Universidad Nacional de General Sarmiento

Programación II - com3

Trabajo Práctico Integrador

Ticketek

Parte 2

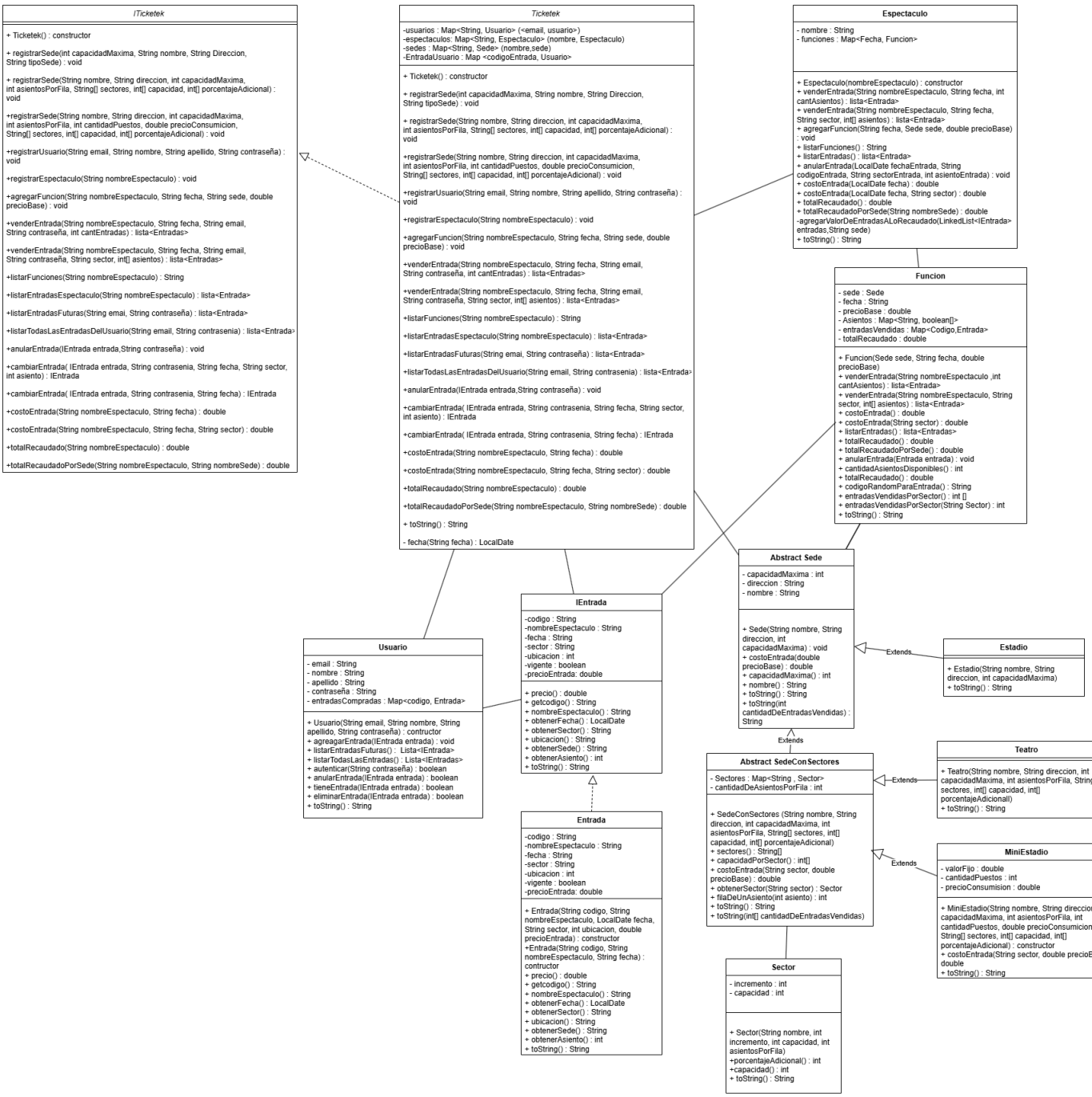
Docentes

- Nores, José
- Gabrielli, Miguel Angel

Integrantes

- Aranda, Rodrigo
- Arias Aguayo, Diego

2025C1



<u>Cómo se usaron los conceptos que vimos:</u>

En el trabajo práctico se usó el concepto de Herencia para la clase MiniEstadio y Teatro, que heredan de la clase SedeConSectores; y para la clase Estadio, que hereda de la clase Sede.

También se usó el concepto de polimorfismo, en la función costoEntrada(), tanto en la clase MiniEstadio como en la clase SedeConSectores.

Además se usó el concepto de sobre escritura, esto se usó en todas las funciones de la clase Entrada y en todas las funciones de la clase Ticketek.

Se usó el concepto de sobrecarga en la función de venderEntrada() en la clase Espectáculo. Dependiendo de si se le pasan los parámetros de una función con sectores o no, este devolverá un resultado diferente.

Se hicieron a la clase Sede y SedeConSectores abstractas porque no notamos que había que instanciarlas.

StringBuilder fue utilizada para los toString, donde era necesario crear un String con diferentes datos que se iban agregando. Como por ejemplo en Función, SedeConSectores y en ToString de Ticketek.

Iteradores y Foreach fueron usados para recorrer los diferentes valores de los Maps. Como en el método entradasVendidasPorSector en la clase Función donde se recorren las claves del map Asientos con un iterador.
