## ***“Piece.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

/**
* types :
*     1 pawn
*     2 rook
*     3 knight
*     4 bishop
*     5 queen
*     6 king
*/

public abstract class Piece {
	private int type;
	private boolean isWhite, hasMoved;
	private Chess_Coord pos;

	public Piece () {
		type = 0;
		hasMoved = false;
	}

	public Piece (int n, Chess_Coord coord, boolean white) {
		type = n;
		isWhite = white;
		hasMoved = false;
		pos = coord;
	}

	public Chess_Coord GetPos () {return pos;}
	public void SetPos (Chess_Coord coord) {pos = coord;}
	public boolean GetColor () {return isWhite;}
	public int GetType () {return type;}
	public void SetType (int n) {type = n;}
	public boolean GetHasMoved () {return hasMoved;}
	public void SetHasMoved () {hasMoved = true;}
	public boolean IsDead () {return (pos.GetX() == -1 && pos.GetY() == -1);}
	public abstract int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p,Chess_Player opponent);
	public abstract boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent);
	public abstract boolean Kill (Piece o, Chess_Board b); // o est la pièce tuée
	public abstract void Die (); // coordonnées (-type, -type) svp
}
```

## ***“Bishop.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

// type 4
public class Bishop extends Piece {
	public Bishop (Chess_Coord pos, boolean white) {
		super(4, pos, white);
	}

	@Override
	public int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p, Chess_Player opponent) {
		if(ValidateMove(coord, b, opponent)) {
			if(OpponentOnCoord(coord, b, opponent)) Kill(b.GetPieceByCoord(coord, opponent), b);
			Chess_Coord temp = new Chess_Coord();
			temp = this.GetPos();
			this.SetPos(coord);
			b.UpdateBoard(this.GetColor(), this.GetType(), coord);
			b.ZeroCoord(temp);
			if(!GetHasMoved()) SetHasMoved();
			return 1;
		}
		else return 0;
	}

	private boolean OpponentOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0) {
			if(opponent.color == b.ReadColorAsInt(n)) return true;
		}
		return false;
	}

	private boolean FriendlyOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0) {
			if(opponent.color != b.ReadColorAsInt(n)) return true;
		}
		return false;
	}
```

```
private int GetDirection (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
            int origX = this.GetPos().GetX();
            int origY = this.GetPos().GetY();
            int res = 0;
            int diag = 1;
            while(origX-diag > 0 && origY-diag > 0) {
                if(FriendlyOnCoord(new Chess_Coord(origX-diag, origY-diag), b, opponent)) res = 0;
                else if(new Chess_Coord(origX-diag, origY-diag).equals(coord)) return 1;
                diag++;
            }
            diag = 1;
        while(origX+diag < 9 && origY-diag > 0) {
            if(FriendlyOnCoord(new Chess_Coord(origX+diag, origY-diag), b, opponent)) res = 0;
            else if(new Chess_Coord(origX+diag, origY-diag).equals(coord)) return 2;
            diag++;
        }
        diag = 1;
        while(origX+diag < 9 && origY+diag < 9) {
            if(FriendlyOnCoord(new Chess_Coord(origX+diag, origY+diag), b, opponent)) res = 0;
            else if(new Chess_Coord(origX+diag, origY+diag).equals(coord)) return 3;
            diag++;
        }
        diag = 1;
        while(origX-diag > 0 && origY+diag < 9) {
            if(FriendlyOnCoord(new Chess_Coord(origX-diag, origY+diag), b, opponent)) res = 0;
            else if(new Chess_Coord(origX-diag, origY+diag).equals(coord)) return 4;
            diag++;
        }
        return res;
}
```

```
private Chess_Coord[] GetMoveOptions (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    Chess_Coord[] moves = new Chess_Coord[7];
    int direction = GetDirection(coord, b, opponent);
    int origX = this.GetPos().GetX();
    int origY = this.GetPos().GetY();

    if(direction == 0) { // si coordonnée erronée
        for(int i = 0 ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
        return moves;
    }

    int index = 0;
    int diag = 1;
    switch(direction) {
        case 1:   diag = 1; index = 0;
                  while(origX-diag > 0 && origY-diag > 0) {
                  int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY-diag));
                  if(n != 0) {
                      if(b.ReadColorAsInt(n) == opponent.color) {
                          moves[index] = new Chess_Coord(origX-diag, origY-diag);
                          for(int i = index+1 ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                          break;
                      }
                      else {
                          for(int i = index ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                          break;
                      }
                  }
                  else moves[index] = new Chess_Coord(origX-diag, origY-diag);
                  diag++; index++;
              }
              break;
```

```
case 2:   diag = 1; index = 0;
          while(origX+diag < 9 && origY-diag > 0) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY-diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX+diag, origY-diag);
                         for(int i = index+1 ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                         break;
                    }
                    else {
                         for(int i = index ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                         break;
                    }
               }
               else moves[index] = new Chess_Coord(origX+diag, origY-diag);
               diag++; index++;
          }
          break;
case 3:   diag = 1; index = 0;
          while(origX+diag < 9 && origY+diag < 9) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY+diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX+diag, origY+diag);
                         for(int i = index+1 ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                         break;
                    }
                    else {
                         for(int i = index ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                         break;
                    }
               }
               else moves[index] = new Chess_Coord(origX+diag, origY+diag);
               diag++; index++;
          }
          break;
```

```
            case 4:   diag = 1; index = 0;
                      while(origX-diag > 0 && origY+diag < 9) {
                          int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY+diag));
                          if(n != 0) {
                              if(b.ReadColorAsInt(n) == opponent.color) {
                                  moves[index] = new Chess_Coord(origX-diag, origY+diag);
                                  for(int i = index+1 ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                                  break;
                              }
                              else {
                                  for(int i = index ; i < 7 ; i++) moves[i] = new Chess_Coord(-10, -10);
                                  break;
                              }
                          }
                          else moves[index] = new Chess_Coord(origX-diag, origY+diag);
                          diag++; index++;
                      }
                      break;
            default:  moves[index] = new Chess_Coord(-10, -10);
                      break;
            }
            return moves;
        }

        @Override
        public boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
            Chess_Coord[] options = new Chess_Coord[7];
            options = GetMoveOptions(coord, b, opponent);
            for(int i = 0 ; i < 7 ; i++) {
                if(options[i].equals(coord)) return true;
            }
            return false;
        }

        @Override
        public boolean Kill (Piece o, Chess_Board b) {
            o.Die();
            return true;
        }

        @Override
        public void Die () {this.GetPos().SetXY(-4, -4);}
    }
```

## ***“King.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

// type 6
public class King extends Piece {
	public King (Chess_Coord pos, boolean white) {
		super(6, pos, white);
	}

	@Override
	public int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p, Chess_Player opponent) {
		// si le roi roque
		if(!this.GetHasMoved()) { // illégal si le roi a bougé
			if(this.GetColor()) {
				if((this.GetPos().GetX() + 2) == coord.GetX() || (this.GetPos().GetX() - 2) == coord.GetX() &&
				coord.GetY() == 8) return Move2(coord, p, b);
			}
			else {
				if((this.GetPos().GetX() + 2) == coord.GetX() || (this.GetPos().GetX() - 2) == coord.GetX() &&
				coord.GetY() == 1) return Move2(coord, p, b);
			}
		}

		if(ValidateMove(coord, b, opponent)) {
			if(OpponentOnCoord(coord, b, opponent)) Kill(b.GetPieceByCoord(coord, opponent), b);
			Chess_Coord temp = new Chess_Coord();
			temp = this.GetPos();
			this.SetPos(coord);
			b.UpdateBoard(this.GetColor(), this.GetType(), coord);
			b.ZeroCoord(temp);
			if(!GetHasMoved()) SetHasMoved();
			return 1;
		}
		else return 0;
	}
```

```
public int Move2 (Chess_Coord coord, Chess_Player p, Chess_Board b) {
    if(p.r1.GetHasMoved() || p.r2.GetHasMoved()) return 0; // la tour a déjà bougé : illégal
    else if(this.GetHasMoved()) return 0;// le roi a déjà bougé : illégal
    else {
        if(p.color == 1) { // joueur blanc
            if(coord.equals(new Chess_Coord(3, 8))) { // tour 1
                if(b.GetPosParamsAsInt(new Chess_Coord(2, 8)) == 0 &&
                b.GetPosParamsAsInt(new Chess_Coord(3, 8)) == 0 &&
                b.GetPosParamsAsInt(new Chess_Coord(4, 8)) == 0) {
                    // roquer
                    Castling1(coord, p, b);
                    if(!GetHasMoved()) SetHasMoved();
                    return 2;
                }
            }
            else if(coord.equals(new Chess_Coord(7, 8))) { // tour 2
                if(b.GetPosParamsAsInt(new Chess_Coord(6, 8)) == 0 &&
                b.GetPosParamsAsInt(new Chess_Coord(7, 8)) == 0) {
                    // roquer
                    Castling2(coord, p, b);
                    if(!GetHasMoved()) SetHasMoved();
                    return 2;
                }
            }
        }
        else if(p.color == 2) { //joueur noir
            if(coord.equals(new Chess_Coord(3, 1))) { // tour 1
                if(b.GetPosParamsAsInt(new Chess_Coord(2, 1)) == 0 &&
                b.GetPosParamsAsInt(new Chess_Coord(3, 1)) == 0 &&
                b.GetPosParamsAsInt(new Chess_Coord(4, 1)) == 0) {
                    // roquer
                    Castling1(coord, p, b);
                    if(!GetHasMoved()) SetHasMoved();
                    return 2;
                }
            }
            else if(coord.equals(new Chess_Coord(7, 1))) { // tour 2
                if(b.GetPosParamsAsInt(new Chess_Coord(6, 1)) == 0 &&
                b.GetPosParamsAsInt(new Chess_Coord(7, 1)) == 0) {
                    // roquer
                    Castling2(coord, p, b);
                    if(!GetHasMoved()) SetHasMoved();
                    return 2;
                }
            }
        }
    }
```

