DESCRIPCIÓN:
La idea del juego es muy simple es un ajedrez para jugar entre dos con reglas estándar que tiene aparte del tablero con todas las piezas un contador que pone la cantidad de piezas que se ha comido cada jugador en la partida.

USO:
El juego se inicia desde la clase juego y empieza directamente le pide que mueva al jugador de las piezas blancas en este caso las letras en mayúscula, le pide la coordenada de la pieza y a la coordenada donde quiere moverla las coordenadas son los numero y letras de arriba e izq desde ahí ya continua el juego normal

ORGANIZACIÓN:
El juego está organizado en varias clases:

Clase Juego: Es la clase que controla el juego la que permite hacer los movimientos de cada pieza correspondiente en el tablero.

```
import java.util.Scanner;

public class Juego {
   private Tablero tablero;
   private boolean turnoBlanco;

   public Juego() {
       tablero = new Tablero();
       turnoBlanco = true;
   }

   public void jugar() {
       Scanner scanner = new Scanner(System.in);

       while (true) {
           tablero.mostrarTablero();

           String colorTurno = turnoBlanco ? "Blanco" : "Negro";
           System.out.println("Turno de las piezas " + colorTurno);
           System.out.println("Ingrese las coordenadas de la pieza
que desea mover (fila columna):");
           String inicioPos = scanner.next();
           String finPos = scanner.next();

           int inicioX = 8 -
Character.getNumericValue(inicioPos.charAt(1));
           int inicioY = (int) inicioPos.charAt(0) - 'a';
           int finX = 8 -
Character.getNumericValue(finPos.charAt(1));
           int finY = (int) finPos.charAt(0) - 'a';
```

```
            if (tablero.moverPieza(inicioX, inicioY, finX, finY)) {
                turnoBlanco = !turnoBlanco;
            } else {
                System.out.println("Movimiento no válido, intente de
nuevo.");
            }
        }
    }

    public static void main(String[] args) {
        Juego juego = new Juego();
        juego.jugar();
    }
}
```

Clase InterfazTerminal: Es la clase de la interfaz de la terminal para que se vea el juego como toca.

```
public class InterfazTerminal {
    private Tablero tablero;

    public InterfazTerminal() {
        tablero = new Tablero();
    }

    public void mostrarTablero() {
        for (int fila = 0; fila < 8; fila++) {
            for (int columna = 0; columna < 8; columna++) {
                Pieza pieza = tablero.obtenerPieza(fila, columna);
                String simbolo = (pieza != null) ?
pieza.obtenerSimbolo() : ".";
                System.out.print(simbolo + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        InterfazTerminal interfaz = new InterfazTerminal();
        interfaz.mostrarTablero();
    }
}
```