El Foreach fue utilizado para lo mismo, como en el método listarEntradas de la clase Función para recorrer las diferentes entradas vendidas.

<u>Irep de la representación:</u>

Sector:

- nombre != null && nombre.length > 2
- incremento >= 0.0
- capacidad >= 0

Usuario:

- email != null && email.length >3
- nombre != null && nombre.length > 2
- apellido != null && apellido.length > 2
- contraseña != null && contraseña.length >= 3
- El email tiene que contener “@” y “.”
- Cada código de la entrada es único por usuario

Entrada:

- código != null && !codigo.isEmpty()
- nombreEspectaculo != null && !nombreEspectaculo.isEmpty()
- fecha != null
- precioEntrada > 0.0
- nombreSede != null && !nombreSede.isEmpty()
- sector != null && !sector.isEmpty()
- asiento >= 0
- fila >= 0
- precioEntrada >= 1

Sede:

- nombre != null && nombre.length > 2
- dirección != null
- capacidadMáxima > 0

Estadio:

- nombre != null && nombre.length > 2
- dirección != null
- capacidadMáxima > 0
- sector == campo
- No tiene asientos numerados

Teatro:

- nombre != null && nombre.length > 2
- dirección != null

- capacidadMáxima > 0
- sectores != null
- capacidad != null
- porcentajeAdicional != null
- asientosPorFila > 0

MiniEstadio:

- nombre != null && nombre.length > 2
- direccion != null
- capacidadMáxima > 0
- sectores != null
- capacidad != null
- porcentajeAdicional != null
- precioConsumicion >= 0.0
- capacidad <= capacidadMáxima
- cantidadPuestos >= 0

Función:

- sede != null
- fecha != null
- precioBase > 0.0

Espectáculo:

- nombre != null && nombre.length >= 2
- funciones != null
- No hay funciones duplicadas, la fecha y sede es única.

