String[] args) {  
        System.out.println("Mr Dupond Martin");  
        System.out.println("56, rue de la fantaisie");  
        System.out.println("99999 VilleImaginaire");  
    }  
}
```

I-B - Afficher "Bonjour" avec des étoiles

Là non plus, rien de très compliqué. Voici comment je m'y suis pris :

Bonjour (avec des étoiles)

```
public class BonjourMonde_LogoEtoile {

    public static void main(String[] args) {
        System.out.println("*****      ***      ***      ***      ***      *      *      ***** ");
        System.out.println();
        System.out.println(" *      *      *      *      *      *      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" *      *      *      *      **      *      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" *      *      *      *      **      *      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" ****      *      *      *      *      *      *      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" *      *      *      *      *      **      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" *      *      *      *      *      **      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" *      *      *      *      *      *      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println(" *      *      *      *      *      *      *      *      *      *      *      *      *      *      * ");
        System.out.println();
        System.out.println("*****      ***      ***      **      ***      ***      ***      ***      *      *      *      *      *      *      * ");
    }
}
```

```
Bonjour (avec des étoiles)
```

```
}
```

```
}
```

Vous aurez peut être noté, au passage, l'appel à `System.out.println()` sans paramètre : on n'a même pas besoin de passer une chaîne vide pour passer la ligne.

II - Chapitre 3

II-A - Création de la classe Voiture

Ceux qui n'en peuvent plus d'attendre peuvent directement atteindre la section "Code complet de la classe Voiture" et éventuellement, revenir sur les explications suivantes plus tard.

II-A-1 - Création de la classe sans les trois méthodes demandées

On commence par créer une nouvelle classe Voiture depuis Eclipse. Remarquez que l'ajout d'une **méthode** main() n'est pas requis puisque ce n'est pas elle qui lance l'application. C'est le rôle de la classe ProprietaireVoiture, que l'on va réaliser en deuxième question. On ne coche donc pas la case "public static void main(String [] args)".

On obtient donc :

Classe Voiture vide

```
public class Voiture {  
  
    // Methodes et variables d'instance vont ici.  
  
}
```

II-A-2 - Petite parenthèse : les commentaires

La ligne " //Methodes et variables d'instance vont ici " est un commentaire. Mais pas de panique : les commentaires sont introduits dans un chapitre ultérieur du livre. Les commentaires sont libres :

- d'être insérés où on le désire ;
- de contenir ce que l'on veut : du français, de l'anglais ☐ même du texte insignifiant.

Pourquoi ? Tout simplement parce que le compilateur les ignorera.

Vous vous demandez alors à quoi cela peut-il bien servir : tout simplement, à apporter des précisions sur le code sans pour autant que la compilation de celui-ci soit entravée.

Ne vous inquiétez pas pour l'instant de savoir comment composer un commentaire, un chapitre ultérieur du livre vous le précisera.

II-A-3 - Ajout des méthodes demarrer() et arreter() dans la classe

Le plus difficile, consiste à passer de la **signature** des trois méthodes (fonctions de la classe) à leur codage complet : du moins à leur structure prêts à accueillir du code.

Étant donné une **signature** telle que :

Simple signature de méthode (fonction)

```
public void demarrer()
```

Il nous suffit alors de :

1) Recopier la **signature** telle quelle entre les accolades destinées à recevoir le code de la classe :

Ajout de la signature

```
public class Voiture {
    // On ajoute la methode ici-même
    public void demarrer()
}
```

2) Ajouter alors le **corps** de la méthode : c'est-à-dire l'endroit où sera codé l'ensemble des instructions de la méthode. On ajoute donc une paire d'accolades à la suite de la signature de la méthode :

Ajout des accolades

```
public class Voiture {
    public void demarrer() { // c'est la paire d'accolades qui definit le corps de la methode
    }
}
```

3) Enfin, on peut coder les **instructions** de la méthode :

Ajout des instructions

```
public class Voiture {
    public void demarrer() {
        // Ici les instructions que la methode demarrer() doit executer
        System.out.println("Je démarre !!!");
    }
}
```

Vous remarquerez également que je n'ai pas mis d'accents ni dans les commentaires, ni dans le code : c'est un choix délibéré. En effet, en passant d'un éditeur à un autre (de Eclipse à Netbeans par exemple), ou d'un système d'exploitation à un autre (de Windows à Ubuntu Linux par exemple), il se peut que l'on se retrouve confronté au problème de conversion des caractères accentués. En fait, tout dépend de "l'encodage de caractères" choisi par les systèmes ou les éditeurs.

Je ne souhaite pas rentrer dans les détails, car je serais alors vraiment hors sujet, mais sachez tout de même que ne pas mettre d'accents dans les codes, est la meilleure manière de ne pas être ennuyé.

On peut coder la méthode arreter() de la même manière :

Méthode arreter

```
public class Voiture {
    public void demarrer() {
        System.out.println("Je démarre !!!") ;
    }
    public void arreter() {
        System.out.println("Je m'arrete !!!") ;
    }
}
```

II-A-4 - Ajout de la méthode rouler() dans la classe

La signature de la méthode rouler() rend la chose un peu plus difficile qu'elle n'était pour les méthodes demarrer() et arreter() :

*) elle retourne un **int** au lieu de renvoyer simplement **void** . Par conséquent, la dernière instruction de la méthode rouler sera :

Retour de méthode

```
return valeurDirecte_Ou_VariableDeTypeInt;
```

*) elle attend obligatoirement un paramètre `duree` de type `int`. On se dit aussitôt qu'il faudra éventuellement, si le besoin se fait ressentir, se reposer sur cette valeur pour adapter le traitement de la méthode `rouler`. Alors doit-on, ne doit-on pas ? Eh bien oui, on doit, car l'énoncé nous précise que c'est cette durée - passée par le code qui utilise la voiture - qui permettra et de calculer et d'afficher la distance parcourue :

formule de la distance parcourue en fonction de la duree

```
distance = duree * 60
```

On peut après se poser la question des unités à employer : distance en km ? Durée en h ? Pour ma part, l'énoncé ne demandant pas de traiter les unités et en plus comme cela me compliquerait plus les choses qu'elles ne les arrangeraient, j'ai décidé de ne pas en utiliser.

Que peut-on conclure des deux points précédents ? Tout simplement qu'il suffira de :

- calculer la distance en fonction de la durée passée ;
- de simuler un traitement avec un affichage écran (`System.out.println()`) ;
- de retourner la distance calculée .

Voici donc un exemple de codage pour la méthode `rouler()` :

Méthode `rouler()`

```
public int rouler(int duree){
    int distance = duree * 60;
    System.out.println("Je roule sur une distance de "+distance+" .");
    return distance;
}
```

II-A-5 - Code complet de la classe Voiture



Code complet de la classe Voiture

```
public class Voiture {

    public void demarrer(){
        System.out.println("Je démarre !!!");
    }

    public void arreter(){
        System.out.println("Je m'arrete !!!");
    }

    public int rouler(int duree){
        int distance = duree * 60;
        System.out.println("Je roule sur une distance de "+distance+" .");
        return distance;
    }

}
```

II-B - Création de la classe ProprietaireVoiture

Dans cette classe, on se contentera de créer une méthode `main()`, afin de tester la classe Voiture.

Il faudra donc **l'instancier**, de manière à disposer d'une variable **de type Voiture**, que l'on testera.

En outre, je crée une variable `distanceTotale`, de type `int`, qui servira -comme son nom l'indique- à garder une trace de la distance globale parcourue : ainsi je pourrais l'afficher avant de quitter l'application. Pour ce faire, il suffit juste d'exploiter les différents retours aux appels à la méthode `rouler()` de la Voiture pour mettre à jour la distance globale :



Propriétaire Voiture

```
public class ProprietaireVoiture {

    public static void main(String[] args) {
        // Cette variable gardera une trace de la distance totale parcourue.
        int distanceTotale = 0;

        // C'est ici que j'instancie la variable de type Voiture, qui sera testee.
        Voiture maVoiture = new Voiture();

        // Les appels des methodes sur la variable maVoiture.
        maVoiture.demarrer();
        distanceTotale = distanceTotale + maVoiture.rouler(10);
        distanceTotale = distanceTotale + maVoiture.rouler(30);
        maVoiture.arreter();

        // Affichage ecran final (compte-rendu)
        System.out.println("Distance totale parcourue : "+distanceTotale + " .");
    }

}
```

II-C - Création de la classe VoitureJamesBond

La classe VoitureJamesBond doit sous-classer la classe Voiture : donc il faudra utiliser le mot-clé "extends".

VoitureJmaesBond hérite de Voiture

```
public class VoitureJamesBond extends Voiture {

    // Inutile de recoder les methodes de la classe Voiture.
}
```

Pour rappel, si on laissait le code tel quel, on pourrait se contenter de remplacer toutes les créations et exploitations d'instances de Voiture par des instances de VoitureJamesBond sans que le résultat de l'application en soit changé.

Dans la classe ProprietaireVoiture, rien ne nous empêche alors d'écrire :

ProprietaireVoitures utilise VoitureJamesBond

```
public class ProprietaireVoiture {
    public static void main(String[] args) {
        int distanceTotale = 0;
        // La seule ligne qui modifiee
        VoitureJamesBond maVoiture = new VoitureJamesBond();
        maVoiture.demarrer();
        distanceTotale = distanceTotale + maVoiture.rouler(10);
        distanceTotale = distanceTotale + maVoiture.rouler(30);
        maVoiture.arreter();
        System.out.println("Distance totale parcourue : "+distanceTotale + " .");
    }
}
```

Rien n'aurait changé pour autant au niveau du résultat.

Mais on souhaite dans l'énoncé que la classe VoitureJamesBond permette de rouler beaucoup plus vite qu'une simple instance de Voiture : on nous demande donc de surcharger la méthode rouler(), pour ne plus simplement dépendre du code défini pour rouler() dans la classe Voiture.

On procède donc comme on l'a fait pour la méthode rouler() de Voiture, mais en changeant la formule (j'ai aussi changé le texte pour être plus fantaisiste) :



VoitureJamesBond

```
public class VoitureJamesBond extends Voiture {  
  
    public int rouler(int duree) {  
        int distance = duree * 180;  
        System.out.println(":=D YAOUHHHHH !!! Je FONCE sur une distance de "+distance+" .");  
        return distance;  
    }  
  
}
```

Maintenant, si l'on relance la méthode main() de la classe ProprietaireVoiture en changeant la création d'une instance de Voiture par une création d'instance de VoitureJamesBond, les distances sont rallongées et le chauffeur semble plus réjoui.

II-D - Synthèse

Ainsi dans ce corrigé nous avons appris à :

- créer une classe Voiture personnelle. Mais aussi à créer nos propres méthodes dans cette même classe ;
- utiliser cette classe dans une méthode main() ;
- à construire une **classe fille** (VoitureJamesBond) de la classe Voiture ;
- à personnaliser l'une des méthodes de la **classe mère** (Voiture) dans la **classe fille** . La méthode rouler() a été **surchargée** dans la classe VoitureJamesBond.

III - Chapitre 4

III-A - La classe ConvertisseurTemperatures

J'ai ici décidé cette fois-ci d'inclure la méthode `main()` dans la classe `ConvertisseurTemperatures` elle-même.

Sur Internet on trouve facilement les formules de conversion. Soient T_c la température en degrés celcius et T_f la température en degrés Fahrenheit.

Formule de conversion d'une température Celcius à partir d'une température Fahrenheit

$$T_c = (5/9) * (T_f - 32)$$

Formule de conversion d'une température Fahrenheit à partir d'une température Celcius

$$T_f = (9/5) * T_c + 32$$

Il convient de traiter les différents cas possibles pour la méthode `convertirTemperature` :

- l'utilisateur a employé le **format 'C'** ;
- l'utilisateur a employé le **format 'F'** ;
- l'utilisateur n'a pas du tout employé un **format** attendu.

J'ai alors pensé à l'utilisation d'une structure **switch**. Sachez que le cas **default** correspond à toute valeur du switch que nous n'avons pas choisi de traiter.

Par conséquent, on peut ici s'en servir pour retourner un message d'erreur (la fonction doit retourner un **String**, profitons-en).

Par contre, ne pas oublier d'utiliser le mot-clé **break** pour chaque cas, hormis le cas **default** étant donné que c'est le dernier à traiter.

III-A-1 - La division entière et la division réelle

Certains d'entre vous ont certainement entendu parler (ou appris) de la division euclidienne. Ce n'est pas compliqué.

Si vous calculez $6 / 3$: pas de problème, vous obtiendrez un nombre entier. Et ce, que cela soit par vous-même ou à l'aide d'un programme Java. En revanche, une opération telle que $9 / 4$ dépendra énormément de la manière dont vous la coderez.

Pourquoi ? Parce que l'ordinateur peut aboutir sur deux valeurs :

- ou bien 2,25 : qui est le résultat auquel, à priori, tout le monde s'attend ;
- ou bien 2 : qui est le quotient (ou simple valeur entière) de la division.

Quand aura-t-on 2,25 et quand aura-t-on 2 ?

- si vous calculez la division avec deux int, vous aurez forcément un int en retour. Donc vous obtiendrez la valeur 2 en calculant $9 / 4$;
- mais si vous calculez la division avec un float et un int, ou avec deux float, alors vous aurez un float en retour. Donc ici : 2,25.

Eh bien, sachez-le, en codant `9f` et non simplement `9`, je déclare : "Je veux utiliser la valeur flottante de 9 et non pas simplement la valeur entière de 9".

Ainsi :

- en utilisant les formules de conversion, le résultat sera beaucoup plus précis ;
- en calculant $5f/9f$ je n'aurais pas la valeur 0 (qui faussera donc toute la formule de conversion en degrés celcius), mais une valeur comprise entre 0 et 1. Avec précision en plus.

Et ça, c'était vraiment le piège à éviter.

III-A-2 - Variable locale et initialisation

Enfin une dernière difficulté technique : toute variable **déclarée** (c'est-à-dire créée) à l'intérieur d'une méthode doit être initialisée (c'est-à-dire qu'on lui donne sa première valeur) avant d'être utilisée.

En d'autres termes, toute variable locale doit être initialisée avant toute utilisation.

Dans le code suivant, la variable **locale** valeurDeRetour n'a pas été initialisée à sa création. Quant à son utilisation, il n'y en a qu'une seule : lors du retour de la méthode convertirTemperature (return valeurDeRetour).

En revanche, dans le switch, tous les cas possibles aboutissent à une initialisation.

C'est-à-dire que pour toutes les valeurs que peut prendre la variable (de type char) convertirEn, on aboutit à une initialisation de la variable valeurDeRetour. Des valeurs à tester, il y en a trop : car un char peut avoir 256 valeurs. D'où l'utilité du mot-clé **default** dans le switch.

III-A-3 - Code



ConvertisseurTempatures

```
public class ConvertisseurTempatures {

    public static void main(String[] args) {
        ConvertisseurTempatures convertisseur = new ConvertisseurTempatures();
        /*
         * Ici j'effectue une concatenation de chaine :
         * Une chaine dont on connaît bien le contenu et le resultat de la conversion de température
         */
        System.out.println("27 degrees celcius font "+convertisseur.convertirTemperature(27, 'F'));
        System.out.println("80 degrees fahrenheit font "+convertisseur.convertirTemperature(80, 'C'));
    }

    public String convertirTemperature (int temperature, char convertirEn){
        String valeurDeRetour;
        switch(convertirEn){
            case 'C':
                float temperatureEnCelcius = (5f/9f) * (temperature - 32);
                valeurDeRetour = temperatureEnCelcius+" celcius";
                // Attention à ne pas oublier le mot-cle break
                break;
            case 'F':
                float temperatureEnFahrenheit = (9f/5f) * temperature + 32;
                valeurDeRetour = temperatureEnFahrenheit+" fahrenheit";
                // Attention à ne pas oublier le mot-cle break
                break;
            default :
                valeurDeRetour = "Erreur de conversion";
        }
        return valeurDeRetour;
    }
}
```

III-B - Modification de la classe BulletinAppreciation

Pour rendre la méthode `convertirNiveaux()` **statique**, rien de plus simple. Sa nouvelle signature sera :

Nouvelle signature de la méthode `convertirNiveaux()`

```
public static char convertirNiveaux(int noteDevoir)
```

On n'a rien à changer dans son **corps**.

Et comme elle est **statique**, nul besoin de créer une **instance** de `BulletinAppreciation` pour appeler la méthode `convertirNiveaux()` :

Appel de la méthode `convertirNiveaux`

```
BulletinAppreciation.convertirNiveaux(valeurIntPassee);
```

Ce qui donne le code suivant :

BulletinAppreciation

```
public class BulletinAppreciation {
    /**
     * Cette methode attend un argument entier ? la note du devoir
     * - et retourne une mention, I, P, A, B, T ou E, en fonction
     * de sa valeur.
     */
    public static char convertirNiveaux(int noteDevoir) {
        char niveau;
        if (noteDevoir >= 18) {
            niveau = 'E';
        }
        else if (noteDevoir >= 16 && noteDevoir < 18){
            niveau = 'T';
        }
        else if (noteDevoir >= 14 && noteDevoir < 16){
            niveau = 'B';
        }
        else if (noteDevoir >= 12 && noteDevoir < 14){
            niveau = 'A';
        }
        else if (noteDevoir >= 10 && noteDevoir < 12){
            niveau = 'P';
        }
        else {
            niveau = 'I';
        }
        return niveau;
    }

    public static void main(String[] args) {
        char tonNiveau = BulletinAppreciation.convertirNiveaux(17);
        System.out.println("Ton premier niveau est " +
            tonNiveau);
        tonNiveau = BulletinAppreciation.convertirNiveaux(15);
        System.out.println("Ton second niveau est " +
            tonNiveau);
    }
}
```

III-C - Exercice pour les petits malins : félicitations aux joueurs

Pour les impatientes, aller directement à la section réponse.

III-C-1 - Le code original avec le mot-clé continue

Voici, à peu de choses près, le code utilisé dans le chapitre avec **continue**. J'ai juste englobé le code dans une classe. Sa méthode main() plus précisément. Et bien sûr, j'ai aussi initialisé les différentes variables utilisées.



FelicitationsAuxJoueurs

```
public class FelicitationsAuxJoueurs {

    public static void main(String[] args) {
        String [] joueurs = {"David", "Daniel", "Anna", "Gregory"};
        int compteur = 0;
        int nombreJoueurs = joueurs.length;

        while (compteur < nombreJoueurs) {
            compteur++;
            String leJoueur = joueurs[compteur];
            if (leJoueur.equals("David")) {
                continue;
            }
            System.out.println("Felicitations, " + leJoueur + " !");
        }
    }
}
```

On comprend aisément que tous les joueurs, à l'exception de David, soient félicités. Si vous n'en êtes pas totalement convaincus, il faut analyser le code plus en détail.

Pour ce faire, on peut "simuler" l'exécution du code par l'ordinateur. Vous pouvez imprimer le code, puis griffonner et gommer au crayon à papier les valeurs des variables. Ceci, au fur et à mesure que vous avancez dans le code. Et lors de vos différents passages dans la boucle while simulez aussi, dans une zone de votre feuille, la sortie de la console.

Si vous parvenez ainsi à deviner le résultat de la console avant même d'avoir lancé le programme : cela voudra dire que vous avez vraiment bien assimilé le concept de **boucles** et d'irrégularités (par **break** / **continue**).

III-C-2 - Le code qu'il nous est demandé d'évaluer

Code à évaluer

```
public class FelicitationsAuxJoueurs {

    public static void main(String[] args) {
        String [] joueurs = {"David", "Daniel", "Anna", "Gregory"};
        int compteur = 0;
        int nombreJoueurs = joueurs.length;

        while (compteur < nombreJoueurs) {
            String leJoueur = joueurs[compteur];
            if (leJoueur.equals("David")) {
                continue;
            }
            System.out.println("Felicitations, " + leJoueur + " !");
            // On a laissé compteur++ a la fin de la boucle while
            compteur++;
        }
    }
}
```

On nous demande alors ce qu'il peut bien se passer.

Vous n'avez pas encore testé ce code à l'ordinateur et vous avez été étonné parce que le code bloquait ? Bravo. Si tel est le cas, vous avez vraiment compris comment fonctionnent les boucles.

En effet, ce code ne s'arrêtera jamais de lui-même. Pourquoi ? Parce que la variable compteur vaudra toujours 0 et par conséquent, le test de la boucle **while** donnera toujours le résultat de $0 < 4$: soit **true**.

Démonstration par la simulation :

Au 1er passage de la boucle while

```
public class FelicitationsAuxJoueurs {

    public static void main(String[] args) {
        String [] joueurs = {"David", "Daniel", "Anna", "Gregory"};
        int compteur = 0;
        int nombreJoueurs = joueurs.length; // soit 4

        while (compteur /* 0 */< nombreJoueurs /* 4 */) { // true
            String leJoueur = joueurs[compteur]; //David
            if (leJoueur.equals("David")) { // true
                continue; // Donc, on revient au test de la boucle while
            }
            System.out.println("Félicitations, " + leJoueur + " !"); // Non execute
            compteur++; // Non execute
        }
    }
}
```

Au 2e passage de la boucle while

```
public class FelicitationsAuxJoueurs {

    public static void main(String[] args) {
        String [] joueurs = {"David", "Daniel", "Anna", "Gregory"};
        int compteur = 0;
        int nombreJoueurs = joueurs.length; // soit 4

        while (compteur /* 0 */< nombreJoueurs /* 4 */) { // true
            String leJoueur = joueurs[compteur]; //David
            if (leJoueur.equals("David")) { // true
                continue; // Donc, on revient au test de la boucle while
            }
            System.out.println("Félicitations, " + leJoueur + " !"); // Non execute
            compteur++; // Non execute
        }
    }
}
```

Au 3e passage de la boucle while

```
public class FelicitationsAuxJoueurs {

    public static void main(String[] args) {
        String [] joueurs = {"David", "Daniel", "Anna", "Gregory"};
        int compteur = 0;
        int nombreJoueurs = joueurs.length; // soit 4

        while (compteur /* 0 */< nombreJoueurs /* 4 */) { // true
            String leJoueur = joueurs[compteur]; //David
            if (leJoueur.equals("David")) { // true
                continue; // Donc, on revient au test de la boucle while
            }
            System.out.println("Félicitations, " + leJoueur + " !"); // Non execute
            compteur++; // Non execute
        }
    }
}
```

Ainsi de suite ...

III-D - Synthèse

Nous avons donc vu :

- comment traiter, dans un **switch**, tous les cas non pris en compte grâce au mot-clé **default** ;
- comment utiliser une méthode **statique** sans initialiser la classe où elle se situe ;
- quel est le danger d'une boucle **while** mal utilisée : on peut bloquer son application.

IV - Chapitre 5

IV-A - Ajout d'un panneau opérations au code Calculatrice.java

Rien de très compliqué.

Pour la disposition du nouveau panneau, j'ai utilisé un GridLayout avec quatre lignes et une colonne. On peut utiliser d'autres layouts à la place, mais pour moi le GridLayout est le plus simple à utiliser dans ce cas de figure.

Voici le code modifié :

Calculatrice

```
import java.awt.BorderLayout;
import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Calculatrice {
    // Declaration de tous les composants de la calculatrice.
    JPanel contenuFenetre;
    JTextField champAffichage;
    JButton bouton0;
    JButton bouton1;
    JButton bouton2;
    JButton bouton3;
    JButton bouton4;
    JButton bouton5;
    JButton bouton6;
    JButton bouton7;
    JButton bouton8;
    JButton bouton9;
    JButton boutonVirgule;
    JButton boutonEgale;
    JPanel panneauChiffres;

    JButton boutonPlus;
    JButton boutonMoins;
    JButton boutonMultiplie;
    JButton boutonDivise;
    JPanel panneauOperations;

    // Le constructeur cree les composants en mémoire
    //et les ajoute au cadre en utilisant une combinaison
    //de BorderLayout et GridLayout
    Calculatrice() {
        contenuFenetre = new JPanel();
        //Affecte un gestionnaire de presentation à ce panneau
        BorderLayout disposition1 = new BorderLayout();
        contenuFenetre.setLayout(disposition1);
        //Cree le champ d'affichage et le positionne dans
        //la zone nord de la fenetre
        champAffichage = new JTextField(30);
        contenuFenetre.add("North", champAffichage);
        //Cree les boutons en utilisant le constructeur de
        //la classe JButton qui prend en parametre le libelle
        //du bouton
        bouton0 = new JButton("0");
        bouton1 = new JButton("1");
        bouton2 = new JButton("2");
        bouton3 = new JButton("3");
        bouton4 = new JButton("4");
        bouton5 = new JButton("5");
```

Calculatrice

```

bouton6 = new JButton("6");
bouton7 = new JButton("7");
bouton8 = new JButton("8");
bouton9 = new JButton("9");
boutonVirgule = new JButton(",");
boutonEgale = new JButton("=");

//Cree le panneau avec le quadrillage qui contient
//les 12 boutons ? les 10 boutons numeriques et ceux
//representant la virgule et le signe egale
panneauChiffres = new JPanel();
GridLayout disposition2 = new GridLayout(4, 3);
panneauChiffres.setLayout(disposition2);
//Ajoute les controles au panneau panneauChiffres
panneauChiffres.add(bouton1);
panneauChiffres.add(bouton2);
panneauChiffres.add(bouton3);
panneauChiffres.add(bouton4);
panneauChiffres.add(bouton5);
panneauChiffres.add(bouton6);
panneauChiffres.add(bouton7);
panneauChiffres.add(bouton8);
panneauChiffres.add(bouton9);
panneauChiffres.add(bouton0);
panneauChiffres.add(boutonVirgule);
panneauChiffres.add(boutonEgale);
//Ajoute panneauChiffres a la zone centrale de la
//fenetre
contenuFenetre.add("Center", panneauChiffres);

// Cree le panneau operations
// Il contiendra les operateurs +, -, *, /
panneauOperations = new JPanel();
// Une grille de 4 lignes * 1 colonne
// suffit pour placer les 4 boutons
// Mais rien ne nous aurais empeche
// de prendre un autre Layout adapté.
// (GridBagLayout par exemple)
GridLayout disposition3 = new GridLayout(4,1);
panneauOperations.setLayout(disposition3);
// Creation des boutons operations
boutonPlus = new JButton("+");
boutonMoins = new JButton("-");
boutonMultiplie = new JButton("*");
boutonDivise = new JButton("/");
// Ajout des boutons operations
panneauOperations.add(boutonPlus);
panneauOperations.add(boutonMoins);
panneauOperations.add(boutonMultiplie);
panneauOperations.add(boutonDivise);
// Ajout du panneau a la zone est
// du contenu de la fenetre
contenuFenetre.add(panneauOperations, BorderLayout.EAST);

//Cree le cadre et lui affecte son contenu
JFrame frame = new JFrame("Calculatrice");
frame.setContentPane(contenuFenetre);
//Affecte a la fenetre des dimensions suffisantes pour
//prendre en compte tous les controles
frame.pack();
//Enfin, affiche la fenetre
frame.setVisible(true);
}

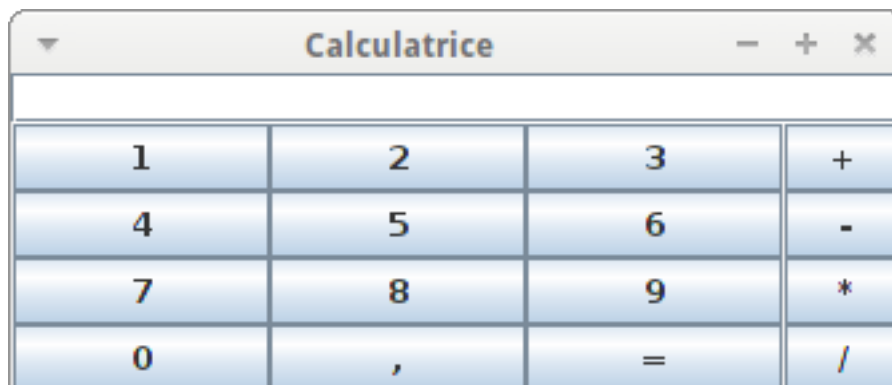
public static void main(String[] args) {
    Calculatrice calc = new Calculatrice();
}

```

Calculatrice

```
}
```

Et voici un aperçu de ce que l'on obtient (suivant votre système d'exploitation, l'apparence sera légèrement différente. L'application tente de réutiliser au maximum l'apparence classique du système) :



Aperçu de la nouvelle version de la calculatrice

IV-B - Alignement de l'écran de la calculatrice à droite

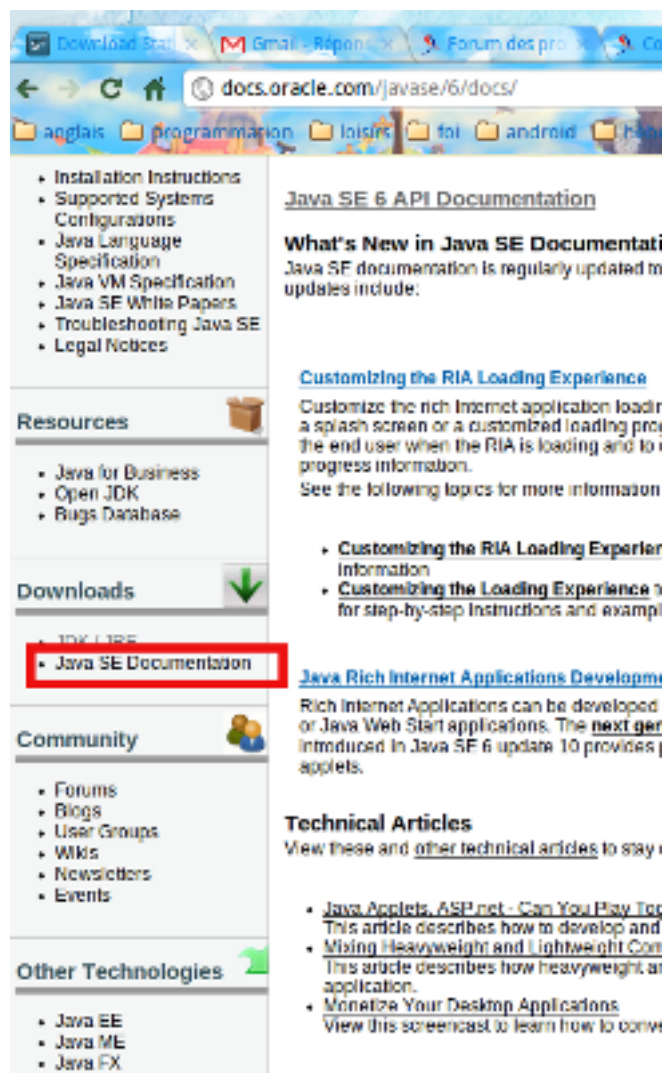
Le premier réflexe que vous devez avoir lorsque vous ne savez pas (ou doutez) comment utiliser un composant, une classe standard de Java : **consulter la javadoc**. C'est donc le cas pour les composants SWING/AWT.

En effet, dans le chapitre 4, vous avez appris que l'outil javadoc vous permet de réaliser vos propres documentations pour vos classes, en respectant un certain standard. Eh bien l'ensemble des classes déjà intégrées à Java a aussi sa documentation Javadoc. Elle est consultable en ligne, sur le Web, mais vous pouvez aussi la télécharger.

La première chose à vérifier, c'est que la documentation que vous consultez correspond à la version de votre JDK : si c'est le JDK 1.5, il vous faudra la javadoc 5. Pour le JDK 1.6 il vous faudra la javadoc 6. Ainsi de suite.

Si vous voulez juste consulter en ligne la javadoc 5 : tapez **java 5 api**, depuis le moteur de recherche Google et accédez au lien qui vous est proposé.

Par contre, pour télécharger la javadoc 5 : il suffit de taper **download javadoc 5**. Ensuite cliquez sur le lien, zone de gauche, **download java se documentation**.



Lien pour télécharger la javadoc

Voici donc comment utiliser la javadoc :



La javadoc

La zone en haut à gauche répertorie les différents packages, la zone en bas à gauche répertorie les classes du package sélectionné dans la zone en haut à gauche et la zone centrale affiche les informations de la classe choisie.

Cliquez sur **All Classes** dans la zone de packages (pour disposer de l'ensemble des classes, au cas où vous auriez déjà cliqué sur autre chose). Puis cliquez sur JTextField dans la zone des classes.

Un résumé du fonctionnement d'un JTextField est alors donné, avant les spécifications techniques en elles-mêmes (Constructeurs, Méthodes voire variables d'instances ou statiques.). Ici ce qui nous intéresse, c'est une méthode pour aligner le contenu à droite.

La méthode setHorizontalAlignement() est ce qu'il nous faut : cliquez sur le lien dans la liste des méthodes. On nous dit qu'il suffit de l'appeler en passant une valeur telle que JTextField.LEFT, JTextField.CENTER, JTextField.RIGHT, □

Il suffit donc tout simplement d'appeler :

Alignement horizontal du texte

