UNIVERSITE DE CAEN **13 nov. 2001**

UFR SCIENCES
DESS NAPI 2000 -2001
Josiane LEFRANÇOIS

DEVOIR JAVA : ECHIQUIER

1.Problème et fonctionnalités:

L'objectif est de réaliser une **applet** représentant un j**eu d'échecs.**
Le jeu est géré par les deux joueus blancs et noirs qui s'affrontent.

L'interface graphique est conçue avec des composants javax.Swing.
Les événements du jeu sont déclenchés par un clic sur les cases.
Les informations concernant le déroulement du jeu(tour , erreurs, nombre de coups joués, pièces prises, échec*) s'affichent sur l'interface.Un bouton « Rejouer » permet de relancer une partie.

Roi allant être pris

2.Diagramme de classes

La gestion de l'Applet et des éléments graphiques est réalisée dans la classe **AppletEchecs** héritant de JApplet.
Cette classe comprend une classe interne , la classe **Case** héritant des fonctionnalités de JButton .

La gestion du jeu se fait dans la classe **Board** , représentant un échiquier fonctionnel composé d'un tableau de pièces.

Les **pièces** héritent toutes d'une **classe abstraite** où sont définies leurs propriétés et méthodes.
Une classe **Position** permet de récupérer les coordonnées des pièces pour gérer les déplacements.
Une classe **EchiquierException** héritant d'Exception permet de gérer les erreurs de jeu.

3.L'interface graphique

La classe **AppletEchecs** représente **graphiquement l'échiquier** avec des composants Swing.

L'initialisation de l'applet provoque le chargement des images représentant les pièces. On utilise la méthode getURL pour renvoyer l'URL de l'image à partir de l'adresse de chargement.

Les ImageIcons crées sont stockées dans un tableau d'Icons indéxé par le type de la piece et sa couleur, ce qui permet de les retrouver facilement. Elles sont ensuite utilisées dans la méthode setPiece de Case pour s'afficher sur l'échiquier.

L'interface est composée d'un JPanel *fond* dont les différents éléments sont organisés par un BorderLayout manager.

Les cases sont stockées dans un tableau pour en faciliter l'accès.

Le JPanel *echiquier* est un gridLayout dont les dimensions sont fixées par les constantes HEIGHT et WIDTH.

Un JPanel *annexe* est ajouté pour contenir les différents éléments d'information

Les différents composants JTextField, JTextArea,JButton , JScrollPane sont ensuite ajoutés à *annexe*.

Le bouton « *Rejouer* « permet de rédémarrer le jeu.

La méthode setEnabledCases est utilisée au cas où la partie précédente se serait terminée par un ECHEC.

Au démarrage de l'Applet **une instance de Board** est crée.L'affichage des cases et des différents éléments d'information est mis à) jour.

La **case** est caractérisée par ses coordonnées ligne et colonne correspondant aux coordonnées de board.

La méthode **jeuEnCours** de Board permet de renseigner l'état du jeu, c'est à dire si une tentative de déplacement est en cours ou non.

Si ce n'est pas le cas, le clic sur une case permet de récupérer les coordonnées de la case origine et lance la méthode **coup** de Board.

En revanche si une tentative de déplacement est déjà en cours on récupère les coordonnées de la **case de destination**, ce qui engrenge les méthodes **deplace** de Board et la **mise à jour de l'échiquier graphique.**

La méthode selectionneCase trace un bord blanc ou noir(selon la couleur de pièce origine) autour de la case sélectionnée.

Les messages déclenchés par les **exceptions générées** dans coup sont affichés dans la zone de texte info.

Si le déplacement de la pièce a provoqué **une prise adverse**, la prise est enregistrée dans la zone de texte correspondante.

4.La gestion du jeu

La classe **Board est un échiquier fonctionnel** représenté par un **tableau de pièces 10 x 10** correspondant aux 8x 8 pièces de l'échiquier plus une rangée de bords.

L'initialisation de Board **positionne les pièces sur l'échiquier**.

Des méthodes permettent d 'accéder aux propriétés des pièces.

Le jeu est caractérisé par son **état,** initialisé à false,le **tour,** initialisé à blancs et le **nombre de coups joués** initialisé à 0.

Des méthodes permettent d 'accéder à ces attributs notamment des classes extérieures.

La méthode **joueurSuivant** permet d'alterner le tour de jeu entre les blancs et les noirs.

La **tentative de déplacement d'une pièce** est prise en charge par la méthode **coup**.

Cette méthode vérifie que la piece n'est pas une pièce vide et que le tour de jeu est correct.

