

- `Random` : Clase utilizada para generar números aleatorios
- `Scanner` : Clase para leer la entrada del usuario.

```
import java.util.Random;
import java.util.Scanner;
```

CLASE DOMINO

Esta es la clase principal, Contendrá la lógica para gestionar el juego.

```
public class Domino {
}
```

CLASE ANIDADA FICHA

¿Por qué una clase anidada?

- Nos deja encapsular la funcionalidad en un solo lugar, facilitando el mantenimiento y comprensión del código
- Facilita la organización del código, ayuda a la navegación y comprensión del mismo

```
static class Ficha {
    int ladoA;
    int ladoB;

    Ficha(int ladoA, int ladoB){
        this.ladoA = ladoA;
        this.ladoB = ladoB;
    }

    void girar(){
        int temp = ladoA;
        ladoA = ladoB;
        ladoB = temp;
    }

    @Override
    public String toString(){
        return "[" + ladoA + "|" + ladoB + "]";
    }
}
```

En este caso nuestra clase es estática porque no necesita acceso a las variables de la instancia

`domino`.

- `int ladoA` , `int ladoB` : Variables que representan los puntos en los extremos de la ficha.
- `Ficha(int ladoA, int ladoB)` : Constructor que inicializa una ficha con puntos en ambos extremos.
- `void girar ()` : Método que intercambia los valores de `ladoA` y `ladoB` .
- `@Override public String toString ()` : Método que devuelve una representación en texto de la ficha.

METODO FICHAS ALEATORIAS

El propósito original de este método es generar 28 fichas de dominó únicas y aleatorias, pero en nuestro caso vamos a generar solo 9 para que en nuestra consola no se vea mal la interfaz.

```
private static Queue<Ficha> generarFichasAleatorias(){
    Queue<Ficha> fichas = new LinkedList<>();
    Random random = new Random();
    boolean [][] generadas = new boolean [7][7];

    while (fichas.size() < 9) {
        int ladoA = random.nextInt(7);
```

```

int ladoB = random.nextInt(7);

if (!generadas[ladoA][ladoB] && !generadas[ladoB][ladoA]){
    fichas.add(new Ficha(ladoA,ladoB));
    generadas[ladoA][ladoB] = true;
    generadas[ladoB][ladoA] = true;
}
}
return fichas;
}

```

- `Queue <Ficha> fichas` : Cola para almacenar las fichas generadas.
- `Random random` : Generador de números aleatorios.
- `boolean [][] generadas` : Matriz para asegurar que no se repitan las combinaciones de fichas.
- `while (fichas.size() < 9)` : Bucle que se ejecuta para generar las 9 fichas (originalmente deberían ser 28).
- `int ladoA = random.nextInt(7) , int ladoB = random.nextInt(7)` : Genera dos números aleatorios entre 0 y 6.
- `if (!generadas[ladoA][ladoB] && !generadas[ladoB][ladoA])` : Verifica si la combinación de puntos no ha sido generada antes.
- `fichas.add(new Ficha(ladoA,ladoB))` : Agrega una nueva ficha a la cola.
- `generadas[ladoA][ladoB] = true , generadas[ladoB][ladoA] = true` : Marca la combinación como generada.

MÉTODO MAIN

```

public static void main(String [] args){
    Queue <Ficha> fichas disponibles = generarFichasAleatorias();
    LinkedList<Ficha> fichasEnJuego = new LinkedList<>();

    Ficha fichaInicial = fichasDisponibles.poll();
    fichasEnJuego.add(fichaInicial);

    Scanner in = new Scanner(System.in);
    boolean juegoTerminado = false;

    while (!fichasDisponibles.isEmpty() && !juegoTerminado){
        System.out.println("Fichas en juego: ");
        for (Ficha ficha : fichasEnJuego){
            System.out.print(ficha + " ");
        }
        System.out.print();

        Ficha primeraFichaEnJuego = fichasEnJuego.getFirst();
        Ficha ultimaFichaEnJuego = fichasEnJuego.getLast();
        System.out.println("Ficha inicial en juego: " + primeraFichaEnJuego);
        System.out.println("Ficha final en juego: " + ultimaFichaEnJuego);
        System.out.println("Tus fichas disponibles: ");

        int index = 1;
        for (Ficha ficha : fichasDisponibles){
            System.out.println(index + ". " + ficha);
            index++;
        }
    }
}