```
champAffichage.setHorizontalAlignement(JTextField.RIGHT);
```

lors de la création du composant JTextField champAffichage. Comme vous pouvez le remarquer, la valeur RIGHT est une valeur **statique** de la classe JTextField.

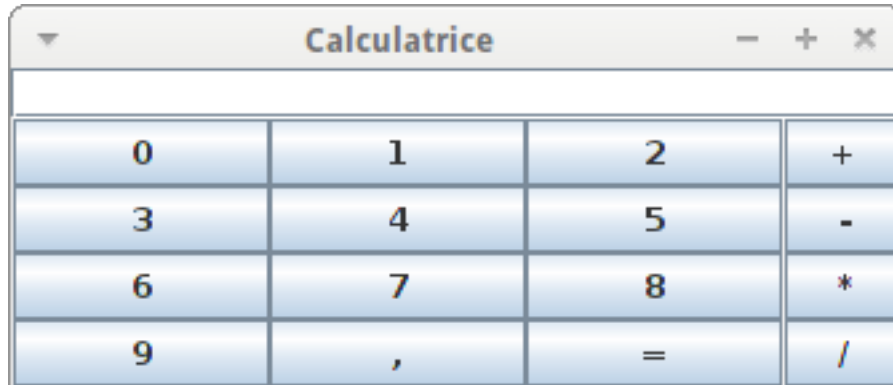
Placez ce code entre la déclaration de champAffichage et son ajout au JPanel contenuFenetre.

IV-C - Exercice pour les petits malins

Là non plus, pas grand chose de compliqué.

Vous remarquerez tout de même que pour l'ajout des boutons au panneau de la fenêtre, je procède différemment. Je commence par ajouter les boutons 1 à 9 à l'aide d'une boucle for, puis j'ajoute le bouton 0. Ceci afin de rester cohérent avec ce qui était déjà réalisé auparavant.

Si vous ajoutez tous les boutons avec juste une seule boucle de 0 à 9, vous aurez quelque chose de similaire à :



Les boutons sont mal organisés

Donc le bouton 0 est mal placé.

Voici donc le code de la nouvelle version (la classe s'appelle Calculatrice_2 au lieu de Calculatrice) :

Calculatrice_2

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.text.DecimalFormat;

import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.text.NumberFormatter;

public class Calculatrice_2 {
    // Declaration de tous les composants de la calculatrice.
    JPanel contenuFenetre;
    JTextField champAffichage;
    JButton [] boutonsChiffres = new JButton[10];
    JButton boutonVirgule;
    JButton boutonEgale;
    JPanel panneauChiffres;

    JButton boutonPlus;
    JButton boutonMoins;
    JButton boutonMultiplie;
    JButton boutonDivise;
    JPanel panneauOperations;

    // Le constructeur cree les composants en mémoire
    //et les ajoute au cadre en utilisant une combinaison
    //de BorderLayout et GridLayout
    Calculatrice_2() {

        contenuFenetre = new JPanel();
        //Affecte un gestionnaire de presentation à ce panneau
        BorderLayout disposition1 = new BorderLayout();
        contenuFenetre.setLayout(disposition1);
```

Calculatrice_2

```
// Cree le champ d'affichage et le positionne dans
// la zone nord de la fenetre
champAffichage = new JTextField();
champAffichage.setColumns(30);
champAffichage.setHorizontalAlignment(SwingConstants.RIGHT);

contenuFenetre.add("North", champAffichage);
// Cree les boutons en utilisant le constructeur de
// la classe JButton qui prend en parametre le libelle
// du bouton
for (int numeroBouton = 0; numeroBouton < 10; numeroBouton++){
    boutonsChiffres[numeroBouton] = new JButton(Integer.toString(numeroBouton));
}
boutonVirgule = new JButton(",");
boutonEgale = new JButton("=");

// Cree le panneau avec le quadrillage qui contient
// les 12 boutons ? les 10 boutons numeriques et ceux
// representant la virgule et le signe egal
panneauChiffres = new JPanel();
GridLayout disposition2 = new GridLayout(4, 3);
panneauChiffres.setLayout(disposition2);
// Ajoute les controles au panneau panneauChiffres
for (int numeroBouton = 1; numeroBouton < 10; numeroBouton++){
    panneauChiffres.add(boutonsChiffres[numeroBouton]);
}
panneauChiffres.add(boutonsChiffres[0]);
panneauChiffres.add(boutonVirgule);
panneauChiffres.add(boutonEgale);
// Ajoute panneauChiffres a la zone centrale de la
// fenetre
contenuFenetre.add("Center", panneauChiffres);

// Cree le panneau operations
// Il contiendra les operateurs +, -, *, /
panneauOperations = new JPanel();
// Une grille de 4 lignes * 1 colonne
// suffit pour placer les 4 boutons
// Mais rien ne nous aurais empeche
// de prendre un autre Layout adapté.
// (GridBagLayout par exemple)
GridLayout disposition3 = new GridLayout(4, 1);
panneauOperations.setLayout(disposition3);
// Creation des boutons operations
boutonPlus = new JButton("+");
boutonMoins = new JButton("-");
boutonMultiplie = new JButton("*");
boutonDivise = new JButton("/");
// Ajout des boutons operations
panneauOperations.add(boutonPlus);
panneauOperations.add(boutonMoins);
panneauOperations.add(boutonMultiplie);
panneauOperations.add(boutonDivise);
// Ajout du panneau a la zone est
// du contenu de la fenetre
contenuFenetre.add(panneauOperations, BorderLayout.EAST);

// Cree le cadre et lui affecte son contenu
JFrame frame = new JFrame("Calculatrice");
frame.setContentPane(contenuFenetre);
// Affecte a la fenetre des dimensions suffisantes pour
// prendre en compte tous les controles
frame.pack();
// Enfin, affiche la fenetre
frame.setVisible(true);
}

public static void main(String[] args) {
    Calculatrice_2 calc = new Calculatrice_2();
}
```


Calculatrice_2

```
}  
  
}
```

IV-D - Synthèse

Ainsi vous avez su :

- créer votre propre panneau de contenus et l'intégrer à l'ensemble existant. De plus, vous êtes resté cohérents dans la disposition graphique de la calculatrice ;
- rechercher des informations sur la javadoc officielle afin d'utiliser des composants / des fonctionnalités qui n'ont pas été présentées lors du cours ;
- ajouter des composants de manière plus dynamique : vous avez remplacé la création de 10 boutons ligne par ligne, par l'utilisation d'un **tableau** et d'une **boucle for**.

V - Chapitre 6

V-A - On adapte la calculatrice de la correction pour petits malins du chapitre 5

V-A-1 - Récupération des anciens codes

La calculatrice que nous avons définie à l'aide de la correction (pour petits malins) du chapitre 5, comporte certains avantages. Notamment, au niveau de la répétition du code pour les différents boutons chiffres.

Je suis donc reparti de cette calculatrice-là (à peu de choses près : le nom de la classe MoteurCalcul est à remplacer par MoteurCalcul_avantCorrection et la classe utilisée n'est plus simplement Calculatrice mais Calculatrice_avantCorrection) : mais sachez qu'il n'est pas difficile d'adapter ce que j'ai utilisé par la suite au code original du chapitre 6.

En revanche, le moteur de calcul reste le même, à quelques noms de variables près. Ce sont les noms des boutons opérateurs de la variable parent qui sont à adapter.

Pour rappel, le code calculatrice que nous avons codé à la correction pour petits malins du chapitre 5 :

Ancien code calculatrice

```
import java.awt.BorderLayout;
import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class Calculatrice_2 {
    // Declaration de tous les composants de la calculatrice.
    JPanel contenuFenetre;
    JTextField champAffichage;
    JButton [] boutonsChiffres = new JButton[10];
    JButton boutonVirgule;
    JButton boutonEgale;
    JPanel panneauChiffres;

    JButton boutonPlus;
    JButton boutonMoins;
    JButton boutonMultiplie;
    JButton boutonDivise;
    JPanel panneauOperations;

    // Le constructeur cree les composants en mémoire
    //et les ajoute au cadre en utilisant une combinaison
    //de BorderLayout et GridLayout
    Calculatrice_2() {

        contenuFenetre = new JPanel();
        //Affecte un gestionnaire de presentation à ce panneau
        BorderLayout disposition1 = new BorderLayout();
        contenuFenetre.setLayout(disposition1);
        //Cree le champ d'affichage et le positionne dans
        //la zone nord de la fenetre
        champAffichage = new JTextField();
        champAffichage.setColumns(30);
        champAffichage.setHorizontalAlignment(SwingConstants.RIGHT);
```

Ancien code calculatrice

```

    contenuFenetre.add("North", champAffichage);
    //Cree les boutons en utilisant le constructeur de
    //la classe JButton qui prend en parametre le libelle
    //du bouton
    for (int numeroBouton = 0; numeroBouton <= 9; numeroBouton++){
        boutonsChiffres[numeroBouton] = new JButton(Integer.toString(numeroBouton));
    }
    boutonVirgule = new JButton(",");
    boutonEgale = new JButton("=");

    //Cree le panneau avec le quadrillage qui contient
    //les 12 boutons : les 10 boutons numeriques et ceux
    //representant la virgule et le signe egale
    panneauChiffres = new JPanel();
    GridLayout disposition2 = new GridLayout(4, 3);
    panneauChiffres.setLayout(disposition2);
    //Ajoute les controles au panneau panneauChiffres
    for (int numeroBouton = 0; numeroBouton <= 9; numeroBouton++){
        panneauChiffres.add(boutonsChiffres[numeroBouton]);
    }
    //panneauChiffres.add(boutonsChiffres[0]);
    panneauChiffres.add(boutonVirgule);
    panneauChiffres.add(boutonEgale);
    //Ajoute panneauChiffres a la zone centrale de la
    //fenetre
    contenuFenetre.add("Center", panneauChiffres);

    // Cree le panneau operations
    // Il contiendra les operateurs +, -, *, /
    panneauOperations = new JPanel();
    // Une grille de 4 lignes * 1 colonne
    // suffit pour placer les 4 boutons
    // Mais rien ne nous aurais empeche
    // de prendre un autre Layout adapté.
    // (GridBagLayout par exemple)
    GridLayout disposition3 = new GridLayout(4,1);
    panneauOperations.setLayout(disposition3);
    // Creation des boutons operations
    boutonPlus = new JButton("+");
    boutonMoins = new JButton("-");
    boutonMultiplie = new JButton("*");
    boutonDivise = new JButton("/");
    // Ajout des boutons operations
    panneauOperations.add(boutonPlus);
    panneauOperations.add(boutonMoins);
    panneauOperations.add(boutonMultiplie);
    panneauOperations.add(boutonDivise);
    // Ajout du panneau a la zone est
    // du contenu de la fenetre
    contenuFenetre.add(panneauOperations, BorderLayout.EAST);

    //Cree le cadre et lui affecte son contenu
    JFrame frame = new JFrame("Calculatrice");
    frame.setContentPane(contenuFenetre);
    //Affecte a la fenetre des dimensions suffisantes pour
    //prendre en compte tous les controles
    frame.pack();
    //Enfin, affiche la fenetre
    frame.setVisible(true);
}

public static void main(String[] args) {
    Calculatrice_2 calc = new Calculatrice_2();
}

}

```

Et voici la classe MoteurCalcul adaptée à nos besoins (pour l'instant remplacez tous les appels à Calculatrice_avantCorrection par Calculatrice_2. Utilisez pour cela le menu Edit->Find/Replace d'Eclipse) :



MoteurCalcul avant la correction

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import java.text.ParsePosition;

import javax.swing.JButton;

public class MoteurCalcul_avantCorrection implements ActionListener {
    Calculatrice_avantCorrection parent; // une reference a la Calculatrice
    char actionSelectionnee = ' '; // +, -, /, ou *
    double resultatCourant = 0;
    NumberFormat formatNombres = NumberFormat.getInstance();
    // un objet capable de lire et presenter les nombres

    // Le constructeur stocke la reference a la fenetre
    // Calculatrice dans la variable membre paren
    MoteurCalcul_avantCorrection(Calculatrice_avantCorrection parent) {
        this.parent = parent;
    }

    public void actionPerformed(ActionEvent evenement) {
        // Retrouve la source de l'action
        JButton boutonClique = (JButton) evenement.getSource();
        String texteChampAffichage =
            parent.champAffichage.getText();
        double valeurAffichee = 0;
        // Retrouve le nombre presente dans le champ texte
        // s'il n'est pas vide
        if (!"".equals(texteChampAffichage)) {
            valeurAffichee =
                // analyse la chaîne de caracteres
                formatNombres.parse(
                    texteChampAffichage,
                    new ParsePosition(0) /* ne sert pas */).
                    // puis donne sa valeur en tant que double
                    doubleValue();
        }
        Object sourceEvenement = evenement.getSource();

        // Pour chaque bouton d'action, memorise l'action
        // selectionnee, +, -, /, ou *, stocke la valeur courante
        // dans la variable resultatCourant et vide le champ
        // Affichage avant l'entree du nombre suivant

        if (sourceEvenement == parent.boutonPlus) {
            actionSelectionnee = '+';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonMoins) {
            actionSelectionnee = '-';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonDivise) {
            actionSelectionnee = '/';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonMultiplie) {
            actionSelectionnee = '*';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
    }
}
```

MoteurCalcul avant la correction

```

else if (sourceEvenement == parent.boutonEgale) {
    // Effectue les calculs en fonction de actionSelectionnee
    // Modifie la valeur de la variable resultatCourant
    // et affiche le resultat
    if (actionSelectionnee == '+') {
        resultatCourant += valeurAffichee;
        // Convertit le resultat en le transformant en String
        // a l'aide de formatNombres
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
    else if (actionSelectionnee == '-') {
        resultatCourant -= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
    else if (actionSelectionnee == '/') {
        resultatCourant /= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
    else if (actionSelectionnee == '*') {
        resultatCourant *= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
}
else {
    // Pour tous les boutons numeriques, ajoute le libelle
    // du bouton au champ texte
    String libelleBoutonClique = boutonClique.getText();
    parent.champAffichage.setText(texteChampAffichage +
        libelleBoutonClique);
}
}
}

```

On ajoute donc une instance de MoteurCalcul à l'ensemble des boutons :

- du panneau panneauChiffres ;
- du panneau panneauOperations.

Pour ce faire :

- je me contente de déclarer une instance de MoteurCalcul une fois pour toutes ;
- pour chaque panneau, je parcours l'ensemble de ses **composants**. Et pour chaque composant de type JButton, j'ajoute le MoteurCalcul en tant qu'**écouteur d'action (ActionListener)**.

V-A-2 - Boucle For each

Avant de vous proposer la nouvelle version de Calculatrice, qui nous permettra de résoudre les exercices du chapitre, je voulais vous parler d'un autre type de **boucle for**. Elle est plus adaptée au cas présent.

Voici sa syntaxe :

Syntaxe for each

```

for (TypeObjet nomObjet : TableauDuTypeObjets){
    // Faire quelque chose avec nomObjet
}

```

Avantages :

- la syntaxe est plus simple ;
- on n'a pas besoin de créer une variable de type entier pour parcourir le tableau d'objets.

Inconvénient :

- on ne connaîtra pas la position de l'objet courant. Pour cela, il faudra en revenir à l'utilisation d'une **boucle for** classique.

Soyez tout de même conscient que cette fonctionnalité n'existe pas avant Java 5. Tout JDK antérieur vous renverra une erreur en employant cette syntaxe.

On appelle cette boucle, une **boucle de type for each**. Car elle définit une action pour toutes les valeurs du tableau.

V-A-3 - La méthode `getComponents()` de `java.awt.Container`

Pour connaître la liste des composants d'un panneau, j'utilise la méthode `getComponents()` de la classe `java.awt.Container`. Si vous parcourez la javadoc officielle, vous constaterez que la classe `javax.swing.JPanel` hérite notamment de la classe `java.awt.Container`.

En revanche, cette méthode renvoie un tableau de `java.awt.Component` et non de `javax.swing.JButton`. La classe `javax.swing.JButton` héritant de `java.awt.Component`, il suffira de :

- tester si le composant courant est de type `JButton` ;
- si cela est concluant, de faire un casting du composant récupéré dans le type `JButton`.

Enfin, pour que cela fonctionne, il faut effectuer ce traitement une fois les boutons ajoutés au panneau concerné, pas avant (logique).

V-A-4 - Le code final de Calculatrice avant les corrections

Voici donc la classe `Calculatrice` que nous allons utiliser (pour l'utiliser, il faut modifier la classe `MoteurCalcul_avantCorrection` pour se baser sur la classe `Calculatrice_avantCorrection` et non `Calculatrice_2`) :

Classe `Calculatrice` avant la correction

```
import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.GridLayout;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

public class Calculatrice_avantCorrection {
    // Declaration de tous les composants de la calculatrice.
    JPanel contenuFenetre;
    JTextField champAffichage;
    JButton [] boutonsChiffres = new JButton[10];
    JButton boutonVirgule;
    JButton boutonEgale;
    JPanel panneauChiffres;

    JButton boutonPlus;
    JButton boutonMoins;
    JButton boutonMultiplie;
    JButton boutonDivise;
    JPanel panneauOperations;
```

Classe Calculatrice avant la correction

```
// Le constructeur cree les composants en mémoire
//et les ajoute au cadre en utilisant une combinaison
//de BorderLayout et GridLayout
Calculatrice_avantCorrection() {
    // Declaration du moteur de calcul, pour cette calculatrice (this).
    MoteurCalcul_avantCorrection moteurCalcul = new MoteurCalcul_avantCorrection(this);

    contenuFenetre = new JPanel();
    //Affecte un gestionnaire de presentation à ce panneau
    BorderLayout disposition1 = new BorderLayout();
    contenuFenetre.setLayout(disposition1);
    //Cree le champ d'affichage et le positionne dans
    //la zone nord de la fenetre
    champAffichage = new JTextField();
    champAffichage.setColumns(30);
    champAffichage.setHorizontalAlignment(SwingConstants.RIGHT);

    contenuFenetre.add("North", champAffichage);
    //Cree les boutons en utilisant le constructeur de
    //la classe JButton qui prend en parametre le libelle
    //du bouton
    for (int numeroBouton = 0; numeroBouton < 10; numeroBouton++){
        boutonsChiffres[numeroBouton] = new JButton(Integer.toString(numeroBouton));
    }
    boutonVirgule = new JButton(",");
    boutonEgale = new JButton("=");

    //Cree le panneau avec le quadrillage qui contient
    //les 12 boutons ? les 10 boutons numeriques et ceux
    //representant la virgule et le signe egale
    panneauChiffres = new JPanel();
    GridLayout disposition2 = new GridLayout(4, 3);
    panneauChiffres.setLayout(disposition2);
    //Ajoute les controles au panneau panneauChiffres
    for (int numeroBouton = 1; numeroBouton < 10; numeroBouton++){
        panneauChiffres.add(boutonsChiffres[numeroBouton]);
    }
    panneauChiffres.add(boutonsChiffres[0]);
    panneauChiffres.add(boutonVirgule);
    panneauChiffres.add(boutonEgale);

    // Definit l'operation MoteurCalcule pour les
    // clics sur les boutons du panneauChiffres
    for (Component composantCourant : panneauChiffres.getComponents()){
        if (composantCourant instanceof JButton){
            JButton boutonCourant = (JButton) composantCourant;
            boutonCourant.addActionListener(moteurCalcul);
        }
    }

    //Ajoute panneauChiffres a la zone centrale de la
    //fenetre
    contenuFenetre.add("Center", panneauChiffres);

    // Cree le panneau operations
    // Il contiendra les operateurs +, -, *, /
    panneauOperations = new JPanel();
    // Une grille de 4 lignes * 1 colonne
    // suffit pour placer les 4 boutons
    // Mais rien ne nous aurais empeche
    // de prendre un autre Layout adapté.
    // (GridBagLayout par exemple)
    GridLayout disposition3 = new GridLayout(4,1);
    panneauOperations.setLayout(disposition3);
    // Creation des boutons operations
    boutonPlus = new JButton("+");
    boutonMoins = new JButton("-");
    boutonMultiplie = new JButton("*");
    boutonDivise = new JButton("/");
}
```

Classe Calculatrice avant la correction

```
// Ajout des boutons operations
panneauOperations.add(boutonPlus);
panneauOperations.add(boutonMoins);
panneauOperations.add(boutonMultiplie);
panneauOperations.add(boutonDivise);

// Definit l'operation MoteurCalcule pour les
// clics sur les boutons du panneauOperations
for (Component composantCourant : panneauOperations.getComponents()) {
    if (composantCourant instanceof JButton) {
        JButton boutonCourant = (JButton) composantCourant;
        boutonCourant.addActionListener(moteurCalcul);
    }
}

// Ajout du panneau a la zone est
// du contenu de la fenetre
contenuFenetre.add(panneauOperations, BorderLayout.EAST);

// Cree le cadre et lui affecte son contenu
JFrame frame = new JFrame("Calculatrice");
frame.setContentPane(contenuFenetre);
//Affecte a la fenetre des dimensions suffisantes pour
//prendre en compte tous les controles
frame.pack();
//Enfin, affiche la fenetre
frame.setVisible(true);
}

public static void main(String[] args) {
    Calculatrice_avantCorrection calc = new Calculatrice_avantCorrection();
}
}
```

En testant le code, vous pouvez vous apercevoir que le fonctionnement est bien celui qui était attendu.

V-B - Division par 0

On nous demande d'afficher un message d'erreur si l'utilisateur tente d'effectuer une division par zéro. On sait d'ores et déjà que la partie graphique est gérée dans la classe Calculatrice et la partie traitements dans la classe MoteurCalcul. C'est donc la classe MoteurCalcul que nous allons ici modifier.

Il nous faut afficher le message d'erreur juste au moment où l'utilisateur appuie sur le bouton égale.

C'est donc cette section de code qui nous intéresse :

Section de code à modifier

```
else if (sourceEvenement == parent.boutonEgale) {
    // Effectue les calculs en fonction de actionSelectionnee
    // Modifie la valeur de la variable resultatCourant
    // et affiche le resultat
    if (actionSelectionnee == '+') {
        resultatCourant += valeurAffichee;
        // Convertit le resultat en le transformant en String
        // a l'aide de formatNombres
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
    else if (actionSelectionnee == '-') {
        resultatCourant -= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
}
```

Section de code à modifier

```

    }
    else if (actionSelectionnee == '/') {
        resultatCourant /= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
    else if (actionSelectionnee == '*') {
        resultatCourant *= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }
}

```

Bien sûr, on s'occupera de la section traitant de la division :

Section traitant de la division

```

    else if (actionSelectionnee == '/') {
        resultatCourant /= valeurAffichee;
        parent.champAffichage.setText(
            formatNombres.format(resultatCourant));
    }

```

Le deuxième opérande de la division, celui qui ne doit pas valoir 0, est forcément la valeur couramment affichée. On peut donc écrire :

Première modification du code

```

    else if (actionSelectionnee == '/') {
        if (valeurAffichee != 0) {
            resultatCourant /= valeurAffichee;
            parent.champAffichage.setText(formatNombres
                .format(resultatCourant));
        }
        else {
            // Afficher message d'erreur
        }
    }

```

Vous savez également qu'une manière facile d'afficher un message d'erreur, consiste à utiliser la classe JOptionPane. Dans le cours vous avez utilisé la méthode statique showConfirmDialog().

Cette fois-ci j'ai utilisé la méthode showMessageDialog(), qui est plus flexible. On peut non seulement changer le message et le titre, mais aussi l'icône à afficher à côté du message. De cette manière, nous afficherons une icône signalant une erreur (encore une fois, n'hésitez pas à aller consulter la javadoc correspondant à votre JDK).

Signatures de showMessageDialog()

```

JOptionPane.showMessageDialog(controleParent, message) ;
JOptionPane.showMessageDialog(controleParent, message, titre, icône) ;

```

Nous utiliserons la deuxième version :

Le code utilisé pour notifier l'utilisateur de l'erreur

```

JOptionPane.showMessageDialog(null, " Division par 0 impossible ", " Erreur de saisie ",
    JOptionPane.ERROR_MESSAGE) ;

```

Voici donc le code du MoteurCalcul mis à jour (le code de la calculatrice reste presque le même, si ce n'est qu'il faut appeler MoteurCalcul_exerciceSimple_corrige cette fois-ci) :

Moteur calcul corrigé (exercice simple)

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

```



Moteur calcul corrigé (exercice simple)

```
import java.text.NumberFormat;
import java.text.ParsePosition;

import javax.swing.JButton;
import javax.swing.JOptionPane;

public class MoteurCalcul_exerciceSimple_corrige implements ActionListener {
    Calculatrice_avantCorrection parent; // une reference a la Calculatrice
    char actionSelectionnee = ' '; // +, -, /, ou *
    double resultatCourant = 0;
    NumberFormat formatNombres = NumberFormat.getInstance();
    // un objet capable de lire et presenter les nombres

    // Le constructeur stocke la reference a la fenetre
    // Calculatrice dans la variable membre parent
    MoteurCalcul_exerciceSimple_corrige(Calculatrice_avantCorrection parent) {
        this.parent = parent;
    }

    public void actionPerformed(ActionEvent evenement) {
        // Retrouve la source de l'action
        JButton boutonClique = (JButton) evenement.getSource();
        String texteChampAffichage =
            parent.champAffichage.getText();
        double valeurAffichee = 0;
        // Retrouve le nombre presente dans le champ texte
        // s'il n'est pas vide
        if (!"".equals(texteChampAffichage)) {
            valeurAffichee =
                // analyse la chaîne de caracteres
                formatNombres.parse(
                    texteChampAffichage,
                    new ParsePosition(0) /* ne sert pas */).
                    // puis donne sa valeur en tant que double
                    doubleValue();
        }
        Object sourceEvenement = evenement.getSource();

        // Pour chaque bouton d'action, memorise l'action
        // selectionnee, +, -, /, ou *, stocke la valeur courante
        // dans la variable resultatCourant et vide le champ
        // Affichage avant l'entree du nombre suivant

        if (sourceEvenement == parent.boutonPlus) {
            actionSelectionnee = '+';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonMoins) {
            actionSelectionnee = '-';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonDivise) {
            actionSelectionnee = '/';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonMultiplie) {
            actionSelectionnee = '*';
            resultatCourant = valeurAffichee;
            parent.champAffichage.setText("");
        }
        else if (sourceEvenement == parent.boutonEgale) {
            // Effectue les calculs en fonction de actionSelectionnee
            // Modifie la valeur de la variable resultatCourant
            // et affiche le resultat
            if (actionSelectionnee == '+') {
                resultatCourant += valeurAffichee;
            }
            // Convertit le resultat en le transformant en String
        }
    }
}
```

Moteur calcul corrigé (exercice simple)

```
// a l'aide de formatNombres
parent.champAffichage.setText(
    formatNombres.format(resultatCourant));
}
else if (actionSelectionnee == '-') {
    resultatCourant -= valeurAffichee;
    parent.champAffichage.setText(
        formatNombres.format(resultatCourant));
}
else if (actionSelectionnee == '/') {
    if (valeurAffichee != 0) {
        resultatCourant /= valeurAffichee;
        parent.champAffichage.setText(formatNombres
            .format(resultatCourant));
    }
    else {
        JOptionPane.showMessageDialog(null, "Impossible de diviser par 0", "Erreur saisie",
            JOptionPane.ERROR_MESSAGE);
    }
}
else if (actionSelectionnee == '*') {
    resultatCourant *= valeurAffichee;
    parent.champAffichage.setText(
        formatNombres.format(resultatCourant));
}
}
else {
    // Pour tous les boutons numeriques, ajoute le libelle
    // du bouton au champ texte
    String libelleBoutonClique = boutonClique.getText();
    parent.champAffichage.setText(texteChampAffichage +
        libelleBoutonClique);
}
}
}
```

V-C - Exercice pour les petits malins

On nous demande de n'autoriser qu'une seule virgule à la fois dans le champ d'affichage. Par conséquent, c'est dans la classe MoteurCalcul que nous allons opérer.

Dans le code, une section se charge d'ajouter, à la zone d'affichage, le contenu du bouton cliqué. Bien sûr si celui-ci n'est pas un bouton opérateur :

La section de code qui nous intéresse

```
else {
    // Pour tous les boutons numeriques, ajoute le libelle
    // du bouton au champ texte
    String libelleBoutonClique = boutonClique.getText();
    parent.champAffichage.setText(texteChampAffichage +
        libelleBoutonClique);
}
```

Il suffit donc de regarder si le bouton cliqué est le bouton virgule et d'adapter le traitement en conséquence :

Modification possible

```
else {
    // Si le bouton n'est pas le bouton virgule
    // On peut se contenter d'ajouter son texte
    // sur l'ecran
    if (boutonClique != parent.boutonVirgule) {
        // Pour tous les boutons numeriques, ajoute le libelle
        // du bouton au champ texte
        String libelleBoutonClique = boutonClique.getText();
        ajouterCeTexteAuChampAffichage(libelleBoutonClique);
    }
}
```

Modification possible

```

    }
    else {
        // traitement pour bouton virgule
    }
}

```

L'exercice nous suggère d'utiliser la méthode `indexOf()` de la classe `String`. La javadoc nous apprend que cette fonction retourne :

- l'index de la première **occurrence** du caractère passé (c'est-à-dire l'index du premier caractère identique qu'elle trouve à partir du début de la chaîne) si celui-ci y figure ;
- -1 si celui-ci n'y figure pas du tout.

Il suffit donc de tester que la valeur de :

```
texteChampAffichage.indexOf(',')
```

vaut -1.

Attention tout de même à passer un caractère et non une chaîne : on utilise donc de simples apostrophes.

