

TRABAJO PRÁCTICO INTEGRADOR  
PROGRAMACIÓN II

**1er. Cuatrimestre 2025 – Primera Parte**

**COM 1**

Integrantes:



* Cisneros Analia
* Pucheta Tamara

Docentes:



* Ángel Juárez
* Lucía Soria

TADS

Posibles TAD’s:

─ Ticketek

─ Espectáculo

─ Entrada

─ Usuario

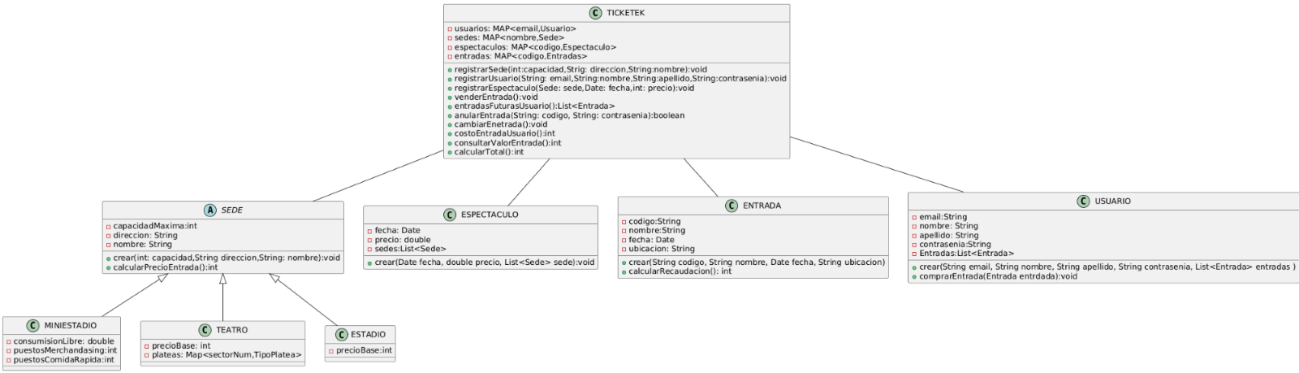
─ Función

─ Sede

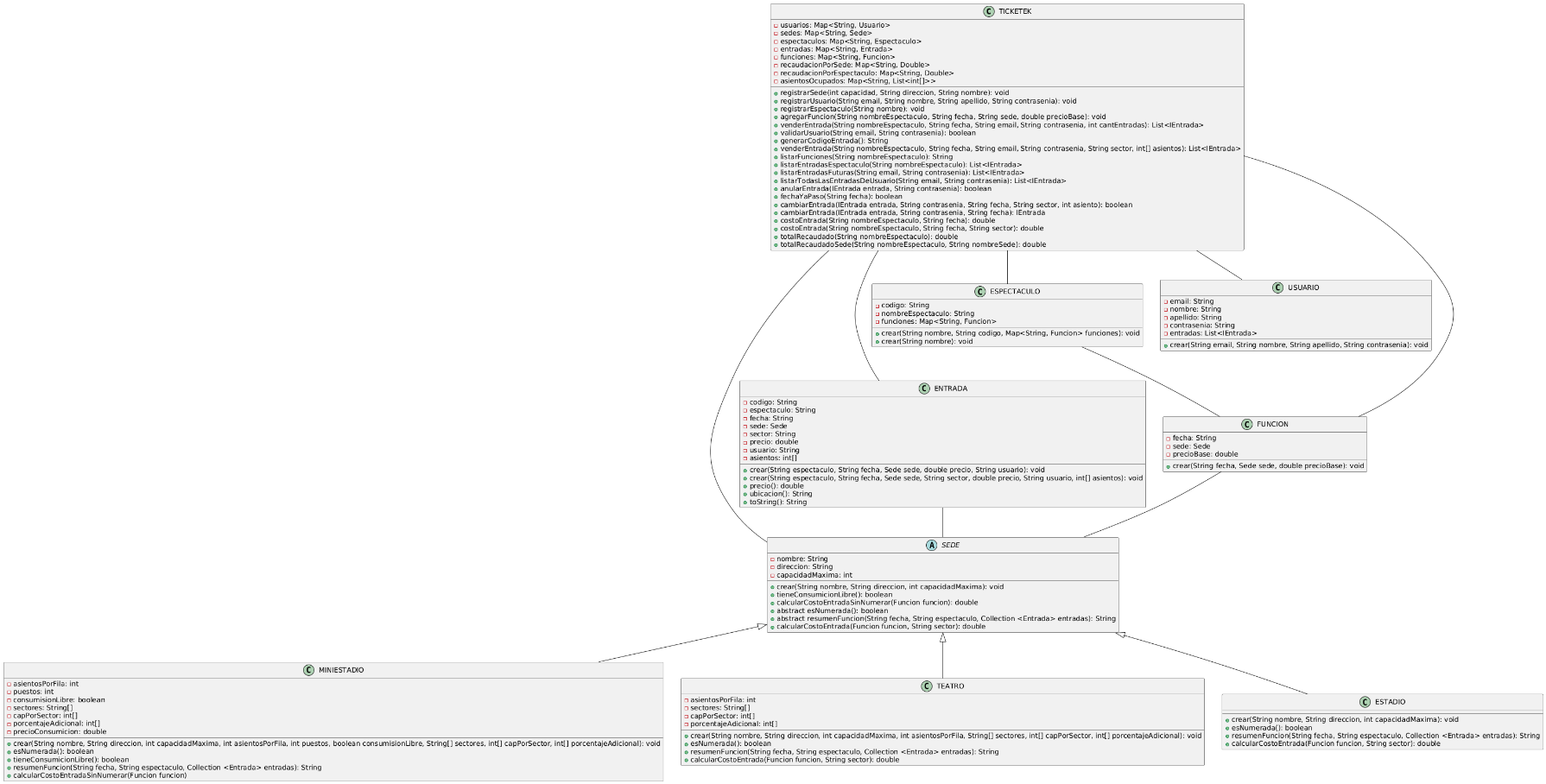
─ MiniEstadio

─ Teatro

─ Estadio

Diagrama de la primera entrega:

Cambios del diagrama de la primera entrega:



**TAD TICKETEK**

**Atributos:**

* Usuarios
* Sedes
* Entradas
* Espectaculos
* Funciones
* RecaudacionPorSede
* RecaudacionPorEspectaculo
* AsientosOcupados

**Operaciones:**

* registrarSede(capacidad, direccion, nombre)
* registrarUsuario(email, nombre, apellido, contrasenia)
* registrarEspectaculo(nombre)
* agregarFuncion(nombreEspectaculo, fecha, sede, precioBase)
* venderEntrada(nombreEspectaculo, fecha, email, contrasenia, cantEntradas)
* venderEntrada(nombreEspectaculo, fecha, email, contrasenia, sector, asientos)
* validarUsuario(email, contrasenia)
* generarCodigo()
* listarFunciones(nombreEspectaculo)
* listarEntradasEspectaculo(nombreEspectaculo)
* listarEntradasFuturas(nombreEspectaculo)
* listarTodasLasEntradasDeUsuario(email, contrasenia)
* anularEntrada(entrada, contrasenia)
* fechaYaPaso(fecha)
* cambiarEntrada(entrada, contrasenia, fecha, sector, asiento)