```

```

System.out.println("Eligre una ficha para jugar (ingrese el número correspondiente): ");
int eleccion = in.nextInt();

if(eleccion < 1 || eleccion > fichasDisponibles.size()){
    System.out.println("Selección inválida. Intenta de nuevo. ");
    continue;
}

Ficha fichasSeleccionada = null;
index = 1;

for (Ficha ficha: fichasDisponibles){
    if (index == eleccion){
        fichaSeleccionada = ficha;
        break;
    }

    index++;
}

boolean fichaColocada = false;
if (ultimaFichaEnJuego.ladoB == fichaSeleccionada.ladoA){
    fichasEnJuego.addLast(fichaSeleccionada);
    fichaColocada = true;
} else if (ultimaFichaEnJuego.ladoB == fichaSeleccioanda.ladoB){
    fichaSeleccioanda.girar();
    fichasEnJuego.addLast(fichaSeleccionada);
    fichaColocada = true;
}

if (!fichaColocada){
    if (primeraFichaEnJuego.ladoA == fichaSeleccionada.ladoB){
        fichasEnJuego.addFirst(fichaSeleccionada);
        fichaColocada = true;

    } else if (primeraFichaEnJuego.ladoA == fichaSeleccionada.ladoA){
        fichasSeleccionada.girar();
        fichasEnJuego.addFirst(fichaSeleccionada);
        fichaColocada = true;
    }
}

if (fichaColocada){
    fichasDisponibles.remove(fichaSeleccionada);
} else {
    System.out.println("No puede colocar esta ficha en ningún lado. Intenta otra vez.");
}

if (fichasDisponibles.isEmpty()){
    System.out.println("No hay más fichas disponibles. Fin del juego");
    juegoTerminado = true;
}
}

System.out.println("Secuencia de fichas jugada: ");
for (Ficha ficha : fichasEnJuego){
    System.out.print(ficha + " ");
}

```

```
in.close();  
}
```

1. Generar fichas aleatorias y prepararlas para el juego:

```
Queue<Ficha> fichasDisponibles = generarFichasAleatorias();  
LinkedList<Ficha> fichasEnJuego = new LinkedList<>();
```

- `fichasDisponibles` : Cola que contiene las fichas generadas aleatoriamente.
- `fichasEnJuego` : Lista enlazada que contiene las fichas en la secuencia del juego.

2. Inicializar el juego con la primera ficha:

```
Ficha fichaInicial = fichasDisponibles.poll();  
fichasEnJuego.add(fichaInicial);
```

- `fichaInicial` : Se saca la primera ficha de las disponibles y se añade a la secuencia de juego.

3. Preparar para recibir la entrada del usuario:

```
Scanner in = new Scanner (System.in);  
boolean juego terminado = false;
```

- `scanner` : para leer la entrada del usuario.
- `juegoTerminado` : Para controlar el fin del juego-

4. Bucle principal del juego:

Este bucle se repite mientras haya fichas disponibles y el juego no haya terminado

```
while (!fichasDisponibles.isEmpty() && !juegoTerminado){  
    // Mostrar estado actual del juego  
    // Pedir al usuario que seleccione una ficha  
    // Intentar colocar la ficha escogida en la secuencia del juego  
    // Verificar si se puede colocar en el lado izquierdo o derecho  
    // Actualizar la secuencia del juego y las fichas disponibles  
    // verificar si el juego termino  
}
```