Les coordonnées de la case cliquée permettent de récupérer l'instance de la pièce et sa couleur. **Les possibilités de déplacement** de la pièce sur le tableau de pièces sont récupérées sous la forme d'un i**térator.**

La méthode hasNext() de Iterator liste ses différents éléments. Elle retourne false si l'Iterator est vide et donc le déplacement impossible ce qui génère une exception « pièce bloquée ».

Si coup n'a pas généré d'exceptions , le jeu est en cours.

La méthode **deplace** compare les positions stockées dans l'iterator de la piece origine avec la position de la pièce de destination.

Si la pièce de destination est le **roi,** il y a echec et le jeu s'arrête.

Sinon le tableau de pièces est mis à jour, le tour passe au joueur suivant et l'état du jeu à false.

Si une piece adverse est prise, elle est stockée dans la variable piecesPrisesB ou N qui ensuite ajoutée à la zone de texte de l'interface graphique.

5.Les pieces

La classe **Piece** est une **classe abstraite dont héritent** les classes Roi, Reine, Cavalier,Fou, Tour, Pion, PieceVide et Bord.

Une pièce est caractérisée par sa **couleur** et son **type,** accessibles aux classes filles.

Les types et les couleurs des pieces sont des **constantes** ce qui en facilite l'accès aux autres classes.

Toutes les constantes sont stockées dans la classe Piece.

Des méthodes permettent d'accéder à la couleur et au type de la pièce.

La méthode toString renvoie le nom de la pièce en fonction de son type et de sa couleur.

Les pièces reine , fou et tour peuvent se déplacer horizontalement, verticalement ou en diagonale sur l'échiquier de pièces.

Ces méthodes sont donc stockées dans la classe Piece.

La méthode **deplacement est abstraite** et redéfinie dans les classes filles.

Les possitions accessibles sont stockées dans des **ArrayList.** L'intérêt des ArrayList est d'avoir une taille modifiable comme Vector mais d'être plus rapide car non synchronisés.

La méthode **iterarator** retourne un Iterator dont les méthodes hasNext et next permettent de lister facilement les éléments.

Les déplacements sont calculés par rapport aux limites de l'échiquier, aux pieces vides et aux pieces adverses.Lorsque la piece correspond aux critères, une **nouvelle position** avec ses coordonnées est ajoutée à l'arrayList.

Pour les pions , on prend en compte le nombre de tours puisque le déplacement de deux cases n'est possible qu'au premier tour.