Voici donc ce que cela peut donner :

```

else {
    // Si le bouton n'est pas le bouton virgule
    // On peut se contenter d'ajouter son texte
    // sur l'écran
    if (boutonClique != parent.boutonVirgule) {
        // Pour tous les boutons numériques, ajoute le libelle
        // du bouton au champ texte
        String libelleBoutonClique = boutonClique.getText();
        parent.champAffichage.setText(texteChampAffichage
            + libelleBoutonClique);
    }
    else {
        boolean virguleNonPresente = texteChampAffichage.indexOf(',') == -1;
        if (virguleNonPresente) {
            parent.champAffichage.setText(texteChampAffichage + ',');
        }
    }
}

```

V-C-1 - Code complet



MoteurCalcul exercice avancé corrigé

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.NumberFormat;
import java.text.ParsePosition;

import javax.swing.JButton;
import javax.swing.JOptionPane;

public class MoteurCalcul_pourMalins_corrige implements ActionListener {
    Calculatrice_avantCorrection parent; // une reference a la Calculatrice
    char actionSelectionnee = '+'; // +, -, /, ou *
    double resultatCourant = 0;
    NumberFormat formatNombres = NumberFormat.getInstance();
    // un objet capable de lire et presenter les nombres

    // Le constructeur stocke la reference a la fenetre

```

MoteurCalcul exercice avancé corrigé

```
// Calculatrice dans la variable membre parent
MoteurCalcul_pourMalins_corrige(Calculatrice_avantCorrection parent) {
    this.parent = parent;
}

public void actionPerformed(ActionEvent evenement) {
    // Retrouve la source de l'action
    JButton boutonClique = (JButton) evenement.getSource();
    String texteChampAffichage =
        parent.champAffichage.getText();
    double valeurAffichee = 0;
    // Retrouve le nombre presente dans le champ texte
    // s'il n'est pas vide
    if (!"".equals(texteChampAffichage)) {
        try {
            valeurAffichee =
                // analyse la chaîne de caracteres
                formatNombres.parse(
                    texteChampAffichage,
                    new ParsePosition(0) /* ne sert pas */).
                    // puis donne sa valeur en tant que double
                    doubleValue();
        } catch (RuntimeException e) {
            JOptionPane.showConfirmDialog(null, "Merci d'entrer un nombre.",
                "Erreur de saisie", JOptionPane.PLAIN_MESSAGE);
        }
    }
    Object sourceEvenement = evenement.getSource();

    // Pour chaque bouton d'action, memorise l'action
    // selectionnee, +, -, /, ou *, stocke la valeur courante
    // dans la variable resultatCourant et vide le champ
    // Affichage avant l'entree du nombre suivant

    if (sourceEvenement == parent.boutonPlus) {
        actionSelectionnee = '+';
        resultatCourant = valeurAffichee;
        parent.champAffichage.setText("");
    }
    else if (sourceEvenement == parent.boutonMoins) {
        actionSelectionnee = '-';
        resultatCourant = valeurAffichee;
        parent.champAffichage.setText("");
    }
    else if (sourceEvenement == parent.boutonDivise) {
        actionSelectionnee = '/';
        resultatCourant = valeurAffichee;
        parent.champAffichage.setText("");
    }
    else if (sourceEvenement == parent.boutonMultiplie) {
        actionSelectionnee = '*';
        resultatCourant = valeurAffichee;
        parent.champAffichage.setText("");
    }
    else if (sourceEvenement == parent.boutonEgale) {
        // Effectue les calculs en fonction de actionSelectionnee
        // Modifie la valeur de la variable resultatCourant
        // et affiche le resultat
        if (actionSelectionnee == '+') {
            resultatCourant += valeurAffichee;
            // Convertit le resultat en le transformant en String
            // a l'aide de formatNombres
            parent.champAffichage.setText(
                formatNombres.format(resultatCourant));
        }
        else if (actionSelectionnee == '-') {
            resultatCourant -= valeurAffichee;
            parent.champAffichage.setText(
                formatNombres.format(resultatCourant));
        }
    }
}
```

MoteurCalcul exercice avancé corrigé

```

else if (actionSelectionnee == '/') {
    if (valeurAffichee != 0) {
        resultatCourant /= valeurAffichee;
        parent.champAffichage.setText(formatNombres
            .format(resultatCourant));
    }
    else {
        JOptionPane.showMessageDialog(null, "Impossible de diviser par 0", "Erreur saisie",
            JOptionPane.ERROR_MESSAGE);
    }
}
else if (actionSelectionnee == '*') {
    resultatCourant *= valeurAffichee;
    parent.champAffichage.setText(
        formatNombres.format(resultatCourant));
}
}
else {
    // Si le bouton n'est pas le bouton virgule
    // On peut se contenter d'ajouter son texte
    // sur l'ecran
    if (boutonClique != parent.boutonVirgule) {
        // Pour tous les boutons numeriques, ajoute le libelle
        // du bouton au champ texte
        String libelleBoutonClique = boutonClique.getText();
        parent.champAffichage.setText(texteChampAffichage
            + libelleBoutonClique);
    }
    else {
        boolean virguleNonPresente = texteChampAffichage.indexOf(',') == -1;
        if (virguleNonPresente) {
            parent.champAffichage.setText(texteChampAffichage+',');
        }
    }
}
}
}
}
}

```

V-D - Synthèse

Ainsi, vous avez pu :

- utiliser une boucle for each pour parcourir l'ensemble des composants d'un panneau ;
- modifier la classe MoteurCalcul afin d'afficher un message d'erreur si l'utilisateur tente une division par 0 ;
- empêcher l'utilisateur d'entrer deux virgules (ou plus) pour le même nombre.

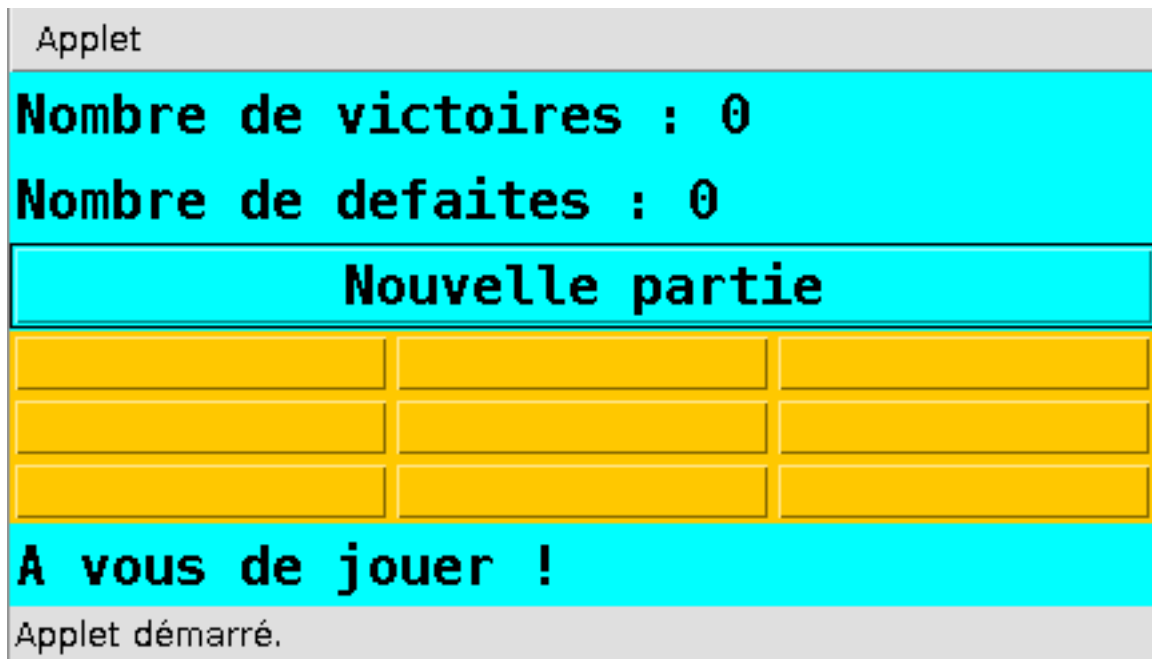
VI - Chapitre 7

VI-A - Exercice 1 : nombre de victoires/défaites

VI-A-1 - Analyses et stratégies

On nous demande d'ajouter au panneau supérieur deux champs affichant le nombre de victoires et de défaites (code ajouté en marron).

Nous allons essayer d'aboutir au résultat suivant :



Aperçu de l'applet

On peut ajouter les champs en tant que composants Label, dont les textes seront mis à jour à chaque fin de partie. On aura aussi besoin de garder une **référence** sur les deux composants, afin que les différentes méthodes de la classe puissent en modifier le texte. J'en fais donc des **variables de classe** et j'y déclare ces deux composants, mais en dehors de toute **méthode**.

```
Button cases[];
Button boutonNouvellePartie;
Label score;
Label labelVictoires;
Label labelDefaites;
```

Mais il convient également de réfléchir quant à l'agencement des composants dans le panneau supérieur. En effet, on n'aura plus affaire à un seul Button qui en occupera tout l'espace, mais à trois composants qu'il faudra faire coopérer le plus harmonieusement possible.

Pour ma part, j'ai opté pour un GridLayout à trois lignes et une colonne. Donc, avant de créer et ajouter les composants, je définis le **LayoutManager** :

```
// Cree le bouton Nouvelle partie et enregistre
// le recepteur d'actions aupres de lui
boutonNouvellePartie = new Button("Nouvelle partie");
boutonNouvellePartie.addActionListener(this);
// Cree deux panneaux et un label et les agence en
```

```
// utilisant le border layout
Panel panneauSuperieur = new Panel();
panneauSuperieur.setLayout(new GridLayout(3,1));
// Cree les deux labels victoires/defaites
labelVictoires = new Label("Nombre de victoires : 0");
labelDefaites = new Label("Nombre de defaites : 0");
// Ajoute les composants au panneau superieur
panneauSuperieur.add(labelVictoires);
panneauSuperieur.add(labelDefaites);
panneauSuperieur.add(boutonNouvellePartie);
this.add(panneauSuperieur, "North");
```

On nous recommande également d'utiliser deux variables de classes et de les utiliser pour comptabiliser le nombre de victoires et de défaites :

```
int casesLibresRestantes = 9;
int nombreVictoires = 0;
int nombreDefaites = 0;
```

Enfin, il ne nous reste plus qu'à incrémenter le nombre de victoires/défaites juste en fin de partie et de mettre à jour les Labels correspondants :

```
} // Fin de la boucle for
    if (gagnant.equals("X")) {
        score.setText("Vous avez gagné !");
        nombreVictoires++;
        labelVictoires.setText("Nombre de victoires : "+nombreVictoires);
    } else if (gagnant.equals("O")) {
        score.setText("Vous avez perdu !");
        nombreDefaites++;
        labelDefaites.setText("Nombre de defaites : "+nombreDefaites);
    } else if (gagnant.equals("T")) {
        score.setText("Partie nulle !");
    }
} // Fin de actionPerformed
```

Vous connaissez déjà l'opérateur d'incrémentation ++, car vous l'avez déjà utilisé pour la boucle for.

```
NombreVictoires++;
```

équivalent à

```
nombreVictoires = nombreVictoires + 1;
```

VI-A-2 - Code complet



Nouvelle version du morpion (ex1)

```
/**
 * Un jeu de morpion sur un plateau 3x3
 */
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class Morpion_ex1 extends Applet implements
ActionListener {
    Button cases[];
    Button boutonNouvellePartie;
    Label score;
    Label labelVictoires;
    Label labelDefaites;

    int casesLibresRestantes = 9;
    int nombreVictoires = 0;
```

Nouvelle version du morpion (ex1)

```

int nombreDefaites = 0;
/**
 * La methode init est comme un constructeur pour l'applet
 */
public void init() {
    // Affecte le gestionnaire de disposition et la couleur
    // de l'applet
    this.setLayout(new BorderLayout());
    this.setBackground(Color.CYAN);
    // Passe la police de l'applet en style gras et taille 20
    Font policeApplet = new Font("Monospaced", Font.BOLD, 20);
    this.setFont(policeApplet);
    // Cree le bouton Nouvelle partie et enregistre
    // le recepteur d'actions aupres de lui
    boutonNouvellePartie = new Button("Nouvelle partie");
    boutonNouvellePartie.addActionListener(this);
    // Cree deux panneaux et un label et les agence en
    // utilisant le border layout
    Panel panneauSuperieur = new Panel();
    panneauSuperieur.setLayout(new GridLayout(3,1));
    // Cree les deux labels victoires/defaites
    labelVictoires = new Label("Nombre de victoires : 0");
    labelDefaites = new Label("Nombre de defaites : 0");
    // Ajoute les composants au panneau superieur
    panneauSuperieur.add(labelVictoires);
    panneauSuperieur.add(labelDefaites);
    panneauSuperieur.add(boutonNouvellePartie);
    this.add(panneauSuperieur, "North");
    Panel panneauCentral = new Panel();
    panneauCentral.setLayout(new GridLayout(3, 3));
    this.add(panneauCentral, "Center");
    score = new Label("A vous de jouer !");
    this.add(score, "South");
    // Cree un tableau pour stocker les references des
    // 9 boutons
    cases = new Button[9];
    // Instancie les boutons, stocke leurs references dans le
    // tableau, enregistre le recepteur aupres d'eux, peint
    // les boutons en orange et les ajoute au panneau central
    for(int i = 0; i < 9; i++) {
        cases[i] = new Button();
        cases[i].addActionListener(this);
        cases[i].setBackground(Color.ORANGE);
        panneauCentral.add(cases[i]);
    }
}
/**
 * Cette methode traite tous les evenements d'action
 * @param evenement l'evenement a traiter
 */
public void actionPerformed(ActionEvent evenement) {
    Button leBouton = (Button) evenement.getSource();
    // S'agit-il du bouton Nouvelle partie ?
    if (leBouton == boutonNouvellePartie) {
        for(int i = 0; i < 9; i++) {
            cases[i].setEnabled(true);
            cases[i].setLabel("");
            cases[i].setBackground(Color.ORANGE);
        }
        casesLibresRestantes = 9;
        score.setText("A vous de jouer !");
        boutonNouvellePartie.setEnabled(false);
        return; // Sort de la methode
    }
    String gagnant = "";
    //S'agit-il de l'une des cases ?
    for (int i = 0; i < 9; i++) {
        if (leBouton == cases[i]) {
            cases[i].setLabel("X");
            gagnant = chercherUnGagnant();
        }
    }
}

```


Nouvelle version du morpion (ex1)

```

        if (!"".equals(gagnant)) {
            terminerLaPartie();
        } else {
            coupOrdinateur();
            gagnant = chercherUnGagnant();
            if (!"".equals(gagnant)) {
                terminerLaPartie();
            }
        }
        break;
    }
} // Fin de la boucle for
if (gagnant.equals("X")) {
    score.setText("Vous avez gagné !");
    nombreVictoires++;
    labelVictoires.setText("Nombre de victoires : "+nombreVictoires);
} else if (gagnant.equals("O")) {
    score.setText("Vous avez perdu !");
    nombreDefaites++;
    labelDefaites.setText("Nombre de defaites : "+nombreDefaites);
} else if (gagnant.equals("T")) {
    score.setText("Partie nulle !");
}
} // Fin de actionPerformed
/**
 * Cette methode est appelee apres chaque coup joue pour
 * voir s'il y a un gagnant. Elle verifie pour chaque ligne,
 * colonne et diagonale, s'il y a trois symboles identiques
 * @return "X", "O", "T" (termine, partie nulle) ou "" (pas
 * fini)
 */
String chercherUnGagnant() {
    String leGagnant = "";
    casesLibresRestantes--;
    //Verifie la ligne 1 - elements 0, 1 et 2 du tableau
    if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[1].getLabel()) &&
        cases[0].getLabel().equals(cases[2].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 1, 2);
        //Verifie la ligne 2 - elements 3, 4 et 5 du tableau
    } else if (!cases[3].getLabel().equals("") &&
        cases[3].getLabel().equals(cases[4].getLabel()) &&
        cases[3].getLabel().equals(cases[5].getLabel())) {
        leGagnant = cases[3].getLabel();
        montrerGagnant(3, 4, 5);
        //Verifie la ligne 3 - elements 6, 7 et 8 du tableau
    } else if (!cases[6].getLabel().equals("") &&
        cases[6].getLabel().equals(cases[7].getLabel()) &&
        cases[6].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[6].getLabel();
        montrerGagnant(6, 7, 8);
        //Verifie la colonne 1 - elements 0, 3 et 6 du tableau
    } else if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[3].getLabel()) &&
        cases[0].getLabel().equals(cases[6].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 3, 6);
        //Verifie la colonne 2 - elements 1, 4 et 7 du tableau
    } else if (!cases[1].getLabel().equals("") &&
        cases[1].getLabel().equals(cases[4].getLabel()) &&
        cases[1].getLabel().equals(cases[7].getLabel())) {
        leGagnant = cases[1].getLabel();
        montrerGagnant(1, 4, 7);
        //Verifie la colonne 3 - elements 2, 5 et 8 du tableau
    } else if (!cases[2].getLabel().equals("") &&
        cases[2].getLabel().equals(cases[5].getLabel()) &&
        cases[2].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[2].getLabel();
        montrerGagnant(2, 5, 8);
        //Verifie la premiere diagonale - elements 0, 4 et 8
    }
}

```

Nouvelle version du morpion (ex1)

```

    } else if (!cases[0].getLabel().equals("") &&
               cases[0].getLabel().equals(cases[4].getLabel()) &&
               cases[0].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 4, 8);
        //Verifie la seconde diagonale - elements 2, 4 et 6
    } else if (!cases[2].getLabel().equals("") &&
               cases[2].getLabel().equals(cases[4].getLabel()) &&
               cases[2].getLabel().equals(cases[6].getLabel())) {
        leGagnant = cases[2].getLabel();
        montrerGagnant(2, 4, 6);
    } else if (casesLibresRestantes == 0) {
        return "T"; // Partie nulle
    }
    return leGagnant;
}
/**
 * Cette methode applique un ensemble de regles afin de
 * trouver le meilleur coup pour l'ordinateur. Si un bon
 * coup ne peut etre trouve, elle choisit une case au
 * hasard.
 */
void coupOrdinateur() {
    int caseSelectionnee;
    //L'ordinateur essaie d'abord de trouver une case
    //vide pres de deux case marquees "O" pour gagner
    caseSelectionnee = trouverCaseVide("O");
    //S'il n'y a pas deux "O" alignes, essaie au moins
    //d'empêcher l'adversaire d'aligner trois "X" en
    //plaçant un "O" pres de deux "X"
    if (caseSelectionnee == -1) {
        caseSelectionnee = trouverCaseVide("X");
    }
    //Si caseSelectionnee vaut toujours -1, essaie d'occuper
    //la case centrale
    if ((caseSelectionnee == -1)
        && (cases[4].getLabel().equals(""))) {
        caseSelectionnee = 4;
    }
    //Pas de chance avec la case centrale non plus...
    //Choisit une case au hasard
    if (caseSelectionnee == -1) {
        caseSelectionnee = choisirCaseAuHasard();
    }
    cases[caseSelectionnee].setLabel("O");
}
/**
 * Cette methode examine chaque ligne, colonne et diagonale
 * pour voir si elle contient deux cases avec le meme label
 * et une case vide.
 * @param joueur "X" pour l'utilisateur ou "O" pour
 * l'ordinateur
 * @return numero de la case vide a utiliser ou le nombre
 * negatif -1 si la recherche est infructueuse
 */
int trouverCaseVide(String joueur) {
    int poids[] = new int[9];
    for (int i = 0; i < 9; i++) {
        if (cases[i].getLabel().equals("O"))
            poids[i] = -1;
        else if (cases[i].getLabel().equals("X"))
            poids[i] = 1;
        else
            poids[i] = 0;
    }
    int deuxPoids = joueur.equals("O") ? -2 : 2;
    // Regarde si la ligne 1 a 2 cases identiques et une vide

    if (poids[0] + poids[1] + poids[2] == deuxPoids) {

```

Nouvelle version du morpion (ex1)

```

    if (poids[0] == 0)
        return 0;
    else if (poids[1] == 0)
        return 1;
    else
        return 2;
}
// Regarde si la ligne 2 a 2 cases identiques et une vide
if (poids[3] + poids[4] + poids[5] == deuxPoids) {

    if (poids[3] == 0)
        return 3;
    else if (poids[4] == 0)
        return 4;
    else
        return 5;
}
//Regarde si la ligne 3 a 2 cases identiques et une vide
if (poids[6] + poids[7] + poids[8] == deuxPoids) {
    if (poids[6] == 0)
        return 6;
    else if (poids[7] == 0)
        return 7;
    else
        return 8;
}
//Regarde si la colonne 1 a 2 cases identiques et une vide
if (poids[0] + poids[3] + poids[6] == deuxPoids) {
    if (poids[0] == 0)
        return 0;
    else if (poids[3] == 0)
        return 3;
    else
        return 6;
}
//Regarde si la colonne 2 a 2 cases identiques et une vide
if (poids[1] + poids[4] + poids[7] == deuxPoids) {
    if (poids[1] == 0)
        return 1;
    else if (poids[4] == 0)
        return 4;
    else
        return 7;
}
//Regarde si la colonne 3 a 2 cases identiques et une vide
if (poids[2] + poids[5] + poids[8] == deuxPoids) {
    if (poids[2] == 0)
        return 2;
    else if (poids[5] == 0)
        return 5;
    else
        return 8;
}
//Regarde si la diagonale 1 a 2 cases identiques et une
//vide
if (poids[0] + poids[4] + poids[8] == deuxPoids) {
    if (poids[0] == 0)
        return 0;
    else if (poids[4] == 0)
        return 4;
    else
        return 8;
}
//Regarde si la diagonale 2 a 2 cases identiques et une
//vide
if (poids[2] + poids[4] + poids[6] == deuxPoids) {
    if (poids[2] == 0)

```

Nouvelle version du morpion (ex1)

```

        return 2;
    else if (poids[4] == 0)
        return 4;
    else
        return 6;
    }
    //Il n'y a pas de cases alignées identiques
    return -1;
} // Fin de trouverCaseVide()
/**
 * Cette methode selectionne une case vide quelconque.
 * @return un numero de case choisi au hasard
 */
int choisirCaseAuHasard() {
    boolean caseVideTrouvee = false;
    int caseSelectionnee = -1;
    do {
        caseSelectionnee = (int) (Math.random() * 9);
        if (cases[caseSelectionnee].getLabel().equals("")) {
            caseVideTrouvee = true; // Pour terminer la boucle
        }
    } while (!caseVideTrouvee);
    return caseSelectionnee;
} // Fin de choisirCaseAuHasard()
/**
 * Cette methode affiche la ligne gagnante en surbrillance.
 * @param gagnante1 premiere case a montrer.
 * @param gagnante2 deuxieme case a montrer.
 * @param gagnante3 troisieme case a montrer.
 */
void montrerGagnant(int gagnante1, int gagnante2, int
    gagnante3) {
    cases[gagnante1].setBackground(Color.CYAN);
    cases[gagnante2].setBackground(Color.CYAN);
    cases[gagnante3].setBackground(Color.CYAN);
}
//Desactive les cases et active le bouton Nouvelle partie
void terminerLaPartie() {
    boutonNouvellePartie.setEnabled(true);
    for (int i = 0; i < 9; i++) {
        cases[i].setEnabled(false);
    }
}
} // Fin de la classe

```

VI-B - Exercice 2 : résolution d'un bogue

VI-B-1 - Quelques astuces

Un bogue grave subsiste : on peut modifier la valeur d'une case déjà jouée !

Pour remédier à ceci, il faut modifier le gestionnaire de clics. Qui dit gestionnaire de clics, dit méthode actionPerformed de l'interface ActionListener.

Il suffira donc, dans la méthode actionPerformed, de n'exécuter la modification du bouton cliqué uniquement si son texte est vide.

Mais d'abord, il ne faut pas perdre de vue que les chaînes de caractères ne sont pas de simples types primitifs (à l'instar des int, boolean, ☐) , mais des données de type Object. Et la meilleure manière de comparer deux objets (on suppose ici que objet1 et objet2 sont comparables, ce qui est le cas si objet1 et objet2 sont du même type ou si objet1 et objet2 ont au moins une classe mère commune), c'est de faire appel à la méthode equals() de l'un des objets :

```
objet1.equals(objet2) ;
```

```
objet2.equals(objet1) ;
```

Donc, si j'ai une chaîne de caractère chaîne1 et que je veuille tester si elle est vide, je pourrais écrire :

```
chaîne1.equals("") ;
```

Enfin le texte d'un composant java.awt.Button s'obtient en appelant sa méthode getLabel().

VI-B-2 - Méthode actionPerformed corrigée

La nouvelle version de la méthode actionPerformed()

```
/**
 * Cette methode traite tous les evenements d'action
 * @param evenement l'evenement a traiter
 */
public void actionPerformed(ActionEvent evenement) {
    Button leBouton = (Button) evenement.getSource();
    // S'agit-il du bouton Nouvelle partie ?
    if (leBouton == boutonNouvellePartie) {
        for(int i = 0; i < 9; i++) {
            cases[i].setEnabled(true);
            cases[i].setLabel("");
            cases[i].setBackground(Color.ORANGE);
        }
        casesLibresRestantes = 9;
        score.setText("A vous de jouer !");
        boutonNouvellePartie.setEnabled(false);
        return; // Sort de la methode
    }
    String gagnant = "";
    //S'agit-il de l'une des cases ?
    for (int i = 0; i < 9; i++) {
        if (leBouton == cases[i] && cases[i].getLabel().equals("")) {
            cases[i].setLabel("X");
            gagnant = chercherUnGagnant();
            if (!"".equals(gagnant)) {
                terminerLaPartie();
            } else {
                coupOrdinateur();
                gagnant = chercherUnGagnant();
                if (!"".equals(gagnant)) {
                    terminerLaPartie();
                }
            }
            break;
        }
    }
    // Fin de la boucle for
    if (gagnant.equals("X")) {
        score.setText("Vous avez gagné !");
        nombreVictoires++;
        labelVictoires.setText("Nombre de victoires : "+nombreVictoires);
    } else if (gagnant.equals("O")) {
        score.setText("Vous avez perdu !");
        nombreDefaites++;
        labelDefaites.setText("Nombre de defaites : "+nombreDefaites);
    } else if (gagnant.equals("T")) {
        score.setText("Partie nulle !");
    }
} // Fin de actionPerformed
```

VI-B-3 - Code complet

Nouvelle version du morpion (ex2)

```
/**
```

Nouvelle version du morpion (ex2)

```

* Un jeu de morpion sur un plateau 3x3
*/
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class Morpion_ex2 extends Applet implements
ActionListener {
    Button cases[];
    Button boutonNouvellePartie;
    Label score;
    Label labelVictoires;
    Label labelDefaites;

    int casesLibresRestantes = 9;
    int nombreVictoires = 0;
    int nombreDefaites = 0;
    /**
     * La methode init est comme un constructeur pour l'applet
     */
    public void init() {
        // Affecte le gestionnaire de disposition et la couleur
        // de l'applet
        this.setLayout(new BorderLayout());
        this.setBackground(Color.CYAN);
        // Passe la police de l'applet en style gras et taille 20
        Font policeApplet = new Font("Monospaced", Font.BOLD, 20);
        this.setFont(policeApplet);
        // Cree le bouton Nouvelle partie et enregistre
        // le recepteur d'actions aupres de lui
        boutonNouvellePartie = new Button("Nouvelle partie");
        boutonNouvellePartie.addActionListener(this);
        // Cree deux panneaux et un label et les agence en
        // utilisant le border layout
        Panel panneauSuperieur = new Panel();
        panneauSuperieur.setLayout(new GridLayout(3,1));
        // Cree les deux labels victoires/defaites
        labelVictoires = new Label("Nombre de victoires : 0");
        labelDefaites = new Label("Nombre de defaites : 0");
        // Ajoute les composants au panneau superieur
        panneauSuperieur.add(labelVictoires);
        panneauSuperieur.add(labelDefaites);
        panneauSuperieur.add(boutonNouvellePartie);
        this.add(panneauSuperieur, "North");
        Panel panneauCentral = new Panel();
        panneauCentral.setLayout(new GridLayout(3, 3));
        this.add(panneauCentral, "Center");
        score = new Label("A vous de jouer !");
        this.add(score, "South");
        // Cree un tableau pour stocker les references des
        // 9 boutons
        cases = new Button[9];
        // Instancie les boutons, stocke leurs references dans le
        // tableau, enregistre le recepteur aupres d'eux, peint
        // les boutons en orange et les ajoute au panneau central
        for(int i = 0; i < 9; i++) {
            cases[i] = new Button();
            cases[i].addActionListener(this);
            cases[i].setBackground(Color.ORANGE);
            panneauCentral.add(cases[i]);
        }
    }
    /**
     * Cette methode traite tous les evenements d'action
     * @param evenement l'evenement a traiter
     */
    public void actionPerformed(ActionEvent evenement) {
        Button leBouton = (Button) evenement.getSource();
        // S'agit-il du bouton Nouvelle partie ?
        if (leBouton == boutonNouvellePartie) {
            for(int i = 0; i < 9; i++) {

```

Nouvelle version du morpion (ex2)

```

        cases[i].setEnabled(true);
        cases[i].setLabel("");
        cases[i].setBackground(Color.ORANGE);
    }
    casesLibresRestantes = 9;
    score.setText("A vous de jouer !");
    boutonNouvellePartie.setEnabled(false);
    return; // Sort de la methode
}
String gagnant = "";
//S'agit-il de l'une des cases ?
for (int i = 0; i < 9; i++) {
    if (leBouton == cases[i] && cases[i].getLabel().equals("")) {
        cases[i].setLabel("X");
        gagnant = chercherUnGagnant();
        if (!"".equals(gagnant)) {
            terminerLaPartie();
        } else {
            coupOrdinateur();
            gagnant = chercherUnGagnant();
            if (!"".equals(gagnant)) {
                terminerLaPartie();
            }
        }
        break;
    }
} // Fin de la boucle for
if (gagnant.equals("X")) {
    score.setText("Vous avez gagne !");
    nombreVictoires++;
    labelVictoires.setText("Nombre de victoires : "+nombreVictoires);
} else if (gagnant.equals("O")) {
    score.setText("Vous avez perdu !");
    nombreDefaites++;
    labelDefaites.setText("Nombre de defaites : "+nombreDefaites);
} else if (gagnant.equals("T")) {
    score.setText("Partie nulle !");
}
} // Fin de actionPerformed
/**
 * Cette methode est appelee apres chaque coup joue pour
 * voir s'il y a un gagnant. Elle verifie pour chaque ligne,
 * colonne et diagonale, s'il y a trois symboles identiques
 * @return "X", "O", "T" (termine, partie nulle) ou "" (pas
 * fini)
 */
String chercherUnGagnant() {
    String leGagnant = "";
    casesLibresRestantes--;
    //Verifie la ligne 1 - elements 0, 1 et 2 du tableau
    if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[1].getLabel()) &&
        cases[0].getLabel().equals(cases[2].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 1, 2);
        //Verifie la ligne 2 - elements 3, 4 et 5 du tableau
    } else if (!cases[3].getLabel().equals("") &&
        cases[3].getLabel().equals(cases[4].getLabel()) &&
        cases[3].getLabel().equals(cases[5].getLabel())) {
        leGagnant = cases[3].getLabel();
        montrerGagnant(3, 4, 5);
        //Verifie la ligne 3 - elements 6, 7 et 8 du tableau
    } else if (!cases[6].getLabel().equals("") &&
        cases[6].getLabel().equals(cases[7].getLabel()) &&
        cases[6].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[6].getLabel();
        montrerGagnant(6, 7, 8);
        //Verifie la colonne 1 - elements 0, 3 et 6 du tableau
    } else if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[3].getLabel()) &&
        cases[0].getLabel().equals(cases[6].getLabel())) {

```

Nouvelle version du morpion (ex2)

```

    leGagnant = cases[0].getLabel();
    montrerGagnant(0, 3, 6);
    //Verifie la colonne 2 - elements 1, 4 et 7 du tableau
} else if (!cases[1].getLabel().equals("") &&
    cases[1].getLabel().equals(cases[4].getLabel()) &&
    cases[1].getLabel().equals(cases[7].getLabel())) {
    leGagnant = cases[1].getLabel();
    montrerGagnant(1, 4, 7);
    //Verifie la colonne 3 - elements 2, 5 et 8 du tableau
} else if (!cases[2].getLabel().equals("") &&
    cases[2].getLabel().equals(cases[5].getLabel()) &&
    cases[2].getLabel().equals(cases[8].getLabel())) {
    leGagnant = cases[2].getLabel();
    montrerGagnant(2, 5, 8);
    //Verifie la premiere diagonale - elements 0, 4 et 8
} else if (!cases[0].getLabel().equals("") &&
    cases[0].getLabel().equals(cases[4].getLabel()) &&
    cases[0].getLabel().equals(cases[8].getLabel())) {
    leGagnant = cases[0].getLabel();
    montrerGagnant(0, 4, 8);
    //Verifie la seconde diagonale - elements 2, 4 et 6
} else if (!cases[2].getLabel().equals("") &&
    cases[2].getLabel().equals(cases[4].getLabel()) &&
    cases[2].getLabel().equals(cases[6].getLabel())) {
    leGagnant = cases[2].getLabel();
    montrerGagnant(2, 4, 6);
} else if (casesLibresRestantes == 0) {
    return "T"; // Partie nulle
}
return leGagnant;
}
/**
 * Cette methode applique un ensemble de regles afin de
 * trouver le meilleur coup pour l'ordinateur. Si un bon
 * coup ne peut etre trouve, elle choisit une case au
 * hasard.
 */
void coupOrdinateur() {
    int caseSelectionnee;
    //L'ordinateur essaie d'abord de trouver une case
    //vide pres de deux case marquees "O" pour gagner
    caseSelectionnee = trouverCaseVide("O");
    //S'il n'y a pas deux "O" alignes, essaie au moins
    //d'empêcher l'adversaire d'aligner trois "X" en
    //plaçant un "O" pres de deux "X"
    if (caseSelectionnee == -1) {
        caseSelectionnee = trouverCaseVide("X");
    }
    //Si caseSelectionnee vaut toujours -1, essaie d'occuper
    //la case centrale
    if ((caseSelectionnee == -1)
        && (cases[4].getLabel().equals(""))) {
        caseSelectionnee = 4;
    }
    //Pas de chance avec la case centrale non plus...
    //Choisit une case au hasard
    if (caseSelectionnee == -1) {
        caseSelectionnee = choisirCaseAuHasard();
    }
    cases[caseSelectionnee].setLabel("O");
}
/**
 * Cette methode examine chaque ligne, colonne et diagonale
 * pour voir si elle contient deux cases avec le meme label
 * et une case vide.
 * @param joueur "X" pour l'utilisateur ou "O" pour
 * l'ordinateur
 * @return numero de la case vide a utiliser ou le nombre
 * negatif -1 si la recherche est infructueuse

```


Nouvelle version du morpion (ex2)