* costoEntrada (nombreEspectaculo, fecha)
* costoEntrada(nombreEspectaculo, fecha, sector)
* TotalRecaudado(nombreEspectaculo)
* totalRecaudadoSede(nombreEspectaculo, nombreSede)

**TAD ESPECTACULO**

**Atributos:**

* nombreEspectaculo
* codigo
* funciones

**Operaciones:**

* crear(nombre, codigo, funciones) //sobrecarga
* crear(nombre)

**TAD ENTRADA**

**Atributos:**

* codigo
* espectáculo
* fecha
* sede
* sector
* precio
* usuario
* asientos

**Operaciones:**

* crear(espectáculo, fecha, sede, precio, usuario)
* crear(espectáculo, fecha, sede, sector, precio, usuario, asientos) //sobrecarga
* precio() //sobreEscritura
* ubicación()//sobreEscritura
* toString()//sobreEscritura

**TAD USUARIO**

**Atributos:**

* email
* nombre
* apellido
* contrasenia
* entradas

**Operaciones:**

* crear(email, nombre, apellido, contrasenia)

**TAD FUNCIÓN**

**Atributos:**

* fecha
* sede
* precioBase

**Operaciones:**

* crear(fecha, sede, precioBase)

**TAD SEDE**

**Atributos:**

* nombre
* direccion
* capacidadMaxima

**Operaciones:**

* crear(nombre, direccion, capacidadMaxima)
* calcularCostoEntradaSinNumerar(Funcion funcion)
* tieneConsumicionLibre()

**TAD MINIESTADIO**

**Atributos:**

* asientosPorFila
* puestos
* consumisionLibre
* sectores
* capPorSector
* porcentajeAdicional
* precioConsumicion