6.Scripts

Les fichiers se trouvent à l'adresse users.info.unicaen.fr/~jlefranc/java/echecs

1Classe AppletEchecs

```
import java.awt.* ;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.net.*;

public class AppletEchecs extends JApplet {

   //attributs

   private JPanel fond, echiquier, annexe;
```

```
private Icon [][] images = new Icon[6][2];

private JTextField info, coups;
private JButton rejouer;
private JTextArea prisesB, prisesN;
private JScrollPane piecesPrisesB, piecesPrisesN;

final static Color CLAIR = new Color(255,250,240);
final static Color FONCE = new Color(211,211,211);

static final int WIDTH=350 , HEIGHT = 350;
private Board board;

private int l,c, ld, cd;
private  Case [][] tabCases;


public void init () {

        //Chargement des images et stockage dans le tableau images

           images [Piece.ROI][Piece.BLANC] = new ImageIcon(getURL(getCodeBase(),"images/roiB.gif"));
           images [Piece.ROI][Piece.NOIR] = new ImageIcon(getURL(getCodeBase(),"images/roiN.gif"));
           images [Piece.REINE][Piece.BLANC] = new ImageIcon(getURL(getCodeBase(),"images/reineB.gif"));
           images [Piece.REINE][Piece.NOIR] = new ImageIcon(getURL(getCodeBase(),"images/reineN.gif"));
           images [Piece.CAV][Piece.BLANC] = new ImageIcon(getURL(getCodeBase(),"images/cavalierB.gif"));
           images [Piece.CAV][Piece.NOIR] = new ImageIcon(getURL(getCodeBase(),"images/cavalierN.gif"));
           images [Piece.TOUR][Piece.BLANC] = new ImageIcon(getURL(getCodeBase(),"images/tourB.gif"));
           images [Piece.TOUR][Piece.NOIR] = new ImageIcon(getURL(getCodeBase(),"images/tourN.gif"));
           images [Piece.FOU][Piece.BLANC] = new ImageIcon(getURL(getCodeBase(),"images/fouB.gif"));
           images [Piece.FOU][Piece.NOIR] = new ImageIcon(getURL(getCodeBase(),"images/fouN.gif"));
           images [Piece.PION][Piece.BLANC] = new ImageIcon(getURL(getCodeBase(),"images/pionB.gif"));
           images [Piece.PION][Piece.NOIR] = new ImageIcon(getURL(getCodeBase(),"images/pionN.gif"));


           /**************JPanel fond*****************/
           fond = new JPanel(new BorderLayout());
           fond.setBackground(CLAIR);
           setContentPane(fond);

          /**************JPanel echiquier*****************/
           echiquier = new JPanel(new GridLayout(8,8));

           // Création du tableau de cases
           tabCases = new Case[10][10];

           for(int i=0; i<10; i++)
                   for(int j=0; j<10; j++)
                      if((i%2 == 0 && j%2 == 0) ||(i%2 != 0 && j%2 != 0))
                              tabCases[i][j] = new Case(i,j,CLAIR);
                      else {
                              tabCases[i][j]= new Case(i,j,FONCE);
                   }

           //ajoute les cases au gridLayout
           for(int i=1; i<9; i++)
                   for(int j=1; j<9; j++)
                      echiquier.add(tabCases[i][j]);

        echiquier.setBorder(BorderFactory.createLineBorder(Color.black,5));
        echiquier.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        fond.add(echiquier, BorderLayout.CENTER);
```

```
/**************JPanel annexe*****************/
annexe = new JPanel();
fond.add(annexe, BorderLayout.EAST);
annexe.setBackground(CLAIR);
annexe.setPreferredSize(new Dimension(WIDTH-100,HEIGHT));
annexe.setOpaque(true);

/**************JTextField info*****************/
info = new JTextField("Les Blancs commencent",20);
info.setBorder(BorderFactory.createCompoundBorder());
info.setHorizontalAlignment(JTextField.CENTER);
info.setBackground(Color.black);
info.setForeground(Color.white);
info.setFont(new Font("Sanserif",Font.BOLD,14));
info.setBorder(BorderFactory.createCompoundBorder());
annexe.add(info);

/**************JTextField coups*****************/

coups = new JTextField("",10);
coups.setBackground(FONCE);
coups.setBorder(BorderFactory.createCompoundBorder());
coups.setHorizontalAlignment(JTextField.CENTER);
annexe.add(coups);

/**************JScrollPane piecesPrises*****************/
prisesB = new JTextArea("Pièces prises\n");
prisesB.setAlignmentY(JComponent.TOP_ALIGNMENT);
annexe.add(prisesB);

piecesPrisesB = new JScrollPane(prisesB);
piecesPrisesB.setBackground(CLAIR);
piecesPrisesB.setPreferredSize(new Dimension(150,150));
piecesPrisesB.setBorder(BorderFactory.createCompoundBorder());
annexe.add(piecesPrisesB);

prisesN = new JTextArea("Pièces prises\n");
prisesN.setAlignmentY(JComponent.TOP_ALIGNMENT);
annexe.add(prisesN);

piecesPrisesN = new JScrollPane(prisesN);
piecesPrisesN.setBackground(CLAIR);
piecesPrisesN.setPreferredSize(new Dimension(150,150));
piecesPrisesN.setBorder(BorderFactory.createCompoundBorder());
annexe.add(piecesPrisesN);

/**************Bouton rejouer*****************/
rejouer = new JButton("Rejouer");
rejouer.setBackground(FONCE);
rejouer.setForeground(Color.black);
rejouer.setBorder(BorderFactory.createCompoundBorder());
rejouer.setPreferredSize(new Dimension(100,20));
rejouer.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            start();
            setEnabledCases(true);
            for(int i=1; i<9; i++)
                for(int j=1; j<9; j++)
                    tabCases[i][j].setBorder(BorderFactory.createCompoundBorder());
        }
   });
```

```
        annexe.add(rejouer);
    }

    public void start() {

        // Création d'un nouveau tableau de pièces
        board = new Board();

        for(int i=0; i<10; i++)
                for(int j=0; j<10; j++)
                        tabCases[i][j].setPiece(board.getType(i,j), board.getCouleur(i,j)) ;
        info.setText("les Blancs commencent");
        coups.setText(board.nCoups());
        prisesB.setText("Pieces Blanches prises");
        prisesN.setText("Pieces Noires prises");

        repaint();
    }

    public void stop() {
        super.stop();
    }

    public void destroy() {

        super.destroy();
    }

    private URL getURL(URL codeBase, String filename){
        URL url = null;
        try{
            url = new URL(codeBase, filename);
        }
        catch(MalformedURLException e){
            System.err.println(filename+" invalide");
            return null;
        }
        return url;
    }

    public void miseAJour(int l, int c, int ld, int cd, Board board){
        tabCases[l][c].setPiece(board.getType(l,c), board.getCouleur(l,c)) ;
        tabCases[l][c].setBorder(BorderFactory.createCompoundBorder());
        tabCases[ld][cd].setPiece(board.getType(ld,cd), board.getCouleur(ld,cd)) ;
        repaint();
    }

    public void setEnabledCases(boolean etat){
        for(int i=1; i<9; i++)
            for(int j=1; j<9; j++)
                    tabCases[i][j].setEnabled(etat);
    }


    /*****************************Classe interne*************************************/

    class Case extends JButton implements ActionListener {

        private int ligne, col;

        // Construit une case avec son numéro de ligne, de colonne et sa couleur
```

```
public Case(int _ligne, int _col ,Color couleur) {
    super();
    ligne = _ligne;
    col = _col;
    setBackground(couleur);
    setBorder(BorderFactory.createCompoundBorder());
    addActionListener(this);
}
//Affecte une piece à la case en fonction du type et de la couleur de la pièce
public void setPiece(int type, int couleur) {
    switch(type) {
    case Piece.ROI :
            if(couleur == Piece.BLANC) setIcon(images[Piece.ROI][Piece.BLANC]);
            if(couleur == Piece.NOIR) setIcon(images[Piece.ROI][Piece.NOIR]);
            break;
    case Piece.REINE :
            if(couleur == Piece.BLANC) setIcon(images[Piece.REINE][Piece.BLANC]);
            if(couleur == Piece.NOIR) setIcon(images[Piece.REINE][Piece.NOIR]);
            break;
    case Piece.CAV :
            if(couleur == Piece.BLANC) setIcon(images[Piece.CAV][Piece.BLANC]);
            if(couleur == Piece.NOIR) setIcon(images[Piece.CAV][Piece.NOIR]);
            break;
    case Piece.TOUR :
            if(couleur == Piece.BLANC) setIcon(images[Piece.TOUR][Piece.BLANC]);
            if(couleur == Piece.NOIR) setIcon(images[Piece.TOUR][Piece.NOIR]);
            break;
    case Piece.FOU :
            if(couleur == Piece.BLANC) setIcon(images[Piece.FOU][Piece.BLANC]);
            if(couleur == Piece.NOIR) setIcon(images[Piece.FOU][Piece.NOIR]);
            break;
    case Piece.PION :
            if(couleur == Piece.BLANC) setIcon(images[Piece.PION][Piece.BLANC]);
            if(couleur == Piece.NOIR) setIcon(images[Piece.PION][Piece.NOIR]);
            break;
    default :
            setIcon(null);
    }
}
//retourne la ligne de la case
public int getL(){
    return ligne;
}
//retourne la colone de la case
public int getC(){
    return col;
}

//dessine un bord autour de la case sélectionnée si la case n'est pas vide
public void selectionneCase(int l,int c) {

    switch(board.getCouleur(l,c)){
            case Piece.BLANC : setBorder(BorderFactory.createLineBorder(Color.white ,3));break;
            case Piece.NOIR : setBorder(BorderFactory.createLineBorder(Color.black ,3));break;
            default: break;
            }
}

public void paintComponent(Graphics g){
    super.paintComponent(g);
}
```

```
        public void actionPerformed (ActionEvent evt) {
            info.setText("");
            if(!board.jeuEnCours()){

                    if(!board.jeuEnCours()){
                        l = ((Case)evt.getSource()).getL();
                        c = ((Case)evt.getSource()).getC();
                        try{
                        board.coup(l,c);
                        selectionneCase(l,c);
                        }
                        catch(EchiquierException e){
                                info.setText(e.getMessage());
                        }
                    }
            }
            else {
                    ld = ((Case)evt.getSource()).getL();
                    cd = ((Case)evt.getSource()).getC();
                    try {
                    board.deplace(l,c,ld,cd);
                    miseAJour(l,c,ld,cd,board);
                    info.setText(board.afficheTour());
                    coups.setText(board.nCoups());
                    if(board.tour()==Piece.BLANC)
                        prisesB.append(board.piecesPrisesB()+"\n");
                    else {
                        prisesN.append(board.piecesPrisesN()+"\n");
                    }
                    }
                    catch(EchiquierException e1){
                        info.setText(e1.getMessage());
                        setEnabledCases(false);
                    }
            }
        }
    }
}
```

