Universidad Nacional de General Sarmiento

Programación II - com3

Trabajo Práctico Integrador

Ticketek

Parte 2

Docentes

- Nores, José
- Gabrielli, Miguel Angel

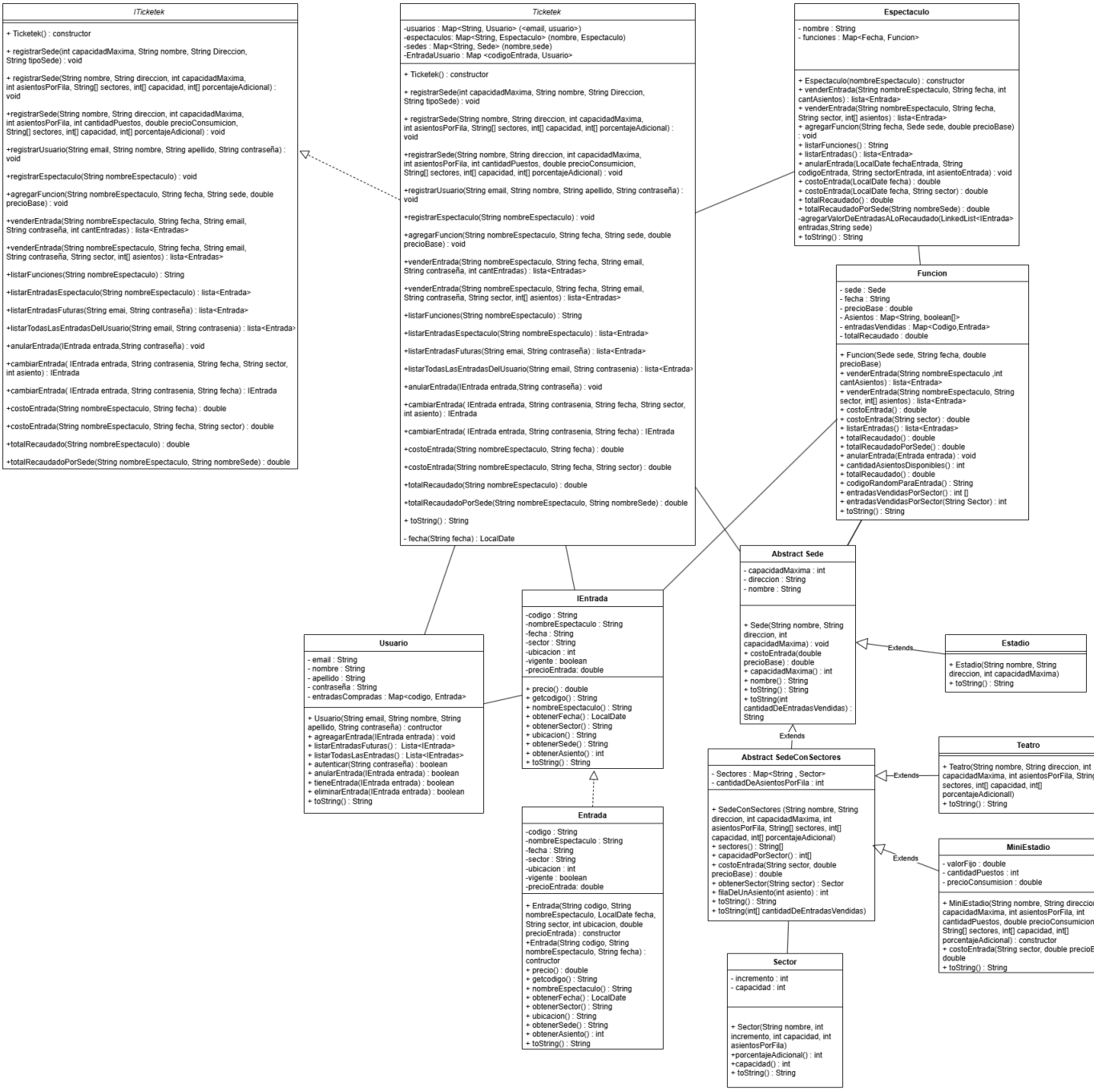
Integrantes

- Aranda, Rodrigo
- Arias Aguayo, Diego

2025C1

Índice

Diagrama de Clases:



Cómo se usaron los conceptos que vimos:

- **Herencia**
  - En el trabajo práctico se usó el concepto de Herencia para la clase Estadio y la clase SedeConSectores, ya que comparten datos como: CapacidadMaxima, direccion y nombre. Luego, además, la clase MiniEstadio y Teatro heredan de la clase SedeConSectores, ya que comparten: cantidadDeAsientosPorFila y Sectores.