```
        return 0;
    }

    private void Castling1 (Chess_Coord coord, Chess_Player p, Chess_Board b) {
        if(p.color == 1) {
            this.GetPos().SetXY(3, 8);
            b.UpdateBoard(this.GetColor(), this.GetType(), coord);
            this.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(5, 8));
            p.r1.GetPos().SetXY(4, 8);
            b.UpdateBoard(this.GetColor(), p.r1.GetType(), p.r1.GetPos());
            p.r1.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(1, 8));
        }
        else {
            this.GetPos().SetXY(3, 1);
            b.UpdateBoard(this.GetColor(), this.GetType(), coord);
            this.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(5, 1));
            p.r1.GetPos().SetXY(4, 1);
            b.UpdateBoard(this.GetColor(), p.r1.GetType(), p.r1.GetPos());
            p.r1.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(1, 1));
        }
    }

    private void Castling2 (Chess_Coord coord, Chess_Player p, Chess_Board b) {
        if(p.color == 1) {
            this.SetPos(new Chess_Coord(7, 8));
            b.UpdateBoard(this.GetColor(), this.GetType(), coord);
            this.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(5, 8));
            p.r2.SetPos(new Chess_Coord(6, 8));
            b.UpdateBoard(this.GetColor(), p.r2.GetType(), p.r2.GetPos());
            p.r2.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(8, 8));
        }
        else {
            this.SetPos(new Chess_Coord(7, 1));
            b.UpdateBoard(this.GetColor(), this.GetType(), coord);
            this.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(5, 1));
            p.r2.SetPos(new Chess_Coord(6, 1));
            b.UpdateBoard(this.GetColor(), p.r2.GetType(), p.r2.GetPos());
            p.r2.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(8, 1));
        }
```

```
}

private boolean OpponentOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    int n = b.GetPosParamsAsInt(coord);
    if(n != 0) {
        if(opponent.color == b.ReadColorAsInt(n)) return true;
    }
    return false;
}

private boolean FriendlyOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    int n = b.GetPosParamsAsInt(coord);
    if(n != 0) {
        if(opponent.color != b.ReadColorAsInt(n)) return true;
    }
    return false;
}

private int GetDirection (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    int origX = this.GetPos().GetX();
    int origY = this.GetPos().GetY();
    int diag = 1;
    if(origX-diag > 0 && origY-diag > 0) {
        if(new Chess_Coord(origX-diag, origY-diag).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 1;
    }
    if(origX+diag < 9 && origY-diag > 0) {
        if(new Chess_Coord(origX+diag, origY-diag).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 2;
    }
    if(origX+diag < 9 && origY+diag < 9) {
        if(new Chess_Coord(origX+diag, origY+diag).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return
        3;
    }
    if(origX-diag > 0 && origY+diag < 9) {
        if(new Chess_Coord(origX-diag, origY+diag).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 4;
    }
    if(origX == coord.GetX() && origY-diag > 0) {
        if(new Chess_Coord(origX, origY-diag).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 5;
    }
    if(origX+diag < 9 && origY == coord.GetY()) {
        if(new Chess_Coord(origX+diag, origY).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 6;
    }
    if(origX == coord.GetX() && origY+diag < 9) {
        if(new Chess_Coord(origX, origY+diag).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 7;
    }
    if(origX-diag > -1 && origY == coord.GetY()) {
        if(new Chess_Coord(origX-diag, origY).equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return 8;
    }
```

```
        return 0;
    }

    private Chess_Coord[] GetMoveOptions (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
        Chess_Coord[] moves = new Chess_Coord[1];
        int direction = GetDirection(coord, b, opponent);
        int origX = this.GetPos().GetX();
        int origY = this.GetPos().GetY();

        if(direction == 0) {
            moves[0] = new Chess_Coord(-10, -10);
            return moves;
        }

        int index = 0;
        int diag = 1;
        switch(direction) {
            case 1:   diag = 1; index = 0;
                      if(origX-diag > 0 && origY-diag > 0) {
                          int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY-diag));
                          if(n != 0) {
                              if(b.ReadColorAsInt(n) == opponent.color)
                                  moves[index] = new Chess_Coord(origX-diag, origY-diag);
                              else break;
                          }
                          else moves[index] = new Chess_Coord(origX-diag, origY-diag);
                      }
                      break;
            case 2:   diag = 1; index = 0;
                      if(origX+diag < 9 && origY-diag > 0) {
                          int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY-diag));
                          if(n != 0) {
                              if(b.ReadColorAsInt(n) == opponent.color)
                                  moves[index] = new Chess_Coord(origX+diag, origY-diag);
                              else break;
                          }
                          else moves[index] = new Chess_Coord(origX+diag, origY-diag);
                      }
                      break;
            case 3:   diag = 1; index = 0;
                      if(origX+diag < 9 && origY+diag < 9) {
                          int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY+diag));
                          if(n != 0) {
                              if(b.ReadColorAsInt(n) == opponent.color)
                                  moves[index] = new Chess_Coord(origX+diag, origY+diag);
                              else break;
                          }
```

```
            else moves[index] = new Chess_Coord(origX+diag, origY+diag);
        }
        break;
case 4: diag = 1; index = 0;
        If(origX-diag > 0 && origY+diag < 9) {
            Int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY+diag));
            If(n != 0) {
                If(b.ReadColorAsInt(n) == opponent.color)
                    moves[index] = new Chess_Coord(origX-diag, origY+diag);
                else break;
            }
            else moves[index] = new Chess_Coord(origX-diag, origY+diag);
        }
        break;
case 5: diag = 1; index = 0;
        if(origX == coord.GetX() && origY-diag > 0) {
            int n = b.GetPosParamsAsInt(new Chess_Coord(origX, origY-diag));
            if(n != 0) {
                if(b.ReadColorAsInt(n) == opponent.color)
                    moves[index] = new Chess_Coord(origX, origY-diag);
                else break;
            }
            else moves[index] = new Chess_Coord(origX, origY-diag);
        }
        break;
case 6: diag = 1; index = 0;
        if(origX+diag < 9 && origY == coord.GetY()) {
            int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY));
            if(n != 0) {
                if(b.ReadColorAsInt(n) == opponent.color)
                    moves[index] = new Chess_Coord(origX+diag, origY);
                else break;
            }
            else moves[index] = new Chess_Coord(origX+diag, origY);
        }
        break;
case 7: diag = 1; index = 0;
        if(origX == coord.GetX() && origY+diag < 9) {
            int n = b.GetPosParamsAsInt(new Chess_Coord(origX, origY+diag));
            if(n != 0) {
                if(b.ReadColorAsInt(n) == opponent.color)
                    moves[index] = new Chess_Coord(origX, origY+diag);
                else break;
            }
            else moves[index] = new Chess_Coord(origX, origY+diag);
        }
        break;
```

```
                case 8:   diag = 1; index = 0;
                          if(origX-diag > 0 && origY == coord.GetY()) {
                               int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY));
                               if(n != 0) {
                                    if(b.ReadColorAsInt(n) == opponent.color)
                                         moves[index] = new Chess_Coord(origX-diag, origY);
                                    else break;
                               }
                               else moves[index] = new Chess_Coord(origX-diag, origY);
                          }
                          break;
                default:  moves[index] = new Chess_Coord(-6, -6);
                          break;
          }
          return moves;
     }

     @Override
     public boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
          Chess_Coord[] options = new Chess_Coord[1];
          options = GetMoveOptions(coord, b, opponent);
          if(options[0].equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return true;
               return false;
          }

     @Override
     public boolean Kill (Piece o, Chess_Board b) {
          o.Die();
          return true;
     }