2Classe Board

```
import jamva.util.*;

Echiquier fonctionnel repésenté
par un tableau de pièces

public class Board {

    private Piece board [][];
    private Iterator iter;
    private boolean jeuEnCours = false;
    private int tour = Piece.BLANC;
    private String piecesPrisesB="" ;
    private String piecesPrisesN="";

    private int nbCoups = 0;

    // Constructeur
    public Board () {
            board = new Piece [10][10];
```

```
        init();
    }
    //Méthodes d'affectation des pieces au tableau
    public void setPiece(int l, int c, Piece piece){
        board[l][c] = piece;
    }

    public void setPieceVide(int l, int c){
        board[l][c]= new PieceVide(Piece.VIDE);
    }

    //Positionnement initial des pièces
    public void init(){
        setPiece(8,5,new Roi(Piece.BLANC));
        setPiece(1,5,new Roi(Piece.NOIR));
        setPiece(8,4,new Reine(Piece.BLANC));
        setPiece(1,4,new Reine(Piece.NOIR));
        setPiece(8,3,new Fou(Piece.BLANC));
        setPiece(8,6,new Fou(Piece.BLANC));
        setPiece(1,3,new Fou(Piece.NOIR));
        setPiece(1,6,new Fou(Piece.NOIR));
        setPiece(8,2,new Cavalier(Piece.BLANC));
        setPiece(8,7,new Cavalier(Piece.BLANC));
        setPiece(1,2,new Cavalier(Piece.NOIR));
        setPiece(1,7,new Cavalier(Piece.NOIR));
        setPiece(8,1,new Tour(Piece.BLANC));
        setPiece(8,8,new Tour(Piece.BLANC));
        setPiece(1,1,new Tour(Piece.NOIR));
        setPiece(1,8,new Tour(Piece.NOIR));

        //pions
        for(int j = 1; j< 9 ; j++)
           setPiece(7,j,new Pion(Piece.BLANC)) ;
        for(int j = 1; j< 9 ; j++)
           setPiece(2,j,new Pion(Piece.NOIR)) ;

        //vides
        for(int i=3; i<7; i++)
            for(int j=1; j<9; j++)
                    setPieceVide(i,j);

        // bords
        for(int j=0; j<board.length; j++){
        setPiece(0,j,new Bord(Piece.BORD));
        setPiece(9,j,new Bord(Piece.BORD));
        }
        for(int i=0; i<board.length; i++){
         setPiece(i,0,new Bord(Piece.BORD));
          setPiece(i,9,new Bord(Piece.BORD));
        }
    }

    // Méthode permettant d'accéder aux pieces

    public Piece getPiece (int l, int c) {
        return board[l][c];
    }

    public int getCouleur (int l , int c) {
        return board[l][c].getCouleur();
    }
```

```
    public int getType (int l , int c) {
            return board[l][c].getType();
    }

    public Piece getRoi(int couleur){

            for(int i=1; i<9; i++)
               for(int j=1 ; j<9; j++)
                       if(getPiece(i,j) instanceof Roi) return board[i][j];
            return null;
    }
    //retourne vrai si piece vide
    public boolean pieceVide(int l, int c){
            return(board[l][c] instanceof PieceVide);
    }
    // retourne vrai si bord
    public boolean bord(int l, int c){
            return(board[l][c] instanceof Bord);
    }
    //retourne vrai si piece de couleur adverse
    public boolean pieceAdverse(int l, int c, int couleur) {
            if (couleur==Piece.BLANC)
            return (getCouleur(l,c) == Piece.NOIR) ;
            if (couleur==Piece.NOIR)
            return (getCouleur(l,c) == Piece.BLANC) ;
            return false;
    }

    public void joueurSuivant(){
            tour = (tour==Piece.BLANC ? Piece.NOIR : Piece.BLANC);
    }

    public void coup(int l, int c) throws EchiquierException {
            if(pieceVide(l,c)) throw new EchiquierException("case vide");

            if(tour == Piece.BLANC && getCouleur(l,c)==Piece.BLANC || tour == Piece.NOIR &&
getCouleur(l,c)==Piece.NOIR){
               iter = getPiece(l,c).deplacement(this,l,c,getCouleur(l,c));
               if(iter.hasNext()==false) throw new EchiquierException("Pièce bloquée");
               else{
               jeuEnCours = true;
               }
            }
            else {
               if(tour==Piece.BLANC)
                       throw new EchiquierException("les blancs jouent");
               else {
                       throw new EchiquierException("les noirs jouent");
               }
            }
    }

    public void deplace (int l, int c, int ld, int cd) throws EchiquierException {
            Position dest = new Position(ld,cd);

                    while(iter.hasNext()){
                       Position p = (Position)iter.next();
                       if(ld==p.getL()&&cd==p.getC()){

                               if(getPiece(ld,cd) instanceof Roi)
                                  throw new EchiquierException("ECHEC");
                               if(getCouleur(ld,cd)!=tour)
```

```
                                if(tour==Piece.NOIR)
                                        piecesPrisesB = getPiece(ld,cd).toString();
                                else {
                                        piecesPrisesN = getPiece(ld,cd).toString();
                                }

                                miseAJour(l,c,ld,cd,getPiece(l,c));
                                jeuEnCours=false;
                                joueurSuivant();
                        }
                        else {
                                jeuEnCours = false;
                        }
                }
    }

    public boolean jeuEnCours(){
            return jeuEnCours;
    }

    public int tour(){
            return tour;
    }

    public String afficheTour(){
            if(tour==Piece.BLANC) return "Aux blancs de jouer";
            if(tour==Piece.NOIR) return "Aux noirs de jouer";
            return "";
    }

    public String nCoups(){
            return "Coups joués: "+nbCoups;
    }


    //retourne le nombre de coups joués
    public int getCoups(){
            return nbCoups;
    }

    public void miseAJour(int l, int c, int ld, int cd, Piece piece){
            setPiece(ld, cd,piece);
            setPieceVide(l,c);
            nbCoups ++;
    }

    public String piecesPrisesB(){
            return piecesPrisesB;
    }

    public String piecesPrisesN(){
            return piecesPrisesN;
    }
}
```