```

*/
int trouverCaseVide(String joueur) {
    int poids[] = new int[9];
    for (int i = 0; i < 9; i++) {
        if (cases[i].getLabel().equals("O"))
            poids[i] = -1;
        else if (cases[i].getLabel().equals("X"))
            poids[i] = 1;
        else
            poids[i] = 0;
    }
    int deuxPoids = joueur.equals("O") ? -2 : 2;
    // Regarde si la ligne 1 a 2 cases identiques et une vide

    if (poids[0] + poids[1] + poids[2] == deuxPoids) {

        if (poids[0] == 0)
            return 0;
        else if (poids[1] == 0)
            return 1;
        else
            return 2;
    }
    // Regarde si la ligne 2 a 2 cases identiques et une vide

    if (poids[3] + poids[4] + poids[5] == deuxPoids) {

        if (poids[3] == 0)
            return 3;
        else if (poids[4] == 0)
            return 4;
        else
            return 5;
    }
    //Regarde si la ligne 3 a 2 cases identiques et une vide
    if (poids[6] + poids[7] + poids[8] == deuxPoids) {
        if (poids[6] == 0)
            return 6;
        else if (poids[7] == 0)
            return 7;
        else
            return 8;
    }
    //Regarde si la colonne 1 a 2 cases identiques et une vide
    if (poids[0] + poids[3] + poids[6] == deuxPoids) {
        if (poids[0] == 0)
            return 0;
        else if (poids[3] == 0)
            return 3;
        else
            return 6;
    }
    //Regarde si la colonne 2 a 2 cases identiques et une vide
    if (poids[1] + poids[4] + poids[7] == deuxPoids) {
        if (poids[1] == 0)
            return 1;
        else if (poids[4] == 0)
            return 4;
        else
            return 7;
    }
    //Regarde si la colonne 3 a 2 cases identiques et une vide
    if (poids[2] + poids[5] + poids[8] == deuxPoids) {
        if (poids[2] == 0)
            return 2;
        else if (poids[5] == 0)
            return 5;
        else
            return 8;
    }
}

```

Nouvelle version du morpion (ex2)

```
//Regarde si la diagonale 1 a 2 cases identiques et une
//vide
if (poids[0] + poids[4] + poids[8] == deuxPoids) {
    if (poids[0] == 0)
        return 0;
    else if (poids[4] == 0)
        return 4;
    else
        return 8;
}
//Regarde si la diagonale 2 a 2 cases identiques et une
//vide
if (poids[2] + poids[4] + poids[6] == deuxPoids) {
    if (poids[2] == 0)
        return 2;
    else if (poids[4] == 0)
        return 4;
    else
        return 6;
}
//Il n'y a pas de cases alignées identiques
return -1;
} // Fin de trouverCaseVide()
/**
 * Cette methode selectionne une case vide quelconque.
 * @return un numero de case choisi au hasard
 */
int choisirCaseAuHasard() {
    boolean caseVideTrouvee = false;
    int caseSelectionnee = -1;
    do {
        caseSelectionnee = (int) (Math.random() * 9);
        if (cases[caseSelectionnee].getLabel().equals("")) {
            caseVideTrouvee = true; // Pour terminer la boucle
        }
    } while (!caseVideTrouvee);
    return caseSelectionnee;
} // Fin de choisirCaseAuHasard()
/**
 * Cette methode affiche la ligne gagnante en surbrillance.
 * @param gagnante1 premiere case a montrer.
 * @param gagnante2 deuxieme case a montrer.
 * @param gagnante3 troisieme case a montrer.
 */
void montrerGagnant(int gagnante1, int gagnante2, int
    gagnante3) {
    cases[gagnante1].setBackground(Color.CYAN);
    cases[gagnante2].setBackground(Color.CYAN);
    cases[gagnante3].setBackground(Color.CYAN);
}
//Desactive les cases et active le bouton Nouvelle partie
void terminerLaPartie() {
    boutonNouvellePartie.setEnabled(true);
    for (int i = 0; i < 9; i++) {
        cases[i].setEnabled(false);
    }
}
} // Fin de la classe
```

VI-C - Exercice 3 : ajout d'une méthode main()

Cet exercice-là peut s'avérer beaucoup plus compliqué qu'il n'y paraît. On nous demande de nous baser sur l'applet existante afin de coder une application à part entière : on doit pouvoir l'exécuter sans l'intégrer dans une page HTML.

Dans la méthode init() de l'applet, on a construit les différents composants. On les a également ajoutés à l'applet : c'est-à-dire à une instance de java.awt.Container. En effet, la javadoc nous apprend que la classe java.applet.Applet est une classe fille de java.awt.Container, même si c'est de manière indirecte.

Or si l'on veut que le jeu puisse être lancé en mode application, il faudra créer nous-même la fenêtre, y ajouter une instance de Panel et ajouter les composants à ce Panel (ainsi qu'ajouter le Panel à la fenêtre). Et la classe `java.awt.Panel` hérite notamment de la classe `java.awt.Container`. Il faudra donc adapter tout ce qui a été fait dans la méthode `init()` pour n'importe quel objet `Container`.

VI-C-1 - Code de la méthode `constuireContainer()`

J'ai donc créé une méthode `construireContainer()` dans la classe `Morpion` : elle prend en paramètre un `Container` et en remplit le contenu avec les composants de notre jeu `Morpion`.

Souvenez-vous que dans le cas d'une Applet, le conteneur parent de tout composant est l'Applet elle-même. Évidemment, dans le cas de l'application, cela sera juste un `Panel` : lequel sera ajouté au `LayoutManager` de la fenêtre.

La méthode `construireContainer()`

```
void construireContainer(Container container) {
    // Affecte le gestionnaire de disposition et la couleur
    // de l'applet
    container.setLayout(new BorderLayout());
    container.setBackground(Color.CYAN);
    // Passe la police de l'applet en style gras et taille 20
    Font policeApplet = new Font("Monospaced", Font.BOLD, 20);
    container.setFont(policeApplet);
    // Cree le bouton Nouvelle partie et enregistre
    // le recepteur d'actions aupres de lui
    boutonNouvellePartie = new Button("Nouvelle partie");
    boutonNouvellePartie.addActionListener(this);
    // Cree deux panneaux et un label et les agence en
    // utilisant le border layout
    Panel panneauSuperieur = new Panel();
    panneauSuperieur.setLayout(new GridLayout(3,1));
    // Cree les deux labels victoires/defaites
    labelVictoires = new Label("Nombre de victoires : 0");
    labelDefaites = new Label("Nombre de defaites : 0");
    // Ajoute les composants au panneau superieur
    panneauSuperieur.add(labelVictoires);
    panneauSuperieur.add(labelDefaites);
    panneauSuperieur.add(boutonNouvellePartie);
    container.add(panneauSuperieur, "North");
    Panel panneauCentral = new Panel();
    panneauCentral.setLayout(new GridLayout(3, 3));
    container.add(panneauCentral, "Center");
    score = new Label("A vous de jouer !");
    container.add(score, "South");
    // Cree un tableau pour stocker les references des
    // 9 boutons
    cases = new Button[9];
    // Instancie les boutons, stocke leurs references dans le
    // tableau, enregistre le recepteur aupres d'eux, peint
    // les boutons en orange et les ajoute au panneau central
    for(int i = 0; i < 9; i++) {
        cases[i] = new Button();
        cases[i].addActionListener(this);
        cases[i].setBackground(Color.ORANGE);
        panneauCentral.add(cases[i]);
    }
}
```

N'oubliez pas que dans le code original, `this` désigne l'instance de `Morpion` courante : il suffit donc, si cela est approprié, de le remplacer par `container`.

VI-C-2 - Nouvelle version de la méthode `init()`

Le code de la méthode `init()` devient donc :

```
public void init() {
    construireContainer(this);
}
```

On peut déjà, à ce stade, vérifier que l'applet fonctionne de la même manière qu'auparavant.

Maintenant on dispose d'une manière simple de remplir un Panel avec le contenu de notre jeu Morpion. On peut se mettre à coder la création de la fenêtre, dans la méthode main().

L'utilisation d'une fenêtre au lieu du codage d'une applet a tout de même quelques répercussions :

- il convient de définir, pour la fenêtre, certains aspects dont on ne se préoccupait pas en développant une applet : le titre de la fenêtre (jusque-là défini par la page HTML), la position de la fenêtre, la taille de la fenêtre ;
- il faut prévoir un moyen de fermer la fenêtre si l'utilisateur clique sur son bouton de fermeture ;
- c'est à nous-même d'afficher la fenêtre.

Afin de prévoir une fermeture de la fenêtre, il nous faut ajouter un WindowListener à la fenêtre créée et coder dans la méthode windowClosing() la fermeture proprement dite. Pour mettre fin à l'application, on appellera :

```
System.exit(0) ;
```

Passer 0 signifie que le programme s'est finalement déroulé normalement. Passer une valeur supérieure à 0 signifie que le programme a dû finir précipitamment. C'est une convention.

La classe WindowListener présentant de nombreuses méthodes dont nous n'aurons pas besoin, on peut se tourner vers la classe WindowAdapter. Il nous suffira alors de **redéfinir** la méthode windowClosing(). J'attire votre attention sur le fait que la méthode windowClosed() est appelée une fois que la fenêtre a été fermée : alors que pour windowClosing(), on se contente de détecter si la fermeture de la fenêtre a été demandée.

VI-C-3 - Code de la méthode main()

Code de la méthode main()

```
public static void main(String[] args) {
    Frame fenetre = new Frame();
    /*
     * Cette fois-ci, c'est le code qui
     * fixe les dimensions de la zone de jeu,
     * et non une page html.
     */
    fenetre.setSize(400, 300);
    /*
     * Cette fois-ci, le titre se définit dans la fenetre.
     */
    fenetre.setTitle("Jeu du morpion");
    /*
     * Centre la fenetre sur l'ecran
     */
    fenetre.setLocationRelativeTo(null);
    Panel panneauFenetre = new Panel();
    fenetre.add(panneauFenetre);

    Morpion logiqueMorpion = new Morpion();
    logiqueMorpion.construireContainer(panneauFenetre);

    fenetre.addWindowListener(new WindowAdapter() {

        @Override
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    })
}
```

Code de la méthode main()

```
});

fenetre.setVisible(true);
}
```

VI-C-4 - Code complet



Nouvelle version du morpion (ex3)

```
/**
 * Un jeu de morpion sur un plateau 3x3
 */
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class Morpion_ex3 extends Applet implements
ActionListener {
    Button cases[];
    Button boutonNouvellePartie;
    Label score;
    Label labelVictoires;
    Label labelDefaites;

    int casesLibresRestantes = 9;
    int nombreVictoires = 0;
    int nombreDefaites = 0;
    /**
     * La methode init est comme un constructeur pour l'applet
     */
    public void init() {
        construireContainer(this);
    }
    /**
     * Cette methode traite tous les evenements d'action
     * @param evenement l'evenement a traiter
     */
    public void actionPerformed(ActionEvent evenement) {
        Button leBouton = (Button) evenement.getSource();
        // S'agit-il du bouton Nouvelle partie ?
        if (leBouton == boutonNouvellePartie) {
            for(int i = 0; i < 9; i++) {
                cases[i].setEnabled(true);
                cases[i].setLabel("");
                cases[i].setBackground(Color.ORANGE);
            }
            casesLibresRestantes = 9;
            score.setText("A vous de jouer !");
            boutonNouvellePartie.setEnabled(false);
            return; // Sort de la methode
        }
        String gagnant = "";
        //S'agit-il de l'une des cases ?
        for (int i = 0; i < 9; i++) {
            if (leBouton == cases[i] && cases[i].getLabel().equals("")) {
                cases[i].setLabel("X");
                gagnant = chercherUnGagnant();
                if(!"".equals(gagnant)) {
                    terminerLaPartie();
                } else {
                    coupOrdinateur();
                    gagnant = chercherUnGagnant();
                    if (!"".equals(gagnant)) {
                        terminerLaPartie();
                    }
                }
            }
        }
        break;
    }
} // Fin de la boucle for
```

Nouvelle version du morpion (ex3)

```

    if (gagnant.equals("X")) {
        score.setText("Vous avez gagné !");
        nombreVictoires++;
        labelVictoires.setText("Nombre de victoires : "+nombreVictoires);
    } else if (gagnant.equals("O")) {
        score.setText("Vous avez perdu !");
        nombreDefaites++;
        labelDefaites.setText("Nombre de defaites : "+nombreDefaites);
    } else if (gagnant.equals("T")) {
        score.setText("Partie nulle !");
    }
} // Fin de actionPerformed
/**
 * Cette methode est appelee apres chaque coup joue pour
 * voir s'il y a un gagnant. Elle verifie pour chaque ligne,
 * colonne et diagonale, s'il y a trois symboles identiques
 * @return "X", "O", "T" (termine, partie nulle) ou "" (pas
 * fini)
 */
String chercherUnGagnant() {
    String leGagnant = "";
    casesLibresRestantes--;
    //Verifie la ligne 1 - elements 0, 1 et 2 du tableau
    if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[1].getLabel()) &&
        cases[0].getLabel().equals(cases[2].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 1, 2);
        //Verifie la ligne 2 - elements 3, 4 et 5 du tableau
    } else if (!cases[3].getLabel().equals("") &&
        cases[3].getLabel().equals(cases[4].getLabel()) &&
        cases[3].getLabel().equals(cases[5].getLabel())) {
        leGagnant = cases[3].getLabel();
        montrerGagnant(3, 4, 5);
        //Verifie la ligne 3 - elements 6, 7 et 8 du tableau
    } else if (!cases[6].getLabel().equals("") &&
        cases[6].getLabel().equals(cases[7].getLabel()) &&
        cases[6].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[6].getLabel();
        montrerGagnant(6, 7, 8);
        //Verifie la colonne 1 - elements 0, 3 et 6 du tableau
    } else if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[3].getLabel()) &&
        cases[0].getLabel().equals(cases[6].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 3, 6);
        //Verifie la colonne 2 - elements 1, 4 et 7 du tableau
    } else if (!cases[1].getLabel().equals("") &&
        cases[1].getLabel().equals(cases[4].getLabel()) &&
        cases[1].getLabel().equals(cases[7].getLabel())) {
        leGagnant = cases[1].getLabel();
        montrerGagnant(1, 4, 7);
        //Verifie la colonne 3 - elements 2, 5 et 8 du tableau
    } else if (!cases[2].getLabel().equals("") &&
        cases[2].getLabel().equals(cases[5].getLabel()) &&
        cases[2].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[2].getLabel();
        montrerGagnant(2, 5, 8);
        //Verifie la premiere diagonale - elements 0, 4 et 8
    } else if (!cases[0].getLabel().equals("") &&
        cases[0].getLabel().equals(cases[4].getLabel()) &&
        cases[0].getLabel().equals(cases[8].getLabel())) {
        leGagnant = cases[0].getLabel();
        montrerGagnant(0, 4, 8);
        //Verifie la seconde diagonale - elements 2, 4 et 6
    } else if (!cases[2].getLabel().equals("") &&
        cases[2].getLabel().equals(cases[4].getLabel()) &&
        cases[2].getLabel().equals(cases[6].getLabel())) {
        leGagnant = cases[2].getLabel();
        montrerGagnant(2, 4, 6);
    } else if (casesLibresRestantes == 0) {

```

Nouvelle version du morpion (ex3)

```

    return "T"; // Partie nulle
  }
  return leGagnant;
}
/**
 * Cette methode applique un ensemble de regles afin de
 * trouver le meilleur coup pour l'ordinateur. Si un bon
 * coup ne peut etre trouve, elle choisit une case au
 * hasard.
 */
void coupOrdinateur() {
  int caseSelectionnee;
  //L'ordinateur essaie d'abord de trouver une case
  //vide pres de deux case marquees "O" pour gagner
  caseSelectionnee = trouverCaseVide("O");
  //S'il n'y a pas deux "O" alignes, essaie au moins
  //d'empecher l'adversaire d'aligner trois "X" en
  //plaçant un "O" pres de deux "X"
  if (caseSelectionnee == -1) {
    caseSelectionnee = trouverCaseVide("X");
  }
  //Si caseSelectionnee vaut toujours -1, essaie d'occuper
  //la case centrale
  if ((caseSelectionnee == -1)
      && (cases[4].getLabel().equals(""))) {
    caseSelectionnee = 4;
  }
  //Pas de chance avec la case centrale non plus...
  //Choisit une case au hasard
  if (caseSelectionnee == -1) {
    caseSelectionnee = choisirCaseAuHasard();
  }
  cases[caseSelectionnee].setLabel("O");
}
/**
 * Cette methode examine chaque ligne, colonne et diagonale
 * pour voir si elle contient deux cases avec le meme label
 * et une case vide.
 * @param joueur "X" pour l'utilisateur ou "O" pour
 *
l'ordinateur
 * @return numero de la case vide a utiliser ou le nombre
 *
negatif -1 si la recherche est infructueuse
 */
int trouverCaseVide(String joueur) {
  int poids[] = new int[9];
  for (int i = 0; i < 9; i++) {
    if (cases[i].getLabel().equals("O"))
      poids[i] = -1;
    else if (cases[i].getLabel().equals("X"))
      poids[i] = 1;
    else
      poids[i] = 0;
  }
  int deuxPoids = joueur.equals("O") ? -2 : 2;
  // Regarde si la ligne 1 a 2 cases identiques et une vide

  if (poids[0] + poids[1] + poids[2] == deuxPoids) {

    if (poids[0] == 0)
      return 0;
    else if (poids[1] == 0)
      return 1;
    else
      return 2;
  }
  // Regarde si la ligne 2 a 2 cases identiques et une vide

  if (poids[3] + poids[4] + poids[5] == deuxPoids) {

```

Nouvelle version du morpion (ex3)

```

    if (poids[3] == 0)
        return 3;
    else if (poids[4] == 0)
        return 4;
    else
        return 5;
}
//Regarde si la ligne 3 a 2 cases identiques et une vide
if (poids[6] + poids[7] + poids[8] == deuxPoids) {
    if (poids[6] == 0)
        return 6;
    else if (poids[7] == 0)
        return 7;
    else
        return 8;
}
//Regarde si la colonne 1 a 2 cases identiques et une vide
if (poids[0] + poids[3] + poids[6] == deuxPoids) {
    if (poids[0] == 0)
        return 0;
    else if (poids[3] == 0)
        return 3;
    else
        return 6;
}
//Regarde si la colonne 2 a 2 cases identiques et une vide
if (poids[1] + poids[4] + poids[7] == deuxPoids) {
    if (poids[1] == 0)
        return 1;
    else if (poids[4] == 0)
        return 4;
    else
        return 7;
}
//Regarde si la colonne 3 a 2 cases identiques et une vide
if (poids[2] + poids[5] + poids[8] == deuxPoids) {
    if (poids[2] == 0)
        return 2;
    else if (poids[5] == 0)
        return 5;
    else
        return 8;
}
//Regarde si la diagonale 1 a 2 cases identiques et une
//vide
if (poids[0] + poids[4] + poids[8] == deuxPoids) {
    if (poids[0] == 0)
        return 0;
    else if (poids[4] == 0)
        return 4;
    else
        return 8;
}
//Regarde si la diagonale 2 a 2 cases identiques et une
//vide
if (poids[2] + poids[4] + poids[6] == deuxPoids) {
    if (poids[2] == 0)
        return 2;
    else if (poids[4] == 0)
        return 4;
    else
        return 6;
}
//Il n'y a pas de cases alignées identiques
return -1;
} // Fin de trouverCaseVide()
/**
 * Cette methode selectionne une case vide quelconque.

```


Nouvelle version du morpion (ex3)

```

* @return un numero de case choisi au hasard
*/
int choisirCaseAuHasard() {
    boolean caseVideTrouvee = false;
    int caseSelectionnee = -1;
    do {
        caseSelectionnee = (int) (Math.random() * 9);
        if (cases[caseSelectionnee].getLabel().equals("")) {
            caseVideTrouvee = true; // Pour terminer la boucle
        }
    } while (!caseVideTrouvee);
    return caseSelectionnee;
} // Fin de choisirCaseAuHasard()
/**
 * Cette methode affiche la ligne gagnante en surbrillance.
 * @param gagnante1 premiere case a montrer.
 * @param gagnante2 deuxieme case a montrer.
 * @param gagnante3 troisieme case a montrer.
 */
void montrerGagnant(int gagnante1, int gagnante2, int
    gagnante3) {
    cases[gagnante1].setBackground(Color.CYAN);
    cases[gagnante2].setBackground(Color.CYAN);
    cases[gagnante3].setBackground(Color.CYAN);
}
//Desactive les cases et active le bouton Nouvelle partie
void terminerLaPartie() {
    boutonNouvellePartie.setEnabled(true);
    for (int i = 0; i < 9; i++) {
        cases[i].setEnabled(false);
    }
}

// Construit le panneau qui contiendra les controles
void construireContainer(Container container){
    // Affecte le gestionnaire de disposition et la couleur
    // de l'applet
    container.setLayout(new BorderLayout());
    container.setBackground(Color.CYAN);
    // Passe la police de l'applet en style gras et taille 20
    Font policeApplet = new Font("Monospaced", Font.BOLD, 20);
    container.setFont(policeApplet);
    // Cree le bouton Nouvelle partie et enregistre
    // le recepteur d'actions aupres de lui
    boutonNouvellePartie = new Button("Nouvelle partie");
    boutonNouvellePartie.addActionListener(this);
    // Cree deux panneaux et un label et les agence en
    // utilisant le border layout
    Panel panneauSuperieur = new Panel();
    panneauSuperieur.setLayout(new GridLayout(3,1));
    // Cree les deux labels victoires/defaites
    labelVictoires = new Label("Nombre de victoires : 0");
    labelDefaites = new Label("Nombre de defaites : 0");
    // Ajoute les composants au panneau superieur
    panneauSuperieur.add(labelVictoires);
    panneauSuperieur.add(labelDefaites);
    panneauSuperieur.add(boutonNouvellePartie);
    container.add(panneauSuperieur, "North");
    Panel panneauCentral = new Panel();
    panneauCentral.setLayout(new GridLayout(3, 3));
    container.add(panneauCentral, "Center");
    score = new Label("A vous de jouer !");
    container.add(score, "South");
    // Cree un tableau pour stocker les references des
    // 9 boutons
    cases = new Button[9];
    // Instancie les boutons, stocke leurs references dans le
    // tableau, enregistre le recepteur aupres d'eux, peint
    // les boutons en orange et les ajoute au panneau central
    for(int i = 0; i < 9; i++) {

```

Nouvelle version du morpion (ex3)

```

        cases[i] = new Button();
        cases[i].addActionListener(this);
        cases[i].setBackground(Color.ORANGE);
        panneauCentral.add(cases[i]);
    }
}

public static void main(String[] args) {
    Frame fenetre = new Frame();
    /*
     * Cette fois-ci, c'est le code qui
     * fixe les dimensions de la zone de jeu,
     * et non une page html.
     */
    fenetre.setSize(400, 300);
    /*
     * Cette fois-ci, le titre se definit dans la fenetre.
     */
    fenetre.setTitle("Jeu du morpion");
    /*
     * Centre la fenetre sur l'ecran
     */
    fenetre.setLocationRelativeTo(null);
    Panel panneauFenetre = new Panel();
    fenetre.add(panneauFenetre);

    Morpion_ex3 logiqueMorpion = new Morpion_ex3();
    logiqueMorpion.construireContainer(panneauFenetre);

    fenetre.addWindowListener(new WindowAdapter() {

        @Override
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }

    });

    fenetre.setVisible(true);
}
} // Fin de la classe

```

Voici ce que le programme peut donner en version fenêtre :



Version fenêtre

VI-D - Exercice pour les petits malins

VI-D-1 - Les tableaux multidimensionnels

VI-D-1-a - Théorie

Jusqu'ici vous n'avez utilisé que les tableaux à une seule dimension, par exemple :

```
int monTableau[] = new int[6] ;
int monTableau[] = {2,4,6,8,10,12} ;
```

Les deux tableaux précédents ont la même dimension.

Mais en fait, un tableau peut avoir, en théorie, une infinité de dimensions :

```
int monTableau3D[][][] = new int [2][6][3] ;
int monTableau3D[][][] = {
{ {5,1,12},{2,-6,18},{8,0,-54},{1,-5,-67},{3,10,20},{0,78,27} },
{ {23,1,134},{25,6,12},{99,6,54},{19,100,-67},{37,-2,18},{5,72,82} }
};
```

Ce sont deux tableaux à trois dimensions et presque équivalents : c'est juste que le premier tableau n'est rempli qu'avec des zéros.

Comment alors parcourir un tel tableau ? Il suffit d'utiliser plusieurs **boucles for** imbriquées : la première boucle for pour parcourir la première dimension, la deuxième pour parcourir la deuxième dimension et la dernière pour parcourir la dernière dimension :

```
for (int dim1 = 0 ; dim1 < monTableau3d.length ; dim1++){
```

```

for (int dim2 = 0 ; dim2 < monTableau3D[dim1].length ; dim2++){
    for (int dim3 = 0 ; dim3 < monTableau3D[dim1][dim2].length ; dim3++){
        System.out.println(monTableau3D[dim1][dim2][dim3]) ;
    }
}
}

```

Certains auront sans doute remarqué comment j'ai consulté la longueur de chaque dimension dans les différentes boucles :

```

monTableau3d.length // pour la première dimension
monTableau3D[dim1].length // pour la deuxième dimension
monTableau3D[dim1][dim2].length // pour la troisième dimension

```

En effet, il faut faire attention à deux choses :

- si l'on écrit juste `monTableau3d.length`, on n'obtient que la longueur de la première dimension ;
- on réutilise les variables `dim1`, `dim2` car il se peut que le tableau ne soit pas de dimension **uniforme** (régulière).

Pour vous en convaincre, essayez de déclarer le tableau comme ceci :

```

int [][][] monTableau3D = {
    { {1,2,3}, {4}, {5,6,7,8}, {9,10} },
    { {11}, {12,13}, {14,15,16}, {17,18, 19,20} }
};

```

Testez-le aussi avec l'extrait de code avec les trois boucles imbriquées, que j'ai donné peu avant. Vous constaterez alors que :

- il n'y a aucune erreur à la compilation (depuis Eclipse, rien ne nous est signalé avant le test) ;
- il n'y a aucune erreur à l'exécution et on a bien l'ensemble des valeurs affichées, dans l'ordre.

Le tableau que vous avez testé est un tableau multidimensionnel légal et la triple boucle imbriquée que je vous ai introduit est tout à fait adaptée pour le tester. Si on ne se basait pas sur la valeur courante de chaque dimension lors des différents tests de longueur des dimensions, on pourrait alors facilement aboutir à une **erreur d'exécution** : **ArrayOutOfBoundsException**.

Donc ceci :

```

for (int dim1 = 0 ; dim1 < monTableau3d.length ; dim1++){
    for (int dim2 = 0 ; dim2 < monTableau3D[0].length ; dim2++){
        for (int dim3 = 0 ; dim3 < monTableau3D[0][0].length ; dim3++){
            System.out.println(monTableau3D[dim1][dim2][dim3]) ;
        }
    }
}

```

est à proscrire pour le tableau irrégulier que je viens de vous donner.

On nous demande ici d'utiliser un tableau à deux dimensions, ce qui ne devrait pas poser plus de problèmes que pour le tableau à trois dimensions que l'on vient de voir.

Enfin, une dernière difficulté technique que l'on se doit de résoudre. La section du code original suivante :

```

// Regarde si la ligne 2 a 2 cases identiques et une vide

if (poids[3] + poids[4] + poids[5] == deuxPoids) {

    if (poids[3] == 0)

```

```

        return 3;
    else if (poids[4] == 0)
        return 4;
    else
        return 5;
}

```

peut nous poser problème. On passe en effet d'un tableau où les neuf cases sont alignées à un tableau à deux dimensions. Il faut donc trouver un moyen de convertir ("à la main") tous les indices qui étaient prévus pour un tableau à une dimension, en une paire d'indices pour notre nouveau tableau.

VI-D-1-b - Mise en pratique

La première question à se poser : la première dimension représentera-t-elle les lignes ou les colonnes ? Pour le savoir, il faut se baser sur la manière dont les boutons sont ajoutés au conteneur. Le panneauCentral, sur lequel sont ajoutés les boutons, a un agencement défini par un GridLayout de trois lignes par trois colonnes.

Les boutons seront donc ajoutés colonne par colonne, en passant à la ligne suivante au besoin. La boucle for la plus interne devrait donc être consacrée au parcours de colonnes. On en déduit donc qu'il sera plus commode d'attribuer les lignes à la première dimension (revoyez au besoin le code de parcours du tableau à trois dimensions : la boucle for la plus externe examine la première dimension du tableau).

La deuxième question à se poser : comment obtenir la ligne et la colonne à partir de l'index initial ?

On peut remarquer que :

```

colonne = index % 3 ; // reste de la division entière de index par 3
ligne = index / 3 ; // quotient de la division index / 3 => c'est un entier
en assumant qu'index est une variable de type int.

```

Par exemple, pour l'index 5 :

```

colonne = 5 % 3 = 2
ligne = 5 / 3 = 1

```

Au lieu d'écrire cases[5], on écrira donc cases[1][2]

VI-D-2 - Code résultat



Nouvelle version du morpion (exMalins)

```

/**
 * Un jeu de morpion sur un plateau 3x3
 */
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class Morpion_exMalins extends Applet implements
ActionListener {
    Button cases[][];
    Button boutonNouvellePartie;
    Label score;
    Label labelVictoires;
    Label labelDefaites;

    int casesLibresRestantes = 9;
    int nombreVictoires = 0;
    int nombreDefaites = 0;

    public static void main(String[] args) {

```

Nouvelle version du morpion (exMalins)

```

Frame fenetre = new Frame();
/*
 * Cette fois-ci, c'est le code qui
 * fixe les dimensions de la zone de jeu,
 * et non une page html.
 */
fenetre.setSize(400, 300);
/*
 * Cette fois-ci, le titre se definit dans la fenetre.
 */
fenetre.setTitle("Jeu du morpion");
/*
 * Centre la fenetre sur l'ecran
 */
fenetre.setLocationRelativeTo(null);
Panel panneauFenetre = new Panel();
fenetre.add(panneauFenetre);

Morpion_exMalins logiqueMorpion = new Morpion_exMalins();
logiqueMorpion.construireContainer(panneauFenetre);

fenetre.addWindowListener(new WindowAdapter() {

    @Override
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }

});

fenetre.setVisible(true);
}

/**
 * La methode init est comme un constructeur pour l'applet
 */
public void init() {
    construireContainer(this);
}

/**
 * Cette methode traite tous les evenements d'action
 * @param evenement l'evenement a traiter
 */
public void actionPerformed(ActionEvent evenement) {
    Button leBouton = (Button) evenement.getSource();
    // S'agit-il du bouton Nouvelle partie ?
    if (leBouton == boutonNouvellePartie) {
        for(int dim1 = 0; dim1 < 3; dim1++) {
            for (int dim2 = 0; dim2 < 3; dim2++){
                cases[dim1][dim2].setEnabled(true);
                cases[dim1][dim2].setLabel("");
                cases[dim1][dim2].setBackground(Color.ORANGE);
            }
        }
        casesLibresRestantes = 9;
        score.setText("A vous de jouer !");
        boutonNouvellePartie.setEnabled(false);
        return; // Sort de la methode
    }
    String gagnant = "";
    //S'agit-il de l'une des cases ?
    for(int dim1 = 0; dim1 < 3; dim1++) {
        for (int dim2 = 0; dim2 < 3; dim2++){
            if (leBouton == cases[dim1][dim2] && cases[dim1][dim2].getLabel().equals("")) {
                cases[dim1][dim2].setLabel("X");
                gagnant = chercherUnGagnant();
                if(!"".equals(gagnant)) {
                    terminerLaPartie();
                } else {
                    coupOrdinateur();
                    gagnant = chercherUnGagnant();
                }
            }
        }
    }
}

```

Nouvelle version du morpion (exMalins)

```

        if (!"".equals(gagnant)) {
            terminerLaPartie();
        }
    }
    break;
}