     @Override
     public void Die () {
          this.GetPos().SetXY(-6, -6);
          if(this.GetColor()) System.out.println("\n\n\t\t\t\033[1;31mPartie terminée !\033[0m \033[5;47;30mBLACK LIVES
          MATTER !!!\033[0m");
          else System.out.println("\033[1;31mPartie terminée !\033[0m \033[5;40;37mWHITE POWER !!!\033[0m");
          System.exit(0);
     }
}
```

## ***“Knight.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

// type 3
public class Knight extends Piece {
	public Knight (Chess_Coord coord, boolean white) {
		super(3, coord, white);
	}

	@Override
	public int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p, Chess_Player opponent) {
		if(ValidateMove(coord, b, opponent)) {
			if(OpponentOnCoord(coord, b, opponent)) Kill(b.GetPieceByCoord(coord, opponent), b);
				Chess_Coord temp = new Chess_Coord();
				temp = this.GetPos();
				this.SetPos(coord);
				b.UpdateBoard(this.GetColor(), this.GetType(), coord);
				b.ZeroCoord(temp);
				if(!GetHasMoved()) SetHasMoved();
				return 1;
		}
		else return 0;
	}

	private boolean OpponentOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0)
		if(opponent.color == b.ReadColorAsInt(n)) return true;
			return false;
		}

	private boolean FriendlyOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0)
			if(opponent.color != b.ReadColorAsInt(n)) return true;
		return false;
	}
```

```
private Chess_Coord[] GetMoveOptions () {
    int coordX = this.GetPos().GetX();
    int coordY = this.GetPos().GetY();
    Chess_Coord[] options = new Chess_Coord[8];
    Chess_Coord temp;
    int tempX, tempY;

    if((tempX = coordX - 2) > 0 && (tempY = coordY - 1) > 0) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[0] = temp;
    if((tempX = coordX - 1) > 0 && (tempY = coordY - 2) > 0) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[1] = temp;
    if((tempX = coordX + 1) < 9 && (tempY = coordY - 2) > 0) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[2] = temp;
    if((tempX = coordX + 2) < 9 && (tempY = coordY - 1) > 0) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[3] = temp;
    if((tempX = coordX + 2) < 9 && (tempY = coordY + 1) < 9) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[4] = temp;
    if((tempX = coordX + 1) < 9 && (tempY = coordY + 2) < 9) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[5] = temp;
    if((tempX = coordX - 1) > 0 && (tempY = coordY + 2) < 9) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[6] = temp;
    if((tempX = coordX - 2) > 0 && (tempY = coordY + 1) < 9) temp = new Chess_Coord(tempX, tempY);
    else temp = new Chess_Coord(-1, -1);
    options[7] = temp;

    return options;
}

@Override
public boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    Chess_Coord[] moves = new Chess_Coord[8];
    moves = GetMoveOptions();
    for(int i = 0 ; i < 8 ; i++)
        if(moves[i].equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return true;
    return false;
}
```

```
    @Override
    public boolean Kill (Piece o, Chess_Board b) {
        o.Die();
        return true;
    }

    @Override
    public void Die () {this.GetPos().SetXY(-3, -3);}
}
```

## ***“Pawn.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

public class Pawn extends Piece {
	public Pawn (Chess_Coord coord, boolean white) {
		super(1, coord, white);
	}

	@Override
	public int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p, Chess_Player opponent) {
		if(ValidateMove(coord, b, opponent)) {
			// cas de promotion
			if(this.GetColor()) {if(coord.GetY() == 1) return 3;}
			else {if(coord.GetY() == 8) return 3;}

			Chess_Coord temp = new Chess_Coord();
			temp = this.GetPos();
			this.SetPos(coord);
			b.UpdateBoard(this.GetColor(), this.GetType(), coord);
			b.ZeroCoord(temp);
			if(!GetHasMoved()) SetHasMoved();
			return 1;
		}
		else return 0;
	}

	private boolean OpponentOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0)
		if(opponent.color == b.ReadColorAsInt(n) && b.ReadColorAsInt(n) != 0) return true;
		return false;
	}

	private boolean FriendlyOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0)
		if(opponent.color != b.ReadColorAsInt(n) && b.ReadColorAsInt(n) != 0) return true;
		return false;
	}
```

```
private Chess_Coord[] GetMoveOptions (Chess_Board b, Chess_Player opponent) {
    int coordX = this.GetPos().GetX();
    int coordY = this.GetPos().GetY();
    Chess_Coord[] options = new Chess_Coord[4];

    if(this.GetColor()) { // joueur BLANC
        if((coordY - 1) > 0) {
            if(b.GetPosParamsAsInt(new Chess_Coord(coordX, coordY-1)) == 0) {
                options[0] = new Chess_Coord(coordX, coordY-1);
                if((coordY - 2) > 0) {
                    if(b.GetPosParamsAsInt(new Chess_Coord(coordX, coordY-2)) == 0 &&
                    !GetHasMoved()) {
                        options[1] = new Chess_Coord(coordX, coordY-2);
                    }
                    else options[1] = new Chess_Coord(-10, -10);
                }
                else options[1] = new Chess_Coord(-10, -10);
            }
            else {
                options[0] = new Chess_Coord(-10, -10);
                options[1] = new Chess_Coord(-10, -10);
            }
        }
        else {
            options[0] = new Chess_Coord(-10, -10);
            options[1] = new Chess_Coord(-10, -10);
        }
        if((coordX - 1) > 0 && (coordY - 1) > 0) {
            if(OpponentOnCoord(new Chess_Coord(coordX-1, coordY-1), b, opponent) &&
            !FriendlyOnCoord(new Chess_Coord(coordX-1, coordY-1), b, opponent)) {
                options[2] = new Chess_Coord(coordX-1, coordY-1);
            }
            else options[2] = new Chess_Coord(-10, -10);
        }
        else options[2] = new Chess_Coord(-10, -10);
        if((coordX + 1) < 9 && (coordY - 1) > 0) {
            if(OpponentOnCoord(new Chess_Coord(coordX+1, coordY-1), b, opponent) &&
            !FriendlyOnCoord(new Chess_Coord(coordX+1, coordY-1), b, opponent)) {
                options[3] = new Chess_Coord(coordX+1, coordY-1);
            }
            else options[3] = new Chess_Coord(-10, -10);
        }
        else options[3] = new Chess_Coord(-10, -10);
    }
```

```
        else { // joueur NOIR
            if((coordY + 1) > 0) {
                if(b.GetPosParamsAsInt(new Chess_Coord(coordX, coordY+1)) == 0) {
                    options[0] = new Chess_Coord(coordX, coordY+1);
                    if((coordY + 2) < 9) {
                        if(b.GetPosParamsAsInt(new Chess_Coord(coordX, coordY+2)) == 0 &&
                        !GetHasMoved()) {
                            options[1] = new Chess_Coord(coordX, coordY+2);
                        }
                        else options[1] = new Chess_Coord(-10, -10);
                    }
                    else options[1] = new Chess_Coord(-10, -10);
                }
                else {
                    options[0] = new Chess_Coord(-10, -10);
                    options[1] = new Chess_Coord(-10, -10);
                }
            }
            else {
                options[0] = new Chess_Coord(-10, -10);
                options[1] = new Chess_Coord(-10, -10);
            }
            if((coordX - 1) > 0 && (coordY + 1) < 9) {
                if(OpponentOnCoord(new Chess_Coord(coordX-1, coordY+1), b, opponent) &&
                !FriendlyOnCoord(new Chess_Coord(coordX-1, coordY+1), b, opponent)) {
                    options[2] = new Chess_Coord(coordX-1, coordY+1);
                }
                else options[2] = new Chess_Coord(-10, -10);
            }
            else options[2] = new Chess_Coord(-10, -10);
            if((coordX + 1) < 9 && (coordY + 1) < 9) {
                if(OpponentOnCoord(new Chess_Coord(coordX+1, coordY+1), b, opponent) &&
                !FriendlyOnCoord(new Chess_Coord(coordX+1, coordY+1), b, opponent)) {
                    options[3] = new Chess_Coord(coordX+1, coordY+1);
                }
                else options[3] = new Chess_Coord(-10, -10);
            }
            else options[3] = new Chess_Coord(-10, -10);
        }
        return options;
    }
```

```
    @Override
    public boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
        Chess_Coord[] moves = new Chess_Coord[4];
        moves = GetMoveOptions(b, opponent);

        for(int i = 0 ; i < 4 ; i++) {
            if(i == 2) {
                if(!moves[i].equals(new Chess_Coord(-10, -10))) {
                    if(this.GetColor()) {
                        b.GetPieceByCoord(coord, opponent).Die();
                    }
                    else {
                        b.GetPieceByCoord(coord, opponent).Die();
                    }
                }
            }
            if(i == 3) {
                if(!moves[i].equals(new Chess_Coord(-10, -10))) {
                    if(this.GetColor()) {
                        b.GetPieceByCoord(coord, opponent).Die();
                    }
                    else {
                        b.GetPieceByCoord(coord, opponent).Die();
                    }
                }
            }
            if(moves[i].equals(coord)) return true;
        }
        return false;
    }

    @Override
    public boolean Kill (Piece o, Chess_Board b) {
        o.Die();
        return true;
    }