- RecaudadoPorSede >= 0

Ticketek:

- usuarios != null
- espectaculos != null
- sedes != null
- usuariosDeEntrada != null
- Cada email es único por usuario
- Cada nombre es único por espectáculo
- Cada nombre es único por sede

SedeConSectores:

- nombre != null && nombre.length > 2
- direccion != null
- capacidadMáxima > 0
- sectores != null
- capacidad > 0
- porcentajeAdicional != null
- capacidad <= capacidadMáxima

ANÁLISIS DE COMPLEJIDAD DEL PUNTO 8

Ticketek:

```
public boolean anularEntrada(IEntrada entrada, String contrasenia) {
            if(entrada == null) {                                                   O(1)
                throw new RuntimeException("Error: La entrada no puede ser nula");   O(1)
            }
            if(contrasenia == null) {                                                       O(1)
                throw new RuntimeException("Error: La contraseña no puede estar vacía");  O(1)
            }

            Usuario usuario = usuariosDeEntrada.get(entrada.getCodigo());       O(1)
            if(usuario == null)                                                 O(1)
                throw new RuntimeException("Error: No se encontró un usuario asociado a esta entrada"); O(1)

            if(!usuario.autenticar(contrasenia))                                        O(1)
                                throw new RuntimeException("Error: La contraseña es incorrecta."); O(1)


            boolean anulacionExitosa = usuario.anularEntrada(entrada);                      O(1)

            if(anulacionExitosa) {                                                          O(1)
                usuariosDeEntrada.remove(entrada.getCodigo());                  O(1)
                Espectaculo espectaculo = espectaculos.get(entrada.nombreEspectaculo());            O(1)
                if(espectaculo != null) {                                               O(1)
                    espectaculo.anularEntrada(entrada.obtenerFecha(), entrada.getCodigo(), entrada.obtenerSector(),
entrada.obtenerAsiento());                                  O(1)
                }
            }

            return anulacionExitosa;                                            O(1)
        }
```

O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) = O(1)
(Sin contar las excepciones porque no son el peor caso).
AnularEntrada es de Orden constante en el peor caso.

Clase Usuario:

```
        public boolean tieneEntrada(IEntrada entrada) {  O(1) + O(1) = O(1)
                if(entrada == null) { O(1)
                        return false;   O(1)
                }
                return entradasCompradas.containsKey(entrada.getCodigo()); O(1)
        }
        public boolean eliminarEntrada(IEntrada entrada) { O(1) + O(1) + O(1) = O(1)
                if(entrada == null || !tieneEntrada(entrada)) {       O(1) + O(1) = O(1)
                        return false;                                           O(1)
                }
                entradasCompradas.remove(entrada.getCodigo());          O(1)
                return true;                                            O(1)
        }
        public boolean anularEntrada(IEntrada entrada) {            O(1)
                return eliminarEntrada(entrada);                                    O(1)
        }
```

Clase Espectáculo:

```
public void anularEntrada(LocalDate fechaEntrada, String codigoEntrada, String sectorEntrada, int
asientoEntrada) {
                    if(!funciones.containsKey(fechaEntrada)){                    O(1)
                              throw new RuntimeException("Error: La fecha de la entrada no concuerda
con las fechas registradas del espectaculo.");                                   O(1)
                    }
                    Funcion funcion = funciones.get(fechaEntrada);               O(1)
                    funcion.anularEntrada(codigoEntrada, sectorEntrada, asientoEntrada);   O(1)
          }
Tiene orden O(1) +  O(1) + O(1) = O(1)
```

Clase Función:

```
public void anularEntrada(String codigoEntrada, String sectorEntrada, int asientoEntrada) {
                    if(entradasVendidas.containsKey(codigoEntrada)) {                    O(1)
                              if(sede instanceof SedeConSectores) {                      O(1)
                                        entradasVendidas.remove(codigoEntrada);          O(1)
                                        boolean[] asientosDelSector = asientos.get(sectorEntrada);O(1)
                                        asientosDelSector[asientoEntrada] = true;        O(1)
                              } else {
                                        entradasVendidas.remove(codigoEntrada);          O(1)
                              }

                    } else {
                              throw new RuntimeException("Error: La entrada no esta registrada en la
Funcion.");                                                                              O(1)
                    }
          }
Es de orden O(1) + O(1) + O(1) + O(1) + O(1) = O(1) en el peor caso.
```