**Operaciones:**

* crear(nombre, direccion, capacidadMaxima, asientosPorFila, puestos, consumisionLibre, sectores, capPorSector, porcentajeAdicional)
* esNumerada()
* tieneConsumicionLibre()

**TAD TEATRO**

**Atributos:**

* asientosPorFila
* sectores
* capPorSector
* porcentajeAdicional

**Operaciones:**

* crear(nombre, direccion, capacidadMaxima, asientosPorFila, sectores, capPorSector, porcentajeAdicional)
* esNumerada()

**TAD ESTADIO**

**Operaciones:**

* crear()
* esNumerada()

INTERFAZ, IMPLEMENTACIÓN

E IREP

* **TAD TICKETEK**

Es la **clase central del programa**, que contiene HashMaps para los Usuarios, Sedes, Entradas y Espectáculos. Los atributos y operaciones que contienen son:

**Atributos:**

* **HashMap** <String, Usuario> Usuarios; // Email, Usuario
* **HashMap** <String, Sede> Sedes; // Nombre, Sede
* **HashMap** <String, Entrada> Entradas; //Código, Entrada
* **HashMap** <String, Espectaculo> Espectaculos; // Nombre, Espectaculo
* **HashMap** <String, Funcion> Funciones; // "Espectaculo-Fecha", Funcion
* **Private HashMap** <String, Double> RecaudacionPorSede; // "Espectaculo-Sede", Recaudación total
* **Private HashMap** <String, Double> RecaudacionPorEspectaculo; // "Espectaculo", Recaudación total por espectáculo
* **Private HashMap** <String, List<Integer>> AsientosOcupados;

**Operaciones:**

* + ***registrarSede(****int capacidad, Strig direccion, String nombre****):void***
  + ***registrarUsuario(****String email, String nombre, String apellido, String contrasenia****):void***
  + ***registrarEspectaculo(****String nombre****):void***
  + ***agregarFuncion(****String nombreEspectactulo, String fecha, String sede, double precioBase****): void***
  + ***venderEntrada(****String nombreEspectaculo, String fecha, String email, String contrasenia, int cantEntradas****): List<IEntrada>***
  + ***venderEntrada(****String nombreEspectaculo, String fecha, String email, String contrasenia, String sector, int[] asientos****): List<IEntrada>***
  + ***private validarUsuario(****String email, String contrasenia****): boolean***
  + ***private generarCodigoEntrada****():* ***String***
  + ***listarFunciones(****String nombreEspectaculo****):String***
  + ***listarEntradasEspectaculo(****String nombreEspectaculo****):List<IEntrada>***
  + ***listarEntradasFuturas(****String email, String contrasenia****):List<IEntrada>***
  + ***listarTodasLasEntradasDelUsuario(****String email, String contrasenia****): List<IEntrada>***
  + ***anularEntrada(****IEntrada: entrada, String: contrasenia****):boolean***
  + ***private fechaYaPaso(****String fecha****): boolean***
  + ***cambiarEntrada(****IEntrada entrada, String contrasenia, String fecha, String sector, int asiento****): IEntrada***
  + ***cambiarEntrada****(IEntrada entrada, String contrasenia, String fecha****): IEntrada***
  + ***costoEntrada****(String nombreEspectaculo, String fecha****):double***
  + ***costoEntrada(****String nombreEspectaculo, String fecha, String sector****): double***
  + ***TotalRecaudado****(String: nombreEspectaculo****):int***
  + ***totalRecaudadoSede(****String: nombreEspectaculo, String: nombreSede****): double***
* **IREP de TICKETEK:**
  + **ATRIBUTOS:**
    - ***Usuarios*** HashMap no vacio, el objeto Usuario debe existir
    - ***Sedes*** HashMap no vacio, el objeto Sede debe existir
    - ***Entradas*** HashMap no vacio, el objeto Entrada debe existir
    - ***Espectaculos*** HashMap no vacio, el objeto Espectaculo debe existir
    - ***Funciones***: HashMap no vacio, cada clave sigue el formato "Espectaculo-Fecha", y el objeto Funcion debe estar asociado a un espectáculo existente
    - ***RecaudacionPorSede:*** HashMap no vacio, cada clave sigue el formato "Espectaculo-Sede", el valor debe ser un double >= 0, y el espectáculo y la sede deben existir
    - ***RecaudacionPorEspectaculo:*** HashMap no vacio,cada clave representa un Espectaculo registrado, el valor debe ser un double >= 0
    - ***AsientosOcupados***: HashMap no vacio, cada clave representa una función ("Espectaculo-Fecha"), y la lista de asientos ocupados debe contener enteros únicos dentro del rango permitido por la sede y el sector correspondiente
* **TAD ESPECTACULO**