} // Fin de la boucle for
if (gagnant.equals("X")) {
    score.setText("Vous avez gagne !");
    nombreVictoires++;
    labelVictoires.setText("Nombre de victoires : "+nombreVictoires);
} else if (gagnant.equals("O")) {
    score.setText("Vous avez perdu !");
    nombreDefaites++;
    labelDefaites.setText("Nombre de defaites : "+nombreDefaites);
} else if (gagnant.equals("T")) {
    score.setText("Partie nulle !");
}
} // Fin de actionPerformed
/**
 * Cette methode est appelee apres chaque coup joue pour
 * voir s'il y a un gagnant. Elle verifie pour chaque ligne,
 * colonne et diagonale, s'il y a trois symboles identiques
 * @return "X", "O", "T" (termine, partie nulle) ou "" (pas
 * fini)
 */
String chercherUnGagnant() {
    String leGagnant = "";
    casesLibresRestantes--;
    //Verifie la ligne 1 - elements 0, 1 et 2 du tableau
    if (!cases[0][0].getLabel().equals("") &&
        cases[0][0].getLabel().equals(cases[0][1].getLabel()) &&
        cases[0][0].getLabel().equals(cases[0][2].getLabel())) {
        leGagnant = cases[0][0].getLabel();
        montrerGagnant(0, 1, 2);
        //Verifie la ligne 2 - elements 3, 4 et 5 du tableau
    } else if (!cases[1][0].getLabel().equals("") &&
        cases[1][0].getLabel().equals(cases[1][1].getLabel()) &&
        cases[1][0].getLabel().equals(cases[1][2].getLabel())) {
        leGagnant = cases[1][0].getLabel();
        montrerGagnant(3, 4, 5);
        //Verifie la ligne 3 - elements 6, 7 et 8 du tableau
    } else if (!cases[2][0].getLabel().equals("") &&
        cases[2][0].getLabel().equals(cases[2][1].getLabel()) &&
        cases[2][0].getLabel().equals(cases[2][2].getLabel())) {
        leGagnant = cases[2][0].getLabel();
        montrerGagnant(6, 7, 8);
        //Verifie la colonne 1 - elements 0, 3 et 6 du tableau
    } else if (!cases[0][0].getLabel().equals("") &&
        cases[0][0].getLabel().equals(cases[1][0].getLabel()) &&
        cases[0][0].getLabel().equals(cases[2][0].getLabel())) {
        leGagnant = cases[0][0].getLabel();
        montrerGagnant(0, 3, 6);
        //Verifie la colonne 2 - elements 1, 4 et 7 du tableau
    } else if (!cases[0][1].getLabel().equals("") &&
        cases[0][1].getLabel().equals(cases[1][1].getLabel()) &&
        cases[0][1].getLabel().equals(cases[2][1].getLabel())) {
        leGagnant = cases[0][1].getLabel();
        montrerGagnant(1, 4, 7);
        //Verifie la colonne 3 - elements 2, 5 et 8 du tableau
    } else if (!cases[0][2].getLabel().equals("") &&
        cases[0][2].getLabel().equals(cases[1][2].getLabel()) &&
        cases[0][2].getLabel().equals(cases[2][2].getLabel())) {
        leGagnant = cases[0][2].getLabel();
        montrerGagnant(2, 5, 8);
        //Verifie la premiere diagonale - elements 0, 4 et 8
    } else if (!cases[0][0].getLabel().equals("") &&
        cases[0][0].getLabel().equals(cases[1][1].getLabel()) &&
        cases[0][0].getLabel().equals(cases[2][2].getLabel())) {
        leGagnant = cases[0][0].getLabel();
    }
}

```

Nouvelle version du morpion (exMalins)

```

    montrerGagnant(0, 4, 8);
    //Verifie la seconde diagonale - elements 2, 4 et 6
} else if (!cases[0][2].getLabel().equals("")) &&
    cases[0][2].getLabel().equals(cases[1][1].getLabel()) &&
    cases[0][2].getLabel().equals(cases[2][0].getLabel())) {
    leGagnant = cases[0][2].getLabel();
    montrerGagnant(2, 4, 6);
} else if (casesLibresRestantes == 0) {
    return "T"; // Partie nulle
}
return leGagnant;
}
/**
 * Cette methode applique un ensemble de regles afin de
 * trouver le meilleur coup pour l'ordinateur. Si un bon
 * coup ne peut etre trouve, elle choisit une case au
 * hasard.
 */
void coupOrdinateur() {
    int caseSelectionnee;
    //L'ordinateur essaie d'abord de trouver une case
    //vide pres de deux case marquees "O" pour gagner
    caseSelectionnee = trouverCaseVide("O");
    //S'il n'y a pas deux "O" alignes, essaie au moins
    //d'empecher l'adversaire d'aligner trois "X" en
    //plaçant un "O" pres de deux "X"
    if (caseSelectionnee == -1) {
        caseSelectionnee = trouverCaseVide("X");
    }
    //Si caseSelectionnee vaut toujours -1, essaie d'occuper
    //la case centrale
    if ((caseSelectionnee == -1)
        && (cases[1][1].getLabel().equals(""))) {
        caseSelectionnee = 4;
    }
    //Pas de chance avec la case centrale non plus...
    //Choisit une case au hasard
    if (caseSelectionnee == -1) {
        caseSelectionnee = choisirCaseAuHasard();
    }
    cases[caseSelectionnee/3][caseSelectionnee%3].setLabel("O");
}
/**
 * Cette methode examine chaque ligne, colonne et diagonale
 * pour voir si elle contient deux cases avec le meme label
 * et une case vide.
 * @param joueur "X" pour l'utilisateur ou "O" pour
 * l'ordinateur
 * @return numero de la case vide a utiliser ou le nombre
 * negatif -1 si la recherche est infructueuse
 */
int trouverCaseVide(String joueur) {
    int poids[][] = new int[3][3];
    for (int dim1 = 0; dim1 < 3; dim1++) {
        for (int dim2 = 0; dim2 < 3; dim2++) {
            if (cases[dim1][dim2].getLabel().equals("O"))
                poids[dim1][dim2] = -1;
            else if (cases[dim1][dim2].getLabel().equals("X"))
                poids[dim1][dim2] = 1;
            else
                poids[dim1][dim2] = 0;
        }
    }
    int deuxPoids = joueur.equals("O") ? -2 : 2;
    // Regarde si la ligne 1 a 2 cases identiques et une vide

    if (poids[0][0] + poids[0][1] + poids[0][2] == deuxPoids) {

        if (poids[0][0] == 0)

```


Nouvelle version du morpion (exMalins)

```

    return 0;
  else if (poids[0][1] == 0)
    return 1;
  else
    return 2;
}
// Regarde si la ligne 2 a 2 cases identiques et une vide
if (poids[1][0] + poids[1][1] + poids[1][2] == deuxPoids) {
  if (poids[1][0] == 0)
    return 3;
  else if (poids[1][1] == 0)
    return 4;
  else
    return 5;
}
//Regarde si la ligne 3 a 2 cases identiques et une vide
if (poids[2][0] + poids[2][1] + poids[2][2] == deuxPoids) {
  if (poids[2][0] == 0)
    return 6;
  else if (poids[2][1] == 0)
    return 7;
  else
    return 8;
}
//Regarde si la colonne 1 a 2 cases identiques et une vide
if (poids[0][0] + poids[1][0] + poids[2][0] == deuxPoids) {
  if (poids[0][0] == 0)
    return 0;
  else if (poids[1][0] == 0)
    return 3;
  else
    return 6;
}
//Regarde si la colonne 2 a 2 cases identiques et une vide
if (poids[0][1] + poids[1][1] + poids[2][1] == deuxPoids) {
  if (poids[0][1] == 0)
    return 1;
  else if (poids[1][1] == 0)
    return 4;
  else
    return 7;
}
//Regarde si la colonne 3 a 2 cases identiques et une vide
if (poids[0][2] + poids[1][2] + poids[2][2] == deuxPoids) {
  if (poids[0][2] == 0)
    return 2;
  else if (poids[1][2] == 0)
    return 5;
  else
    return 8;
}
//Regarde si la diagonale 1 a 2 cases identiques et une
//vide
if (poids[0][0] + poids[1][1] + poids[2][2] == deuxPoids) {
  if (poids[0][0] == 0)
    return 0;
  else if (poids[1][1] == 0)
    return 4;
  else
    return 8;
}
//Regarde si la diagonale 2 a 2 cases identiques et une
//vide
if (poids[0][2] + poids[1][1] + poids[2][0] == deuxPoids) {
  if (poids[0][2] == 0)
    return 2;
  else if (poids[1][1] == 0)

```

Nouvelle version du morpion (exMalins)

```

        return 4;
    }
    else
        return 6;
    }
    //Il n'y a pas de cases alignées identiques
    return -1;
} // Fin de trouverCaseVide()
/**
 * Cette methode selectionne une case vide quelconque.
 * @return un numero de case choisi au hasard
 */
int choisirCaseAuHasard() {
    boolean caseVideTrouvee = false;
    int caseSelectionnee = -1;
    do {
        caseSelectionnee = (int) (Math.random() * 9);
        if (cases[caseSelectionnee/3][caseSelectionnee%3].getLabel().equals("")) {
            caseVideTrouvee = true; // Pour terminer la boucle
        }
    } while (!caseVideTrouvee);
    return caseSelectionnee;
} // Fin de choisirCaseAuHasard()
/**
 * Cette methode affiche la ligne gagnante en surbrillance.
 * @param gagnante1 premiere case a montrer.
 * @param gagnante2 deuxieme case a montrer.
 * @param gagnante3 troisieme case a montrer.
 */
void montrerGagnant(int gagnante1, int gagnante2, int
    gagnante3) {
    cases[gagnante1/3][gagnante1%3].setBackground(Color.CYAN);
    cases[gagnante2/3][gagnante2%3].setBackground(Color.CYAN);
    cases[gagnante3/3][gagnante3%3].setBackground(Color.CYAN);
}
//Desactive les cases et active le bouton Nouvelle partie
void terminerLaPartie() {
    boutonNouvellePartie.setEnabled(true);
    for (int dim1 = 0; dim1 < 3; dim1++) {
        for (int dim2 = 0; dim2 < 3; dim2++) {
            cases[dim1][dim2].setEnabled(false);
        }
    }
}

void construireContainer(Container container){
    // Affecte le gestionnaire de disposition et la couleur
    // de l'applet
    container.setLayout(new BorderLayout());
    container.setBackground(Color.CYAN);
    // Passe la police de l'applet en style gras et taille 20
    Font policeApplet = new Font("Monospaced", Font.BOLD, 20);
    container.setFont(policeApplet);
    // Cree le bouton Nouvelle partie et enregistre
    // le recepteur d'actions aupres de lui
    boutonNouvellePartie = new Button("Nouvelle partie");
    boutonNouvellePartie.addActionListener(this);
    // Cree deux panneaux et un label et les agence en
    // utilisant le border layout
    Panel panneauSuperieur = new Panel();
    panneauSuperieur.setLayout(new GridLayout(3,1));
    // Cree les deux labels victoires/defaites
    labelVictoires = new Label("Nombre de victoires : 0");
    labelDefaites = new Label("Nombre de defaites : 0");
    // Ajoute les composants au panneau superieur
    panneauSuperieur.add(labelVictoires);
    panneauSuperieur.add(labelDefaites);
    panneauSuperieur.add(boutonNouvellePartie);
    container.add(panneauSuperieur, "North");
    Panel panneauCentral = new Panel();
    panneauCentral.setLayout(new GridLayout(3, 3));

```

Nouvelle version du morpion (exMalins)

```
container.add(panneauCentral, "Center");
score = new Label("A vous de jouer !");
container.add(score, "South");
// Cree un tableau pour stocker les references des
// 9 boutons
cases = new Button[3][3];
// Instancie les boutons, stocke leurs references dans le
// tableau, enregistre le recepteur aupres d'eux, peint
// les boutons en orange et les ajoute au panneau central
for(int ligne = 0; ligne < 3; ligne++) {
    for (int col = 0; col < 3; col++) {
        cases[ligne][col] = new Button();
        cases[ligne][col].addActionListener(this);
        cases[ligne][col].setBackground(Color.ORANGE);
        panneauCentral.add(cases[ligne][col]);
    }
}
} // Fin de la classe
```

VI-E - Synthèse

Ces exercices ont pu vous apprendre :

- à ajouter des composants dans une interface AWT et les mettre régulièrement à jour grâce à l'utilisation de variables de classe ;
- à résoudre simplement un bogue ;
- à rendre une applet utilisable en mode application, en y ajoutant une méthode main(). Mais aussi à adapter au besoin le code ;
- à utiliser un tableau à deux dimensions, sans obtenir d'erreur à l'exécution.

VII - Chapitre 8

VII-A - Application commande de vélos

On dispose déjà des classes EcranCommande et TropDeVelosException. Utilisez la fonction rechercher/remplacer de l'éditeur, pour "retirer" tous les accents dans les deux classes.

La classe EcranCommande doit être complétée :

- pour y déclarer les composants (variables de classe) et les construire (dans le constructeur) ;
- pour vérifier, dans la méthode actionPerformed(), que la quantité commandée ne dépasse pas la quantité disponible.

VII-A-1 - Nouvelle version de TropDeVelosException

Mais avant cela, il faut d'abord remarquer que l'on a besoin d'un constructeur prenant un paramètre de type String, dans la classe TropDeVelosException. Pourquoi ? À cause de la ligne suivante de la classe EcranCommande (cherchez dans la méthode verifierCommande()) :

```
throw new TropDeVelosException ("Impossible de livrer " + quantite + " vélos du modele " +
modeleVelo + " en une fois.");
```

On remarque que:

- c'est un appel au constructeur de la classe TropDeVelosException. En effet, on utilise le mot-clé new ;
- le paramètre passé est une chaîne de caractères (obtenue par une **concaténation** : pour rappel, une chaîne construite avec un mélange de chaînes et de **types primitifs**. Et ce grâce à l'**opérateur "+"**). Or ce constructeur n'existe pas encore dans la classe TropDeVelosException.

De plus, la classe mère Exception de TropDeVelosException dispose d'un constructeur acceptant un String.

Voici donc une nouvelle version de la classe TropDeVelosException :



Nouvelle version de TropDeVelosException

```
class TropDeVelosException extends Exception {
    // Constructeur
    TropDeVelosException() {
        // Appelle simplement le constructeur de la superclasse
        // et lui passe le message d'erreur à afficher
        super("Impossible de livrer autant de vélos en une fois.");
    }

    TropDeVelosException(String message) {
        /*
         * Initialise l'objet avec le message desire,
         * grace au constructeur de la classe mere.
         */
        super(message);
    }
}
```

VII-A-2 - Correction d'une légère anomalie dans le code original de EcranCommande

Dans la classe EcranCommande, méthode actionPerformed(), cette ligne comporte encore deux légères erreurs :

```
commandeVelos.verifierCommande(modeleChoisi, quantiteChoisie);
```

En effet :

=> la méthode `verifierCommande` est déjà présente dans la classe `EcranCommande`. Aussi on n'a pas besoin d'appeler :

```
commandeVelos.verifierCommande(modeleChoisi, quantiteChoisie);
```

mais on peut tout simplement se contenter de :

```
verifierCommande(modeleChoisi, quantiteChoisie);
```

=> le deuxième paramètre de la méthode `verifierCommande` devant être un `int`, ce n'est pas la valeur de `quantiteChoisie` (`String`) qui doit être passée, mais le paramètre `quantite` (`int`).

On corrige donc par la simple ligne :

```
verifierCommande(modeleChoisi, quantite);
```

VII-A-3 - EcranCommande sera une fenêtre Swing

Il nous est demandé de réaliser une application Swing. On peut faire hériter la classe `EcranCommande` de `javax.swing.JFrame` et procéder à quelques réglages (modification du constructeur sans arguments) :

Première ébauche pour EcranCommandes

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JFrame;

class EcranCommande extends JFrame implements ActionListener {

    public EcranCommande() {
        // titre
        setTitle("Commande de velos");
        // taille
        setSize(400, 300);
        // centrer la fenetre sur l'ecran
        setLocationRelativeTo(null);
        // on quitte l'application si la fenetre est fermee
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent evt) {
        // L'utilisateur a clique sur le bouton Commander
        String modeleChoisi = champTexteModele.getText();
        String quantiteChoisie = champTexteQuantite.getText();
        int quantite = Integer.parseInt(quantiteChoisie);
        try {
            commandeVelos.verifierCommande(modeleChoisi,
                quantiteChoisie);
            // Cette ligne sera sautee en cas d'exception
            champTexteConfirmationCommande.setText(
                "Votre commande est prete.");
        } catch (TropDeVelosException e) {
            champTexteConfirmationCommande.setText(e.getMessage());
        }
    }

    void verifierCommande(String modeleVelo, int quantite)
        throws TropDeVelosException {
        // Ecris le code qui verifie que la quantite demandee
        // du modele selectionne entre bien dans la camionnette.
    }
}
```

Première ébauche pour EcranCommandes

```
// Si ça n'entre pas, faire ce qui suit :
throw new TropDeVelosException ("Impossible de livrer " +
    quantite + " velos du modele " + modeleVelo +
    " en une fois.");
}
```

N'oubliez pas que la définition des interfaces implémentées vient après la définition de l'éventuelle classe mère. Donc, lorsque l'on déclare une classe, on précise toujours d'abord la classe mère (par le mot-clé `extends`) avant les interfaces implémentées (par le mot-clé `implements`).

VII-A-4 - Déclaration et construction des composants de la classe EcranCommande

On devine aisément, d'après le code EcranCommande, quels sont les composants à déclarer :

- un JTextField nommé champTexteModele et donc également un JLabel identifiant le champ ;
- un JTextField nommé champTexteQuantite et donc un JLabel l'identifiant ;
- un JTextField nommé champTexteConfirmationCommande et donc un JLabel l'identifiant.

Mais il faudra également créer un bouton permettant à l'utilisateur de soumettre sa commande.

On ajoutera l'ensemble de ces composants à un JPanel, qui sera intégré à la fenêtre. Pour agencer les composants, j'ai choisi un GridLayout de quatre lignes par deux colonnes ; avec un espacement de cinq pixels entre les composants (en largeur : troisième paramètre du constructeur et en hauteur : quatrième paramètre du constructeur).

Enfin, il faut donner au bouton son écouteur d'événements : on peut lui attribuer l'instance courante de EcranCommande, à savoir **this**. La classe EcranCommande implémente déjà l'interface ActionListener et sa méthode actionPerformed est déjà prévue pour exécuter le code de validation/annulation de commande.

VII-A-5 - Vérification de la commande et la nouvelle classe ModeleVelo

La méthode verifierCommande est incomplète : elle se contente de renvoyer une exception TropDeVelosException, sans même vérifier si la commande est valable.

À chaque commande, il faut vérifier :

- si le modèle existe ;
- si la quantité commandée pour ce modèle n'excède pas la quantité maximale autorisée.

Pour procéder, on peut créer une classe ModeleVelo. Elle servira à nous renseigner à la fois sur le nom et la quantité maximale d'un modèle :

Première ébauche de ModeleVelo

```
public class ModeleVelo {

    String model;
    int quantiteMaximumCommandable;

    public ModeleVelo(String model,    int quantiteMaximumCommandable) {
        this.model = model;
        this.quantiteMaximumCommandable = quantiteMaximumCommandable;
    }

}
```

On crée alors un tableau de type `ModeleVelo` (eh oui, vous en avez le droit car maintenant `ModeleVelo` est un type reconnu pour votre programme.) dans la classe `EcranCommande` :

```
ModeleVelo modelesDisponibles[] = {
    new ModeleVelo("BikeMike 500", 5),
    new ModeleVelo("RideAMax 130", 2),
    new ModeleVelo("BigRace 200", 8),
    new ModeleVelo("TrickyCourt 820", 7),
    new ModeleVelo("MaxiRide 20", 3),
    new ModeleVelo("RidersHeaven 760", 9),
};
```

Cette construction vous semble bizarre ? Pourtant elle est correcte : souvenez-vous que vous n'êtes pas obligés de conserver une référence sur l'objet créé avec `new`. Ici le tableau, dont on a conservé une référence, s'en chargera pour nous.

Pour tester si un modèle fait partie de la liste, il suffira donc d'exécuter une boucle `for` (ou `for each`) sur tout le tableau :

```
String modeleRecherche = "BikeMike 500";
for (ModeleVelo modeleCourant : modelesDisponibles) {
    if (modeleRecherche.equals(modeleCourant.modele)) {
        /*
         * Faire ce que l'on veut faire avec modeleCourant :
         * c'est l'objet ModeleVelo correspondant au nom
         * de modele recherche.
         */
    }
}
```

Enfin, il ne faut pas oublier d'ajouter une méthode `main()`. On y créera une instance d'`EcranCommande` et on l'affichera.

VII-A-6 - Code complet de la classe `EcranCommande`

CommandeVelosPremiereVersion

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

class EcranCommandePremiereVersion extends JFrame implements ActionListener {

    JTextField champTexteModele, champTexteQuantite,
    champTexteConfirmationCommande;
    JLabel labelModele, labelQuantite, labelConfirmationCommande;
    JButton boutonValidation;
    ModeleVelo modelesDisponibles[] = {
        /*
         * Nul besoin de stocker la reference
         * sur chaque objet du tableau.
         */
        new ModeleVelo("BikeMike 500", 5),
        new ModeleVelo("RideAMax 130", 2),
        new ModeleVelo("BigRace 200", 8),
        new ModeleVelo("TrickyCourt 820", 7),
        new ModeleVelo("MaxiRide 20", 3),
        new ModeleVelo("RidersHeaven 760", 9),
    }
```

CommandeVeloPremiereVersion

```
};

public EcranCommandePremiereVersion() {
    // titre
    setTitle("Commande de velos");
    // centrer la fenetre sur l'ecran
    setLocationRelativeTo(null);
    // on quitte l'application si la fenetre est fermee
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel panneauInterne = new JPanel();
    // La couleur de fond du panneau sera rose.
    panneauInterne.setBackground(Color.PINK);
    /*
     * Il y aura toujours 5 pixels d'espaces entre
     * les composants : que ce soit en largeur ou
     * en hauteur.
     */
    panneauInterne.setLayout(new GridLayout(4,2, 5,5));
    // Un JTextField avec 30 colonnes affichees.
    labelModele = new JLabel("Modele de velo : ");
    panneauInterne.add(labelModele);
    champTexteModele = new JTextField(30);
    panneauInterne.add(champTexteModele);
    labelQuantite = new JLabel("Quantite : ");
    panneauInterne.add(labelQuantite);
    champTexteQuantite = new JTextField(30);
    panneauInterne.add(champTexteQuantite);
    labelConfirmationCommande = new JLabel("Commande confirmee ? ");
    panneauInterne.add(labelConfirmationCommande);
    champTexteConfirmationCommande = new JTextField(30);
    panneauInterne.add(champTexteConfirmationCommande);
    boutonValidation = new JButton("Passer la commande");
    panneauInterne.add(boutonValidation);

    boutonValidation.addActionListener(this);

    add(panneauInterne);
    /*
     * La fenetre aura juste la taille
     * qui lui laisse placer tous ses composants :
     * donc la taille qui laisse le panneauInterne
     * placer tous ses composants.
     */
    pack();
}

public void actionPerformed(ActionEvent evt) {
    // L'utilisateur a clique sur le bouton Commander
    String modeleChoisi = champTexteModele.getText();
    String quantiteChoisie = champTexteQuantite.getText();
    int quantite = Integer.parseInt(quantiteChoisie);
    try {
        verifierCommande(modeleChoisi, quantite);
        // Cette ligne sera sautee en cas d'exception
        champTexteConfirmationCommande.setText(
            "Votre commande est prete.");
    } catch (TropDeVelosException e) {
        champTexteConfirmationCommande.setText(e.getMessage());
    } catch (Exception e) {
        champTexteConfirmationCommande.setText(e.getMessage());
    }
}

void verifierCommande(String modeleVelo, int quantite)
    throws TropDeVelosException,Exception {
    ModeleVelo modeleCorrespondant = null;
    for (ModeleVelo modeleCourant : modelesDisponibles) {
        if (modeleVelo.equals(modeleCourant.model)) {
            modeleCorrespondant = modeleCourant;
            break;
        }
    }
}
```


CommandeVelosPremiereVersion

```

    }
    if (modeleCorrespondant == null)
        throw new Exception("Le modele "+modeleVelo+" n'existe pas.");
    if (quantite > modeleCorrespondant.quantiteMaximumCommandable)
        throw new TropDeVelosException ("Impossible de livrer " + quantite + " velos du modele " +
modeleVelo + " en une fois.");
    }

    public static void main(String[] args) {
        /*
         * Pas besoin de stocker une reference
         * sur la fenetre cree : elle sera automatiquement
         * detruite en quittant l'application.
         */
        new EcranCommandePremiereVersion().setVisible(true);
    }
}

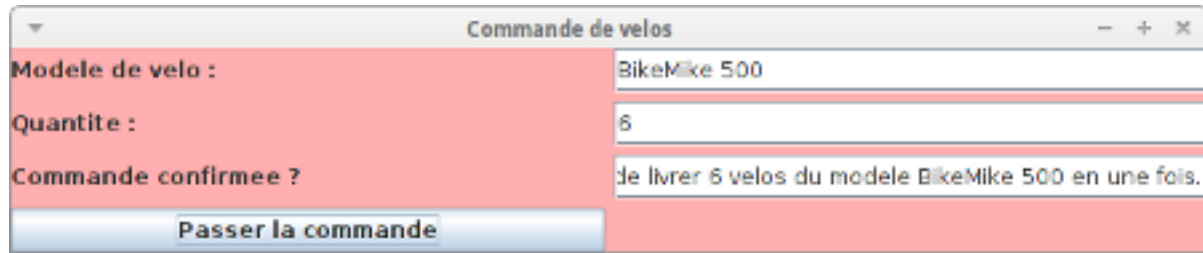
```

Voici un aperçu de la nouvelle interface :

Si le modèle de vélo entré n'est pas reconnu, le champ "Commande confirmée ?" indique "Le modèle de vélo n'existe pas."

Si le modèle de vélo entré est reconnu et la quantité commandée acceptable, l'interface renvoie le message : "Votre commande est prête."

Toutefois, si la commande excède la limite autorisée pour le modèle, un message d'erreur sera affiché. Un message du genre : " Impossible de livrer six vélos du modèle BikeMike 500 en une seule commande. ".



VII-B - Exercices pour les petits malins : liste déroulante

VII-B-1 - Construire l'objet JComboBox

Il y a plusieurs manières pour initialiser et remplir un JComboBox. On peut utiliser le constructeur par défaut et ensuite, remplir la liste en ajoutant les éléments un par un.

Mais il y a un moyen de réutiliser, presque directement, le tableau de ModeleVelo que nous avons employé précédemment. Pour cela j'utilise le constructeur :

```
JComboBox (ComboBoxModel aModel)
```

Ce constructeur va nous permettre de lui passer un DefaultComboBoxModel. Cette dernière classe nous permet simplement d'initialiser la liste avec un tableau d'Object (c'est-à-dire avec toute classe existante, car toute classe hérite de Object). Voici son constructeur :

```
DefaultComboBoxModel (Object[] items)
```

Pour créer et initialiser notre liste en même temps, il nous suffira donc d'écrire :

```
new JComboBox (new DefaultComboBoxModel (modelesDisponibles));
```

VII-B-2 - La méthode toString() de tout objet et nouvelle version de la classe ModeleVelo

Toute classe hérite de la classe Object. Et la classe Object définit un certain nombre de méthodes, qui peuvent donc être surchargées (souvenez-vous de la méthode dire() créée dans la classe Poisson du chapitre 3 : elle surcharge celle de la classe AnimalFamiliier).

Parmi elles, la méthode toString(), destinée à donner une description de l'objet en question. Si pour certaines classes cette méthode donne une indication précise sur la valeur de l'objet (et éventuellement sa classe), cela ne sera pas le cas pour la majorité. Au contraire, on obtiendra quelque chose contenant beaucoup de caractères étranges.

Et quand le JComboBox stocke des objets (et non de simples types primitifs), il se base sur le texte renvoyé par la méthode toString() de ces objets. C'est pourquoi vous obtiendrez un texte bizarre en lieu et place des différentes possibilités de la liste, si vous lancez l'application.

Néanmoins, on a la possibilité, si on le désire, de remplacer cette description rébarbative, par une chaîne beaucoup plus élégante. Pour cela, il suffit de surcharger la méthode toString().

Vous l'aurez certainement compris, nous allons surcharger la méthode toString() de la classe ModeleVelo afin qu'elle renvoie le modèle du vélo.



ModeleVelo (nouvelle version)

ModeleVelo (nouvelle version)

```
public class NouveauModeleVelo {

    String modele;
    int quantiteMaximumCommandable;

    public NouveauModeleVelo(String modele,    int quantiteMaximumCommandable) {
        this.modele = modele;
        this.quantiteMaximumCommandable = quantiteMaximumCommandable;
    }

    public String toString(){
        return modele;
    }

}
```

C'est aussi simple que cela.

Maintenant il faut savoir comment réagir quand l'utilisateur change l'élément sélectionné dans la liste.

VII-B-3 - L'interface ItemListener et la nouvelle version de EcranCommande

L'interface ItemListener est l'interface à utiliser pour tout objet devant être informé quand la sélection d'un JComboBox est modifiée. Elle ne comporte qu'une méthode de signature :

```
void itemStateChanged(ItemEvent e)
```

On obtient alors un objet ItemEvent, qui nous renseigne sur différents aspects de l'évènement. Notamment sur l'objet qui a été sélectionné. Pour cela on fera appel à la méthode getItem() de la classe ItemEvent :

```
Object getItem()
```

On obtient une instance de Object, un cast sera donc nécessaire.

On peut donc :

- créer une variable de classe qui servira à renseigner sur le dernier modèle sélectionné : modeleCourant. On l'initialisera avec le modèle qui sera affiché au lancement de l'application : c'est-à-dire le 1er modèle de la liste ;
- mettre à jour cette variable dans l'écouteur de type ItemListener ;
- définir la classe EcranCommande elle-même comme écouteur de type ItemListener.

Évidemment, il y a encore de petits ajustements à faire dans EcranCommande :

- dans la méthode actionPerformed(), la variable modeleChoisi ne sert plus ;
- on n'a plus besoin de la version verifierCommande() à deux paramètres : mais on peut la surcharger avec une version ne prenant que la quantité demandée.

Ce qui nous donne donc :

EcranCommande (nouvelle version)

```
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
```

EcranCommande (nouvelle version)

```
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

class EcranCommandeDeuxiemeVersion extends JFrame implements ActionListener, ItemListener {

    JComboBox champListeModele;
    JTextField champTexteQuantite, champTexteConfirmationCommande;
    JLabel labelModele, labelQuantite, labelConfirmationCommande;
    JButton boutonValidation;
    NouveauModeleVelo modelesDisponibles[] = {
        /*
         * Nul besoin de stocker la reference
         * sur chaque objet du tableau.
         */
        new NouveauModeleVelo("BikeMike 500", 5),
        new NouveauModeleVelo("RideAMax 130", 2),
        new NouveauModeleVelo("BigRace 200", 8),
        new NouveauModeleVelo("TrickyCourt 820", 7),
        new NouveauModeleVelo("MaxiRide 20", 3),
        new NouveauModeleVelo("RidersHeaven 760", 9),
    };
    NouveauModeleVelo modeleCourant = modelesDisponibles[0];

    public EcranCommandeDeuxiemeVersion() {
        // titre
        setTitle("Commande de velos");
        // centrer la fenetre sur l'ecran
        setLocationRelativeTo(null);
        // on quitte l'application si la fenetre est fermee
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panneauInterne = new JPanel();
        // La couleur de fond du panneau sera rose.
        panneauInterne.setBackground(Color.PINK);
        /*
         * Il y aura toujours 5 pixels d'espaces entre
         * les composants : que ce soit en largeur ou
         * en hauteur.
         */
        panneauInterne.setLayout(new GridLayout(4, 2, 5, 5));
        // Un JTextField avec 30 colonnes affichees.
        labelModele = new JLabel("Modele de velo : ");
        panneauInterne.add(labelModele);
        champListeModele = new JComboBox(new DefaultComboBoxModel(modelesDisponibles));
        panneauInterne.add(champListeModele);
        labelQuantite = new JLabel("Quantite : ");
        panneauInterne.add(labelQuantite);
        champTexteQuantite = new JTextField(30);
        panneauInterne.add(champTexteQuantite);
        labelConfirmationCommande = new JLabel("Commande confirmee ? ");
        panneauInterne.add(labelConfirmationCommande);
        champTexteConfirmationCommande = new JTextField(30);
        panneauInterne.add(champTexteConfirmationCommande);
        boutonValidation = new JButton("Passer la commande");
        panneauInterne.add(boutonValidation);

        boutonValidation.addActionListener(this);
        champListeModele.addItemListener(this);

        add(panneauInterne);
        /*
         * La fenetre aura juste la taille
         * qui lui laisse placer tous ses composants :
         * donc la taille qui laisse le panneauInterne
         * placer tous ses composants.
         */
    }
}
```

EcranCommande (nouvelle version)

```

    */
    pack();
}

public void actionPerformed(ActionEvent evt) {
    // L'utilisateur a cliqué sur le bouton Commander
    String quantiteChoisie = champTexteQuantite.getText();
    int quantite = Integer.parseInt(quantiteChoisie);
    try {
        verifierCommandeDuModeleCourant(quantite);
        // Cette ligne sera sautée en cas d'exception
        champTexteConfirmationCommande.setText(
            "Votre commande est prête.");
    } catch (TropDeVelosException e) {
        champTexteConfirmationCommande.setText(e.getMessage());
    } catch (Exception e) {
        champTexteConfirmationCommande.setText(e.getMessage());
    }
}

public void itemStateChanged(ItemEvent evt) {
    Object objetSelectionne = evt.getItem();
    if (objetSelectionne instanceof NouveauModeleVelo)
        modeleCourant = (NouveauModeleVelo) objetSelectionne;
}

/*
 * Cette version est devenue inutile
 */
void verifierCommande(String modeleVelo, int quantite)
    throws TropDeVelosException, Exception {
    NouveauModeleVelo modeleCorrespondant = null;
    for (NouveauModeleVelo modeleCourant : modelesDisponibles) {
        if (modeleVelo.equals(modeleCourant.modele)) {
            modeleCorrespondant = modeleCourant;
            break;
        }
    }
    if (modeleCorrespondant == null)
        throw new Exception("Le modele " + modeleVelo + " n'existe pas.");
    if (quantite > modeleCorrespondant.quantiteMaximumCommandable)
        throw new TropDeVelosException ("Impossible de livrer " +
            quantite + " velos du modele " + modeleVelo +
            " en une fois.");
}

void verifierCommandeDuModeleCourant(int quantite)
    throws TropDeVelosException, Exception {
    if (modeleCourant == null)
        throw new Exception("Erreur : aucun modele choisi.");
    if (quantite > modeleCourant.quantiteMaximumCommandable)
        throw new TropDeVelosException ("Impossible de livrer " +
            quantite + " velos du modele " + modeleCourant.modele +
            " en une fois.");
}

public static void main(String[] args) {
    /*
     * Pas besoin de stocker une reference
     * sur la fenetre cree : elle sera automatiquement
     * detruite en quittant l'application.
     */
    new EcranCommandeDeuxiemeVersion().setVisible(true);
}
}

```

Voici un aperçu de la nouvelle interface (le comportement reste le même).



VII-C - Synthèse

Les exercices de ce chapitre vous ont permis :

- de construire votre propre exception, en héritant de la classe Exception ;
- de créer un tableau contenant vos propres objets ;
- de construire une liste déroulante, la remplir et réagir quand l'utilisateur en change l'élément choisit ;
- de profiter du potentiel offert par la méthode toString(), en la redéfinissant.

VIII - Chapitre 9

VIII-A - Exercice 1 : Copier un fichier (mode console)

On nous demande de réaliser une copie de fichier, à l'aide de deux paramètres passés dans la console, lors du lancement du programme. Une grande partie du code de ce chapitre, peut être réutilisée : on s'inspirera des codes d'utilisation des flux d'octets et des codes d'utilisation des tampons.

Cet exercice ne présente pas de difficulté majeure. Mais tout de même quelques remarques par rapport au code que je vais vous soumettre :

- j'ai pris soin de vérifier que l'on a bien deux arguments dans la ligne de commandes ;
- voici un exemple d'erreur de type FileNotFoundException :

```
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(FileInputStream.java:137)
at java.io.FileInputStream.<init>(FileInputStream.java:96)
at CopieFichier.copier(CopieFichier.java:182)
at CopieFichier.main(CopieFichier.java:171)
```

Le message de l'exception est alors ce qui se situe en première ligne, juste après les deux points.

Comme le premier contenu du message est le nom du fichier que l'on a tenté d'ouvrir, on va tenter de l'extraire. En appelant la méthode `split(" ")` - la chaîne en paramètre se constitue d'un seul espace - sur une chaîne de caractères, on coupe la chaîne selon ses espaces.

