Entornos de Desarrollo (ED) Unidad de Trabajo 8 (UT8) Tarea 03

OBJETIVOS DE APRENDIZAJE

- Dibujar diagramas de clases siguiendo unas especificaciones.
- Valorar la utilidad de los diagramas UML para transmitir información técnica de manera fiel y eficiente.
- Relacionar los diagramas de clases con los conceptos de la programación orientada a objetos.

TA03 Genera diagramas de clases

Dibuja diagramas de clases a partir de las especificaciones y el código siguientes

- Utiliza una de las herramientas que hemos seleccionado en la tarea 1
- Responde al final de este documento, en el mismo orden en el que te muestro los enunciados e indicando el número de enunciado

1. Restaurante

1a. Descripción del proyecto

En un sistema de pedidos de un restaurante, queremos modelar la clase `Pedido`. Esta clase debe cumplir con los siguientes requisitos:

- **Atributos:**
  - `numeroPedido`: un identificador único para cada pedido.
  - `mesa`: número de la mesa que realizó el pedido.
  - `estado`: estado del pedido (`pendiente`, `en preparación`, `servido`).
  - `platos`: un array de platos incluidos en el pedido.
  - `precioTotal`: comienza en `0.0`.
- **Métodos:**
  - `calcularTotal()`: Calcula el precio total sumando los precios de todos los platos en el pedido.
  - `cambiarEstado(nuevoEstado: String)`: Cambia el estado del pedido.
  - `imprimirRecibo()`: Muestra la información del pedido.

1b. Código Java

```
public class Pedido {
    private int numeroPedido;
    private int mesa;
    protected String estado;
    String[] platos;
    private double precioTotal = 0.0;
```

```
    public Pedido(int numeroPedido, int mesa, String[] platos) {
        this.numeroPedido = numeroPedido;
        this.mesa = mesa;
        this.platos = platos;
        this.estado = "pendiente";
    }

    public double calcularTotal() {
        precioTotal = platos.length * 10.0;
        return precioTotal;
    }

    public void cambiarEstado(String nuevoEstado) {
        this.estado = nuevoEstado;
    }

    protected void imprimirRecibo() {
        System.out.println("Pedido #" + numeroPedido);
        System.out.println("Mesa: " + mesa);
        System.out.println("Estado: " + estado);
        System.out.println("Total: " + precioTotal + "€");
    }
}
```

incrusta aquí tu respuesta

2. Transporte

2a. Descripción del proyecto

En un sistema de gestión de transporte, queremos modelar la relación entre un `Autobús` y sus `Pasajeros`. Un autobús puede transportar múltiples pasajeros, y los pasajeros pueden existir independientemente del autobús al que suban.

- **Clase `Autobus`**:
  - **Atributos:**
    - `matricula`: identifica el autobús.
    - `capacidad`: número máximo de pasajeros.
    - `pasajeros`: array de objetos `Pasajero`.
  - **Métodos:**
    - `subirPasajero(Pasajero pasajero)`: Añade un pasajero al autobús si hay espacio disponible.
    - `bajarPasajero(Pasajero pasajero)`: Elimina un pasajero del autobús si está a bordo.
- **Clase `Pasajero`**:
  - **Atributos:**

- `nombre`: nombre del pasajero.
- `dni`: documento de identidad del pasajero.

- **Métodos:**

  - `obtenerDatos()`: Devuelve una cadena con el nombre y DNI del pasajero.

2b. Código Java

```
public class Pasajero {
    private String nombre;
    private String dni;

    public Pasajero(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
    }

    public String obtenerDatos() {
        return nombre + " (" + dni + ")";
    }
}

public class Autobus {
    private String matricula;
    private int capacidad;
    private Pasajero[] pasajeros;
    private int ocupacionActual = 0;

    public Autobus(String matricula, int capacidad) {
        this.matricula = matricula;
        this.capacidad = capacidad;
        this.pasajeros = new Pasajero[capacidad];
    }

    public boolean subirPasajero(Pasajero pasajero) {
        if (ocupacionActual < capacidad) {
            pasajeros[ocupacionActual] = pasajero;
            ocupacionActual++;
            return true;
        }
        return false;
    }

    public boolean bajarPasajero(Pasajero pasajero) {
        for (int i = 0; i < ocupacionActual; i++) {
            if (pasajeros[i].equals(pasajero)) {
                pasajeros[i] = pasajeros[ocupacionActual - 1];
                pasajeros[ocupacionActual - 1] = null;
                ocupacionActual--;
                return true;
```

```
            }
        }
        return false;
    }
}
```

---

incrusta aquí tu respuesta

3. Salud

3a. Descripción del proyecto

En un sistema de gestión de pacientes de un hospital, queremos modelar la relación entre un `Paciente` y sus `Órganos`. Un paciente **siempre tiene** órganos, y estos **no pueden existir sin el paciente**.