3Classe Piece

```
import java.util.*;

public  abstract class Piece {
```

```
//Une piece est caractérisée par sa couleur et son type
//(roi, reine ....)
// ces attributs sont accessibles aux classes filles

protected int couleur;
protected int type;

//Constantes
static final int VIDE = -1;
static final int BORD = -2;
static final int ROI = 0;
static final int REINE = 1;
static final int CAV = 2;
static final int TOUR = 3;
static final int FOU = 4;
static final int PION = 5;
final static int BLANC = 0;
final static int NOIR = 1;

// constructeur
public Piece(int couleur) {
	this.couleur = couleur;
}

// constructeur de recopie
public Piece(Piece piece) {
	this.couleur = piece.couleur;
	this.type = piece.type;
}

//retourne la couleur de la piece
protected int getCouleur() {
	if (couleur== BLANC) return BLANC;
	if (couleur== NOIR) return NOIR;
	return VIDE;
}

//retourne le type de la piece
protected int getType() {
	return type;
}

public String toString(){

	switch(type){
	case Piece.ROI :
	   if(couleur==Piece.BLANC) return "Roi Blanc";
	   if(couleur==Piece.NOIR) return "Roi Noir";
	   break;

	case Piece.REINE :
	   if(couleur==Piece.BLANC) return "Reine Blanche";
	   if(couleur==Piece.NOIR) return "Reine Noire";
	   break;

	case Piece.CAV :
	   if(couleur==Piece.BLANC) return "Cavalier Blanc";
	   if(couleur==Piece.NOIR) return "Cavalier Noir";
	   break;

	case Piece.FOU :
	   if(couleur==Piece.BLANC) return "Fou Blanc";
```

```
            if(couleur==Piece.NOIR) return "Fou Noir";
            break;

        case Piece.TOUR :
            if(couleur==Piece.BLANC) return "Tour Blanche";
            if(couleur==Piece.NOIR) return "Tour Noire";
            break;

        case Piece.PION :
            if(couleur==Piece.BLANC) return "Pion blanc";
            if(couleur==Piece.NOIR) return "Pion Noir";
            break;
        default : return "";
        }
        return "";
}

//les pièces peuvent se déplacer horizontalement, verticalement ou en diagonale
//Déplacements possibles si piece vide etlimités par présence pièce adverse
//Les positions des pieces accessibles sont stockées dans des ArrayLists
protected List parcoursHorizontal ( int l, int c, int couleur, Board b) {
        List list = new ArrayList();
        int tmpC = c;

        while(--c > 0) {
            if(b.pieceVide(l,c)) list.add(new Position(l,c));
            else {
                    if(b.pieceAdverse(l,c,couleur))
                        list.add(new Position(l,c)); break;
            }
        }
        c = tmpC;
        while(++c < 9 ){
            if(b.pieceVide(l,c))list.add(new Position(l,c));
            else {
                    if(b.pieceAdverse(l,c,couleur))
                        list.add(new Position(l,c));break;
                    }
        }
        return list;
}

protected List parcoursVertical (int l, int c, int couleur, Board b) {
        List list = new ArrayList();
        int tmpL = l;

        while(--l > 0) {
            if(b.pieceVide(l,c)) list.add(new Position(l,c));
            else {
                    if(b.pieceAdverse(l,c,couleur))
                        list.add(new Position(l,c)); break;
            }
        }
        l = tmpL;
        while(++l < 9 ){
            if(b.pieceVide(l,c))list.add(new Position(l,c));
            else {
                    if(b.pieceAdverse(l,c,couleur))
                        list.add(new Position(l,c));break;
                    }
        }
        return list;
```

```
    }

    protected List parcoursDiagonal (int l, int c, int couleur, Board b) {

            List list = new ArrayList();
            int tmpL = l;
            int tmpC = c;

            while(--l > 0 && --c >0 && diagonale(l,tmpL,c,tmpC)) {
               if(b.pieceVide(l,c)) list.add(new Position(l,c));
               else {
                        if(b.pieceAdverse(l,c,couleur))
                           list.add(new Position(l,c)); break;
               }
            }
            l = tmpL ; c = tmpC;
            while(++l < 9  && ++c <9 && diagonale(l,tmpL,c,tmpC)){
               if(b.pieceVide(l,c))list.add(new Position(l,c));
               else {
                        if(b.pieceAdverse(l,c,couleur))
                           list.add(new Position(l,c));break;
                        }
            }

            l = tmpL ; c = tmpC;
            while(++l < 9  && --c >0 && diagonale(l,tmpL,c,tmpC)){
               if(b.pieceVide(l,c))list.add(new Position(l,c));
               else {
                        if(b.pieceAdverse(l,c,couleur))
                           list.add(new Position(l,c));break;
                        }
            }

            l = tmpL ; c = tmpC;
            while(--l > 0  && ++c < 9 && diagonale(l,tmpL,c,tmpC)){
               if(b.pieceVide(l,c))list.add(new Position(l,c));
               else {
                        if(b.pieceAdverse(l,c,couleur))
                           list.add(new Position(l,c));break;
                        }
            }
            return list;
    }

    //Iterator des positions possibles dans le tableau de pièces
    public abstract Iterator deplacement(Board b ,int l, int c , int couleur);

    public boolean diagonale(int la, int lb, int ca, int cb) {
            return (Math.abs(la - lb)  == Math.abs(ca - cb)) ;
    }
}
```