5. Mostrar las fichas en juego y las disponibles:

Imprimir la secuencia actual del juego.

```
System.out.println("Fichas en juego: ");  
for (Ficha ficha : fichasEnJuego){  
    System.out.print(ficha + " ");  
}  
System.out.println();
```

6. Pedir al usuario que elija un ficha:


```
System.out.println("Elige una ficha para jugar (ingresa el número correspondiente): ");
int eleccion = in.nextInt;
```

7. Validar la elección del usuario:

Verifica si la elección del usuario está dentro del rango de las fichas disponibles. Si no lo está, muestra un mensaje de error y vuelve al inicio del bucle.

```
if ( eleccion < 1 || eleccion > fichasDisponibles.size()){
    System.out.println("Selección inválida. Intenta de nuevo.");
    continue;
}
```

8. Obtener la ficha seleccionada por el usuario:

Recorre las fichas disponibles hasta encontrar la ficha seleccionada por el usuario

```
Ficha fichaSeleccionada = null;

index = 1;
for ( Ficha ficha : fichasDisponibles){
    if (index == eleccion){
        fichaSeleccionada = ficha;
        break;
    }
    index++;
}
```

9. Intentar colocar la ficha seleccionada en la secuencia de juego:

Se verifica si la ficha seleccionada puede ser colocada en algún extremo de la secuencia de juego. Si se puede, se añade a la secuencia correspondiente (inicio o fin).

```
boolean fichaColocada = false;

if (ultimaFichaEnJuego.ladoB == fichaSeleccionada.ladoA){
    fichasEnJuego.addLast(fichaSeleccionada);
    fichaColocada = true;
} else if (ultimaFichaEnJuego.ladoB == fichaSeleccionada.ladoB){
    fichaSeleccionada.girar();
    fichasEnJuego.addLast(fichaSeleccionada);
    fichaColocada = true;
}

if (!fichaColocada){
    if (primeraFichaEnJuego.ladoA == fichaSeleccioanda.ladoB){
        fichasEnJuego.addFirst(fichaSeleccionada);
        fichaColocada = true;
    } else if ( primeraFichaEnJuego.ladoA == fichaSeleccionada.ladoA){
        fichaSeleccionada.girar();
        fichasEnJuego.addFirst(fichaSeleccionada);
        fichaColocada = true;
    }
}
```

```
}  
}
```

10. Actualizar las fichas disponibles y verificar si el juego ha terminado

```
if (fichaColocada){  
    fichasDisponible.remove(fichaSeleccionada)  
} else {  
    System.out.println("No puedes colocar esta ficha en ningún extremo. Intenta de nuevo.");  
}  
  
if (fichasDisponibles.isEmpty()){  
    System.out.println("No hay más fichas disponibles. Fin del juego,");  
    juegoTerminado = true;  
}
```

- Si la ficha se pudo colocar en la secuencia del juego, se elimina de las fichas disponibles. Si no, se muestra un mensaje de error.
- Se verifica si ya no quedan fichas disponibles en el mazo. Si es así, se indica que el juego ha terminado.

11. Mostrar la secuencia de fichas jugadas y cerrar el scanner:

```
System.out.println("Secuencia de fichas jugadas: ");  
for (Ficha ficha : fichasEnJuego){  
    System.out.print(ficha + " ");  
}  
  
in.close();
```

CONCLUSIONES

La división del código en funciones cortas y clases anidadas facilita la comprensión de cada componente por separado. Cada función tiene una tarea específica, lo que simplifica su comprensión.

El uso de las estructuras de datos como colas y listas enlazadas es apropiado para representar las fichas de domino y gestionar el estado del juego, los datos y estructura se pueden manipular y organizar fácilmente.

El flujo de ejecución del programa sigue un patrón lógico basado en el primer mapa mental, esto facilitó seguir la secuencia de acciones realizadas durante el juego y ayudó a estructurar el código de forma que se ejecute en el orden correcto.