- **Clase `Paciente`**:
  - **Atributos:**
    - `nombre`: nombre del paciente.
    - `dni`: documento de identidad del paciente.
    - `organos`: array de órganos del paciente (se crean al instanciar `Paciente`).
  - **Métodos:**
    - `listarOrganos()`: Muestra todos los órganos del paciente.
- **Clase `Organo`**:
  - **Atributos:**
    - `nombre`: nombre del órgano.
    - `funcion`: descripción de la función del órgano.
  - **Métodos:**
    - `obtenerDescripcion()`: Devuelve una cadena con el nombre y función del órgano.

3b. Código Java

```
public class Organo {
    private String nombre;
    private String funcion;

    public Organo(String nombre, String funcion) {
        this.nombre = nombre;
        this.funcion = funcion;
    }
```

```
    public String obtenerDescripcion() {
        return nombre + ": " + funcion;
    }
}

public class Paciente {
    private String nombre;
    private String dni;
    private Organo[] organos;

    public Paciente(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
        this.organos = new Organo[] {
            new Organo("Corazón", "Bombea sangre"),
            new Organo("Pulmones", "Permiten la respiración"),
            new Organo("Hígado", "Filtra toxinas"),
            new Organo("Riñones", "Regulan los líquidos")
        };
    }

    public void listarOrganos() {
        System.out.println("Órganos de " + nombre + ":");
        for (Organo organo : organos) {
            System.out.println(organo.obtenerDescripcion());
        }
    }
}
```

---

incrusta aquí tu respuesta

4. Gestión de archivos

4a. Descripción del proyecto

En un sistema de gestión de archivos, queremos modelar una jerarquía de clases donde `Fichero` es la clase base y sus subclases representan distintos tipos de archivos.

- **Clase `Fichero` (superclase)**:
  - **Atributos:**
    - `nombre`: nombre del archivo.
    - `tamaño`: tamaño en KB.
  - **Métodos:**
    - `abrir()`: Muestra un mensaje indicando que el archivo se ha abierto.
- **Subclases que heredan de `Fichero`**:
  - **Clase `Texto`**:

- **Atributo**: `codificacion`: formato de texto (UTF-8, ASCII, etc.).
- **Método**: `contarPalabras()`: Devuelve un número simulado de palabras en el archivo.

- **Clase** `Imagen`:
  - **Atributo**: `resolucion`: resolución en píxeles (ej. 1920x1080).
  - **Método**: `cambiarResolucion(nuevaResolucion)`: Cambia la resolución de la imagen.

- **Clase** `Audio`:
  - **Atributo**: `duracion`: duración del audio en segundos.
  - **Método**: `reproducir()`: Muestra un mensaje indicando que el audio se está reproduciendo.

4b. Código Java

```
public class Fichero {
    protected String nombre;
    protected int tamaño;

    public Fichero(String nombre, int tamaño) {
        this.nombre = nombre;
        this.tamaño = tamaño;
    }

    public void abrir() {
        System.out.println("Abriendo el archivo: " + nombre);
    }
}

public class Texto extends Fichero {
    private String codificacion;

    public Texto(String nombre, int tamaño, String codificacion) {
        super(nombre, tamaño);
        this.codificacion = codificacion;
    }

    public int contarPalabras() {
        return tamaño * 2; // Simulación: cada KB equivale a 2 palabras
    }
}

public class Imagen extends Fichero {
    private String resolucion;

    public Imagen(String nombre, int tamaño, String resolucion) {
        super(nombre, tamaño);
        this.resolucion = resolucion;
    }
```

```
    public void cambiarResolucion(String nuevaResolucion) {
        this.resolucion = nuevaResolucion;
        System.out.println("Resolución cambiada a: " + nuevaResolucion);
    }
}

public class Audio extends Fichero {
    private int duracion;

    public Audio(String nombre, int tamaño, int duracion) {
        super(nombre, tamaño);
        this.duracion = duracion;
    }

    public void reproducir() {
        System.out.println("Reproduciendo audio: " + nombre);
    }
}
```

---

incrusta aquí tu respuesta

5. Autenticación de usuarios

5a. Descripción del proyecto

En un sistema de autenticación, queremos modelar la relación entre una **interfaz** `Autenticable` y dos clases que la implementan: `Usuario` y `Administrador`.

- **Interfaz** `Autenticable`:
  - **Métodos**:
    - `login(usuario, contraseña)`: Devuelve `true` si las credenciales son correctas.
    - `logout()`: Cierra la sesión del usuario.
- **Clases que implementan** `Autenticable`:
  - **Clase** `Usuario`:
    - **Atributo**: `email`: correo electrónico del usuario.
    - **Método**: `login(usuario, contraseña)`: Devuelve `true` si la contraseña es válida.
  - **Clase** `Administrador`:
    - **Atributo**: `nivelAcceso`: nivel de permisos del administrador.
    - **Método**: `gestionarUsuarios()`: Muestra un mensaje indicando que el administrador gestiona usuarios.