**Atributos:**

* String nombreEspectaculo;
* String codigo;
* List<Funcion> funciones;

**Operaciones:**

* crear(String nombre, String codigo, HashMap<String, Funcion> funciones): void
* crear(String nombre); void
* **IREP de ESPECTACULO:**
  + **ATRIBUTOS:**
    - ***nombreEspectaculo***: debe ser único y no debe estar vacío
    - ***codigo:*** debe ser un codigo existente
    - ***funciones:*** no debe estar vacío
* **TAD ENTRADA**

**Atributos:**

* String codigo;
* String espectaculo;
* String fecha;
* Sede sede;
* String sector;
* double precio;
* String usuario;
* int[] asientos;

**Operaciones:**

* crear(String sede, String espectaculo, String codigo, String fecha, String ubicacion, double precio)
* crear(String espectaculo, String fecha, Sede sede, String sector, double precio, String usuario, int[] asientos);
* double precio();
* String ubicacion();
* String toString();
* **IREP de ENTRADA:**
  + **ATRIBUTOS:**
    - ***sede:*** debe ser una Sede existente, no debe estar vacío & !=null
    - ***espectaculo:*** debe ser un espectáculo existente, no debe estar vacío
    - ***código:*** debe ser único y no debe estar vacío
    - ***fecha:*** debe ser una fecha existente, no debe tener dos fechas
    - ***usuario:*** el usuario debe existir
    - ***sector:*** debe ser un sector existente
    - ***asientos:*** no debe estar vacio
    - ***precio: > 0***
* **TAD USUARIO**

**Atributos:**

* String email;
* String nombre;
* String apellido;
* String contrasenia;
* List<Entrada> entradas;

**Operación:**

* crear(String email, String nombre, String apellido, String contrasenia): void
* **IREP de USUARIO:**
  + **ATRIBUTOS:**
    - ***email:*** debe ser único, no debe estar vacío
    - ***nombre:*** no debe estar vacío
    - ***apellido:*** no debe estar vacío
    - ***contrasenia:*** no debe estar vacío
    - ***entradas:*** no debe estar vacío & !=null
* **TAD FUNCIÓN**

**Atributos:**

* String fecha;
* String sede;
* double precioBase;

**Operación:**

* crear(String fecha, String sede, double precioBase): void
* **IREP de FUNCIÓN:**
  + **ATRIBUTOS:**
    - ***fecha:*** una fecha existente, no puede estar vacio
    - ***sede:*** una sede existente, no puede estar vacio
    - ***precioBase:*** > 0
* **TAD SEDE**

**Atributos:**

* + - String nombre;
    - String direccion;
    - int capacidadMaxima;

**Operación:**

* crear(String nombre, String direccion, int capacidadMaxima): void
* tieneConsumicionLibre(); boolean
* abstract esNumerada(): boolean
* abstract calcularCostoEntrada(): double
* abstract resumenFuncion(): String
* **IREP de SEDE:**
  + **ATRIBUTOS:**
    - ***nombre***: debe ser único, no puede estar vacio
    - ***dirección:*** no puede estar vacio
    - ***capacidadMaxima:*** > 0
* **TAD MINIESTADIO :**

**Atributos:**

* + - int asientosPorFila;
    - int puestos;
    - String[] sectores;
    - Int[] capPorSector;
    - Int[] porcentajeAdicional;
    - double precioConsumicion;

**Operaciones:**

* crear(String nombre, String direccion, int capacidadMaxima, int asientosPorFila,int puestos, double consumision, String[] sectores, int[] capPorSector, int[] porcentajeAdicional)
* esNumerada(): boolean
* tieneConsumisionLibre(): boolean
* calcularCostoEntradaSinNumerar(): double
* resumenFuncion(): String
* calcularCostoEntrada(): double
* **IREP de MINIESTADIO:**
  + **ATRIBUTOS:**
    - ***asientosPorFila:*** >0
    - ***puestos:*** > 0
    - ***precioConsumision***: >0
    - ***sectores:*** no debe estar vacio
    - ***capPorSector:*** no debe estar vacio
    - ***porcentajeAdicional:*** no debe estar vacio
* **TAD TEATRO**

**Atributos:**

* + - int asientosPorFila;
    - String[] sectores;
    - int[] capPorSector;
    - int[] porcentajeAdicional;