4Classe Roi

```
import java.util.*;

public class Roi extends Piece {

    List list;

    public Roi(int couleur) {
```

```
        super(couleur);
        super.type = Piece.ROI;
    }

     public Iterator deplacement (Board b, int l, int c, int couleur) {

        list = new ArrayList();

        if(b.pieceVide(l+1,c) || b.pieceAdverse(l+1,c,couleur)) list.add(new Position(l+1,c));
        if(b.pieceVide(l+1,c-1) || b.pieceAdverse(l+1,c-1,couleur)) list.add(new Position(l+1,c-1));
        if(b.pieceVide(l,c-1) || b.pieceAdverse(l,c-1,couleur)) list.add(new Position(l,c-1));
        if(b.pieceVide(l-1,c-1) || b.pieceAdverse(l-1,c-1,couleur)) list.add(new Position(l-1,c-1));
        if(b.pieceVide(l-1,c) || b.pieceAdverse(l-1,c,couleur)) list.add(new Position(l-1,c));
        if(b.pieceVide(l-1,c+1) || b.pieceAdverse(l-1,c+1,couleur)) list.add(new Position(l-1,c+1));
        if(b.pieceVide(l,c+1)|| b.pieceAdverse(l,c+1,couleur)) list.add(new Position(l,c+1));
        if(b.pieceVide(l+1,c+1)|| b.pieceAdverse(l+1,c+1,couleur)) list.add(new Position(l+1,c+1));

        return list.iterator();
    }
}
```