Clase Tablero: Es la clase que maneja el tablero y lo que puede hacer cada pieza en el, también controla los métodos del contador de piezas comidas y si hay victoria al comer el rey rival.

```
public class Tablero {
    private Pieza[][] tablero;
    private int blancasComidas;
    private int negrasComidas;
    private boolean victoriaBlancas;
    private boolean victoriaNegras;

    public Tablero() {
        tablero = new Pieza[8][8];
        prepararTablero();
        contarBlancasComidas();
        contarNegrasComidas();
        victoriaBlancas = false;
        victoriaNegras = false;
    }

    public Pieza obtenerPieza(int x, int y) {
        return tablero[x][y];
    }

    public void colocarPieza(int x, int y, Pieza pieza) {
        tablero[x][y] = pieza;
    }

    public void prepararTablero() {
        // Peones
        for (int i = 0; i < 8; i++) {
            tablero[1][i] = new Peon(false);
            tablero[6][i] = new Peon(true);
        }

        // Torres
        tablero[0][0] = new Torre(false);
        tablero[0][7] = new Torre(false);
        tablero[7][0] = new Torre(true);
        tablero[7][7] = new Torre(true);

        // Caballos
        tablero[0][1] = new Caballo(false);
        tablero[0][6] = new Caballo(false);
        tablero[7][1] = new Caballo(true);
        tablero[7][6] = new Caballo(true);

        // Alfiles
```

```
        tablero[0][2] = new Alfil(false);
        tablero[0][5] = new Alfil(false);
        tablero[7][2] = new Alfil(true);
        tablero[7][5] = new Alfil(true);

        // Reinas
        tablero[0][3] = new Reina(false);
        tablero[7][3] = new Reina(true);

        // Reyes
        tablero[0][4] = new Rey(false);
        tablero[7][4] = new Rey(true);
    }

    public boolean moverPieza(int inicioX, int inicioY, int finX, int
finY) {
        Pieza pieza = obtenerPieza(inicioX, inicioY);
        if (pieza != null && pieza.esMovimientoValido(this, inicioX,
inicioY, finX, finY)) {
            colocarPieza(finX, finY, pieza);
            colocarPieza(inicioX, inicioY, null);
            return true;
        }
        return false;
    }

    public void mostrarTablero() {
        System.out.println("  a b c d e f g h");
        for (int i = 0; i < 8; i++) {
            System.out.print((8 - i) + " ");
            for (int j = 0; j < 8; j++) {
                Pieza pieza = tablero[i][j];
                System.out.print((pieza != null ?
pieza.obtenerSimbolo() : ".") + " ");
            }
            System.out.println();
        }
    }
    public void contarBlancasComidas() {
        blancasComidas = 0;
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                Pieza pieza = tablero[i][j];
                if (pieza != null && pieza.estaComida() &&
pieza.esBlanco()) {
                    blancasComidas++;
                }
            }
```

```
        }
    }
    public void contarNegrasComidas() {
        negrasComidas = 0;
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                Pieza pieza = tablero[i][j];
                if (pieza != null && pieza.estaComida() &&
!pieza.esBlanco()) {
                    negrasComidas++;
                }
            }
        }
    }
    public boolean Victoria() {
        if (blancasComidas >= 16) {
            victoriaNegras = true;
            return true;
        }
        if (negrasComidas >= 16) {
            victoriaBlancas = true;
            return true;
        }

        return false;
    }

    public boolean hayVictoriaBlancas() {
        return victoriaBlancas;
    }

    public boolean hayVictoriaNegras() {
        return victoriaNegras;
    }

    public static void main(String[] args) {
        Tablero tablero = new Tablero();
        tablero.mostrarTablero();
    }
}
```

Clase Pieza: Es la clase que controla todo lo común de cada pieza como el comer otras piezas.

```
public abstract class Pieza {
    private boolean esBlanco;
    private String simbolo;
```

```
    public Pieza(boolean esBlanco, String simbolo) {
        this.esBlanco = esBlanco;
        this.simbolo = simbolo;
    }

    public abstract boolean esMovimientoValido(Tablero tablero, int
inicioX, int inicioY, int finX, int finY);

    public boolean esBlanco() {
        return esBlanco;
    }

    public String obtenerSimbolo() {
        return simbolo;
    }

    public void capturarPieza(Pieza piezaCapturada) {
        if (piezaCapturada != null) {
            String mensaje = esBlanco ? "Las blancas han capturado
una " : "Las negras han capturado una ";
            mensaje += piezaCapturada.getClass().getSimpleName();
            System.out.println(mensaje);
        }
    }
    private boolean comida;

    public boolean estaComida() {
        return comida;
    }

    public void setComida(boolean comida) {
        this.comida = comida;
    }
}
```

Clase Peon, Torre, Caballo, Alfil, Reina y Rey: Que están adjuntas a la clase pieza a base de una herencia y son las clases de cada pieza que controla el movimiento individual de cada una de ellas.

Peon:

```
public class Peon extends Pieza {
    private boolean primerMovimiento;

    public Peon(boolean esBlanco) {
        super(esBlanco, esBlanco ? "P" : "p");
        primerMovimiento = true;
    }
```

```
    @Override
    public boolean esMovimientoValido(Tablero tablero, int inicioX,
int inicioY, int finX, int finY) {
        int direccion = esBlanco() ? -1 : 1;
        int deltaX = finX - inicioX;
        int deltaY = finY - inicioY;

        if (deltaX == direccion && deltaY == 0 &&
tablero.obtenerPieza(finX, finY) == null) {
            primerMovimiento = false;
            return true;
        }

        if (primerMovimiento && deltaX == 2 * direccion && deltaY
== 0 && tablero.obtenerPieza(finX, finY) == null &&
                tablero.obtenerPieza(inicioX + direccion, inicioY)
== null) {
            primerMovimiento = false;
            return true;
        }

        if (Math.abs(deltaX) == 1 && deltaY == 1 &&
tablero.obtenerPieza(finX, finY) != null &&
                tablero.obtenerPieza(finX, finY).esBlanco() !=
esBlanco()) {
            primerMovimiento = false;
            return true;
        }

        return false;
    }
}
```