```
String s = "Ceci est, un exemple, un simple exemple.";
String [] morceaux = s.split(" ");
```

On obtient donc le tableau `morceaux` suivant :

```
{" Ceci ", " est, ", " un ", " exemple, ", " un ", " simple ", " exemple. "}
```

Ainsi pour extraire le nom du fichier, on appellera :

```
exception.getMessage().split[0];
```

- j'ai préféré construire une méthode qui gère la copie, plutôt que de coder le tout en dur dans le `main()` ;
- enfin on referme, dans la clause `finally`, les différents flux. On prend soin de fermer les flux de lecture avant les flux d'écriture (et un tampon avant le flux auquel il est relié).

Voici donc le code que j'ai à vous proposer :

Copie de fichier (mode console)

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.swing.JOptionPane;

public class CopieFichierModeConsole {

    public static void main(String[] args) {
```

Copie de fichier (mode console)

```
// On verifie que l'on a bien deux arguments
if (args.length != 2){
    System.err.println("Utilisation : CopierFichier source destination");
    System.exit(1);
}

CopieFichierModeConsole copieurFichier = new CopieFichierModeConsole();
copieurFichier.copier(args[0], args[1]);
}

public void copier(String source, String destination){
    FileInputStream fichierLecture = null;
    FileOutputStream fichierSauvegarde = null;
    BufferedInputStream tamponFichierLecture = null;
    BufferedOutputStream tamponFichierSauvegarde = null;

    try {
        fichierLecture = new FileInputStream(source);
        tamponFichierLecture = new BufferedInputStream(fichierLecture);
        fichierSauvegarde = new FileOutputStream(destination);
        tamponFichierSauvegarde = new BufferedOutputStream(fichierSauvegarde);

        while(true){
            int valeurEntiereOctet = tamponFichierLecture.read();
            if (valeurEntiereOctet == -1)
                break;
            tamponFichierSauvegarde.write(valeurEntiereOctet);
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        // Decoupe la chaine selon les espaces
        String [] motsDuMessage = e.getMessage().split(" ");
        String message = "Impossible d'ouvrir le fichier "+motsDuMessage[0]+" ."
            +"\nVerifiez que le fichier existe et qu'il ne s'agisse pas" +
            " du tout d'un repertoire.";
        JOptionPane.showMessageDialog(null,message,
            "Erreur d'ouverture de fichier", JOptionPane.ERROR_MESSAGE);
    } catch (IOException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null,
            "Un erreur de lecture ou d'écriture diverse est survenue.",
            "Erreur diverse", JOptionPane.ERROR_MESSAGE);
    }
    finally{
        try {
            tamponFichierSauvegarde.flush();
            tamponFichierLecture.close();
            fichierLecture.close();
            tamponFichierSauvegarde.close();
            fichierSauvegarde.close();
        }
        catch (IOException exception){
            // l'un des flux est deja ferme
            exception.printStackTrace();
        }
    }
}
}
```

VIII-B - Exercice pour les petits malins : copier un fichier (mode graphique)

VIII-B-1 - Utilisation du composant Swing JFileChooser

Ce composant permet d'afficher une boîte de dialogue Ouvrir/EnregistrerSous, nous permettant donc de choisir un fichier grâce à une fenêtre conviviale.

Tout d'abord, il faut créer une instance de JFileChooser. On peut ensuite se contenter d'appeler :

- sa méthode `showOpenDialog` pour afficher une fenêtre de sélection de fichier pour une ouverture ;
- sa méthode `showSaveDialog` pour afficher une fenêtre de sélection de fichier pour une sauvegarde.

Dans les deux cas, on obtient un `int` en retour, nous indiquant de quelle manière l'utilisateur a quitté la fenêtre. S'il a validé sa sélection, on obtient la valeur `JFileChooser.APPROVE_OPTION`.

Ceci fait, on dispose de plusieurs méthodes nous renseignant sur la dernière sélection utilisateur, dont les plus utiles ici seront :

- `getSelectedFile()` : on obtient un objet de type `File`, correspondant au fichier sélectionné ;
- `getCurrentDirectory()` : on obtient le répertoire du fichier sélectionné.

Enfin, le constructeur

```
JFileChooser(File currentDirectory)
```

permet de choisir le répertoire affiché lors du lancement du menu : une valeur de `null` laissera le menu afficher le répertoire par défaut.

VIII-B-2 - L'utilisation de classe locale et des constantes

Vous connaissiez les variables **locales** : eh bien sachez maintenant que vous pouvez aussi déclarer des classes localement à une méthode. Bien sûr les inconvénients principaux d'une telle pratique :

- la classe ne sera pas réutilisable en dehors de la méthode où elle a été déclarée ;
- cela ne va pas en améliorant la clarté, la lisibilité du code.

Alors pourquoi utiliser de telles classes ? J'ai choisi de déclarer localement deux classes implémentant l'interface `ActionListener` dans le constructeur, parce que de cette manière les variables locales au constructeur peuvent être directement réutilisées dans la classe interne. Seule condition : que l'on déclare les variables réutilisées comme des constantes, grâce au mot-clé **final**.

```
final int monEntierConstant = 10 ;
```

Une constante est une variable qui ne pourra jamais être modifiée, sous peine d'obtenir une erreur de compilation. Les constantes peuvent aussi être utiles pour éviter de coder les variables en dur. Ainsi au lieu d'écrire, pour redimensionner une fenêtre :

```
fenetre.setSize(10,20) ;
```

Vous pouvez écrire :

```
final int tailleEnX = 10 ;
final int tailleEnY = 20 ;
/* plus loin dans le code (par exemple) */
fenetre.setSize(tailleEnX, tailleEnY) ;
```

Voici un exemple de déclaration/utilisation d'une classe **interne** dans une méthode :

Exemple de classe interne locale à une méthode

```
void construireFenetre() {
    /* du code avant (volontairement omis) */
    final String texte1 = "MonTexte1" ;
    final String texte2 = "AutreTexte2" ;
```

Exemple de classe interne locale à une méthode

```
final JButton monBouton = new JButton(texte1) ;
/* la classe interne */
class MonActionPourBouton implements ActionListener {
    public void actionPerformed(ActionEvent evt) {
        if (texte1.equals(monBouton.getText()) {
            monBouton.setText(texte2) ;
        }
        else if (texte2.equals(monBouton.getText()) {
            monBouton.setText(texte1) ;
        }
    }
}; // attention, le point virgule est très important
/* du code apres (volontairement omis) */
}
monBouton.addActionListener(new MonActionPourBouton()) ;
```

Ici le texte du bouton basculera entre le contenu de texte1 et le contenu de texte2 à chaque clic.

VIII-B-3 - Dernières considérations

J'ai réutilisé la méthode copier() de l'exercice précédent presque telle quelle : dans chaque close catch, je n'affiche plus mon message d'erreur personnalisé, mais je me contente de renvoyer une exception du même type en remplaçant le message par le mien.

La classe CopieFichier est maintenant une classe fille de la classe JFrame.

VIII-B-4 - Le code complet



Copie de fichier (version graphique)

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class CopieFichierModeGraphique extends JFrame {

    File fichierSource = null, fichierDestination = null;
    File repertoireSource = null, repertoireDestination = null;

    public CopieFichierModeGraphique() {
        setTitle("Copie de fichiers");
        // Fenetre fermee ? => On quitte l'application
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setLayout(new GridLayout(4, 3));

        add(new JLabel("Copier depuis : "));
        /*
         * Cette variable locale est aussi utilisée dans une classe interne
         */
    }
}
```

Copie de fichier (version graphique)

```

    * (voir explications plus loin) : on donc la declarer final.
    */
    final JTextField texteSource = new JTextField(30);
    /*
    * L'utilisateur ne pourra pas en
    * modifier le contenu.
    */
    texteSource.setEditable(false);
    add(texteSource);
    JButton boutonSource = new JButton("Parcourir ...");
    /*
    * Classe interne et locale au constructeur :
    * nous permet d'utiliser directement les variables
    * locales du constructeur, lesquelles doivent etre declarees en final.
    * Par contre, on n'est pas tenu d'utiliser final pour les variables de classe
    * que l'on souhaite utiliser dans la classe interne.
    *
    *
    * Remarquez comment la classe locale, elle aussi,
    * implemente une interface.
    */
    class ActionBoutonSource implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent evt) {
            /*
            * Si le repertoireSource n'est pas null,
            * on proposera ce repertoire a l'affichage du menu.
            */
            JFileChooser fenetreChoixFichier = new JFileChooser(repertoireSource);
            /*
            * Le resultatOuverture indique sur quel bouton
            * l'utilisateur a clique en fermant la fenetre de choix
            * de fichier.
            * Attention : ici un simple this nous
            * donnerais la classe interne ActionBoutonSource.
            * Enfin, on ouvre le menu en mode "Ouverture de fichier".
            */
            int modeFermeture = fenetreChoixFichier.showOpenDialog(CopieFichierModeGraphique.this);
            if (modeFermeture == JFileChooser.APPROVE_OPTION) {
                /*
                * Sauvegarde le repertoire pour la prochaine ouverture
                * du menu d'ouverture de fichier.
                */
                repertoireSource = fenetreChoixFichier.getCurrentDirectory();
                fichierSource = fenetreChoixFichier.getSelectedFile();
                /*
                * getAbsolutePath de la class File retourne
                * le chemin complet du fichier pointe.
                */
                texteSource.setText(fichierSource.getAbsolutePath());
            }
        }
    };

    boutonSource.addActionListener(new ActionBoutonSource());
    add(boutonSource);

    add(new JLabel("Copier vers : "));
    final JTextField texteDestination = new JTextField(30);
    texteDestination.setEditable(false);
    add(texteDestination);

    JButton boutonDestination = new JButton("Parcourir ...");
    class ActionBoutonDestination implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent evt) {
            JFileChooser fenetreChoixFichier = new JFileChooser(repertoireDestination);
            /*
            * On ouvre le menu en mode "Sauvegarde de fichier".

```

Copie de fichier (version graphique)

```

        */
        int modeFermeture = fenetreChoixFichier.showSaveDialog(CopieFichierModeGraphique.this);
        if (modeFermeture == JFileChooser.APPROVE_OPTION) {
            repertoireDestination = fenetreChoixFichier.getCurrentDirectory();
            fichierDestination = fenetreChoixFichier.getSelectedFile();
            texteDestination.setText(fichierDestination.getAbsolutePath());
        }
    }

    };
    boutonDestination.addActionListener(new ActionBoutonDestination());
    add(boutonDestination);

    add(new JLabel());
    JButton boutonCopier = new JButton("Copier");
    class ActionBoutonCopier implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent evt) {
            if (fichierSource == null) {
                JOptionPane.showMessageDialog(CopieFichierModeGraphique.this,
                    "Veuillez sélectionner un fichier source.",
                    "Selection incomplete",
                    JOptionPane.ERROR_MESSAGE
                );
                return;
            }
            if (fichierDestination == null) {
                JOptionPane.showMessageDialog(CopieFichierModeGraphique.this,
                    "Veuillez sélectionner un fichier destination.",
                    "Selection incomplete",
                    JOptionPane.ERROR_MESSAGE
                );
                return;
            }
            try {
                copier(fichierSource.getAbsolutePath(),
                    fichierDestination.getAbsolutePath());
            } catch (FileNotFoundException e) {
                JOptionPane.showMessageDialog(CopieFichierModeGraphique.this, "Le fichier a ete copie avec success.");
                e.printStackTrace();
                JOptionPane.showMessageDialog(CopieFichierModeGraphique.this, e.getMessage(),
                    "Fichier non trouve", JOptionPane.ERROR_MESSAGE);
            } catch (IOException e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(CopieFichierModeGraphique.this, e.getMessage(),
                    "Erreur de lecture/ecriture diverse", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    boutonCopier.addActionListener(new ActionBoutonCopier());
    add(boutonCopier);
    add(new JLabel());

    pack();
    /*
     * Centrer la fenetre :
     * c'est mieux de le faire apres l'appel a JFrame#pack()
     */
    setLocationRelativeTo(null);
}

public static void main(String[] args) {
    new CopieFichierModeGraphique().setVisible(true);
}

public void copier(String source, String destination)
throws FileNotFoundException, IOException {

```

Copie de fichier (version graphique)

```

FileInputStream fichierLecture = null;
FileOutputStream fichierSauvegarde = null;
BufferedInputStream tamponFichierLecture = null;
BufferedOutputStream tamponFichierSauvegarde = null;

try {
    fichierLecture = new FileInputStream(source);
    tamponFichierLecture = new BufferedInputStream(fichierLecture);
    fichierSauvegarde = new FileOutputStream(destination);
    tamponFichierSauvegarde = new BufferedOutputStream(fichierSauvegarde);

    while(true){
        int valeurEntiereOctet = tamponFichierLecture.read();
        if (valeurEntiereOctet == -1)
            break;
        tamponFichierSauvegarde.write(valeurEntiereOctet);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
    // Decoupe la chaine selon les espaces
    String [] motsDuMessage = e.getMessage().split(" ");
    String message = "Impossible d'ouvrir le fichier "+motsDuMessage[0]+" ."
        +"\nVerifiez que le fichier existe et qu'il ne s'agisse pas" +
        " du tout d'un repertoire.";
    throw new FileNotFoundException(message);
} catch (IOException e) {
    e.printStackTrace();
    String message = "Un erreur de lecture ou d'ecriture diverse est survenue.";
    throw new IOException(message);
}
finally{
    try {
        tamponFichierSauvegarde.flush();
        tamponFichierLecture.close();
        fichierLecture.close();
        tamponFichierSauvegarde.close();
        fichierSauvegarde.close();
    }
    catch (IOException exception){
        // l'un des flux est deja ferme
        exception.printStackTrace();
    }
}
}

```

VIII-C - Synthèse

Avec les exercices de ce chapitre, vous avez appris :

- à utiliser un flux de lecture d'octets et un flux d'écriture d'octets ;
- à réaliser une copie de fichier ;
- à utiliser les fenêtres Ouvrir/EnregistrerSous.

IX - Chapitre 10

IX-A - Exercice 1 : nouveau constructeur pour la classe Poisson

Cet exercice ne présente pas de difficulté particulière.

Bien sûr, nous travaillerons avec les nouvelles versions des classes Poisson et MaitrePoisson : les classes obtenues dans ce chapitre (j'y ai remplacé tous les accents).

Voici donc la classe Poisson modifiée :

Poisson (nouvelle version)

```
public class Poisson extends AnimalFamilier {
    int profondeurCourante = 0;
    final int PROFONDEUR_PLONGEE = 5;

    public Poisson() {
        profondeurCourante = 10;
    }

    public int plonger() {
        profondeurCourante = profondeurCourante +
            PROFONDEUR_PLONGEE;
        if (profondeurCourante > 100) {
            System.out.println("Je suis un petit"
                + " poisson et je ne peux pas plonger"
                + " plus profond que 100 metres");
            profondeurCourante = profondeurCourante
                - PROFONDEUR_PLONGEE;
        } else {
            System.out.println("Plongee de " + PROFONDEUR_PLONGEE
                + " metres");
            System.out.println("Je suis a " + profondeurCourante
                + " metres sous le niveau de la mer");
        }
        return profondeurCourante;
    }

    public int plonger(int combienDePlus) {
        profondeurCourante = profondeurCourante + combienDePlus;
        if (profondeurCourante > 100) {
            System.out.println("Je suis un petit"
                + " poisson et je ne peux pas plonger"
                + " plus profond que 100 metres");
            profondeurCourante = profondeurCourante
                - combienDePlus;
        } else {
            System.out.println("Plongee de " + combienDePlus
                + " metres");
            System.out.println("Je suis a " + profondeurCourante
                + " metres sous le niveau de la mer");
        }
        return profondeurCourante;
    }

    public String dire(String unMot) {
        return "Ne sais-tu pas que les poissons ne"
            + " parlent pas ?";
    }

    // Constructeur
    Poisson(int positionDepart) {
        profondeurCourante = positionDepart;
    }
}
```

Voici la nouvelle version de la classe MaitrePoisson :



MaitrePoisson nouvelle version

```
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class MaitrePoisson {
    public static void main(String[] args) {
        Poisson monPoisson = new Poisson();
        String chaineNombreDeMetres = "";
        int entierNombreDeMetres;
        // Cree un lecteur de flux d'entree connecte a
        // System.in et le passe au lecteur a tampon
        BufferedReader entreeStandard = new BufferedReader
            (new InputStreamReader(System.in));
        // Continue a plonger tant que l'utilisateur
        // ne tape pas "Q"
        while (true) {
            System.out.println("Pret a plonger. De combien ?");
            try {
                chaineNombreDeMetres = entreeStandard.readLine();
                if (chaineNombreDeMetres.equals("Q")) {
                    // Sort du programme
                    System.out.println("Au revoir !");
                    System.exit(0);
                } else {
                    // Convertit chaineNombreDeMetres en entier
                    // et plonge de la valeur de entierNombreDeMetres
                    entierNombreDeMetres =
                        Integer.parseInt(chaineNombreDeMetres);
                    monPoisson.plonger(entierNombreDeMetres);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

La classe AnimalFamiliier reste la même :



AnimalFamiliier

```
public class AnimalFamiliier {
    int age;
    float poids;
    float taille;
    String couleur;
    public void dormir() {
        System.out.println("Bonne nuit, a demain");
    }
    public void manger() {
        System.out.println(
            "J'ai si faim... Donne-moi un biscuit !");
    }
    public String dire(String unMot) {
        String réponseAnimal = "OK !! OK !! " + unMot;
        return réponseAnimal;
    }
}
```

Et enfin, un exemple de la sortie console :

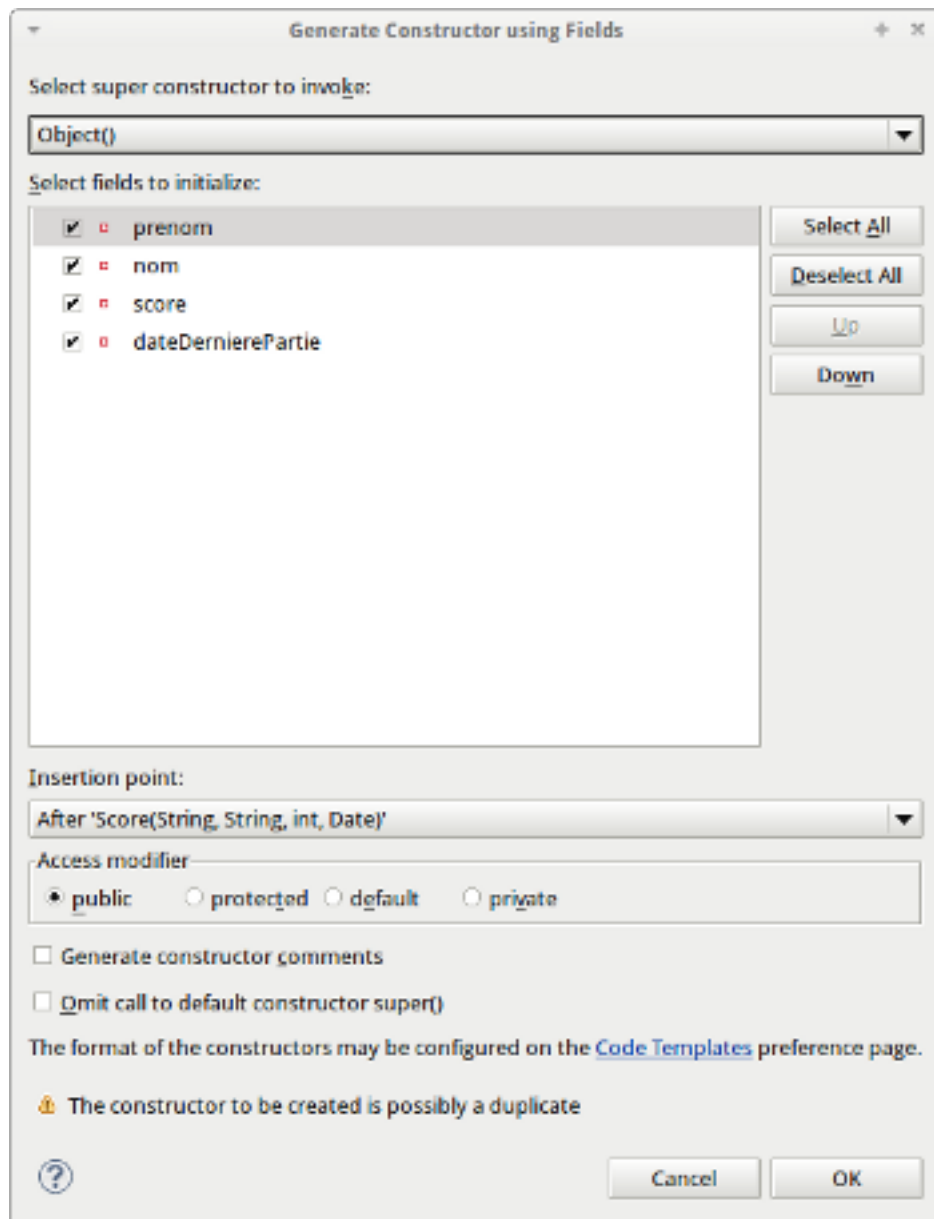
```
Pret a plonger. De combien ?
10
Plongee de 10 metres
Je suis a 20 metres sous le niveau de la mer
Pret a plonger. De combien ?
```

```
3
Plongee de 3 metres
Je suis a 23 metres sous le niveau de la mer
Pret a plonger. De combien ?
Q
Au revoir !
```

IX-B - Exercice 2 : nouveau constructeur pour la classe Score

Cet exercice ne présente pas non plus de difficulté particulière.

Sachez tout de même que vous pouvez, sous Eclipse, lancer le menu Source->Generate Constructor using fields. Ce menu vous permet de générer un constructeur, mais vous pouvez choisir quels champs de votre classe seront initialisés :



Eclipse : générer un constructeur

Voici la nouvelle version de la classe Score :



Scores : nouvelle version

```
import java.util.Date;
import java.text.SimpleDateFormat;
public class Score {
    private String prenom;
    private String nom;
    private int score;
    private Date dateDernierePartie;

    public Score(String prenom, String nom, int score, Date dateDernierePartie) {
        this.prenom = prenom;
        this.nom = nom;
        this.score = score;
        this.dateDernierePartie = dateDernierePartie;
    }

    public String lirePrenom() {
        return prenom;
    }
    public void affecterPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String lireNom() {
        return nom;
    }
    public void affecterNom(String nom) {
        this.nom = nom;
    }
    public int lireScore() {
        return score;
    }
    public void affecterScore(int score) {
        this.score = score;
    }
    public Date lireDateDernierePartie() {
        return dateDernierePartie;
    }
    public void affecterDateDernierePartie(Date
        dateDernierePartie) {
        this.dateDernierePartie = dateDernierePartie;
    }
    // Concatene tous les attributs en une chaîne
    // et y ajoute un caractere fin de ligne.
    // Cette methode est pratique, par exemple pour
    // afficher toutes les valeurs d'un coup, comme ceci :
    // System.out.println(myScore.toString());
    // NDT : comme cette methode surcharge Object.toString(),
    // tu es certain que la bonne representation de Score
    // est utilisee partout où Java en a besoin (c'est pour
    // cela que nous ne l'avons pas appelee convertirEnString).
    public String toString() {
        String chaîneScore = prenom + " " +
            nom + " " + score + " " + SimpleDateFormat.
                getDateInstance().format(dateDernierePartie) +
            System.getProperty("line.separator");
        return chaîneScore;
    }
}
```

Et voici la classe EnregistreurScores3 :



EnregistreurScores3

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
```

EnregistreurScores3

```
import java.util.Date;

public class EnregistreurScores3 {
    /**
    La methode main execute les actions suivantes :
    1. Cree une instance de tableau
    2. Cree les objets Score et les stocke dans le tableau
    3. Ecrit les donnees de scores dans un fichier
    */
    public static void main(String[] args) {
        FileWriter monFichier = null;
        BufferedWriter tampon = null;
        Date ceJour = new Date();
        Score scores[] = new Score[3];
        // Joueur n°1
        scores[0]= new Score("Jean", "Dupont", 250, ceJour);
        // Joueur n°2
        scores[1]= new Score("Anne", "Durand", 300, ceJour);
        // Joueur n°3
        scores[2]= new Score("Eskil", "Pemieuf", 190, ceJour);

        try {
            monFichier = new FileWriter("c:\\scores2.txt");
            tampon = new BufferedWriter(monFichier);
            for (int i = 0; i < scores.length; i++) {
                // Convertit chaque Score en String
                // et l'ecrit dans scores2.txt
                tampon.write(scores[i].toString());
                System.out.println("Ecriture des donnees de " +
                    scores[i].lireNom());
            }
            System.out.println("Ecriture du fichier terminee");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                tampon.flush();
                tampon.close();
                monFichier.close();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

Enfin je souhaite attirer votre attention sur l'utilisation du mot-clé `this` dans le constructeur : il permet ici de distinguer les paramètres des variables de classe. Ceci parce que paramètres et variables de classes ont les mêmes noms.

IX-C - Exercice pour les petits malins : DemoVector.java

Il n'y a pas d'énorme différence entre la classe `Vector` et la classe `ArrayList` : si ce n'est que la classe `ArrayList` est préférable du fait qu'elle est beaucoup plus récente. Vous pouvez trouver un petit comparatif des deux classes sur la page suivante de la FAQ Java de developpez.com : http://java.developpez.com/faq/java/index.php?page=langage_donnees#LANGAGE_COLLECTIONS_info_list

La classe `java.util.Vector` nous permet ici de réaliser exactement la même chose que dans le code original, presque sans aucune modification. En effet les deux classes disposent des méthodes `add(Object)`, `size()` et `get(int)` et ces méthodes ont le même comportement. Quand je parle de la méthode `get(int)` : je parle tout simplement de la version de la méthode `get()` acceptant un paramètre de type `int`.

Voici donc la classe `DemoVector` :

**DemoVector**

```
import java.util.Vector;

public class DemoVector {

    public static void main(String[] args) {
        Vector amis = new Vector();
        amis.add("Marie");
        amis.add("Anne");
        amis.add("David");
        amis.add("Remi");

        int nombreAmis = amis.size();

        for(int i = 0; i < nombreAmis; i++){
            System.out.println("L'ami(e) numero "+i+" est "+amis.get(i));
        }
    }
}
```

IX-D - Synthèse

Les exercices de ce chapitre vous ont permis de vous exercer à :

- surcharger un constructeur d'une classe parent ;
- créer un constructeur pour initialiser les champs de vos classes ;
- à utiliser la classe Vector.

X - Chapitre 11

X-A - Exercice 1 : affecter les coordonnées du point

X-A-1 - Le support de travail

Pour cet exercice, nous avons besoin de la première version du jeu : c'est-à-dire celle qui nous est donnée avant la section sur les fils d'exécution (Thread). Comme vous avez sûrement modifié les codes originaux pour les adapter à la section fil d'exécution, je les reposte ici (j'ai remplacé les accents).