5Classe Reine

```
import java.util.*;

public class Reine extends Piece {

    List list;

    public Reine(int couleur) {
        super(couleur);
        super.type = Piece.REINE;
    }

    public Iterator deplacement (Board b, int l, int c, int couleur) {

        list = new ArrayList();

        list.addAll(parcoursHorizontal(l,c,couleur,b));
        list.addAll(parcoursVertical(l,c,couleur,b));
        list.addAll(parcoursDiagonal(l,c,couleur,b));

        return list.iterator();
        }
}
```

6Classe Cavalier

```
import java.util.*;

public class Cavalier extends Piece {

    List list;

    public Cavalier(int couleur) {
        super(couleur);
        super.type = Piece.CAV;
    }
```

```
  public Iterator deplacement (Board b, int l, int c, int couleur) {

            list = new ArrayList();

            if(!b.bord(l+1,c-1) && (b.pieceVide(l+2,c-1)|| b.pieceAdverse(l+2,c-1,couleur))) list.add(new Position(l+2,c-1));
            if(!b.bord(l+1,c-1) && (b.pieceVide(l+1,c-2)|| b.pieceAdverse(l+1,c-2,couleur))) list.add(new Position(l+1,c-2));
            if(!b.bord(l-1,c-1) && (b.pieceVide(l-1,c-2)|| b.pieceAdverse(l-1,c-2,couleur))) list.add(new Position(l-1,c-2));
            if(!b.bord(l-1,c-1) && (b.pieceVide(l-2,c-1)|| b.pieceAdverse(l-2,c-1,couleur))) list.add(new Position(l-2,c-1));
            if(!b.bord(l-1,c+1) && (b.pieceVide(l-2,c+1)|| b.pieceAdverse(l-2,c+1,couleur))) list.add(new Position(l-2,c+1));
            if(!b.bord(l-1,c+1) && (b.pieceVide(l-1,c+2)|| b.pieceAdverse(l-1,c+2,couleur))) list.add(new Position(l-1,c+2));
            if(!b.bord(l+1,c+1) && (b.pieceVide(l+1,c+2)|| b.pieceAdverse(l+1,c+2,couleur))) list.add(new Position(l+1,c+2));
            if(!b.bord(l+1,c+1) && (b.pieceVide(l+2,c+1)|| b.pieceAdverse(l+2,c+1,couleur))) list.add(new Position(l+2,c+1));

            return list.iterator();
    }
}
```

