Ejercicio: La máquina de mate

Tiene 4 estados:

-Caliente: el agua está muy caliente

-Fría: el agua está muy fría

-Ok: el agua está ok

-Vacío: no hay más agua

Representar la máquina de mate con una serie de métodos:

-cebarMate()

-enfriarAgua()

-calentarAgua()

-llenarTermo()

Métodos de la máquina:
cebarMate(): depende del estado
enfriarAgua() enfría el agua y cambia el estado
calentarAgua() calienta el agua y cambia el estado
llenarTermo() llena el termo y cambia el estado

Métodos del estado: cebarMate()
¿Qué hace EstadoFria?
-No se puede cebar -> envía calentarAgua()
¿Qué hace EstadoCaliente?
-No se puede cebar -> envía enfriarAgua()
¿Qué hace EstadoOk?
-Ceba mate
¿Qué hace EstadoVacio?
-No se puede cebar -> envía llenarTermo()

Implementemos:

MaquinaMate

```
Estado estado;
double temperatura;
double nivel;

public MaquinaMate() {
   temperatura = 90;
   nivel = 0;
   estado = new EstadoVacio();
}

public void cebarMate(){
       getEstado().puedoCebarMate(this);
}

public void abrirCanilla() {
...
  if (nivel == 0)
         setEstado(new EstadoVacio());
  if (temperatura < 85)
       setEstado(new EstadoFria());
   if (temperatura > 95)
       setEstado(new EstadoCaliente());
}

public void calentarAgua(){
```

```
  ....
   //ahora esta ok
}

public void enfriarAgua(){
  ....
   //ahora esta ok
}

public void llenarTermo(){
  ....
   setEstado(new EstadoOk());
}

public abstract class Estado() {
    public void puedoCebarMate(MaquinaMate mm);
}

public class EstadoOk extends Estado() {
    public void puedoCebarMate(MaquinaMate mm) {
        mm.abrirCanilla();
    }
}

public class EstadoFria extends Estado() {
    public void puedoCebarMate(MaquinaMate mm) {
        mm.calentarAgua();
    }
}
```

```
public class EstadoCaliente extends Estado() {
   public void puedoCebarMate(MaquinaMate mm) {
      mm.enfriarAgua();
   }
}

public class EstadoVacio extends Estado() {
   public void puedoCebarMate(MaquinaMate mm) {
      mm.llenarTermo();
   }
}
```

Ejercicio: Recorrido desde Jujuy hasta Río Gallegos.
Proponer un recorrido por Argentina:
ViajeTuristico: Auto por Ruta 40
ViajeRapido: Avión
ViajeEconomico: Mochilero por rutas internas

Cada viaje toma una ruta diferente

Pensemos

Ejercicio: Generación de monstruos

Algoritmo para dibujar monstruos:

```
public void dibujar() {
	dibujarCabeza();
	dibujarCuerpo();
	dibujarBrazos();
	dibujarPiernas();
	dibujarCola();
}

public void dibujarCabeza() {
	getCabeza().dibujar();
}
```

Necesito generar monstruos aleatorios:
Cabezas: dragón, redonda y de robot
Cuerpos: peludo, con pinches, musculoso
Brazos: cortito tipo dinosaurio, pulpo
Piernas: peludas de yeti, pata corta
Colas: cerdo, dragón

¿Cuántas tipos de monstruos permiten generar?

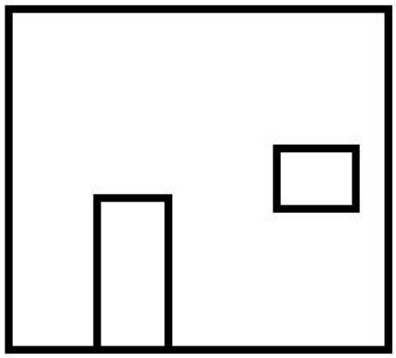
¿Cuántas clases necesito para generar todos?
Juego

```
Montruo mio = new Monstruo();
mio.setCabeza(new CabezaDragon());
mio.setCuerpo(new CuerpoPeludo());
```

```
mio.setBrazos(new BrazoCorto());
...
mio.dibujar();
```

Ejercicio: Dibujar una casita.
Base: Un cuadrado con una puerta y una ventana
Detalles y terminaciones: cortinas, balcón, techo a 2 aguas, chimenea, camino, alero de puerta, árbol, flores, perro, nubes, sol, lluvia, rayos.

¿Cuántos atributos le puedo agregar? ¿Cuántas combinaciones puedo generar?



Ejercicio: sala de espera:

Pasos:

- El paciente tiene turno pero no llegó
- El paciente llegó, se le carga el bono de consulta y queda esperando que lo llamen
- El paciente está siendo llamado
- El paciente ingresó al consultorio
- El paciente salió del consultorio

1. Identificar: Clases, subclases, métodos
2. Modelar las diferentes situaciones: cargarBono(), llamar() y bajo qué condiciones se puede llamar a cada método
3. Pensar en diferentes aplicaciones para modelar: ej: enviar un sms cuando lo llaman, mostrar en un TV, etc.

Implementemos

```
class Turno {
  EstadoTurno estadoTurno;

  cambiarEstado(EstadoTurno estadoTurno){
    this.estadoTurno =  estadoTurno;
  }

  cargarBono(Bono bono){
      estadoTurno.puedoCargarBono(this, bono);
  }
  processCargarBono(Bono bono) {
     // Guardo el bono en la bd
     cambiarEstado(new EnEspera());
  }

  llamar() {
     estadoTurno.puedoLlamar(this);
 }
 processLLamar() {
    //envio SMS y muestro en TV
    cambiarEstado(new EstadoSiendoLlamado());
 }

 ingreso() {
        estadoTurno.ingreso(this);
  }
}
```

```
abstract class EstadoTurno {
  puedoCargarBono(Turno turno, Bono bono){}
  puedoLlamar(Turno turno) {}
  ingreso(Turno turno) {}
}

class EstadoTurnoLlego extends EstadoTurno {
    puedoCargarBono(Turno turno, Bono bono){
          turno.processCargarBono(bono);
    }
}

class EstadoEnEspera extends EstadoTurno {
  puedoLlamar(Turno turno) {
       turno.processLlamar();
   }
}

class EstadoSiendoLlamado extends EstadoTurno {
  puedoLlamar(Turno turno) {
       turno.processLlamar();
  }
  ingreso(Turno turno) {
       turno.cambiarEstado(new Ingreso());
  }
}
```

Muy importante identificar que

1. El estado es del Turno y no del Paciente.

2. Las acciones se ejecutan sobre el turno, pero dependiendo de su estado.
3. Nunca preguntar por el estado, siempre delegar
4. Puede haber double dispatching o no.