5b. Código Java

```
public interface Autenticable {
    boolean login(String usuario, String contraseña);
    void logout();
}

public class Usuario implements Autenticable {
    private String email;

    public Usuario(String email) {
        this.email = email;
    }

    @Override
    public boolean login(String usuario, String contraseña) {
        return usuario.equals(email) && contraseña.equals("1234"); // Simulación
de autenticación
    }

    @Override
    public void logout() {
        System.out.println("Usuario " + email + " ha cerrado sesión.");
    }
}

public class Administrador implements Autenticable {
    private int nivelAcceso;

    public Administrador(int nivelAcceso) {
        this.nivelAcceso = nivelAcceso;
    }

    @Override
    public boolean login(String usuario, String contraseña) {
        return contraseña.equals("admin123"); // Simulación de autenticación
    }

    @Override
    public void logout() {
        System.out.println("Administrador con nivel " + nivelAcceso + " ha cerrado
sesión.");
    }

    public void gestionarUsuarios() {
        System.out.println("Gestionando usuarios...");
    }
}
```

---

incrusta aquí tu respuesta

6. Gestión de hoteles

6a. Descripción del proyecto

En un sistema de reservas de hoteles, queremos modelar la relación entre un `Hotel`, sus `Habitaciones`, las `Reservas` que gestiona y los `Clientes` que realizan dichas reservas.

- **Clase `Hotel`**:
  - **Atributos**:
    - `nombre`: nombre del hotel.
    - `direccion`: ubicación del hotel.
    - `habitaciones`: array de habitaciones disponibles en el hotel.
  - **Métodos**:
    - `obtenerDisponibilidad()`: Devuelve `true` si hay habitaciones disponibles.
- **Clase `Habitacion`**:
  - **Atributos**:
    - `numero`: número de la habitación.
    - `tipo`: tipo de habitación (individual, doble, suite).
  - **Métodos**:
    - `verPrecio()`: Devuelve el precio de la habitación según su tipo.
- **Clase `Cliente`**:
  - **Atributos**:
    - `nombre`: nombre del cliente.
    - `dni`: documento de identidad del cliente.
  - **Métodos**:
    - `hacerReserva(Hotel hotel, Habitacion habitacion)`: Realiza una reserva en el hotel.
- **Clase `Reserva`**:
  - **Atributos**:
    - `fechaInicio`: fecha de inicio de la reserva.
    - `fechaFin`: fecha de fin de la reserva.
    - `cliente`: cliente que ha realizado la reserva.
    - `habitacion`: habitación reservada.
  - **Métodos**:
    - `confirmarReserva()`: Muestra un mensaje confirmando la reserva.

6b. Código Java

```
public class Hotel {
    private String nombre;
    private String direccion;
    private Habitacion[] habitaciones;

    public Hotel(String nombre, String direccion, Habitacion[] habitaciones) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.habitaciones = habitaciones;
    }

    public boolean obtenerDisponibilidad() {
        return habitaciones.length > 0;
    }
}

public class Habitacion {
    private int numero;
    private String tipo;

    public Habitacion(int numero, String tipo) {
        this.numero = numero;
        this.tipo = tipo;
    }

    public double verPrecio() {
        switch (tipo) {
            case "individual": return 50.0;
            case "doble": return 80.0;
            case "suite": return 150.0;
            default: return 0.0;
        }
    }
}

public class Cliente {
    private String nombre;
    private String dni;

    public Cliente(String nombre, String dni) {
        this.nombre = nombre;
        this.dni = dni;
    }

    public Reserva hacerReserva(Hotel hotel, Habitacion habitacion, String
fechaInicio, String fechaFin) {
        return new Reserva(fechaInicio, fechaFin, this, habitacion);
    }
}
```

```
public class Reserva {
    private String fechaInicio;
    private String fechaFin;
    private Cliente cliente;
    private Habitacion habitacion;

    public Reserva(String fechaInicio, String fechaFin, Cliente cliente,
Habitacion habitacion) {
        this.fechaInicio = fechaInicio;
        this.fechaFin = fechaFin;
        this.cliente = cliente;
        this.habitacion = habitacion;
    }

    public void confirmarReserva() {
        System.out.println("Reserva confirmada para " + cliente.nombre + " en la
habitación " + habitacion.numero + " del " + fechaInicio + " al " + fechaFin);
    }
}
```

---

incrusta aquí tu respuesta

Rúbrica

Calificación	Descripción
0	No se entrega la tarea o se entrega sin sentido ni conexión con el enunciado
3	Se han dibujado correctamente menos de 5 diagramas de clases
6	Se han dibujado correctamente todos los diagramas, excepto el 5 o el 6
9	Se han dibujado correctamente todos los diagramas
10	Se han aportado elementos personales más allá de lo solicitado