    @Override
    public void Die () {this.GetPos().SetXY(-1, -1);}
}
```

## ***“Queen.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

// type 5
public class Queen extends Piece {
	public Queen (Chess_Coord pos, boolean white) {
		super(5, pos, white);
	}

	@Override
	public int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p, Chess_Player opponent) {
		if(ValidateMove(coord, b, opponent)) {
			if(OpponentOnCoord(coord, b, opponent)) Kill(b.GetPieceByCoord(coord, opponent), b);
			Chess_Coord temp = new Chess_Coord();
			temp = this.GetPos();
			this.SetPos(coord);
			b.UpdateBoard(this.GetColor(), this.GetType(), coord);
			b.ZeroCoord(temp);
			if(!GetHasMoved()) SetHasMoved();
			return 1;
		}
		else return 0;
	}

	private boolean OpponentOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0) {
			if(opponent.color == b.ReadColorAsInt(n)) return true;
		}
		return false;
	}

	private boolean FriendlyOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
		int n = b.GetPosParamsAsInt(coord);
		if(n != 0) {
			if(opponent.color != b.ReadColorAsInt(n)) return true;
		}
		return false;
	}
```

```
private Chess_Coord[] GetMoveOptions (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    Chess_Coord[] moves = new Chess_Coord[8];
    int direction = GetDirection(coord, b, opponent);
    System.out.println("direction queen : " + direction);
    int origX = this.GetPos().GetX();
    int origY = this.GetPos().GetY();

    if(direction == 0) { // si coordonnée erronée
        for(int i = 0 ; i < 8 ; i++) moves[i] = new Chess_Coord(-10, -10);
        return moves;
    }

    int index = 0;
    int diag = 1;
    switch(direction) {
        case 1:   diag = 1; index = 0;
                  while(origX-diag > 0 && origY-diag > 0) {
                      int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY-diag));
                      if(n != 0) {
                          if(b.ReadColorAsInt(n) == opponent.color) {
                              moves[index] = new Chess_Coord(origX-diag, origY-diag);
                              if(index < 8) {
                                  index++;
                                  while(index < 8) {
                                      moves[index] = new Chess_Coord(-10, -10);
                                      index++;
                                  }
                                  break;
                              }
                          }
                          else {
                              if(index < 8) {
                                  while(index < 8) {
                                      moves[index] = new Chess_Coord(-10, -10);
                                      index++;
                                  }
                                  break;
                              }
                          }
                      }
                      else moves[index] = new Chess_Coord(origX-diag, origY-diag);
                      diag++; index++;
                  }
                  break;
```

```
case 2:   diag = 1; index = 0;
          while(origX+diag < 9 && origY-diag > 0) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY-diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX+diag, origY-diag);
                         if(index < 8) {
                              index++;
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 8) {
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX+diag, origY-diag);
               diag++; index++;
          }
          break;
```

```
case 3:   diag = 1; index = 0;
          while(origX+diag < 9 && origY+diag < 9) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY+diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX+diag, origY+diag);
                         if(index < 8) {
                              index++;
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 8) {
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX+diag, origY+diag);
               diag++; index++;
          }
          break;
```

```
case 4:   diag = 1; index = 0;
          while(origX-diag > 0 && origY+diag < 9) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY+diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX-diag, origY+diag);
                         if(index < 8) {
                              index++;
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 8) {
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX-diag, origY+diag);
               diag++; index++;
          }
          break;
```

```
case 5:   diag = 1; index = 0;
          while(origX == coord.GetX() && origY-diag > 0) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX, origY-diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX, origY-diag);
                         if(index < 8) {
                              index++;
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 8) {
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX, origY-diag);
               diag++; index++;
          }
          break;
```

```
case 6:   diag = 1; index = 0;
                while(origX+diag < 9 && origY == coord.GetY()) {
                int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY));
                if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                        moves[index] = new Chess_Coord(origX+diag, origY);
                        if(index < 8) {
                            index++;
                            while(index < 8) {
                                moves[index] = new Chess_Coord(-10, -10);
                                index++;
                            }
                            break;
                        }
                    }
                    else {
                        if(index < 8) {
                            while(index < 8) {
                                moves[index] = new Chess_Coord(-10, -10);
                                index++;
                            }
                            break;
                        }
                    }
                }
                else moves[index] = new Chess_Coord(origX+diag, origY);
                diag++; index++;
            }
            break;
```

```
case 7:   diag = 1; index = 0;
          while(origX == coord.GetX() && origY+diag < 9) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX, origY+diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                    moves[index] = new Chess_Coord(origX, origY+diag);
                         if(index < 8) {
                              index++;
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 8) {
                              while(index < 8) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX, origY+diag);
               diag++; index++;
          }
          break;
```

```
            case 8:   diag = 1; index = 0;
                      while(origX-diag > 0 && origY == coord.GetY()) {
                          int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY));
                          if(n != 0) {
                              if(b.ReadColorAsInt(n) == opponent.color) {
                                  moves[index] = new Chess_Coord(origX-diag, origY);
                                  if(index < 8) {
                                      index++;
                                      while(index < 8) {
                                          moves[index] = new Chess_Coord(-10, -10);
                                          index++;
                                      }
                                      break;
                                  }
                              }
                              else {
                                  if(index < 8) {
                                      while(index < 8) {
                                          moves[index] = new Chess_Coord(-10, -10);
                                          index++;
                                      }
                                      break;
                                  }
                              }
                          }
                          else moves[index] = new Chess_Coord(origX-diag, origY);
                          diag++; index++;
                      }
                      break;
            default:  moves[index] = new Chess_Coord(-10, -10);
                      break;
        }
        return moves;
    }
```

```
private int GetDirection (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    int origX = this.GetPos().GetX();
    int origY = this.GetPos().GetY();
    int res = 0;
    int diag = 1;
    while(origX-diag > 0 && origY-diag > 0) {
        if(FriendlyOnCoord(new Chess_Coord(origX-diag, origY-diag), b, opponent)) res = 0;
        if(new Chess_Coord(origX-diag, origY-diag).equals(coord)) return 1;
        diag++;
    }
    diag = 1;
    while(origX+diag < 9 && origY-diag > 0) {
        if(FriendlyOnCoord(new Chess_Coord(origX+diag, origY-diag), b, opponent)) res = 0;
        if(new Chess_Coord(origX+diag, origY-diag).equals(coord)) return 2;
        diag++;
    }
    diag = 1;
    while(origX+diag < 9 && origY+diag < 9) {
        if(FriendlyOnCoord(new Chess_Coord(origX+diag, origY+diag), b, opponent)) res = 0;
        if(new Chess_Coord(origX+diag, origY+diag).equals(coord)) return 3;
        diag++;
    }
    diag = 1;
    while(origX-diag > 0 && origY+diag < 9) {
        if(FriendlyOnCoord(new Chess_Coord(origX-diag, origY+diag), b, opponent)) res = 0;
        if(new Chess_Coord(origX-diag, origY+diag).equals(coord)) return 4;
        diag++;
    }
    diag = 1;
    while(origX == coord.GetX() && origY-diag > 0) {
        if(FriendlyOnCoord(new Chess_Coord(origX, origY-diag), b, opponent)) res = 0;
        if(new Chess_Coord(origX, origY-diag).equals(coord)) return 5;
        diag++;
    }
    diag = 1;
    while(origX+diag < 9 && origY == coord.GetY()) {
        if(FriendlyOnCoord(new Chess_Coord(origX+diag, origY), b, opponent)) res = 0;
        if(new Chess_Coord(origX+diag, origY).equals(coord)) return 6;
        diag++;
    }
    diag = 1;
    while(origX == coord.GetX() && origY+diag < 9) {
        if(FriendlyOnCoord(new Chess_Coord(origX, origY+diag), b, opponent)) res = 0;
        if(new Chess_Coord(origX, origY+diag).equals(coord)) return 7;
        diag++;
    }
    diag = 1;
```

```
        while(origX-diag > 0 && origY == coord.GetY()) {
            if(FriendlyOnCoord(new Chess_Coord(origX-diag, origY), b, opponent)) res = 0;
            if(new Chess_Coord(origX-diag, origY).equals(coord)) return 8;
            diag++;
        }
        return res;
    }

    @Override
    public boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
        Chess_Coord[] options = new Chess_Coord[8];
        options = GetMoveOptions(coord, b, opponent);
        for(int i = 0 ; i < 8 ; i++) {
            if(options[i].equals(new Chess_Coord(-10, -10))) return false;
            if(options[i].equals(coord)) return true;
        }
        return false;
    }

    @Override
    public boolean Kill (Piece o, Chess_Board b) {
        o.Die();
        return true;
    }

    @Override
    public void Die () {this.GetPos().SetXY(-5, -5);}
}
```