TableVertePingPong

```
package support_travail_exercice_1;

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;

import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.WindowConstants;

/**
 * Cette classe dessine une table de ping-pong verte
 * et affiche les coordonnees du point où l'utilisateur
 * a clique.
 */
public class TableVertePingPong
extends JPanel
implements ConstantesDuJeu {
    JLabel label;
    public Point point = new Point(0,0);
    public int raquetteOrdinateur_X = 15;
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    Dimension taillePreferee = new
        Dimension(LARGEUR_TABLE, HAUTEUR_TABLE);
    // Cette methode affecte sa taille au cadre.
    // Elle est appelee par Java.
    public Dimension getPreferredSize() {
        return taillePreferee;
    }
    // Constructeur.
    TableVertePingPong() {
        MoteurJeuPingPong moteurJeu =
            new MoteurJeuPingPong(this);
        // Reçoit les clics pour l'affichage de leurs coordonnees
        addMouseListener(moteurJeu);
        // Reçoit les mouvements de la souris pour
        // le deplacement des raquettes
        addMouseMotionListener(moteurJeu);
    }
    // Ajoute à une fenetre un panneau contenant cette table et
    // un JLabel
    void ajouteAuCadre(Container conteneur) {
        conteneur.setLayout(new BoxLayout(conteneur,
            BoxLayout.Y_AXIS));
        conteneur.add(this);
        label = new JLabel("Coordonnees...");
        conteneur.add(label);
    }
    // Repeint la fenetre. Cette methode est appelee par Java
    // quand il est necessaire de rafraichir l'ecran ou quand
```

TableVertePingPong

```
// la methode repaint() est appelee par le
// MoteurJeuPingPong.
public void paintComponent(Graphics contexteGraphique) {
    super.paintComponent(contexteGraphique);
    // Dessine la table verte
    contexteGraphique.setColor(Color.GREEN);
    contexteGraphique.fillRect(
        0, 0, LARGEUR_TABLE, HAUTEUR_TABLE);
    // Dessine la raquette droite
    contexteGraphique.setColor(Color.yellow);
    contexteGraphique.fillRect(RAQUETTE_ENFANT_X_DEPART,
        raquetteEnfant_Y, 5, 30);
    // Dessine la raquette gauche
    contexteGraphique.setColor(Color.blue);
    contexteGraphique.fillRect(
        raquetteOrdinateur_X, 100, 5, 30);
    // Dessine la balle
    contexteGraphique.setColor(Color.red);
    contexteGraphique.fillOval(25, 110, 10, 10);
    // Dessine les lignes
    contexteGraphique.setColor(Color.white);
    contexteGraphique.drawRect(10, 10, 300, 200);
    contexteGraphique.drawLine(160, 10, 160, 210);

    // Affiche un point sous forme de rectangle de 2x2 pixels
    if (point != null) {
        label.setText("Coordonnees (x, y) : " +
            point.x + ", " + point.y);
        contexteGraphique.fillRect(point.x, point.y, 2, 2);
    }
}

// Affecte sa position courante à la raquette de l'enfant
public void positionnerRaquetteEnfant_Y(int y) {
    this.raquetteEnfant_Y = y;
}

// Retourne la position courante de la raquette de l'enfant
public int coordonneeRaquetteEnfant_Y() {
    return raquetteEnfant_Y;
}

public static void main(String[] args) {
    // Cree une instance du cadre
    JFrame monCadre = new JFrame("Table verte de ping-pong");
    // Permet la fermeture de la fenetre par clic sur la
    // petite croix dans le coin.
    monCadre.setDefaultCloseOperation(
        WindowConstants.EXIT_ON_CLOSE);
    TableVertePingPong table = new TableVertePingPong();
    table.ajouteAuCadre(monCadre.getContentPane());
    // Affecte sa taille au cadre et le rend visible.
    monCadre.pack();
    monCadre.setVisible(true);
}
}
```



ConstantesDuJeu

```
package support_travail_exercice_1;

public interface ConstantesDuJeu {
    public final int LARGEUR_TABLE = 320;
    public final int HAUTEUR_TABLE = 220;
    public final int RAQUETTE_ENFANT_Y_DEPART = 100;
    public final int RAQUETTE_ENFANT_X_DEPART = 300;
    public final int HAUT_TABLE = 12;
    public final int BAS_TABLE = 180;
    public final int INCREMENT_RAQUETTE = 4;
}
```



MoteurJeuPingPong

```
package support_travail_exercice_1;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

public class MoteurJeuPingPong implements
MouseListener, MouseMotionListener, ConstantesDuJeu {
    TableVertePingPong table;
    public int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    // Constructeur. Stocke une reference à la table.
    public MoteurJeuPingPong(TableVertePingPong tableVerte) {
        table = tableVerte;
    }
    // Methodes requises par l'interface MouseListener.
    public void mousePressed(MouseEvent evenement) {
        // Recupere les coordonnees X et Y du pointeur de la
        // souris et les affecte au "point blanc" sur la table.
        table.point.x = evenement.getX();
        table.point.y = evenement.getY();
        // La methode repaint appelle en interne la methode
        // paintComponent() de la table qui rafraîchit la
        // fenetre.
        table.repaint();
    }
    public void mouseReleased(MouseEvent evenement) {}
    public void mouseEntered(MouseEvent evenement) {}
    public void mouseExited(MouseEvent evenement) {}
    public void mouseClicked(MouseEvent evenement) {}
    // Methodes requises par l'interface MouseMotionListener.
    public void mouseDragged(MouseEvent evenement) {}
    public void mouseMoved(MouseEvent evenement) {
        int souris_Y = evenement.getY();
        // Si la souris est au-dessus de la raquette de l'enfant
        // et que la raquette n'a pas depasse la limite
        // superieure de la table, la deplace vers le haut ;
        // sinon, la deplace vers le bas.
        if (souris_Y < raquetteEnfant_Y
            && raquetteEnfant_Y > HAUT_TABLE) {
            raquetteEnfant_Y -= INCREMENT_RAQUETTE;
        } else if (raquetteEnfant_Y < BAS_TABLE) {
            raquetteEnfant_Y += INCREMENT_RAQUETTE;
        }
        // Affecte la nouvelle position de la raquette dans la
        // classe table
        table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
        table.repaint();
    }
}
```

X-A-2 - Une correction possible

Cet exercice ne présente pas de grande difficulté. Vous pourrez juste remarquer, dans la méthode `mousePressed()` de la classe `MoteurJeuPingPong`, l'appel à

```
table.affecteCoordonneesPoint(evenement.getX(), evenement.getY());
```

sans même stocker les valeurs de `evenement.getX()` et `evenement.getY()` : en effet, on n'aura pas besoin de les réutiliser.

Voici les versions modifiées de `TableVertePingPong.java` et `MoteurJeuPingPong.java`.

Créez un package pour placer les deux fichiers de l'exercice 1 donnés ci-dessous. Dans Eclipse, importez les deux fichiers dans le nouveau package (clic droit sur le package en question et Import->General->FileSystem pour choisir le dossier où ils se situent). Copiez alors le fichier ConstantesDuJeu du package support_travail_exercice_1 vers le nouveau package (il suffit de faire clic droit->copy sur le fichier en question puis clic droit->paste sur le nouveau package.) Vous disposerez alors d'une copie adaptée pour le nouveau package.



TableVertePingPong

```
package exercice1;

import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Point;

import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.WindowConstants;

/**
 * Cette classe dessine une table de ping-pong verte
 * et affiche les coordonnées du point où l'utilisateur
 * a clique.
 */
public class TableVertePingPong
    extends JPanel
    implements ConstantesDuJeu {
    JLabel label;
    private Point point = new Point(0,0);
    public int raquetteOrdinateur_X = 15;
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    Dimension taillePreferee = new
        Dimension(LARGEUR_TABLE, HAUTEUR_TABLE);
    // Cette methode affecte sa taille au cadre.
    // Elle est appelee par Java.
    public Dimension getPreferredSize() {
        return taillePreferee;
    }
    // Constructeur.
    TableVertePingPong() {
        MoteurJeuPingPong moteurJeu =
            new MoteurJeuPingPong(this);
        // Reçoit les clics pour l'affichage de leurs coordonnées
        addMouseListener(moteurJeu);
        // Reçoit les mouvements de la souris pour
        // le déplacement des raquettes
        addMouseMotionListener(moteurJeu);
    }
    // Ajoute à une fenetre un panneau contenant cette table et
    // un JLabel
    void ajouteAuCadre(Container conteneur) {
        conteneur.setLayout(new BoxLayout(conteneur,
            BoxLayout.Y_AXIS));
        conteneur.add(this);
        label = new JLabel("Coordonnees...");
        conteneur.add(label);
    }
    // Repeint la fenetre. Cette methode est appelee par Java
    // quand il est necessaire de rafraîchir l'ecran ou quand
    // la methode repaint() est appelee par le
    // MoteurJeuPingPong.
    public void paintComponent(Graphics contexteGraphique) {
        super.paintComponent(contexteGraphique);
        // Dessine la table verte
        contexteGraphique.setColor(Color.GREEN);
        contexteGraphique.fillRect(
```

TableVertePingPong

```

        0, 0, LARGEUR_TABLE, HAUTEUR_TABLE);
// Dessine la raquette droite
contexteGraphique.setColor(Color.yellow);
contexteGraphique.fillRect(RAQUETTE_ENFANT_X_DEPART,
    raquetteEnfant_Y, 5, 30);
// Dessine la raquette gauche
contexteGraphique.setColor(Color.blue);
contexteGraphique.fillRect(
    raquetteOrdinateur_X, 100, 5, 30);
// Dessine la balle
contexteGraphique.setColor(Color.red);
contexteGraphique.fillOval(25, 110, 10, 10);
// Dessine les lignes
contexteGraphique.setColor(Color.white);
contexteGraphique.drawRect(10, 10, 300, 200);
contexteGraphique.drawLine(160, 10, 160, 210);

// Affiche un point sous forme de rectangle de 2x2 pixels
if (point != null) {
    label.setText("Coordonnees (x, y) : " +
        point.x + ", " + point.y);
    contexteGraphique.fillRect(point.x, point.y, 2, 2);
}
}
// Affecte sa position courante à la raquette de l'enfant
public void positionnerRaquetteEnfant_Y(int y) {
    this.raquetteEnfant_Y = y;
}
// Retourne la position courante de la raquette de l'enfant
public int coordonneeRaquetteEnfant_Y() {
    return raquetteEnfant_Y;
}
public void affecteCoordonneesPoint(int x, int y){
    point = new Point(x, y);
}
public static void main(String[] args) {
    // Cree une instance du cadre
    JFrame monCadre = new JFrame("Table verte de ping-pong");
    // Permet la fermeture de la fenetre par clic sur la
    // petite croix dans le coin.
    monCadre.setDefaultCloseOperation(
        WindowConstants.EXIT_ON_CLOSE);
    TableVertePingPong table = new TableVertePingPong();
    table.ajouteAuCadre(monCadre.getContentPane());
    // Affecte sa taille au cadre et le rend visible.
    monCadre.pack();
    monCadre.setVisible(true);
}
}

```



MoteurJeuPingPong

```

package exercicel;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

public class MoteurJeuPingPong implements
MouseListener, MouseMotionListener, ConstantesDuJeu {
    TableVertePingPong table;
    public int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    // Constructeur. Stocke une reference a la table.
    public MoteurJeuPingPong(TableVertePingPong tableVerte) {
        table = tableVerte;
    }
    // Methodes requises par l'interface MouseListener.
    public void mousePressed(MouseEvent evenement) {
        // Recupere les coordonnees X et Y du pointeur de la

```


MoteurJeuPingPong

```
// souris et les affecte au "point blanc" sur la table.
table.affecteCoordonneesPoint(evenement.getX(), evenement.getY());
// La methode repaint appelle en interne la methode
// paintComponent() de la table qui rafraichit la
// fenetre.
table.repaint();
}

public void mouseReleased(MouseEvent evenement) {}
public void mouseEntered(MouseEvent evenement) {}
public void mouseExited(MouseEvent evenement) {}
public void mouseClicked(MouseEvent evenement) {}
// Methodes requises par l'interface MouseMotionListener.
public void mouseDragged(MouseEvent evenement) {}
public void mouseMoved(MouseEvent evenement) {
    int souris_Y = evenement.getY();
    // Si la souris est au-dessus de la raquette de l'enfant
    // et que la raquette n'a pas depasse la limite
    // superieure de la table, la deplace vers le haut ;
    // sinon, la deplace vers le bas.
    if (souris_Y < raquetteEnfant_Y
        && raquetteEnfant_Y > HAUT_TABLE) {
        raquetteEnfant_Y -= INCREMENT_RAQUETTE;
    } else if (raquetteEnfant_Y < BAS_TABLE) {
        raquetteEnfant_Y += INCREMENT_RAQUETTE;
    }
    // Affecte la nouvelle position de la raquette dans la
    // classe table
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
    table.repaint();
}
}
```

X-B - Exercice 2 : résolution du bogue

X-B-1 - Support de travail

Cette fois-ci on doit travailler sur la version finale du jeu. (Sachez que dans les versions corrigées des différentes classes, tous les caractères accentués auront été remplacés.)

Je vous repasse donc les fichiers de la version finale du jeu de ping-pong, tel qu'il a été défini à la fin du chapitre (sans les accents).



TableVertePingPong

```
package support_travail_exercice_2;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Graphics;

import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
/**
 * Cette classe dessine la table de ping-pong, la balle et
 * les raquettes et affiche le score
 */
public class TableVertePingPong extends JPanel
implements ConstantesDuJeu {
    private JLabel label;
    private int raquetteOrdinateur_Y =
        RAQUETTE_ORDINATEUR_Y_DEPART;
```

TableVertePingPong

```

private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
private int balle_X = BALLE_X_DEPART;
private int balle_Y = BALLE_Y_DEPART;
Dimension taillePreferee = new
    Dimension(LARGEUR_TABLE, HAUTEUR_TABLE);
// Cette methode affecte sa taille au cadre.
// Elle est appelee par Java.
public Dimension getPreferredSize() {
    return taillePreferee;
}
// Constructeur. Cree un recepteur d'evenements souris.
TableVertePingPong() {
    MoteurJeuPingPong moteurJeu =
        new MoteurJeuPingPong(this);
    // Reçoit les mouvements de la souris pour deplacer la
    // raquette.
    addMouseListener(moteurJeu);
    // Reçoit les evenements clavier.
    addKeyListener(moteurJeu);
}
//Ajoute a un cadre la table et un JLabel
void ajouteAuCadre(Container conteneur) {
    conteneur.setLayout(new BorderLayout(conteneur,
        BorderLayout.Y_AXIS));
    conteneur.add(this);
    label = new JLabel(
        "Taper N pour une nouvelle partie, S pour servir"+
        " ou Q pour quitter");
    conteneur.add(label);
}
//Repeint la fenetre. Cette methode est appelee par Java
//quand il est necessaire de rafraîchir l'ecran ou quand
//la methode repaint() est appelee.
public void paintComponent(Graphics contexteGraphique) {
    super.paintComponent(contexteGraphique);
    //Dessine la table verte
    contexteGraphique.setColor(Color.GREEN);
    contexteGraphique.fillRect(
        0, 0, LARGEUR_TABLE, HAUTEUR_TABLE);
    //Dessine la raquette droite
    contexteGraphique.setColor(Color.yellow);
    contexteGraphique.fillRect(
        RAQUETTE_ENFANT_X,
        raquetteEnfant_Y,
        LARGEUR_RAQUETTE, LONGUEUR_RAQUETTE);
    //Dessine la raquette gauche
    contexteGraphique.setColor(Color.blue);
    contexteGraphique.fillRect(RAQUETTE_ORDINATEUR_X,
        raquetteOrdinateur_Y,
        LARGEUR_RAQUETTE, LONGUEUR_RAQUETTE);
    //Dessine la balle
    contexteGraphique.setColor(Color.red);
    contexteGraphique.fillOval(balle_X, balle_Y, 10, 10);
    //Dessine les lignes blanches
    contexteGraphique.setColor(Color.white);
    contexteGraphique.drawRect(10, 10, 300, 200);
    contexteGraphique.drawLine(160, 10, 160, 210);
    //Donne le focus a la table, afin que le recepteur de
    //touches envoie les commandes a la table
    requestFocus();
}
//Affecte sa position courante a la raquette de l'enfant
public void positionnerRaquetteEnfant_Y(int y) {
    this.raquetteEnfant_Y = y;
    repaint();
}
//Retourne la position courante de la raquette de l'enfant
public int coordonneeRaquetteEnfant_Y() {
    return raquetteEnfant_Y;
}
//Affecte sa position courante a la raquette de

```

TableVertePingPong

```
//l'ordinateur
public void positionnerRaquetteOrdinateur_Y(int y) {
    this.raquetteOrdinateur_Y = y;
    repaint();
}

//Affecte le texte du message du jeu
public void affecterTexteMessage(String texte) {
    label.setText(texte);
    repaint();
}

//Positionne la balle
public void positionnerBalle(int x, int y) {
    balle_X = x;
    balle_Y = y;
    repaint();
}

public static void main(String[] args) {
    //Cree une instance du cadre
    JFrame monCadre = new JFrame("Table verte de ping-pong");
    //Permet la fermeture de la fenetre par clic sur la
    //petite croix dans le coin.
    monCadre.setDefaultCloseOperation(
        WindowConstants.EXIT_ON_CLOSE);
    TableVertePingPong table = new TableVertePingPong();
    table.ajouteAuCadre(monCadre.getContentPane());
    //Affecte sa taille au cadre et le rend visible.
    monCadre.setBounds(0, 0, LARGEUR_TABLE + 5,
        HAUTEUR_TABLE + 40);
    monCadre.setVisible(true);
}
}
```



MoteurJeuPingPong

```
package support_travail_exercice_2;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
/**
 * Cette classe est un recepteur de souris et de clavier.
 * Elle calcule les déplacements de la balle et des raquettes
 * et change leurs coordonnees.
 */
public class MoteurJeuPingPong implements Runnable,
    MouseMotionListener, KeyListener, ConstantesDuJeu {
    private TableVertePingPong table; // Reference à la table.
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    private int raquetteOrdinateur_Y =
        RAQUETTE_ORDINATEUR_Y_DEPART;
    private int scoreEnfant;
    private int scoreOrdinateur;
    private int balle_X; // position X de la balle
    private int balle_Y; // position Y de la balle
    private boolean deplacementGauche = true;
    private boolean balleServie = false;
    //Valeur en pixels du deplacement vertical de la balle.
    private int deplacementVertical;
    // Constructeur. Stocke une reference à la table.
    public MoteurJeuPingPong(TableVertePingPong tableVerte) {
        table = tableVerte;
        Thread travailleur = new Thread(this);
        travailleur.start();
    }
    // Methodes requises par l'interface MouseMotionListener
    // (certaines sont vides, mais doivent etre incluses dans
    // la classe de toute façon).
    public void mouseDragged(MouseEvent evenement) {
    }
}
```

MoteurJeuPingPong

```

public void mouseMoved(MouseEvent evenement) {
    int souris_Y = evenement.getY();
    // Si la souris est au-dessus de la raquette de l'enfant
    // et que la raquette n'a pas depasse le haut de la
    // table, la deplace vers le haut ;
    // sinon, la deplace vers le bas.
    if (souris_Y < raquetteEnfant_Y &&
        raquetteEnfant_Y > HAUT_TABLE) {
        raquetteEnfant_Y -= INCREMENT_RAQUETTE;
    } else if (raquetteEnfant_Y < BAS_TABLE) {
        raquetteEnfant_Y += INCREMENT_RAQUETTE;
    }
    // Affecte la nouvelle position de la raquette
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}
// Methodes requises par l'interface KeyListener.
public void keyPressed(KeyEvent evenement) {
    char touche = evenement.getKeyChar();
    if ('n' == touche || 'N' == touche) {
        demarrerNouvellePartie();
    } else if ('q' == touche || 'Q' == touche) {
        terminerJeu();
    } else if ('s' == touche || 'S' == touche) {
        serviceEnfant();
    }
}

public void keyReleased(KeyEvent evenement) {}
public void keyTyped(KeyEvent evenement) {}
// Demarre une nouvelle partie.
public void demarrerNouvellePartie() {
    scoreOrdinateur = 0;
    scoreEnfant = 0;
    table.affecterTexteMessage("Scores - Ordinateur : 0"
        + "Enfant : 0");
    serviceEnfant();
}
// Termine le jeu.
public void terminerJeu() {
    System.exit(0);
}
//La methode run() est requise par l'interface Runnable.
public void run() {
    boolean rebondPossible = false;
    while (true) {
        if (balleServie) { // Si la balle est en mouvement
            //Etape 1. La balle se deplace-t-elle vers la
            //gauche ?
            if (deplacementGauche && balle_X > BALLE_X_MIN) {
                rebondPossible = (balle_Y >= raquetteOrdinateur_Y
                    && balle_Y < (raquetteOrdinateur_Y +
                        LONGUEUR_RAQUETTE) ? true : false);
                balle_X -= INCREMENT_BALLE;
                //Ajoute un deplacement vertical à chaque
                //mouvement horizontal de la balle.
                balle_Y -= deplacementVertical;
                table.positionnerBalle(balle_X, balle_Y);
                //La balle peut-elle rebondir ?
                if (balle_X <= RAQUETTE_ORDINATEUR_X
                    && rebondPossible) {
                    deplacementGauche = false;
                }
            }
            //Etape 2. La balle se deplace-t-elle vers la
            //droite ?
            if (!deplacementGauche && balle_X <= BALLE_X_MAX) {
                rebondPossible = (balle_Y >= raquetteEnfant_Y &&
                    balle_Y < (raquetteEnfant_Y +
                        LONGUEUR_RAQUETTE) ? true : false);
                balle_X += INCREMENT_BALLE;
                table.positionnerBalle(balle_X, balle_Y);
                //La balle peut-elle rebondir ?
            }
        }
    }
}

```

MoteurJeuPingPong

```

        if (balle_X >= RAQUETTE_ENFANT_X &&
            rebondPossible) {
            deplacementGauche = true;
        }
    }
    //Etape 3. Deplace la raquette de l'ordinateur vers le
    // haut ou vers le bas pour bloquer la balle.
    if (raquetteOrdinateur_Y < balle_Y
        && raquetteOrdinateur_Y < BAS_TABLE) {
        raquetteOrdinateur_Y += INCREMENT_RAQUETTE;
    } else if (raquetteOrdinateur_Y > HAUT_TABLE) {
        raquetteOrdinateur_Y -= INCREMENT_RAQUETTE;
    }
    table.positionnerRaquetteOrdinateur_Y(
        raquetteOrdinateur_Y);
    // Etape 4. Sommeiller un peu
    try {
        Thread.sleep(DUREE_SOMMEIL);
    } catch (InterruptedException exception) {
        exception.printStackTrace();
    }
    // Etape 5. Mettre le score à jour si la balle est
    // dans la surface verte mais ne bouge plus.
    if (balleSurLaTable()) {
        if (balle_X > BALLE_X_MAX ) {
            scoreOrdinateur++;
            afficherScore();
        } else if (balle_X < BALLE_X_MIN) {
            scoreEnfant++;
            afficherScore();
        }
    }
    } // Fin du if balleServie
    } // Fin du while
} // Fin de run()
// Sert depuis la position courante de la raquette
// de l'enfant.
private void serviceEnfant() {
    balleServie = true;
    balle_X = RAQUETTE_ENFANT_X - 1;
    balle_Y = raquetteEnfant_Y;
    if (balle_Y > HAUTEUR_TABLE / 2) {
        deplacementVertical = -1;
    } else {
        deplacementVertical = 1;
    }
    table.positionnerBalle(balle_X, balle_Y);
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}
private void afficherScore() {
    balleServie = false;
    if (scoreOrdinateur == SCORE_GAGNANT) {
        table.affecterTexteMessage("L'ordinateur a gagne ! " +
            scoreOrdinateur +
            " : " + scoreEnfant);
    } else if (scoreEnfant == SCORE_GAGNANT) {
        table.affecterTexteMessage ("Tu as gagne ! " +
            scoreEnfant +
            " : " + scoreOrdinateur);
    } else {
        table.affecterTexteMessage ("Ordinateur : " +
            scoreOrdinateur +
            " Enfant: " + scoreEnfant);
    }
}
// Verifie que la balle n'a pas depasse la limite
// inferieure ou superieure de la table.
private boolean balleSurLaTable() {
    if (balle_Y >= BALLE_Y_MIN && balle_Y <= BALLE_Y_MAX) {
        return true;
    }
}

```

MoteurJeuPingPong

```

    } else {
        return false;
    }
}
}

```



ConstantesDuJeu

```

package support_travail_exercice_2;
/**
 * Cette interface contient toutes les définitions des
 * variables invariantes utilisées dans le jeu.
 */
public interface ConstantesDuJeu {
    // Taille de la table de ping-pong
    public final int LARGEUR_TABLE = 320;
    public final int HAUTEUR_TABLE = 220;
    public final int HAUT_TABLE = 12;
    public final int BAS_TABLE = 180;
    // Incrément du mouvement de la balle en pixels
    public final int INCREMENT_BALLE = 4;
    // Coordonnées maximum et minimum permises pour la balle
    public final int BALLE_X_MIN = 1 + INCREMENT_BALLE;
    public final int BALLE_Y_MIN = 1 + INCREMENT_BALLE;
    public final int BALLE_X_MAX =
        LARGEUR_TABLE - INCREMENT_BALLE;
    public final int BALLE_Y_MAX =
        HAUTEUR_TABLE - INCREMENT_BALLE;
    // Position de départ de la balle
    public final int BALLE_X_DEPART = LARGEUR_TABLE / 2;
    public final int BALLE_Y_DEPART = HAUTEUR_TABLE / 2;
    // Taille, positions et incrément des raquettes
    public final int RAQUETTE_ENFANT_X = 300;
    public final int RAQUETTE_ENFANT_Y_DEPART = 100;
    public final int RAQUETTE_ORDINATEUR_X = 15;
    public final int RAQUETTE_ORDINATEUR_Y_DEPART = 100;
    public final int INCREMENT_RAQUETTE = 2;
    public final int LONGUEUR_RAQUETTE = 30;
    public final int LARGEUR_RAQUETTE = 5;
    public final int SCORE_GAGNANT = 21;
    // Ralentit mes ordinateurs rapides ;
    // modifier la valeur si nécessaire.
    public final int DUREE_SOMMEIL = 10; // En millisecondes.
}

```

X-B-2 - Correction possible

Cet exercice ne présente pas de difficulté particulière.

Une manière simple de procéder consiste à vérifier, au moment où l'utilisateur appuie sur la touche S (que ce soit en minuscule ou en majuscule), que les scores du joueur **et** de l'ordinateur sont **inférieurs au score gagnant**. Et l'on sait que c'est dans la méthode keyPressed de la classe MoteurJeuPingPong qu'est effectuée la gestion des touches.

Enfin par prudence on peut déplacer l'étape 4 de la méthode run(), celle qui consiste à faire sommeiller le Thread, juste en dehors de la condition if (balleServie). Donc juste au début de la boucle while(true). De cette manière, si le service n'a pas été effectué, on sera quand même en mesure de faire le Thread se reposer un peu et éviter qu'il ne bloque tout.

Voici donc la nouvelle version de MoteurJeuPingPong.java :



MoteurJeuPingPong amélioré

```
package exercice2;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
/**
 * Cette classe est un recepteur de souris et de clavier.
 * Elle calcule les déplacements de la balle et des raquettes
 * et change leurs coordonnees.
 */
public class MoteurJeuPingPong implements Runnable,
MouseMotionListener, KeyListener, ConstantesDuJeu {
    private TableVertePingPong table; // Reference a la table.
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    private int raquetteOrdinateur_Y =
        RAQUETTE_ORDINATEUR_Y_DEPART;
    private int scoreEnfant;
    private int scoreOrdinateur;
    private int balle_X; // position X de la balle
    private int balle_Y; // position Y de la balle
    private boolean deplacementGauche = true;
    private boolean balleServie = false;
    //Valeur en pixels du deplacement vertical de la balle.
    private int deplacementVertical;
    // Constructeur. Stocke une reference a la table.
    public MoteurJeuPingPong(TableVertePingPong tableVerte) {
        table = tableVerte;
        Thread travailleur = new Thread(this);
        travailleur.start();
    }
    // Methodes requises par l'interface MouseMotionListener
    // (certaines sont vides, mais doivent etre incluses dans
    // la classe de toute façon).
    public void mouseDragged(MouseEvent evenement) {
    }

    public void mouseMoved(MouseEvent evenement) {
        int souris_Y = evenement.getY();
        // Si la souris est au-dessus de la raquette de l'enfant
        // et que la raquette n'a pas depasse le haut de la
        // table, la deplace vers le haut ;
        // sinon, la deplace vers le bas.
        if (souris_Y < raquetteEnfant_Y &&
            raquetteEnfant_Y > HAUT_TABLE) {
            raquetteEnfant_Y -= INCREMENT_RAQUETTE;
        } else if (raquetteEnfant_Y < BAS_TABLE) {
            raquetteEnfant_Y += INCREMENT_RAQUETTE;
        }
        // Affecte la nouvelle position de la raquette
        table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
    }
    // Methodes requises par l'interface KeyListener.
    public void keyPressed(KeyEvent evenement) {
        char touche = evenement.getKeyChar();
        if ('n' == touche || 'N' == touche) {
            demarrerNouvellePartie();
        } else if ('q' == touche || 'Q' == touche) {
            terminerJeu();
        } else if ('s' == touche || 'S' == touche) {
            if (scoreEnfant < SCORE_GAGNANT && scoreOrdinateur < SCORE_GAGNANT) {
                serviceEnfant();
            }
        }
    }
    public void keyReleased(KeyEvent evenement) {}
    public void keyTyped(KeyEvent evenement) {}
    // Demarre une nouvelle partie.
    public void demarrerNouvellePartie() {
        scoreOrdinateur = 0;
    }
}
```

MoteurJeuPingPong amélioré

```

    scoreEnfant = 0;
    table.affecterTexteMessage("Scores - Ordinateur : 0"
        + " Enfant : 0");
    serviceEnfant();
}
// Termine le jeu.
public void terminerJeu() {
    System.exit(0);
}

// La methode run() est requise par l'interface Runnable.
public void run() {
    boolean rebondPossible = false;
    while (true) {
        // Etape 4. Sommeiller un peu
        try {
            Thread.sleep(DUREE_SOMMEIL);
        } catch (InterruptedException exception) {
            exception.printStackTrace();
        }
        if (balleServie) { // Si la balle est en mouvement
            // Etape 1. La balle se deplace-t-elle vers la
            // gauche ?
            if (deplacementGauche && balle_X > BALLE_X_MIN) {
                rebondPossible = (balle_Y >= raquetteOrdinateur_Y
                    && balle_Y < (raquetteOrdinateur_Y +
                        LONGUEUR_RAQUETTE)) ? true : false);
                balle_X -= INCREMENT_BALLE;
                // Ajoute un deplacement vertical a chaque
                // mouvement horizontal de la balle.
                balle_Y -= deplacementVertical;
                table.positionnerBalle(balle_X, balle_Y);
                // La balle peut-elle rebondir ?
                if (balle_X <= RAQUETTE_ORDINATEUR_X
                    && rebondPossible) {
                    deplacementGauche = false;
                }
            }
            // Etape 2. La balle se deplace-t-elle vers la
            // droite ?
            if (!deplacementGauche && balle_X <= BALLE_X_MAX) {
                rebondPossible = (balle_Y >= raquetteEnfant_Y &&
                    balle_Y < (raquetteEnfant_Y +
                        LONGUEUR_RAQUETTE)) ? true : false);
                balle_X += INCREMENT_BALLE;
                table.positionnerBalle(balle_X, balle_Y);
                // La balle peut-elle rebondir ?
                if (balle_X >= RAQUETTE_ENFANT_X &&
                    rebondPossible) {
                    deplacementGauche = true;
                }
            }
            // Etape 3. Deplace la raquette de l'ordinateur vers le
            // haut ou vers le bas pour bloquer la balle.

            if (raquetteOrdinateur_Y < balle_Y
                && raquetteOrdinateur_Y < BAS_TABLE) {
                raquetteOrdinateur_Y += INCREMENT_RAQUETTE;
            } else if (raquetteOrdinateur_Y > HAUT_TABLE) {
                raquetteOrdinateur_Y -= INCREMENT_RAQUETTE;
            }
            table.positionnerRaquetteOrdinateur_Y(
                raquetteOrdinateur_Y);

            // Etape 5. Mettre le score a jour si la balle est
            // dans la surface verte mais ne bouge plus.
            if (balleSurLaTable()) {
                if (balle_X > BALLE_X_MAX ) {
                    scoreOrdinateur++;
                    afficherScore();
                } else if (balle_X < BALLE_X_MIN) {

```


MoteurJeuPingPong amélioré

```

        scoreEnfant++;
        afficherScore();
    }
} // Fin du if balleServie
} // Fin du while
} // Fin de run()
// Sert depuis la position courante de la raquette
// de l'enfant.
private void serviceEnfant() {
    balleServie = true;
    balle_X = RAQUETTE_ENFANT_X - 1;
    balle_Y = raquetteEnfant_Y;
    if (balle_Y > HAUTEUR_TABLE / 2) {
        deplacementVertical = -1;
    } else {
        deplacementVertical = 1;
    }
    table.positionnerBalle(balle_X, balle_Y);
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}

private void afficherScore() {
    balleServie = false;
    if (scoreOrdinateur == SCORE_GAGNANT) {
        table.affecterTexteMessage("L'ordinateur a gagne ! " +
            scoreOrdinateur +
            " : " + scoreEnfant);
    } else if (scoreEnfant == SCORE_GAGNANT) {
        table.affecterTexteMessage ("Tu as gagne ! " +
            scoreEnfant +
            " : " + scoreOrdinateur);
    } else {
        table.affecterTexteMessage ("Ordinateur : " +
            scoreOrdinateur +
            " Enfant : " + scoreEnfant);
    }
}
// Verifie que la balle n'a pas depasse la limite
// inferieure ou superieure de la table.
private boolean balleSurLaTable() {
    if (balle_Y >= BALLE_Y_MIN && balle_Y <= BALLE_Y_MAX) {
        return true;
    } else {
        return false;
    }
}
}

```

X-C - Exercice pour les petits malins 1 : vitesse de la raquette

On peut s'y prendre de plusieurs manières pour ajuster le niveau de jeu. Ici, nous nous contenterons d'ajuster la vitesse de déplacement de la raquette du joueur en fonction du niveau.

