Trabajo Práctico 1

Nombre: Ramiro Garies

Padrón: 108092

Email: rgareis@fi.uba.ar

DNI: 43447079

1. Supuestos

Los supuestos que tome en cuenta para realizar la solución del problema fueron entre otros:

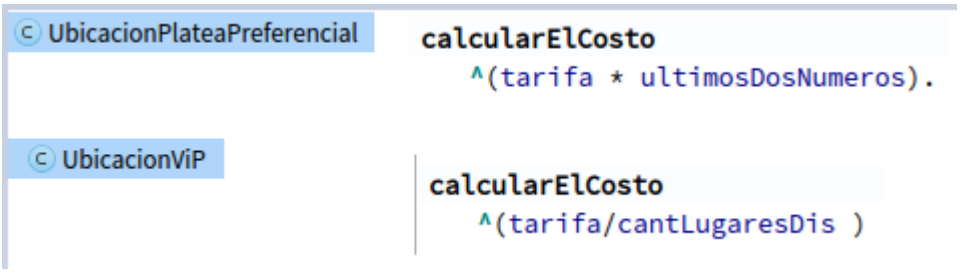
- El programa está enfocado en un solo un Evento, el cual solo tiene un Artista. Nunca exploró la posibilidad de que hubiera 2 artistas en un mismo Evento.
- No pueden haber dos ubicaciones registradas con el mismo nombre
- La clase efectivo y tarjeta solo devuelven el descuento que les toca, no toman registro de nada más
- Los lugares disponibles para vip son ilimitados, es decir puedo poner 50000 lugares disponibles y no va haber ningún problema
- No se puede registrar una ubicacionVip con 0 lugares disponibles, porque no tiene sentido registrar una ubicación la cual no se puede sacar una entrada
- La tarifas de las ubicaciones nunca pueden ser 0, es decir que no puede haber entradas gratis
- En platea preferencial solo se puede ingresar ubicaciones con 4 dígitos, y si o si los últimos 2 deben ser números, además no puede existir una ubicación que sus ultimos 2 digitos sean 00

2. Detalles de implementación

Un pilar que vi extremadamente necesarios usar para el desarrollo de mi programa fue la herencia, ya que en mi trabajo tengo 3 clases madres, como lo son ***Ubicacion***, ***MediosDePago*** y ***Artistas***, es lo pense asi debido a que

varias clases compartían el término “es un”, además de compartir varios métodos, por dar un ejemplo. *Ubicaciones* tiene tres clases hijas **UbicacionVip**, **UbicacionCampo** y **UbicacionPlateaPreferencial**, y yo debía saber de alguna manera el precio que podían llegar a tener cada una de estas clases, pero la manera de calcular los precios esa distinto en cada una, entonces la solucion que se me ocurrio para esto era usa Polimorfismo, y la manera más fácil de poder aplicarlo es con la herencia, y como las 3 clases cumplian el “es un”, en este caso las 3 eran “Ubicaciones”, cree una clase abstracta Ubicaciones la cual le cree 2 métodos abstractos, “calcularElCosto” y “sacarEntrada”. Debido a que son métodos abstractos, estos métodos en las clases hijas pueden actuar de manera diferente en cada una.

Aca un ejemplo con código del uso de polimorfismo gracias a la herencia:



Y para que me sirvió el polimorfismo, me sirvió para evitar usar el uso de condicionales, ya que no tengo que preguntar que ubicicaicones es y tener que llamar a su metodo el cual haga su respectiva cuenta, solo llamo el método y este me devuelve la cuenta, lo que hace también que haya encapsulamiento, por lo tanto hace que el código esté más protegido de agentes externo y además de la prolijidad, ya que solo se ve el método, nunca se ve el como se hico la cuenta(hablando del caso de calcular el precio de una ubicación)

También utilice mucha la delegación, por ejemplo **AlgoTek**, todos los métodos son delegados hacia la clase **Evento**, el cual este se encarga de devolverle todos los datos que se le pidieron. Voy a dejar otro ejemplo más explícito, en **sacarEntrada** en Evento llegó a delegar cosas a 3 clases distintas, el precio lo obtengo de sacarEntrada() de la clase Ubicaciones, el impuesto del artista, se lo delegó a la clase Artista el cual tiene un método calcularImpuesto(), y el descuento se encarga MedioDePago, que tiene un