## ***“Rook.java”***

```
package pieces;

import board.*;
import game.Chess_Player;

// type 2
public class Rook extends Piece {
	public Rook (Chess_Coord pos, boolean white) {
		super(2, pos, white);
	}

	@Override
	public int Move1 (Chess_Coord coord, Chess_Board b, Chess_Player p, Chess_Player opponent) {
		boolean direction = GoingVertical(coord);
		if(direction == GoingHorizontal(coord)) return 0;

		if(ValidateMove(coord, b, opponent)) {
			if(OpponentOnCoord(coord, b, opponent)) Kill(b.GetPieceByCoord(coord, opponent), b);
			Chess_Coord temp = new Chess_Coord();
			temp = this.GetPos();
			this.SetPos(coord);
			b.UpdateBoard(this.GetColor(), this.GetType(), coord);
			b.ZeroCoord(temp);
			if(!GetHasMoved()) SetHasMoved();
			return 1;
		}
		else return 0;
	}

	@Deprecated
	public boolean Move2 (Chess_Coord coord, Chess_Player p, Chess_Board b) {
		if(p.k.GetHasMoved()) return false; // le roi a déjà bougé : illégal
		else if(this.GetHasMoved()) return false;// la tour a déjà bougé : illégal
		else {
			if(p.color == 1) { // joueur blanc
				if(this.equals(p.r1)) { // tour 1
					if(b.GetPosParamsAsInt(new Chess_Coord(2, 8)) == 0 &&
					b.GetPosParamsAsInt(new Chess_Coord(3, 8)) == 0 &&
					b.GetPosParamsAsInt(new Chess_Coord(4, 8)) == 0) {
						// roquer
						Castling1(coord, p, b);
						return true;
					}
				}
				else if(this.equals(p.r2)) { // tour 2
					if(b.GetPosParamsAsInt(new Chess_Coord(6, 8)) == 0 &&
```

```
                    b.GetPosParamsAsInt(new Chess_Coord(7, 8)) == 0) {
                        // roquer
                        Castling2(coord, p, b);
                        return true;
                    }
                }
            }
            else if(p.color == 2) { //joueur noir
                if(this.equals(p.r1)) { // tour 1
                    if(b.GetPosParamsAsInt(new Chess_Coord(2, 1)) == 0 &&
                    b.GetPosParamsAsInt(new Chess_Coord(3, 1)) == 0 &&
                    b.GetPosParamsAsInt(new Chess_Coord(4, 1)) == 0) {
                        // roquer
                        Castling1(coord, p, b);
                    }
                }
                else if(this.equals(p.r2)) { // tour 2
                    if(b.GetPosParamsAsInt(new Chess_Coord(6, 1)) == 0 &&
                    b.GetPosParamsAsInt(new Chess_Coord(7, 1)) == 0) {
                        // roquer
                        Castling2(coord, p, b);
                    }
                }
            }
        }
        return false;
    }

    @Deprecated
    private void Castling1 (Chess_Coord coord, Chess_Player p, Chess_Board b) {
        if(p.color == 1) {
            p.k.SetPos(new Chess_Coord(3, 8));
            p.k.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(5, 8));
            p.r1.SetPos(new Chess_Coord(4, 8));
            p.r1.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(1, 8));
        }
        else {
            p.k.SetPos(new Chess_Coord(3, 1));
            p.k.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(5, 1));
            p.r1.SetPos(new Chess_Coord(4, 1));
            p.r1.SetHasMoved();
            b.ZeroCoord(new Chess_Coord(1, 1));
        }
    }
```

```
@Deprecated
private void Castling2 (Chess_Coord coord, Chess_Player p, Chess_Board b) {
	if(p.color == 1) {
		p.k.SetPos(new Chess_Coord(7, 8));
		p.k.SetHasMoved();
		b.ZeroCoord(new Chess_Coord(5, 8));
		p.r2.SetPos(new Chess_Coord(6, 8));
		p.r2.SetHasMoved();
		b.ZeroCoord(new Chess_Coord(8, 8));
	}
	else {
		p.k.SetPos(new Chess_Coord(7, 1));
		p.k.SetHasMoved();
		b.ZeroCoord(new Chess_Coord(5, 1));
		p.r2.SetPos(new Chess_Coord(6, 1));
		p.r2.SetHasMoved();
		b.ZeroCoord(new Chess_Coord(8, 1));
	}
}

private boolean OpponentOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
	int n = b.GetPosParamsAsInt(coord);
	if(n != 0) {
		if(opponent.color == b.ReadColorAsInt(n)) return true;
	}
	return false;
}

private boolean FriendlyOnCoord (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
	int n = b.GetPosParamsAsInt(coord);
	if(n != 0) {
		if(opponent.color != b.ReadColorAsInt(n)) return true;
	}
	return false;
}

@Deprecated
private boolean GoingVertical (Chess_Coord coord) {
	if(this.GetPos().GetX() == coord.GetX() && this.GetPos().GetY() != coord.GetY()) return true;
	else return false;
}
```

```
@Deprecated
private boolean GoingHorizontal (Chess_Coord coord) {
    if(this.GetPos().GetY() == coord.GetY() &&
    this.GetPos().GetX() != coord.GetX()) return true;
    else return false;
}

private int GetDirection (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    int origX = this.GetPos().GetX();
    int origY = this.GetPos().GetY();
    int diag = 1;
    while(origX == coord.GetX() && origY-diag > 0) {
        if(FriendlyOnCoord(new Chess_Coord(origX, origY-diag), b, opponent)) return 0;
        if(new Chess_Coord(origX, origY-diag).equals(coord)) return 1;
        diag++;
    }
    diag = 1;
    while(origX+diag < 9 && origY == coord.GetY()) {
        if(FriendlyOnCoord(new Chess_Coord(origX+diag, origY), b, opponent)) return 0;
        if(new Chess_Coord(origX+diag, origY).equals(coord)) return 2;
        diag++;
    }
    diag = 1;
    while(origX == coord.GetX() && origY+diag < 9) {
        if(FriendlyOnCoord(new Chess_Coord(origX, origY+diag), b, opponent)) return 0;
        if(new Chess_Coord(origX, origY+diag).equals(coord)) return 3;
        diag++;
    }
    diag = 1;
    while(origX-diag > 0 && origY == coord.GetY()) {
        if(FriendlyOnCoord(new Chess_Coord(origX-diag, origY), b, opponent)) return 0;
        if(new Chess_Coord(origX-diag, origY).equals(coord)) return 4;
        diag++;
    }
    return 0;
}
```

```
private Chess_Coord[] GetMoveOptions (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
    int direction = GetDirection(coord, b, opponent);
    int origX = this.GetPos().GetX();
    int origY = this.GetPos().GetY();
    Chess_Coord[] moves = new Chess_Coord[7];

    if(direction == 0) {
        for(int i = 0 ; i < 7 ; i++) {
            moves[i] = new Chess_Coord(-10, -10);
        }
        return moves;
    }

    int index = 0;
    int diag = 1;
    switch(direction) {
        case 1:   diag = 1; index = 0;
                  while(origX == coord.GetX() && origY-diag > 0) {
                      int n = b.GetPosParamsAsInt(new Chess_Coord(origX, origY-diag));
                      if(n != 0) {
                          if(b.ReadColorAsInt(n) == opponent.color) {
                              moves[index] = new Chess_Coord(origX, origY-diag);
                              if(index < 7) {
                                  index++;
                                  while(index < 7) {
                                      moves[index] = new Chess_Coord(-10, -10);
                                      index++;
                                  }
                                  break;
                              }
                          }
                          else {
                              if(index < 7) {
                                  while(index < 7) {
                                      moves[index] = new Chess_Coord(-10, -10);
                                      index++;
                                  }
                                  break;
                              }
                          }
                      }
                      else moves[index] = new Chess_Coord(origX, origY-diag);
                      diag++; index++;
                  }
                  break;
```

```
case 2:   diag = 1; index = 0;
          while(origX+diag < 9 && origY == coord.GetY()) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX+diag, origY));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX+diag, origY);
                         if(index < 7) {
                              index++;
                              while(index < 7) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 7) {
                              while(index < 7) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX+diag, origY);
               diag++; index++;
          }
          break;
```

```
case 3:   diag = 1; index = 0;
          while(origX == coord.GetX() && origY+diag < 9) {
               int n = b.GetPosParamsAsInt(new Chess_Coord(origX, origY+diag));
               if(n != 0) {
                    if(b.ReadColorAsInt(n) == opponent.color) {
                         moves[index] = new Chess_Coord(origX, origY+diag);
                         if(index < 7) {
                              index++;
                              while(index < 7) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
                    else {
                         if(index < 7) {
                              while(index < 7) {
                                   moves[index] = new Chess_Coord(-10, -10);
                                   index++;
                              }
                              break;
                         }
                    }
               }
               else moves[index] = new Chess_Coord(origX, origY+diag);
               diag++; index++;
          }
          break;
```

```
            case 4:   diag = 1; index = 0;
                      while(origX-diag > 0 && origY == coord.GetY()) {
                          int n = b.GetPosParamsAsInt(new Chess_Coord(origX-diag, origY));
                          if(n != 0) {
                              if(b.ReadColorAsInt(n) == opponent.color) {
                                  moves[index] = new Chess_Coord(origX-diag, origY);
                                  if(index < 7) {
                                      index++;
                                      while(index < 7) {
                                          moves[index] = new Chess_Coord(-10, -10);
                                          index++;
                                      }
                                      break;
                                  }
                              }
                              else {
                                  if(index < 7) {
                                      while(index < 7) {
                                          moves[index] = new Chess_Coord(-10, -10);
                                          index++;
                                      }
                                      break;
                                  }
                              }
                          }
                          else moves[index] = new Chess_Coord(origX-diag, origY);
                          diag++; index++;
                      }
                      break;
            default:  moves[index] = new Chess_Coord(-10, -10);
                      break;
        }
        return moves;
    }

    @Override
    public boolean ValidateMove (Chess_Coord coord, Chess_Board b, Chess_Player opponent) {
        Chess_Coord[] options = new Chess_Coord[7];
        options = GetMoveOptions(coord, b, opponent);
        for(int i = 0 ; i < 7 ; i++) {
            if(options[i].equals(new Chess_Coord(-10, -10))) return false;
            if(options[i].equals(coord) && !FriendlyOnCoord(coord, b, opponent)) return true;
        }
        return false;
    }
```

```
    @Override
    public boolean Kill (Piece o, Chess_Board b) {
        o.Die();
        return true;
    }

    @Override
    public void Die () {this.GetPos().SetXY(-2, -2);}
}
```

## ***“Chess_Coord.java”***

```
package board;

public class Chess_Coord {
	private int x, y;

	public Chess_Coord () {
		x = 0;
		y = 0;
	}

	public Chess_Coord (int a, int b) {
		x = a;
		y = b;
	}

	public int GetX () {return x;}
	public int GetY () {return y;}
	public void SetX (int n) {x = n;}
	public void SetY (int n) {y = n;}
	public void SetXY (int a, int b) {x = a; y = b;}

	public boolean equals (Chess_Coord o) {
		if(this.x == o.x && this.y == o.y) return true;
		else return false;
	}
}
```