**Operaciones:**

* + - crear(String nombre, String direccion, int capacidadMaxima, int asientosPorFila, String[] sectores, int[] capPorSector, int[] porcentajeAdicional): void
    - esNumerada(): boolean
    - resumenFuncion(): String
    - calcularCostoEntrada(): double
* **IREP de TEATRO:**
  + **ATRIBUTOS:**
    - ***asientosPorFila:*** >0
    - ***sectores:*** no debe estar vacio
    - ***capPorSector:*** no debe estar vacio
    - ***porcentajeAdicional:*** no debe estar vacio
* **TAD ESTADIO**

**Operación:**

* + - crear(String nombre, String direccion, int capacidadMaxima): void
    - esNumerada(): boolean
    - resumenFuncion(): String
    - calcularCostoEntrada(): double

***USO DE CONCEPTOS***

**Herencia:**

El concepto de ***herencia*** fue utilizado en las clases **Espectaculo**, **Entrada**, **Usuario**, **Función** y **Sede**. Estas **heredan** las características de la clase ***Ticketek***. Así mismo, ***Sede*** también utiliza el concepto de **herencia** en las clases de **Estadio**, **Miniestadio** y **Teatro**.

**Clase abstracta:**

El concepto de ***clases abstractas*** se utiliza en la clase **Sede**, ya que todos sus datos y todas sus operaciones se encuentran vacíos. Esto se debe a que es una clase padre, y en el caso de un buen funcionamiento nunca deberían ejecutarse.

**Polimorfismo:**

El concepto de ***polimorfismo*** se aplica en dos casos. Por un lado, se usa mediante la interfaz **IEntrada**, que permite que métodos como ***venderEntrada(), listarEntradasFuturas(), cambiarEntrada(),*** entre otros, trabajen con cualquier tipo de entrada sin importar su implementación concreta, siempre que implemente dicha interfaz. Por otro lado, se aplica en las clases **Teatro**, **MiniEstadio** y **Estadio**, ya que todas extienden a **Sede** y redefinen métodos como ***esNumerada() o tieneConsumicionLibre()*** con comportamientos distintos según la subclase.

**Sobrecarga:**

El concepto de ***sobrecarga*** se utiliza en el constructor de **Entrada**, donde se encuentran dos operaciones con el mismo nombre, pero con distinta cantidad y tipos de parámetros.

**Encapsulamiento:**

El concepto de ***encapsulamiento*** se aplica en las clases al tener los atributos declarados como private, utilizando getters y setters para acceder o modificar su valor. Esto permite proteger los datos y controlar de forma segura el acceso a ellos.

**Interfaz:**  
 El concepto de ***interfaz*** se aplica mediante la interfaz IEntrada, que define los métodos que debe tener cualquier entrada del sistema. Gracias a esto se pueden escribir métodos como venderEntrada o cambiarEntrada que funcionan sin importar la clase exacta de la entrada, siempre que implemente dicha interfaz.

***USO DE TECNOLOGÍAS***

**StringBuilder:**

Usado en listarFunciones(...) para construir cadenas de manera eficiente al listar múltiples funciones.

**Foreach:**

Utilizado en múltiples métodos para recorrer listas o mapas como usuarios, entradas o funciones.

**Iterator:**

En el método cambiarEntrada(...), se utiliza un Iterator para eliminar de forma segura una entrada de la lista de entradas del usuario. Esto permite modificar la colección mientras se la recorre, permitiendo un mayor control sobre qué elementos se eliminan.

***COMPLEJIDAD DE ANULARENTRADA***

public boolean anularEntrada(IEntrada entrada, String contrasenia) {

if (entrada == null) { *// O(1)*

throw new RuntimeException("La entrada no puede ser null");

}

Entrada ticket = (Entrada) entrada; // O(1)

if (!entradas.containsKey(ticket.codigo)) { // O(1

throw new RuntimeException("La entrada no existe");

}

Usuario user = usuarios.get(ticket.usuario); // O(1)

if (user == null || !user.contrasenia.equals(contrasenia)) {// O(1)

throw new RuntimeException("Usuario inválido o contraseña incorrecta");

}

if (fechaYaPaso(ticket.fecha)) { // O(1

return false;

}

entradas.remove(ticket.codigo); // O(1)

int[] asientos = ticket.asientos; // O(1)

if (asientos != null) { // O(1)

String clave = ticket.espectaculo + "-" +

ticket.fecha + "-" + ticket.sector; // O(1)

Set<Integer> ocupados = asientosOcupados.get(clave); // O(1)

if (ocupados != null) { // O(1)

for (int asiento : asientos) { // O(k)

ocupados.remove(asiento); // O(1)

}

}

}

return true; // O(1) – retorno

}