- **Polimorfismo**
  - Se usó el concepto de polimorfismo, en la función costoEntrada(), tanto en la clase MiniEstadio como en la clase SedeConSectores.

- **Sobrescritura**
  - El concepto de sobre escritura se usó en todas las funciones de la clase Entrada y en las de la clase Ticketek, ya que estas implementan las interfaces IEntrada y ITicketek respectivamente.

- **Sobrecarga**
  - 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 debido a los diferentes tipos de Sedes.

- **Abstracción**
  - Las clases Sede y SedeConSectores se declararon abstractas porque representan conceptos genéricos que no deben instanciarse directamente. Por ejemplo, no tiene sentido crear un objeto Sede, ya que una sede debe ser específica (un teatro, un estadio o miniEstadio).

- **StringBuilder**
  - 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**
  - 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.

Irep de las clases:

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

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.
```

- if(entradasVendidas.containsKey(codigoEntrada) es O(1) porque se trata de un Map.
- entradasVendidas.remove(codigoEntrada) es O(1) porque él remover un elemento de un Map es de complejidad O(1).
- asientos.get(sectorEntrada) es O(1) porque se asientos es un Map.
- asientosDelSector[asientoEntrada] = true es O(1) porque se solo se cambia el valor que tenía el boolean de la posición “asientoEntrada”.
- Los throw new RuntimeException son de O(1).
- El peor caso seria aquel en el que sede sea instancia de SedeConSectores, por lo que seria de orden: O(5) = O(1)
- Por lo tanto es la funcion anularEntrada de de Orden(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)
```

Aclaraciones:

- funciones.containsKey(fechaEntrada) y funciones.get(fechaEntrada) es de O(1) ya que funciones es un Map.
- funcion.anularEntrada(codigoEntrada, sectorEntrada, asientoEntrada) es de O(1) segun el conteo de complejidad anterior.
- Por lo tanto anularEntrada es O(1).

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)
}
```

- entradasCompradas.containsKey(entrada.getCodigo()) es O(1) porque entradasCompradas es un Map.
- En suma, tieneEntrada(IEntrada entrada) es de O(1).

- entradasCompradas.remove(entrada.getCodigo()) como entradasCompradas es un Map entonces remover un elemento es O(1)
- En conclusion, eliminarEntrada es O(1), que es es: O(1) + O(1) + O(1) = O(1)
- Y como anularEntrada solo llama y devuelve eliminarEntrada, entonces, anularEntrada tambien es de O(1).

```
//Verifica si la contraseña ingresada coincide con la del usuario.
public boolean autenticar(String contrasenia) {
        if(contrasenia == null) { O(1)
                return false;
        }
        return this.contraseña.equals(contrasenia); O(1)
}
```

- equals es un método de orden contraste, por lo que autentificar es de O(1)

Ticketek:

```
private Usuario autentificarUsuarioDeEntrada(Entrada entrada, String contrasenia) {
  if(contrasenia == null) {  O(1)
     throw new RuntimeException("Error: La contraseña no puede estar vacía");
   }

   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");
   }

   if(!usuario.autenticar(contrasenia)) { O(1)
     throw new RuntimeException("Error: Contraseña incorrecta");
   }
   return usuario; O(1)
}
```

- UsuariosDeEntrada es un Map por lo que usuariosDeEntrada.get(...) es de Orden 1. Ademas, como autentificar es de O(1), se concluye que autentificarUsuarioDeEntrada(...) es de O(1).

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

    Usuario usuario = autentificarUsuarioDeEntrada(entradaObjeto, contrasenia); O(1)

     boolean anulacionExitosa = usuario.anularEntrada(entradaObjeto); 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)
(Sin contar las excepciones porque no son el peor caso).
AnularEntrada es de Orden constante en el peor caso. Aquel donde se logre la anulación.

- usuariosDeEntrada.get(entrada.getCodigo()) y usuariosDeEntrada.remove(...) son de O(1) ya que usuariosDeEntrada es un Map, y obtener un valor de un Map o eliminarlo es de O(1).
- Como espectaculos es tambien un Map, espectaculos.get(..) es de O(1).
- Como vimos anteriormente, autentificarUsuarioDeEntrada(...) es de orden 1.
- También vimos que usuario.anularEntrada(entradaObjeto) es de O(1).
- Finalmente espectaculo.anularEntrada(...) ya vimos que es de O(1).
- Concluimos que como el método anularEntrada(...) se compone siempre de la suma de 10 O(1). Por lo que el método es de O(1).