## ***“Chess_Board.java”***

```
package board;

import game.Chess_Player;
import pieces.Pawn;
import pieces.Piece;

/**
* couleur :
*     1 - white (true)
*     2 - black (false)
* type :
*     1 - pawn
*     2 - rook
*     3 - knight
*     4 - bishop
*     5 - queen
*     6 - king
* x et y : entre 1 et 8 inclusivement
*/

public class Chess_Board {
	private int[][] situation; // 4 chiffres : couleur.type.x.y

	public Chess_Board () {
		Init();
	}

	private void Init () {
		situation = new int[8][8];
		// origine coin supérieur gauche (0, 0)
		for(int i = 0 ; i < 8 ; i++) {
			for(int j = 0 ; j < 8 ; j++) {
				situation[i][j] = 0; // 0 = vide
			}
		}
	}
```

```
@Deprecated
public void DisplaySituation () {
	System.out.println("\n\t\t\033[5;1;37;101mSituation :\033[25;0m\n");
	for(int i = 0 ; i < 8 ; i++) {
		for(int j = 0 ; j < 8 ; j++) {
			if(situation[i][j] == 0) System.out.print("\033[31m" + situation[i][j] + "\033[0m ");
			else System.out.print("\033[5;1;33m" + situation[i][j] + "\033[0m ");
		}
		System.out.println("\n");
	}
}

public void DisplayBoard () {
	boolean tile = true;

	System.out.print(" ");
	for(int i = 0 ; i < 8 ; i++) System.out.print("\033[1;31m " + (i+1) + "\033[0m ");
	System.out.println();
	for(int i = 0 ; i < 8 ; i++) {
		if(tile) System.out.print("\n \033[1;31m" + (i+1) + " \033[46;0m ");
		else System.out.print("\n \033[1;31m" + (i+1) + " \033[42;0m ");
		for(int j = 0 ; j < 8 ; j++) {
			if(GetTypeAsChar(situation[i][j]) != 0) {
				if(ReadColorAsInt(situation[i][j]) == 1) {
					if(tile) System.out.print("\033[46;1;97m" + GetTypeAsChar(situation[i][j]) + " \033[0m");
					else System.out.print("\033[42;1;97m" + GetTypeAsChar(situation[i][j]) + " \033[0m");
				}
				else {
					if(tile) System.out.print("\033[46;1;90m" + GetTypeAsChar(situation[i][j]) + " \033[0m");
					else System.out.print("\033[42;1;90m" + GetTypeAsChar(situation[i][j]) + " \033[0m");
				}
			}
			else {
				if(tile) System.out.print("\033[46;1;34m" + GetTypeAsChar(situation[i][j]) + " \033[0m");
				else System.out.print("\033[42;1;34m" + GetTypeAsChar(situation[i][j]) + " \033[0m");
			}
			tile = ToggleTile(tile);
		}
		if(tile) System.out.print("\n \033[46m \033[42m \033[46m \033[42m \033[46m \033[42m \033[46m \
		033[42m \033[0m");
		else System.out.print("\n \033[42m \033[46m \033[42m \033[46m \033[42m \033[46m \033[42m \033[46m \
		033[0m");
		tile = ToggleTile(tile);
	}
	System.out.println("\n\033[0m");
}
```

```
private boolean ToggleTile(boolean b) {
	if(b) return false;
	else return true;
}

public void UpdateBoard (boolean color, int type, Chess_Coord pos) {
	int col; // ici color devient un int
	if(color) col = 1;
	else col = 2;
	situation[pos.GetY()-1][pos.GetX()-1] = ConvertParamsAsInt(col, type, pos);
}

public void ZeroCoord (Chess_Coord pos) {
	situation[pos.GetY()-1][pos.GetX()-1] = 0;
}

public int GetPosParamsAsInt (Chess_Coord pos) {
	return situation[pos.GetY()-1][pos.GetX()-1];
}

private int ConvertParamsAsInt (int color, int type, Chess_Coord pos) {
	int x_ = (pos.GetX()) * 10;
	int y_ = pos.GetY();
	int type_, color_;
	type_ = type * 100;
	color_ = color * 1000;
	return (color_ + type_ + x_ + y_);
}

public Piece GetPieceByCoord (Chess_Coord coord, Chess_Player p) {
	Pawn temp = new Pawn(new Chess_Coord(-10, -10), true);
	int n = GetPosParamsAsInt(coord);

	if(n != 0) {
		int typ = ReadTypeAsInt(n);
		switch(typ) {
			case 1:	if(coord.equals(p.p1.GetPos())) return p.p1;
				else if(coord.equals(p.p2.GetPos())) return p.p2;
				else if(coord.equals(p.p3.GetPos())) return p.p3;
				else if(coord.equals(p.p4.GetPos())) return p.p4;
				else if(coord.equals(p.p5.GetPos())) return p.p5;
				else if(coord.equals(p.p6.GetPos())) return p.p6;
				else if(coord.equals(p.p7.GetPos())) return p.p7;
				else if(coord.equals(p.p8.GetPos())) return p.p8;
				break;
```

```
                case 2:   if(coord.equals(p.r1.GetPos())) return p.r1;
                          else if(coord.equals(p.r2.GetPos())) return p.r2;
                          else { // recherche parmis les extras
                          return SearchExtras(coord, p);
                          }
                case 3:   if(coord.equals(p.n1.GetPos())) return p.n1;
                          else if(coord.equals(p.n2.GetPos())) return p.n2;
                          else { // recherche parmis les extras
                          return SearchExtras(coord, p);
                          }
                case 4:   if(coord.equals(p.b1.GetPos())) return p.b1;
                          else if(coord.equals(p.b2.GetPos())) return p.b2;
                          else { // recherche parmis les extras
                          return SearchExtras(coord, p);
                          }
                case 5:   if(coord.equals(p.q.GetPos())) return p.q;
                          else { // recherche parmis les extras
                          return SearchExtras(coord, p);
                          }
                case 6:   if(coord.equals(p.k.GetPos())) return p.k;
                          break;
                default:  return temp;
            }
        }
        return temp;
}

private Piece SearchExtras (Chess_Coord coord, Chess_Player p) {
        int prom = p.GetPromCount();
        Piece temp = new Pawn(new Chess_Coord(-1, -1), true);

        if(prom > 0) {
            for(int i = 1 ; i <= prom ; i++) {
                switch(i) {
                    case 1:   if(p.e1.GetPos().equals(coord)) return p.e1;
                    case 2:   if(p.e2.GetPos().equals(coord)) return p.e2;
                    case 3:    if(p.e3.GetPos().equals(coord)) return p.e3;
                    case 4:   if(p.e4.GetPos().equals(coord)) return p.e4;
                    case 5:   if(p.e5.GetPos().equals(coord)) return p.e5;
                    case 6:   if(p.e6.GetPos().equals(coord)) return p.e6;
                    case 7:   if(p.e7.GetPos().equals(coord)) return p.e7;
                    case 8:   if(p.e8.GetPos().equals(coord)) return p.e8;
                }
            }
        }
        return temp;
}
```

```java
/**
* Fonctions de lecture de position XXXX
* @param n
* @return
*/
public int ReadColorAsInt (int n) {return n / 1000;}

public int ReadTypeAsInt (int n) {
	int i = ReadColorAsInt(n) * 1000;
	i = n - i;
	return i / 100;
}

public int ReadXAsInt (int n) {
	int i = ReadColorAsInt(n) * 1000;
	i = n - i;
	int j = ReadTypeAsInt(n) * 100;
	j = i - j;
	return j / 10;
}

public int ReadYAsInt (int n) {
	int i = ReadColorAsInt(n) * 1000;
	i = n - i;
	int j = ReadTypeAsInt(n) * 100;
	j = i - j;
	int k = ReadXAsInt(n);
	k = j - k * 10;
	return k;
}
```

```
    public char GetTypeAsChar (int n) {
        int type = ReadTypeAsInt(n);
        char c;
        switch(type) {
            case 1:   c = 'P';
                      break;
            case 2:   c = 'R';
                      break;
            case 3:   c = 'N';
                      break;
            case 4:   c = 'B';
                      break;
            case 5:   c = 'Q';
                      break;
            case 6:   c = 'K';
                      break;
            default:  c = '*';
        }
        return c;
    }
}
```