7Classe Fou

```
import java.util.*;

public class Fou extends Piece {

    List list;

    public Fou (int couleur) {
            super(couleur);
            super.type = Piece.FOU;

    }
   public Iterator deplacement (Board b, int l, int c, int couleur) {

            list = new ArrayList();
            list.addAll(parcoursDiagonal(l,c,couleur,b));

            return list.iterator();
   }
}
```

8Classe Tour

```
import java.util.*;

public class Tour extends Piece {

    List list;

    public Tour(int couleur) {
            super(couleur);
            super.type = Piece.TOUR;
    }

    public Iterator deplacement (Board b,int l, int c, int couleur) {

            list = new ArrayList();
            list.addAll(parcoursHorizontal(l,c,couleur,b));
            list.addAll(parcoursVertical(l,c,couleur,b));

            return list.iterator();
            }
```

```
}
```

9Classe Pion

```
import java.util.*;

public class Pion extends Piece {

   List list;

   public Pion(int couleur) {
          super(couleur);
          super.type = Piece.PION;
   }

   //les pions ne peuvent pas reculer
   // le pion peut avancer de 2 cases au premier tour de chacun des joueurs

    public Iterator deplacement(Board b ,int l, int c , int couleur) {

          list = new ArrayList();

          if(b.getCoups()<2) {
             if(couleur == Piece.BLANC) {
                   if(b.pieceVide(l-1,c)) list.add(new Position(l-1,c));
                   if(b.pieceVide(l-2,c)) list.add(new Position(l-2,c));
             }
             else {
                   if(b.pieceVide(l+1,c)) list.add(new Position(l+1,c));
                   if(b.pieceVide(l+2,c)) list.add(new Position(l+2,c));
             }
             return list.iterator();
          }
          else {
             if(couleur == Piece.BLANC) {
                   if(b.pieceVide(l-1,c)) list.add(new Position(l-1,c));
                   if(l-1 > 0 && c-1 >0 && b.pieceAdverse(l-1,c-1,couleur)) list.add(new Position(l-1,c-1));
                   if(l-1 > 0 && c+1 <9 && b.pieceAdverse(l-1,c+1,couleur)) list.add(new Position(l-1,c+1));
             }
             else {
                   if(b.pieceVide(l+1,c))  list.add(new Position(l+1,c));
                   if(l+1 < 9 && c-1 >0 && b.pieceAdverse(l+1,c-1,couleur)) list.add(new Position(l+1,c-1));
                   if(l+1 < 9 && c+1 <9 && b.pieceAdverse(l+1,c+1,couleur)) list.add(new Position(l+1,c+1));
             }
             return list.iterator();
          }
   }
}
```

10Classe PieceVide

```
import java.util.*;


public class PieceVide extends Piece {

   public PieceVide (int couleur) {
          super(couleur);
          super.type = Piece.VIDE;
   }
```

```
    // constructeur de recopie

    public PieceVide(PieceVide pieceVide) {
            super(pieceVide);
    }

    public Iterator deplacement(Board b,int l, int c , int couleur){

            return null;
    }
}
```

2Classe Bord

```
import java.util.*;

public class Bord extends Piece {

    public Bord (int couleur) {
            super(couleur);
            super.type = Piece.BORD;
    }

     public Iterator deplacement(Board b ,int l, int c , int couleur){

            return null;
     }
}
```

11Classe Position

```
public class Position {

    private int ligne;

    private int col;

    public Position(int _ligne,int  _col){

            ligne = _ligne;

            col= _col;
    }

    public boolean equals (Position p1){

            if(this.ligne==p1.ligne && this.col==p1.col)return true;
            return false;
    }

    public int getL(){

            return ligne;
    }
    public int getC(){

            return col;
    }
}
```

12Classe EchiquierException

```
public class EchiquierException extends Exception {

   public EchiquierException(){
          super();
   }

   public EchiquierException(String m){
          super(m);
   }
}
```

13Fichier index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">

<html>
<head>
<title></title>
<style type="text/css">

body {
background-color : #fff0f5;
}
h1{
font-family : sans-serif;
text-align:center;
}
#applet {
text-align: center;
}
</style>

</head>
<body>
<h1>JEU D'ECHECS</h1>

<div id="applet">
<applet code="AppletEchecs.class" width="650" height="400" >
</div>

</applet>

<hr />
</body>
</html>
```