Pour cela, la formule suivante est utilisée :

```
incrémentRaquetteJoueur = INCREMENT_RAQUETTE + 20 ? 2*niveauJoueur
```

Si l'on prend la valeur 2 pour INCREMENT_RAQUETTE, on aura notamment les valeurs suivantes :

niveauJoueur	incrementRaquetteJoueur
1	20
5	12
10	2

Ainsi, la vitesse sera la plus élevée pour le niveau 1 et la plus faible pour le niveau 10.

Dans le fichier ConstantesDeJeu, on élargit la hauteur du composant table verte, afin d'y ajouter un nouveau panneau pour le choix du niveau. Mais on y définit aussi les niveaux minimum et maximum.

ConstantesDuJeu

```
package ex_malins_1;
/**
 * Cette interface contient toutes les definitions des
 * variables invariantes utilisees dans le jeu.
 */
public interface ConstantesDuJeu {
    // Taille de la table de ping-pong
    public final int LARGEUR_TABLE = 320;
    public final int HAUTEUR_TABLE = 260;
    public final int HAUT_TABLE = 12;
    public final int BAS_TABLE = 180;
    // Increment du mouvement de la balle en pixels
    public final int INCREMENT_BALLE = 4;
    // Coordonnees maximum et minimum permises pour la balle
    public final int BALLE_X_MIN = 1 + INCREMENT_BALLE;
    public final int BALLE_Y_MIN = 1 + INCREMENT_BALLE;
    public final int BALLE_X_MAX =
        LARGEUR_TABLE - INCREMENT_BALLE;
    public final int BALLE_Y_MAX =
        HAUTEUR_TABLE - INCREMENT_BALLE;
    // Position de depart de la balle
    public final int BALLE_X_DEPART = LARGEUR_TABLE / 2;
    public final int BALLE_Y_DEPART = HAUTEUR_TABLE / 2;
    // Taille, positions et increment des raquettes
    public final int RAQUETTE_ENFANT_X = 300;
    public final int RAQUETTE_ENFANT_Y_DEPART = 100;
    public final int RAQUETTE_ORDINATEUR_X = 15;
    public final int RAQUETTE_ORDINATEUR_Y_DEPART = 100;
    public final int INCREMENT_RAQUETTE = 2;
    public final int LONGUEUR_RAQUETTE = 30;
    public final int LARGEUR_RAQUETTE = 5;
    public final int SCORE_GAGNANT = 21;
    // Ralentit mes ordinateurs rapides ;
    // modifier la valeur si necessaire.
    public final int DUREE_SOMMEIL = 10; // En millisecondes.
    // Valeurs minimale et maximale pour le niveau
    public final int NIVEAU_MIN = 1;
    public final int NIVEAU_MAX = 10;
}
```

On peut ajouter une méthode publique `fixerNiveauDeJeu()` dans la classe `MoteurDeJeu` afin de permettre à la classe `TableVertePingPong` de le modifier.

Dans la classe `TableVertePingPong`, on va ajouter une `JComboBox` permettant de choisir le niveau. On implémente l'interface `ItemListener` dans la classe `TableVertePingPong` afin de réagir au choix d'un élément de la `JComboBox`. Elle se compose de la méthode unique `itemStateChanged()`.

Vous remarquerez l'appel à

```
String.valueOf(i);
```

où i est de type int. Ceci permet simplement de convertir un entier en chaîne de caractères. La méthode valueOf est aussi surchargée pour accepter un boolean, un char...

Enfin, une précaution est prise, dans la méthode itemStateChanged(), pour vérifier que l'on réponde bien à un évènement de sélection et non de désélection : car quand on choisit un élément, l'écouteur de type ItemListener réagit deux fois consécutives. L'une pour la désélection de l'élément précédent et l'autre pour la sélection du nouvel élément. Ceci par le biais du code suivant :

```
if (event.getStateChange() == ItemEvent.SELECTED) {
```

où event est le paramètre de la méthode itemStateChanged().

Voici le code de la nouvelle version de TableVertePingPong.java :

TableVertePingPong

```
package ex_malins_1;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.BoxLayout;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.WindowConstants;
/**
 * Cette classe dessine la table de ping-pong, la balle et
 * les raquettes et affiche le score
 */
public class TableVertePingPong extends JPanel
implements ConstantesDuJeu, ItemListener {
    private MoteurJeuPingPong moteurJeu;
    private JLabel label;
    private int raquetteOrdinateur_Y =
        RAQUETTE_ORDINATEUR_Y_DEPART;
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    private int balle_X = BALLE_X_DEPART;
    private int balle_Y = BALLE_Y_DEPART;
    Dimension taillePreferee = new
        Dimension(LARGEUR_TABLE, HAUTEUR_TABLE);
    // Cette methode affecte sa taille au cadre.
    // Elle est appelee par Java.
    public Dimension getPreferredSize() {
        return taillePreferee;
    }
    // Constructeur. Cree un recepteur d'evenements souris.
    TableVertePingPong() {
        moteurJeu =
            new MoteurJeuPingPong(this);
        // Reçoit les mouvements de la souris pour deplacer la
        // raquette.
        addMouseListener(moteurJeu);
        // Reçoit les evenements clavier.
        addKeyListener(moteurJeu);
    }
    // Ajoute a un cadre la table, un JLabel,
    // et un ensemble JLabel + JComboBox
    void ajouteAuCadre(Container conteneur) {
        conteneur.setLayout(new BoxLayout(conteneur,
            BoxLayout.Y_AXIS));
        conteneur.add(this);
    }
}
```

TableVertePingPong

```

    label = new JLabel(
        "Taper N pour une nouvelle partie, S pour servir" +
        " ou Q pour quitter");
    conteneur.add(label);

    JPanel panneauChoixNiveau = new JPanel();
    panneauChoixNiveau.setLayout(new FlowLayout());
    panneauChoixNiveau.add(new JLabel("Niveau"));
    JComboBox controleNiveaux = new JComboBox();
    int i;
    for(i = NIVEAU_MIN; i <= NIVEAU_MAX; i++){
        controleNiveaux.addItem(String.valueOf(i));
    }
    controleNiveaux.addItemListener(this);
    panneauChoixNiveau.add(controleNiveaux);
    conteneur.add(panneauChoixNiveau);
}

// Repeint la fenetre. Cette methode est appelee par Java
// quand il est necessaire de rafraîchir l'ecran ou quand
// la methode repaint() est appelee.
public void paintComponent(Graphics contexteGraphique) {
    super.paintComponent(contexteGraphique);
    // Dessine la table verte
    contexteGraphique.setColor(Color.GREEN);
    contexteGraphique.fillRect(
        0, 0, LARGEUR_TABLE, HAUTEUR_TABLE);
    // Dessine la raquette droite
    contexteGraphique.setColor(Color.yellow);
    contexteGraphique.fillRect(
        RAQUETTE_ENFANT_X,
        raquetteEnfant_Y,
        LARGEUR_RAQUETTE, LONGUEUR_RAQUETTE);
    // Dessine la raquette gauche
    contexteGraphique.setColor(Color.blue);
    contexteGraphique.fillRect(RAQUETTE_ORDINATEUR_X,
        raquetteOrdinateur_Y,
        LARGEUR_RAQUETTE, LONGUEUR_RAQUETTE);
    // Dessine la balle
    contexteGraphique.setColor(Color.red);
    contexteGraphique.fillOval(balle_X, balle_Y, 10, 10);
    // Dessine les lignes blanches
    contexteGraphique.setColor(Color.white);
    contexteGraphique.drawRect(10, 10, 300, 200);
    contexteGraphique.drawLine(160, 10, 160, 210);
    // Donne le focus a la table, afin que le recepteur de
    // touches envoie les commandes a la table
    requestFocus();
}

// Affecte sa position courante a la raquette de l'enfant
public void positionnerRaquetteEnfant_Y(int y) {
    this.raquetteEnfant_Y = y;
    repaint();
}

// Retourne la position courante de la raquette de l'enfant
public int coordonneeRaquetteEnfant_Y() {
    return raquetteEnfant_Y;
}

// Affecte sa position courante a la raquette de
// l'ordinateur
public void positionnerRaquetteOrdinateur_Y(int y) {
    this.raquetteOrdinateur_Y = y;
    repaint();
}

// Affecte le texte du message du jeu
public void affecterTexteMessage(String texte) {
    label.setText(texte);
    repaint();
}

// Positionne la balle
public void positionnerBalle(int x, int y) {
    balle_X = x;

```

TableVertePingPong

```

    balle_Y = y;
    repaint();
}

public static void main(String[] args) {
    // Cree une instance du cadre
    JFrame monCadre = new JFrame("Table verte de ping-pong");
    // Permet la fermeture de la fenetre par clic sur la
    // petite croix dans le coin.
    monCadre.setDefaultCloseOperation(
        WindowConstants.EXIT_ON_CLOSE);
    TableVertePingPong table = new TableVertePingPong();
    table.ajouteAuCadre(monCadre.getContentPane());
    // Affecte sa taille au cadre et le rend visible.
    monCadre.setBounds(0, 0, LARGEUR_TABLE + 5,
        HAUTEUR_TABLE + 40);
    monCadre.setVisible(true);
}

public void itemStateChanged(ItemEvent event) {
    if (event.getStateChange() == ItemEvent.SELECTED) {
        String niveauChoisi = (String) event.getItem();
        moteurJeu.fixerNiveauDeJeu(Integer.valueOf(niveauChoisi));
    }
}
}

```

Voici la nouvelle version de MoteurJeuPingPong.java :



MoteurJeuPingPong

```

package ex_malins_1;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
/**
 * Cette classe est un recepteur de souris et de clavier.
 * Elle calcule les deplacements de la balle et des raquettes
 * et change leurs coordonnees.
 */
public class MoteurJeuPingPong implements Runnable,
    MouseMotionListener, KeyListener, ConstantesDuJeu {
    private TableVertePingPong table; // Reference a la table.
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    private int raquetteOrdinateur_Y =
        RAQUETTE_ORDINATEUR_Y_DEPART;
    private int scoreEnfant;
    private int scoreOrdinateur;
    private int balle_X; // position X de la balle
    private int balle_Y; // position Y de la balle
    private boolean deplacementGauche = true;
    private boolean balleServie = false;
    //Valeur en pixels du deplacement vertical de la balle.
    private int deplacementVertical;
    // Niveau de jeu : vaudra entre 1 et 10
    private int niveauDeJeu = 10;
    // Constructeur. Stocke une reference a la table.
    public MoteurJeuPingPong(TableVertePingPong tableVerte) {
        table = tableVerte;
        Thread travailleur = new Thread(this);
        travailleur.start();
    }
    // Methodes requises par l'interface MouseMotionListener
    // (certaines sont vides, mais doivent etre incluses dans
    // la classe de toute façon).
    public void mouseDragged(MouseEvent evenement) {
    }

    public void mouseMoved(MouseEvent evenement) {
    }
}

```

MoteurJeuPingPong

```

    int souris_Y = evenement.getY();
    // Si la souris est au-dessus de la raquette de l'enfant
    // et que la raquette n'a pas depasse le haut de la
    // table, la deplace vers le haut ;
    // sinon, la deplace vers le bas.
    if (souris_Y < raquetteEnfant_Y &&
        raquetteEnfant_Y > HAUT_TABLE) {
        raquetteEnfant_Y -= INCREMENT_RAQUETTE + 20 - 2*niveauDeJeu;
    } else if (raquetteEnfant_Y < BAS_TABLE) {
        raquetteEnfant_Y += INCREMENT_RAQUETTE + 20 - 2*niveauDeJeu;
    }
    // Affecte la nouvelle position de la raquette
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}
// Methodes requises par l'interface KeyListener.
public void keyPressed(KeyEvent evenement) {
    char touche = evenement.getKeyChar();
    if ('n' == touche || 'N' == touche) {
        demarrerNouvellePartie();
    } else if ('q' == touche || 'Q' == touche) {
        terminerJeu();
    } else if ('s' == touche || 'S' == touche) {
        if (scoreEnfant < SCORE_GAGNANT && scoreOrdinateur < SCORE_GAGNANT) {
            serviceEnfant();
        }
    }
}

public void keyReleased(KeyEvent evenement) {}
public void keyTyped(KeyEvent evenement) {}
// Demarre une nouvelle partie.
public void demarrerNouvellePartie() {
    scoreOrdinateur = 0;
    scoreEnfant = 0;
    table.affecterTexteMessage("Scores - Ordinateur : 0"
        + " Enfant : 0");
    serviceEnfant();
}
// Termine le jeu.
public void terminerJeu() {
    System.exit(0);
}

// La methode run() est requise par l'interface Runnable.
public void run() {
    boolean rebondPossible = false;
    while (true) {
        // Etape 4. Sommeiller un peu
        try {
            Thread.sleep(DUREE_SOMMEIL);
        } catch (InterruptedException exception) {
            exception.printStackTrace();
        }
        if (balleServie) { // Si la balle est en mouvement
            // Etape 1. La balle se deplace-t-elle vers la
            // gauche ?
            if (deplacementGauche && balle_X > BALLE_X_MIN) {
                rebondPossible = (balle_Y >= raquetteOrdinateur_Y
                    && balle_Y < (raquetteOrdinateur_Y +
                        LONGUEUR_RAQUETTE) ? true : false);
                balle_X -= INCREMENT_BALLE;
                // Ajoute un deplacement vertical a chaque
                // mouvement horizontal de la balle.
                balle_Y -= deplacementVertical;
                table.positionnerBalle(balle_X, balle_Y);
                // La balle peut-elle rebondir ?
                if (balle_X <= RAQUETTE_ORDINATEUR_X
                    && rebondPossible) {
                    deplacementGauche = false;
                }
            }
            // Etape 2. La balle se deplace-t-elle vers la

```

MoteurJeuPingPong

```
// droite ?
if (!deplacementGauche && balle_X <= BALLE_X_MAX) {
    rebondPossible = (balle_Y >= raquetteEnfant_Y &&
        balle_Y < (raquetteEnfant_Y +
            LONGUEUR_RAQUETTE) ? true : false);
    balle_X += INCREMENT_BALLE;
    table.positionnerBalle(balle_X, balle_Y);
    // La balle peut-elle rebondir ?
    if (balle_X >= RAQUETTE_ENFANT_X &&
        rebondPossible) {
        deplacementGauche = true;
    }
}

// Etape 3. Deplace la raquette de l'ordinateur vers le
// haut ou vers le bas pour bloquer la balle.

if (raquetteOrdinateur_Y < balle_Y
    && raquetteOrdinateur_Y < BAS_TABLE) {
    raquetteOrdinateur_Y += INCREMENT_RAQUETTE;
} else if (raquetteOrdinateur_Y > HAUT_TABLE) {
    raquetteOrdinateur_Y -= INCREMENT_RAQUETTE;
}
table.positionnerRaquetteOrdinateur_Y(
    raquetteOrdinateur_Y);
// Etape 5. Mettre le score a jour si la balle est
// dans la surface verte mais ne bouge plus.
if (balleSurLaTable()) {
    if (balle_X > BALLE_X_MAX) {
        scoreOrdinateur++;
        afficherScore();
    } else if (balle_X < BALLE_X_MIN) {
        scoreEnfant++;
        afficherScore();
    }
}
} // Fin du if balleServie
} // Fin du while
} // Fin de run()

public void fixerNiveauDeJeu(int niveauDeJeu){
    /*
     * On s'assure que le niveau de jeu
     * reste compris entre 1 et 10
     */
    niveauDeJeu = niveauDeJeu < 1 ? 1 : niveauDeJeu;
    niveauDeJeu = niveauDeJeu > 10 ? 10 : niveauDeJeu;

    /*
     * Ici, on doit utiliser le
     * mot clé this pour distinguer la variable locale
     * et la variable de classe.
     */
    this.niveauDeJeu = niveauDeJeu;
}

// Sert depuis la position courante de la raquette
// de l'enfant.
private void serviceEnfant() {
    balleServie = true;
    balle_X = RAQUETTE_ENFANT_X - 1;
    balle_Y = raquetteEnfant_Y;
    if (balle_Y > HAUTEUR_TABLE / 2) {
        deplacementVertical = -1;
    } else {
        deplacementVertical = 1;
    }
    table.positionnerBalle(balle_X, balle_Y);
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}
```

MoteurJeuPingPong

```
private void afficherScore() {
    balleServie = false;
    if (scoreOrdinateur == SCORE_GAGNANT) {
        table.affecterTexteMessage("L'ordinateur a gagne ! " +
            scoreOrdinateur +
            " : " + scoreEnfant);
    } else if (scoreEnfant == SCORE_GAGNANT) {
        table.affecterTexteMessage("Tu as gagne ! " +
            scoreEnfant +
            " : " + scoreOrdinateur);
    } else {
        table.affecterTexteMessage("Ordinateur : " +
            scoreOrdinateur +
            " Enfant : " + scoreEnfant);
    }
}

// Verifie que la balle n'a pas depasse la limite
// inferieure ou superieure de la table.
private boolean balleSurLaTable() {
    if (balle_Y >= BALLE_Y_MIN && balle_Y <= BALLE_Y_MAX) {
        return true;
    } else {
        return false;
    }
}
}
```

Une dernière remarque. J'ai utilisé l'**opérateur ternaire** pour affecter la variable niveauDeJeu :

```
niveauDeJeu = niveauDeJeu < 1 ? 1 : niveauDeJeu
```

Il vous a été expliqué dans le livre. Plus précisément, dans le code de **la version finale du jeu de ping pong** (Chapitre 11, section "Fin du jeu ping-pong"). Il est utilisé au début de la méthode run() de la classe MoteurJeuPingPong et l'auteur nous en explique le fonctionnement juste avant le code. Pour rappel cet opérateur permet "d'économiser" l'utilisation d'une structure if/else.

X-D - Exercice pour les petits malins 2 : direction du rebond de la balle

Pour résoudre cet exercice, il convient d'abord de se poser les questions suivantes :

- où se situe le milieu de la table ?
- où doit-on modifier le code afin de gérer le rebond ?

Pour répondre à la première question, nous allons déclarer une nouvelle constante : HAUTEUR_MILIEU_TABLE. Nous avons :

```
HAUTEUR_MILIEU_TABLE = (BAS_TABLE ? HAUT_TABLE) / 2
```

Voici donc la nouvelle version du fichier ConstantesDuJeu.java :

ConstantesDuJeu

```
package ex_malins_2;
/**
 * Cette interface contient toutes les definitions des
 * variables invariantes utilisees dans le jeu.
 */
public interface ConstantesDuJeu {
    // Taille de la table de ping-pong
    public final int LARGEUR_TABLE = 320;
```


ConstantesDuJeu

```

public final int HAUTEUR_TABLE = 260;
public final int HAUT_TABLE = 12;
public final int BAS_TABLE = 180;
public final int MILIEU_HAUTEUR_TABLE = (BAS_TABLE - HAUT_TABLE) / 2;
// Increment du mouvement de la balle en pixels
public final int INCREMENT_BALLE = 4;
// Coordonnees maximum et minimum permises pour la balle
public final int BALLE_X_MIN = 1 + INCREMENT_BALLE;
public final int BALLE_Y_MIN = 1 + INCREMENT_BALLE;
public final int BALLE_X_MAX =
    LARGEUR_TABLE - INCREMENT_BALLE;
public final int BALLE_Y_MAX =
    HAUTEUR_TABLE - INCREMENT_BALLE;
// Position de depart de la balle
public final int BALLE_X_DEPART = LARGEUR_TABLE / 2;
public final int BALLE_Y_DEPART = HAUTEUR_TABLE / 2;
// Taille, positions et increment des raquettes
public final int RAQUETTE_ENFANT_X = 300;
public final int RAQUETTE_ENFANT_Y_DEPART = 100;
public final int RAQUETTE_ORDINATEUR_X = 15;
public final int RAQUETTE_ORDINATEUR_Y_DEPART = 100;
public final int INCREMENT_RAQUETTE = 2;
public final int LONGUEUR_RAQUETTE = 30;
public final int LARGEUR_RAQUETTE = 5;
public final int SCORE_GAGNANT = 21;
// Ralentit mes ordinateurs rapides ;
// modifier la valeur si necessaire.
public final int DUREE_SOMMEIL = 10; // En millisecondes.
// Valeurs minimale et maximale pour le niveau
public final int NIVEAU_MIN = 1;
public final int NIVEAU_MAX = 10;
}

```

Quant à la gestion du rebond sur la raquette enfant, il fallait regarder dans la méthode `run()` de la classe `MoteurJeuPingPong.java`, étape 2. Pourquoi l'étape 2 ? Tout simplement parce que c'est lorsque que la balle se déplace vers la droite qu'elle est susceptible de rebondir sur la raquette de l'enfant.

Section de code pertinente

```

// Etape 2. La balle se deplace-t-elle vers la
// droite ?
if (!deplacementGauche && balle_X <= BALLE_X_MAX) {
    rebondPossible = (balle_Y >= raquetteEnfant_Y &&
        balle_Y < (raquetteEnfant_Y +
            LONGUEUR_RAQUETTE) ? true : false);
    balle_X += INCREMENT_BALLE;
    table.positionnerBalle(balle_X, balle_Y);
    // La balle peut-elle rebondir ?
    if (balle_X >= RAQUETTE_ENFANT_X &&
        rebondPossible) {
        deplacementGauche = true;
    }
}

```

Pour l'instant, le code se contente de renvoyer la balle dans la direction verticale qui a été donnée lors du service de l'enfant (cette variable n'est modifiée nulle part ailleurs que dans la méthode `serviceEnfant()`). Or, il faut que la direction change en fonction de la position de `raquette_Y`.

Nous allons donc ajouter un test `if()`, pour voir si la raquette de l'enfant se situe au-dessus du milieu de l'écran et modifier la direction verticale de la balle en conséquence.

```

// Etape 2. La balle se deplace-t-elle vers la
// droite ?
if (!deplacementGauche && balle_X <= BALLE_X_MAX) {
    rebondPossible = (balle_Y >= raquetteEnfant_Y &&
        balle_Y < (raquetteEnfant_Y +

```

```

        LONGUEUR_RAQUETTE) ? true : false);
    balle_X += INCREMENT_BALLE;
    table.positionnerBalle(balle_X, balle_Y);
    // La balle peut-elle rebondir ?
    if (balle_X >= RAQUETTE_ENFANT_X &&
        rebondPossible) {
        if (raquetteEnfant_Y < MILIEU_HAUTEUR_TABLE) {
            deplacementVertical = -1;
        }
        else {
            deplacementVertical = 1;
        }
        deplacementGauche = true;
    }
}

```

Voici donc le code complet de la nouvelle version de MoteurJeuPingPong.java :



MoteurJeuPingPong

```

package ex_malins_2;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
/**
 * Cette classe est un recepteur de souris et de clavier.
 * Elle calcule les déplacements de la balle et des raquettes
 * et change leurs coordonnees.
 */
public class MoteurJeuPingPong implements Runnable,
    MouseMotionListener, KeyListener, ConstantesDuJeu {
    private TableVertePingPong table; // Reference a la table.
    private int raquetteEnfant_Y = RAQUETTE_ENFANT_Y_DEPART;
    private int raquetteOrdinateur_Y =
        RAQUETTE_ORDINATEUR_Y_DEPART;
    private int scoreEnfant;
    private int scoreOrdinateur;
    private int balle_X; // position X de la balle
    private int balle_Y; // position Y de la balle
    private boolean deplacementGauche = true;
    private boolean balleServie = false;
    //Valeur en pixels du deplacement vertical de la balle.
    private int deplacementVertical;
    // Niveau de jeu : vaudra entre 1 et 10
    private int niveauDeJeu = 10;
    // Constructeur. Stocke une reference a la table.
    public MoteurJeuPingPong(TableVertePingPong tableVerte) {
        table = tableVerte;
        Thread travailleur = new Thread(this);
        travailleur.start();
    }
    // Methodes requises par l'interface MouseMotionListener
    // (certaines sont vides, mais doivent etre incluses dans
    // la classe de toute façon).
    public void mouseDragged(MouseEvent evenement) {
    }

    public void mouseMoved(MouseEvent evenement) {
        int souris_Y = evenement.getY();
        // Si la souris est au-dessus de la raquette de l'enfant
        // et que la raquette n'a pas depasse le haut de la
        // table, la deplace vers le haut ;
        // sinon, la deplace vers le bas.
        if (souris_Y < raquetteEnfant_Y &&
            raquetteEnfant_Y > HAUT_TABLE) {
            raquetteEnfant_Y -= INCREMENT_RAQUETTE + 20 - 2*niveauDeJeu;
        } else if (raquetteEnfant_Y < BAS_TABLE) {
            raquetteEnfant_Y += INCREMENT_RAQUETTE + 20 - 2*niveauDeJeu;
        }
    }
}

```

MoteurJeuPingPong

```
// Affecte la nouvelle position de la raquette
table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}
// Methodes requises par l'interface KeyListener.
public void keyPressed(KeyEvent evenement) {
    char touche = evenement.getKeyChar();
    if ('n' == touche || 'N' == touche) {
        demarrerNouvellePartie();
    } else if ('q' == touche || 'Q' == touche) {
        terminerJeu();
    } else if ('s' == touche || 'S' == touche) {
        if (scoreEnfant < SCORE_GAGNANT && scoreOrdinateur < SCORE_GAGNANT) {
            serviceEnfant();
        }
    }
}

public void keyReleased(KeyEvent evenement) {}
public void keyTyped(KeyEvent evenement) {}
// Demarre une nouvelle partie.
public void demarrerNouvellePartie() {
    scoreOrdinateur = 0;
    scoreEnfant = 0;
    table.afficherTexteMessage("Scores - Ordinateur : 0"
        + " Enfant : 0");
    serviceEnfant();
}

// Termine le jeu.
public void terminerJeu() {
    System.exit(0);
}

// La methode run() est requise par l'interface Runnable.
public void run() {
    boolean rebondPossible = false;
    while (true) {
        // Etape 4. Sommeiller un peu
        try {
            Thread.sleep(DUREE_SOMMEIL);
        } catch (InterruptedException exception) {
            exception.printStackTrace();
        }
        if (balleServie) { // Si la balle est en mouvement
            // Etape 1. La balle se deplace-t-elle vers la
            // gauche ?
            if (deplacementGauche && balle_X > BALLE_X_MIN) {
                rebondPossible = (balle_Y >= raquetteOrdinateur_Y
                    && balle_Y < (raquetteOrdinateur_Y +
                        LONGUEUR_RAQUETTE)) ? true : false);
                balle_X -= INCREMENT_BALLE;
                // Ajoute un deplacement vertical a chaque
                // mouvement horizontal de la balle.
                balle_Y -= deplacementVertical;
                table.positionnerBalle(balle_X, balle_Y);
                // La balle peut-elle rebondir ?
                if (balle_X <= RAQUETTE_ORDINATEUR_X
                    && rebondPossible) {
                    deplacementGauche = false;
                }
            }
            // Etape 2. La balle se deplace-t-elle vers la
            // droite ?
            if (!deplacementGauche && balle_X <= BALLE_X_MAX) {
                rebondPossible = (balle_Y >= raquetteEnfant_Y &&
                    balle_Y < (raquetteEnfant_Y +
                        LONGUEUR_RAQUETTE)) ? true : false);
                balle_X += INCREMENT_BALLE;
                table.positionnerBalle(balle_X, balle_Y);
                // La balle peut-elle rebondir ?
                if (balle_X >= RAQUETTE_ENFANT_X &&
                    rebondPossible) {

```

MoteurJeuPingPong

```

        if (raquetteEnfant_Y < MILIEU_HAUTEUR_TABLE) {
            deplacementVertical = -1;
        }
        else {
            deplacementVertical = 1;
        }
        deplacementGauche = true;
    }

    // Etape 3. Deplace la raquette de l'ordinateur vers le
    // haut ou vers le bas pour bloquer la balle.

    if (raquetteOrdinateur_Y < balle_Y
        && raquetteOrdinateur_Y < BAS_TABLE) {
        raquetteOrdinateur_Y += INCREMENT_RAQUETTE;
    } else if (raquetteOrdinateur_Y > HAUT_TABLE) {
        raquetteOrdinateur_Y -= INCREMENT_RAQUETTE;
    }
    table.positionnerRaquetteOrdinateur_Y(
        raquetteOrdinateur_Y);
    // Etape 5. Mettre le score a jour si la balle est
    // dans la surface verte mais ne bouge plus.
    if (balleSurLaTable()) {
        if (balle_X > BALLE_X_MAX ) {
            scoreOrdinateur++;
            afficherScore();
        } else if (balle_X < BALLE_X_MIN) {
            scoreEnfant++;
            afficherScore();
        }
    }
    } // Fin du if balleServie
    } // Fin du while
} // Fin de run()

public void fixerNiveauDeJeu(int niveauDeJeu) {
    /*
     * On s'assure que le niveau de jeu
     * reste compris entre 1 et 10
     */
    niveauDeJeu = niveauDeJeu < 1 ? 1 : niveauDeJeu;
    niveauDeJeu = niveauDeJeu > 10 ? 10 : niveauDeJeu;

    /*
     * Ici, on doit utiliser le
     * mot clé this pour distinguer la variable locale
     * et la variable de classe.
     */
    this.niveauDeJeu = niveauDeJeu;
}

// Sert depuis la position courante de la raquette
// de l'enfant.
private void serviceEnfant() {
    balleServie = true;
    balle_X = RAQUETTE_ENFANT_X - 1;
    balle_Y = raquetteEnfant_Y;
    if (raquetteEnfant_Y > HAUTEUR_TABLE / 2) {
        deplacementVertical = +1;
    } else {
        deplacementVertical = -1;
    }
    table.positionnerBalle(balle_X, balle_Y);
    table.positionnerRaquetteEnfant_Y(raquetteEnfant_Y);
}

private void afficherScore() {
    balleServie = false;
    if (scoreOrdinateur == SCORE_GAGNANT) {
        table.affecterTexteMessage("L'ordinateur a gagne ! " +

```

MoteurJeuPingPong

```

        scoreOrdinateur +
        " : " + scoreEnfant);
    } else if (scoreEnfant == SCORE_GAGNANT) {
        table.affecterTexteMessage ("Tu as gagne ! " +
        scoreEnfant +
        " : " + scoreOrdinateur);
    } else {
        table.affecterTexteMessage ("Ordinateur : " +
        scoreOrdinateur +
        " Enfant : " + scoreEnfant);
    }
}
// Verifie que la balle n'a pas depasse la limite
// inferieure ou superieure de la table.
private boolean balleSurLaTable() {
    if (balle_Y >= BALLE_Y_MIN && balle_Y <= BALLE_Y_MAX) {
        return true;
    } else {
        return false;
    }
}
}
}

```

Pour tester le package depuis Eclipse (clic droit sur son nœud → RunAs → JavaApplication), il vous faut copier la source TableVertePingPong du package de l'exercice 1 des petits malins (l'exercice précédent).

X-E - Synthèse

Les exercices de ce chapitre vous ont permis :

- d'encapsuler une variable de classe et de modifier les autres classes en conséquence ;
- de résoudre un bogue simple ;
- d'ajuster le niveau de jeu ;
- de résoudre un bogue moins évident.

XI - Remerciements

Je tiens à remercier les responsables Java [keulkeul](#), [mlny84](#) et le modérateur [Robin56](#) pour leurs relectures et les conseils qu'ils m'ont donné.

Je tiens également à remercier [jacques_jean](#) pour sa relecture très précise.

Je tiens aussi à remercier le rédacteur [TheSeb](#) et le modérateur [le y@m's](#) pour leurs aides quant à la gabarisation de l'article.

Je tiens à remercier les auteurs des différentes ressources et outils disponibles facilitant la rédaction d'articles : notamment [djibril](#) qui m'a assisté dans l'utilisation du kit developpez.

Enfin, je tiens à remercier les gérants du site developpez.com de m'avoir permis d'héberger mon article.