## ***“Chess_Game.java”***

```
package game;

import java.util.Scanner;
import java.io.IOException;
import board.*;
import pieces.*;

public class Chess_Game {
	private Chess_Player p1, p2;
	private Chess_Board b;
	private Chess_Coord startCoord, endCoord;
	private boolean turn; // p1 = true, p2 = false
	private Chess_Setup setup;
	private Scanner scan;

	public Chess_Game () {
		turn = true;
		b = new Chess_Board();
		p1 = new Chess_Player(1);
		p2 = new Chess_Player(2);
		setup = new Chess_Setup(); // setup initialise le board
		p1 = setup.GetPlayer(1);
		p2 = setup.GetPlayer(2);
		b = setup.GetBoard();
		scan = new Scanner(System.in);
		Splash();
	}

	public void PlayTurn () {
		/**
		* Si joueur 1 : Si joueur 2 :
		* p1 = joueur courant p2 = joueur courant
		* p2 = joueur 'opponent' p1 = joueur 'opponent'
		*/
		Chess_Coord[] coords = new Chess_Coord[2];
		boolean ok = false;
		int moveResult;
		do {
			DisplayBoard();
			coords = InputCoord(turn);
			int newType;
			if(turn) {
				moveResult = b.GetPieceByCoord(coords[0], p1).Move1(coords[1], b, p1, p2);
				if(moveResult == 3) {
					newType = Promote(p1); // retourne le nouveau type (int)
					// détruire pawn + créer pièce de nouveau type + updater le board
```

```
                    Piece oldPiece = b.GetPieceByCoord(coords[1], p1); // pièce qui "disparaît"
                    Piece newPiece;
                    newPiece = ExtraFactory(p1, newType, coords[1]);
                    oldPiece.Die();
                    b.ZeroCoord(coords[0]);
                    b.UpdateBoard(newPiece.GetColor(), newPiece.GetType(), newPiece.GetPos());
                }
                if(moveResult > 0 && moveResult < 4) ok = true;
                // (pour les tests)
                //b.DisplaySituation();
                //scan.nextLine();
            }
            else {
                moveResult = b.GetPieceByCoord(coords[0], p2).Move1(coords[1], b, p2, p1);
                if(moveResult == 3) {
                    newType = Promote(p2); // retourne le nouveau type (int)
                    // détruire pawn + créer pièce de nouveau type + updater le board
                    Piece oldPiece = b.GetPieceByCoord(coords[1], p2); // pièce qui "disparaît"
                    Piece newPiece;
                    newPiece = ExtraFactory(p2, newType, coords[1]);
                    oldPiece.Die();
                    b.ZeroCoord(coords[0]);
                    b.UpdateBoard(newPiece.GetColor(), newPiece.GetType(), newPiece.GetPos());
                }
                if(moveResult > 0 && moveResult < 4) ok = true;
                // (pour les tests)
                //b.DisplaySituation();
                //scan.nextLine();
            }
        }
        while(!ok);
        EndTurn();
    }

    // attention, l'erreur de type n'est pas définie
    private Piece ExtraFactory(Chess_Player p, int type_, Chess_Coord coord) {
        int prom = p.GetPromCount();
        switch(prom) { // promCount a déjà été incrémenté à ce point
            case 1:   switch(type_) {
                        case 2:   p.e1 = new Rook(coord, turn);
                                  return p.e1;
                        case 3:   p.e1 = new Knight(coord, turn);
                                  return p.e1;
                        case 4:   p.e1 = new Bishop(coord, turn);
                                  return p.e1;
                        case 5:   p.e1 = new Queen(coord, turn);
                                  return p.e1    ;
```

```
            }
        case 2:   switch(type_) {
                    case 2:   p.e2 = new Rook(coord, turn);
                              return p.e2;
                    case 3:   p.e2 = new Knight(coord, turn);
                              return p.e2;
                    case 4:   p.e2 = new Bishop(coord, turn);
                              return p.e2;
                    case 5:   p.e2 = new Queen(coord, turn);
                              return p.e2;
            }
        case 3:   switch(type_) {
                    case 2:   p.e3 = new Rook(coord, turn);
                              return p.e3;
                    case 3:   p.e3 = new Knight(coord, turn);
                              return p.e3;
                    case 4:   p.e3 = new Bishop(coord, turn);
                              return p.e3;
                    case 5:   p.e3 = new Queen(coord, turn);
                              return p.e3;
            }
        case 4:   switch(type_) {
                    case 2:   p.e4 = new Rook(coord, turn);
                              return p.e4;
                    case 3:   p.e4 = new Knight(coord, turn);
                              return p.e4;
                    case 4:   p.e4 = new Bishop(coord, turn);
                              return p.e4;
                    case 5:   p.e4 = new Queen(coord, turn);
                              return p.e4;
                    }
        case 5:   switch(type_) {
                    case 2:   p.e5 = new Rook(coord, turn);
                              return p.e5;
                    case 3:   p.e5 = new Knight(coord, turn);
                              return p.e5;
                    case 4:   p.e5 = new Bishop(coord, turn);
                              return p.e5;
                    case 5:   p.e5 = new Queen(coord, turn);
                              return p.e5;
            }
        case 6:   switch(type_) {
                    case 2:   p.e6 = new Rook(coord, turn);
                              return p.e6;
                    case 3:   p.e6 = new Knight(coord, turn);
                              return p.e6;
                    case 4:   p.e6 = new Bishop(coord, turn);
```

```
                        return p.e6;
                    case 5:   p.e6 = new Queen(coord, turn);
                        return p.e6;
                }
            case 7:   switch(type_) {
                    case 2:   p.e7 = new Rook(coord, turn);
                        return p.e7;
                    case 3:   p.e7 = new Knight(coord, turn);
                        return p.e7;
                    case 4:   p.e7 = new Bishop(coord, turn);
                        return p.e7;
                    case 5:   p.e7 = new Queen(coord, turn);
                        return p.e7;
                }
            case 8:   switch(type_) {
                    case 2:   p.e8 = new Rook(coord, turn);
                        return p.e8;
                    case 3:   p.e8 = new Knight(coord, turn);
                        return p.e8;
                    case 4:   p.e8 = new Bishop(coord, turn);
                        return p.e8;
                    case 5:   p.e8 = new Queen(coord, turn);
                        return p.e8;
                }
            default:  Pawn temp = new Pawn(new Chess_Coord(-1, -1), true);
                return temp;
        }
    }

    private void DisplayBoard () {
        Clear();
        b.DisplayBoard();
    }

    private Chess_Coord[] InputCoord (boolean who) {
        String s1, s2;
        Chess_Coord[] vector = new Chess_Coord[2];
        boolean ok = true;
        if(who) System.out.println("\n\t\t\033[40;1;37m** Joueur BLANC **\033[0m");
        else System.out.println("\n\t\t\033[47;1;30m** Joueur NOIR **\033[0m");
        while(ok) {
            do {
                System.out.print("\t\033[1;36mCoordonnées de départ \033[31m(PIÈCE.X.Y)\033[1;36m : \033[1;33m");
                s1 = scan.nextLine();
            }
            while(!ValidateCoordString(s1, true));
            do{
```

```
                System.out.print("\t\033[1;36mCoordonnées de fin \033[31m(X.Y)\033[1;36m : \033[1;33m");
                s2 = scan.nextLine();
            }
            while(!ValidateCoordString(s2, false));
            System.out.print("\033[0m");

            // vérifie si la pièce est autorisée
            startCoord = new Chess_Coord(GetXFromString(s1, true), GetYFromString(s1, true));
            if(turn) {
                if(GetPieceFromString(s1, turn, true).GetType() == GetTypeFromString(s1) &&
                GetPieceFromString(s1, turn, true).GetColor() == true) {
                    endCoord = new Chess_Coord(GetXFromString(s2, false), GetYFromString(s2, false));
                    vector[0] = startCoord;
                    vector[1] = endCoord;
                    ok = false;
                }
            }
            else {
                if(GetPieceFromString(s1, turn, true).GetType() == GetTypeFromString(s1) &&
                GetPieceFromString(s1, turn, true).GetColor() == false) {
                    endCoord = new Chess_Coord(GetXFromString(s2, false), GetYFromString(s2, false));
                    vector[0] = startCoord;
                    vector[1] = endCoord;
                    ok = false;
                }
            }
        }
        return vector;
    }

    private boolean ValidateCoordString (String s, boolean isStart) {
        String[] tokens = s.split("[.]");
        if(isStart) { // pour coordonnées de départ
            if(s.length() == 5) {
                if(tokens[0].length() == 1) {
                    if(tokens[0].toLowerCase().equalsIgnoreCase("p") ||
                    tokens[0].toLowerCase().equalsIgnoreCase("r") ||
                    tokens[0].toLowerCase().equalsIgnoreCase("n") ||
                    tokens[0].toLowerCase().equalsIgnoreCase("b") ||
                    tokens[0].toLowerCase().equalsIgnoreCase("q") ||
                    tokens[0].toLowerCase().equalsIgnoreCase("k")) {
                        if(tokens[1].length() == 1) {
                            int x = Character.getNumericValue(tokens[1].charAt(0));
                            if(x > 0 && x < 9) {
                                if(tokens[2].length() == 1) {
                                    int y = Character.getNumericValue(tokens[2].charAt(0));
                                    if(y > 0 && y < 9) return true;
```

```
                                }
                            }
                        }
                    }
                }
            }
        }
        else { // pour coordonnées de fin
            if(s.length() == 3) {
                if(tokens[0].length() == 1) {
                    int x = Character.getNumericValue(tokens[0].charAt(0));
                    if(x > 0 && x < 9) {
                        if(tokens[1].length() == 1) {
                            int y = Character.getNumericValue(tokens[1].charAt(0));
                            if(y > 0 && y < 9) return true;
                        }
                    }
                }
            }
        }
        return false;
    }

    private int Promote (Chess_Player p) {
        String s;
        System.out.print("\n\033[5;1;91mNouveau type : \033[25;1;33m");
        s = scan.nextLine();
        System.out.print("\033[0m");
        int i = ValidatePromoteString(s);
        p.SetPromCount();
        return i;
    }

    private int ValidatePromoteString (String s) {
        s = s.toLowerCase();
        if(s.charAt(0) == 'r') return 2;
        else if(s.charAt(0) == 'n') return 3;
        else if(s.charAt(0) == 'b') return 4;
        else if(s.charAt(0) == 'q') return 5;
        else return 0;
    }

    private int GetTypeFromString (String s) {
        if(Character.valueOf(s.charAt(0)) == 'P' || Character.valueOf(s.charAt(0)) == 'p') return 1;
        else if(Character.valueOf(s.charAt(0)) == 'R' || Character.valueOf(s.charAt(0)) == 'r') return 2;
        else if(Character.valueOf(s.charAt(0)) == 'N' || Character.valueOf(s.charAt(0)) == 'n') return 3;
        else if(Character.valueOf(s.charAt(0)) == 'B' || Character.valueOf(s.charAt(0)) == 'b') return 4;
```

```
        else if(Character.valueOf(s.charAt(0)) == 'Q' || Character.valueOf(s.charAt(0)) == 'q') return 5;
        else if(Character.valueOf(s.charAt(0)) == 'K' || Character.valueOf(s.charAt(0)) == 'k') return 6;
        else return 0;
    }

    private int GetXFromString (String s, boolean opt1) {
        if(opt1) return Character.getNumericValue(s.charAt(2));
        else return Character.getNumericValue(s.charAt(0));
    }

    private int GetYFromString (String s, boolean opt1) {
        if(opt1) return Character.getNumericValue(s.charAt(4));
        else return Character.getNumericValue(s.charAt(2));
    }

    private Piece GetPieceFromString (String s, boolean turn_, boolean opt1) {
        if(turn_) return b.GetPieceByCoord(new Chess_Coord(GetXFromString(s, opt1), GetYFromString(s, opt1)), p1);
        else return b.GetPieceByCoord(new Chess_Coord(GetXFromString(s, opt1), GetYFromString(s, opt1)), p2);
    }

    private void EndTurn () {
        if(turn) turn = false;
        else turn = true;
    }

    public boolean GetTurn () {return turn;}

    public void Clear () {
        try {
            if(System.getProperty("os.name").contains("Windows"))
                new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
            else System.out.print("\033\143");
        }
        catch (IOException | InterruptedException ex) {System.out.println("Clear : " + ex.getMessage());}
    }

    private void Splash () {
        Clear();
        System.out.println("\n\n\n");
        System.out.println("\033[1;47;30m ################ #### #### ################
################## ");
        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("\033[1;40;37m###### ##################### ############
################## ");
            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(" ############### #### #### ################
################## \033[0m");
            System.out.println();
            System.out.print("\t\t\t\033[32mAppuyer sur une touche ...\033[0m");
            scan.nextLine();
        }
}
```