Torre:

```
public class Torre extends Pieza {
    public Torre(boolean esBlanco) {
        super(esBlanco, esBlanco ? "T" : "t");
    }

    @Override
    public boolean esMovimientoValido(Tablero tablero, int inicioX,
int inicioY, int finX, int finY) {
        if (inicioX == finX) {
            int step = (inicioY < finY) ? 1 : -1;
            for (int i = inicioY + step; i != finY; i += step) {
                if (tablero.obtenerPieza(inicioX, i) != null) {
                    return false;
                }
```

```
            }
            return true;
        } else if (inicioY == finY) {
            int step = (inicioX < finX) ? 1 : -1;
            for (int i = inicioX + step; i != finX; i += step) {
                if (tablero.obtenerPieza(i, inicioY) != null) {
                    return false;
                }
            }
            return true;
        } else {
            return false;
        }
    }
}
```

Caballo:

```
public class Caballo extends Pieza {
    public Caballo(boolean esBlanco) {
        super(esBlanco, esBlanco ? "C" : "c");
    }

    @Override
    public boolean esMovimientoValido(Tablero tablero, int inicioX,
int inicioY, int finX, int finY) {
        int dx = Math.abs(finX - inicioX);
        int dy = Math.abs(finY - inicioY);
        return (dx == 2 && dy == 1) || (dx == 1 && dy == 2);
    }
}
```

Alfil:

```
public class Alfil extends Pieza {
    public Alfil(boolean esBlanco) {
        super(esBlanco, esBlanco ? "A" : "a");
    }

    @Override
    public boolean esMovimientoValido(Tablero tablero, int inicioX,
int inicioY, int finX, int finY) {
        int dx = Math.abs(finX - inicioX);
        int dy = Math.abs(finY - inicioY);
        if (dx != dy) {
            return false;
        }

        int stepX = (finX > inicioX) ? 1 : -1;
        int stepY = (finY > inicioY) ? 1 : -1;
```

```
        for (int i = inicioX + stepX, j = inicioY + stepY; i !=
finX; i += stepX, j += stepY) {
            if (tablero.obtenerPieza(i, j) != null) {
                return false;
            }
        }
        return true;
    }
}
```

Reina:

```
public class Reina extends Pieza {
    public Reina(boolean esBlanco) {
        super(esBlanco, esBlanco ? "R" : "r");
    }

    @Override
    public boolean esMovimientoValido(Tablero tablero, int inicioX,
int inicioY, int finX, int finY) {
        int dx = Math.abs(finX - inicioX);
        int dy = Math.abs(finY - inicioY);

        if (dx == dy || inicioX == finX || inicioY == finY) {
            if (inicioX == finX) {
                int step = (inicioY < finY) ? 1 : -1;
                for (int i = inicioY + step; i != finY; i += step)
{
                    if (tablero.obtenerPieza(inicioX, i) != null) {
                        return false;
                    }
                }
                return true;
            } else if (inicioY == finY) {
                int step = (inicioX < finX) ? 1 : -1;
                for (int i = inicioX + step; i != finX; i += step)
{
                    if (tablero.obtenerPieza(i, inicioY) != null) {
                        return false;
                    }
                }
                return true;
            } else {
                int stepX = (finX > inicioX) ? 1 : -1;
                int stepY = (finY > inicioY) ? 1 : -1;
                for (int i = inicioX + stepX, j = inicioY + stepY;
i != finX; i += stepX, j += stepY) {
                    if (tablero.obtenerPieza(i, j) != null) {
                        return false;
```

```
                    }
                }
                return true;
            }
        } else {
            return false;
        }
    }
}
```

Rey:

```
public class Rey extends Pieza {
    public Rey(boolean esBlanco) {
        super(esBlanco, esBlanco ? "K" : "k");
    }

    @Override
    public boolean esMovimientoValido(Tablero tablero, int inicioX,
int inicioY, int finX, int finY) {
        int dx = Math.abs(finX - inicioX);
        int dy = Math.abs(finY - inicioY);
        return dx <= 1 && dy <= 1;
    }
}
```