## ***“Chess_Player.java”***

```
package game;

import board.*;
import pieces.*;

public class Chess_Player {
	public Pawn p1, p2, p3, p4, p5, p6, p7, p8;
	public Rook r1, r2;
	public Knight n1, n2;
	public Bishop b1, b2;
	public Queen q;
	public King k;
	public int color;
	public Piece e1, e2, e3, e4, e5, e6, e7, e8;
	private int promCount;

	public Chess_Player (int num) {
		color = num;
		promCount = 0;
		Init();
	}

	private void Init () {
		if(color == 1) {
			// set régulier
			p1 = new Pawn(new Chess_Coord(1, 7), true);
			p2 = new Pawn(new Chess_Coord(2, 7), true);
			p3 = new Pawn(new Chess_Coord(3, 7), true);
			p4 = new Pawn(new Chess_Coord(4, 7), true);
			p5 = new Pawn(new Chess_Coord(5, 7), true);
			p6 = new Pawn(new Chess_Coord(6, 7), true);
			p7 = new Pawn(new Chess_Coord(7, 7), true);
			p8 = new Pawn(new Chess_Coord(8, 7), true);
			r1 = new Rook(new Chess_Coord(1, 8), true);
			r2 = new Rook(new Chess_Coord(8, 8), true);
			n1 = new Knight(new Chess_Coord(2, 8), true);
			n2 = new Knight(new Chess_Coord(7, 8), true);
			b1 = new Bishop(new Chess_Coord(3, 8), true);
			b2 = new Bishop(new Chess_Coord(6, 8), true);
			q = new Queen(new Chess_Coord(4, 8), true);
			k = new King(new Chess_Coord(5, 8), true);

			// set de test
			/*p1 = new Pawn(new Chess_Coord(1, 7), true);
			p2 = new Pawn(new Chess_Coord(2, 7), true);
			p3 = new Pawn(new Chess_Coord(3, 7), true);
```

```
        p4 = new Pawn(new Chess_Coord(4, 7), true);
        p5 = new Pawn(new Chess_Coord(5, 7), true);
        p6 = new Pawn(new Chess_Coord(6, 7), true);
        p7 = new Pawn(new Chess_Coord(7, 7), true);
        p8 = new Pawn(new Chess_Coord(8, 7), true);
        r1 = new Rook(new Chess_Coord(1, 6), true);
        r2 = new Rook(new Chess_Coord(8, 8), true);
        n1 = new Knight(new Chess_Coord(2, 6), true);
        n2 = new Knight(new Chess_Coord(7, 6), true);
        b1 = new Bishop(new Chess_Coord(3, 6), true);
        b2 = new Bishop(new Chess_Coord(6, 6), true);
        q = new Queen(new Chess_Coord(4, 6), true);
        k = new King(new Chess_Coord(5, 3), true);*/
    }
    else {
        // set régulier
        p1 = new Pawn(new Chess_Coord(1, 2), false);
        p2 = new Pawn(new Chess_Coord(2, 2), false);
        p3 = new Pawn(new Chess_Coord(3, 2), false);
        p4 = new Pawn(new Chess_Coord(4, 2), false);
        p5 = new Pawn(new Chess_Coord(5, 2), false);
        p6 = new Pawn(new Chess_Coord(6, 2), false);
        p7 = new Pawn(new Chess_Coord(7, 2), false);
        p8 = new Pawn(new Chess_Coord(8, 2), false);
        r1 = new Rook(new Chess_Coord(1, 1), false);
        r2 = new Rook(new Chess_Coord(8, 1), false);
        n1 = new Knight(new Chess_Coord(2, 1), false);
        n2 = new Knight(new Chess_Coord(7, 1), false);
        b1 = new Bishop(new Chess_Coord(3, 1), false);
        b2 = new Bishop(new Chess_Coord(6, 1), false);
        q = new Queen(new Chess_Coord(4, 1), false);
        k = new King(new Chess_Coord(5, 1), false);

        // set de test
        /*p1 = new Pawn(new Chess_Coord(1, 7), false);
        p2 = new Pawn(new Chess_Coord(2, 2), false);
        p3 = new Pawn(new Chess_Coord(3, 2), false);
        p4 = new Pawn(new Chess_Coord(4, 2), false);
        p5 = new Pawn(new Chess_Coord(5, 2), false);
        p6 = new Pawn(new Chess_Coord(6, 2), false);
        p7 = new Pawn(new Chess_Coord(7, 2), false);
        p8 = new Pawn(new Chess_Coord(8, 2), false);
        r1 = new Rook(new Chess_Coord(1, 1), false);
        r2 = new Rook(new Chess_Coord(8, 1), false);
        n1 = new Knight(new Chess_Coord(2, 3), false);
        n2 = new Knight(new Chess_Coord(7, 3), false);
        b1 = new Bishop(new Chess_Coord(3, 3), false);
```

```
                b2 = new Bishop(new Chess_Coord(6, 3), false);
                q = new Queen(new Chess_Coord(4, 3), false);
                k = new King(new Chess_Coord(5, 1), false);*/
            }
        }

        public Piece GetPiece (Piece o) {
            if(o.equals(p1)) return p1;
            else if(o.equals(p2)) return p2;
            else if(o.equals(p3)) return p3;
            else if(o.equals(p4)) return p4;
            else if(o.equals(p5)) return p5;
            else if(o.equals(p6)) return p6;
            else if(o.equals(p7)) return p7;
            else if(o.equals(p8)) return p8;
            else if(o.equals(r1)) return r1;
            else if(o.equals(r2)) return r2;
            else if(o.equals(n1)) return n1;
            else if(o.equals(n2)) return n2;
            else if(o.equals(b1)) return b1;
            else if(o.equals(b2)) return b2;
            else if(o.equals(q)) return q;
            else return k;
        }

        public int GetPromCount () {return promCount;}

        public void SetPromCount () {promCount++;}
}
```

## ***“Chess_Setup.java”***

```
package game;

import board.*;
import pieces.*;

public class Chess_Setup {
	private Chess_Board gBoard;
	private Chess_Player player1, player2;

	public Chess_Setup () {
		Init();
		InitPiecesOnBoard(player1);
		InitPiecesOnBoard(player2);
	}

	private void Init () {
		gBoard = new Chess_Board();
		player1 = new Chess_Player(1);
		player2 = new Chess_Player(2);
	}

	private void InitPiecesOnBoard (Chess_Player p) {
		SetPieceOnBoard(p.p1);
		SetPieceOnBoard(p.p2);
		SetPieceOnBoard(p.p3);
		SetPieceOnBoard(p.p4);
		SetPieceOnBoard(p.p5);
		SetPieceOnBoard(p.p6);
		SetPieceOnBoard(p.p7);
		SetPieceOnBoard(p.p8);
		SetPieceOnBoard(p.r1);
		SetPieceOnBoard(p.r2);
		SetPieceOnBoard(p.n1);
		SetPieceOnBoard(p.n2);
		SetPieceOnBoard(p.b1);
		SetPieceOnBoard(p.b2);
		SetPieceOnBoard(p.q);
		SetPieceOnBoard(p.k);
	}

	private void SetPieceOnBoard (Piece o) {
		gBoard.UpdateBoard(o.GetColor(), o.GetType(), o.GetPos());
	}
```

```
    public Chess_Board GetBoard () {return gBoard;}

    public Chess_Player GetPlayer (int num) {
        if(num == 1) return player1;
        else return player2;
    }
}
```

## ***“Chess.java”***

```
import game.*;

public class Chess {
	public static void main (String[] args) throws NullPointerException {
		Chess_Game game = new Chess_Game();
		try {
			while(true) game.PlayTurn();
		}
		catch(NullPointerException e) {System.out.println("Erreur : " + e.getMessage());}
	}
}
```