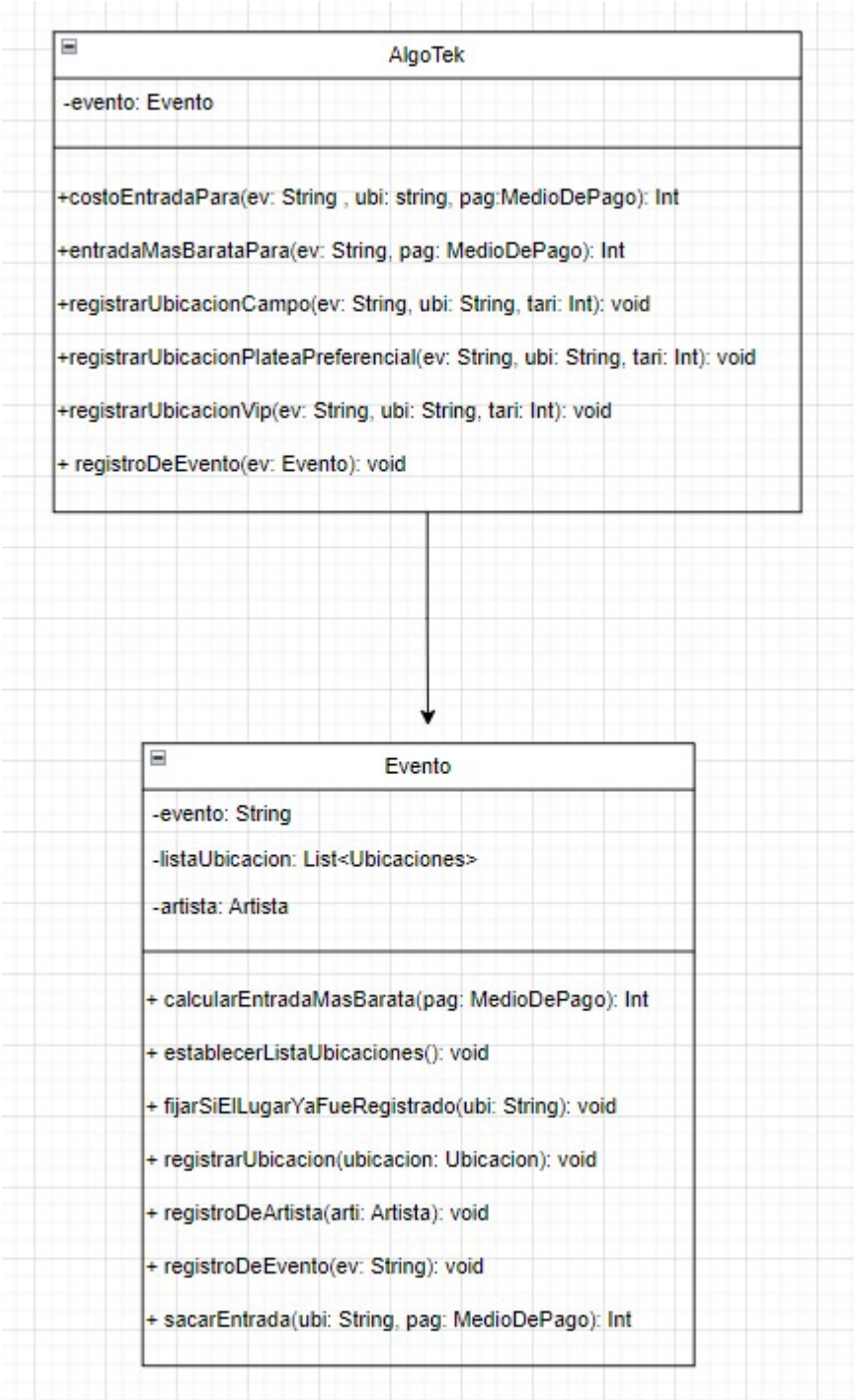
método aplicarDescuento: Int que lo que hace es recibir un número y hacer la cuenta para su debido descuento.
Tanto el uso de polimorfismo y delegación ayudan a la mejor lectura del programa y el evitar tener código repetido

3. Excepciones:

- *NoSePuedeIngresarDosCerosEnLaUbicacionDePlateaPreferencial* esta es necesaria, ya que en “platea preferencial” el precio se calcula atravez de los ultimos 2 digitos de la ubicion, y si se ponen dos 0, el precio termina quedando en 0, lo cual no tiene mucho sentido que digamos
- *NoSePuedeIngresarUnaUbicacionLaCualNoTengaCuatroCaracteres* Yo puse como un supuesto que no tenga mas o menos de 4 caracteres la ubicacion, porque yo veo bien que se estandarice que la ubicacion de “plateapreferencial” solo pueda tener 4 caracteres.
- *NoSePuedeIngresarUnaUbicacionLaCualSusUltimosDosCaracteresNoSeanNumeros* Como ya lo dije antes, el precio depende de los ultimos dos numeros de la ubicaicon, y si hay un caracter que no sea un numero decimal la cuenta del precio nunca se va poder ejecutar de manera correcta
- *NoSePuedeIngresarUnStringVacio* No tiene sentido el registrar algo con un valor vacío
- *NoSePuedeIngresarValorNegativo*
- *NoSePuedeIngresarValorQueDeComoResultadoNegativoOCero* Un ejemplo que salta esta exception es cuando se pone en la tarifa de una ubicacion cero
- *NoSePuedenRegistrarDosUbiacionesConElMismoNombre* Cuando registrar 2 ubicaciones con el mismo nombre lanza esta excepción.
- *YaNoQuedanMasEntradasDiaponibles* Una vez que ya no quedan mas lugares disponibles en el Vip y querer sacar otra entrada salta está exception.

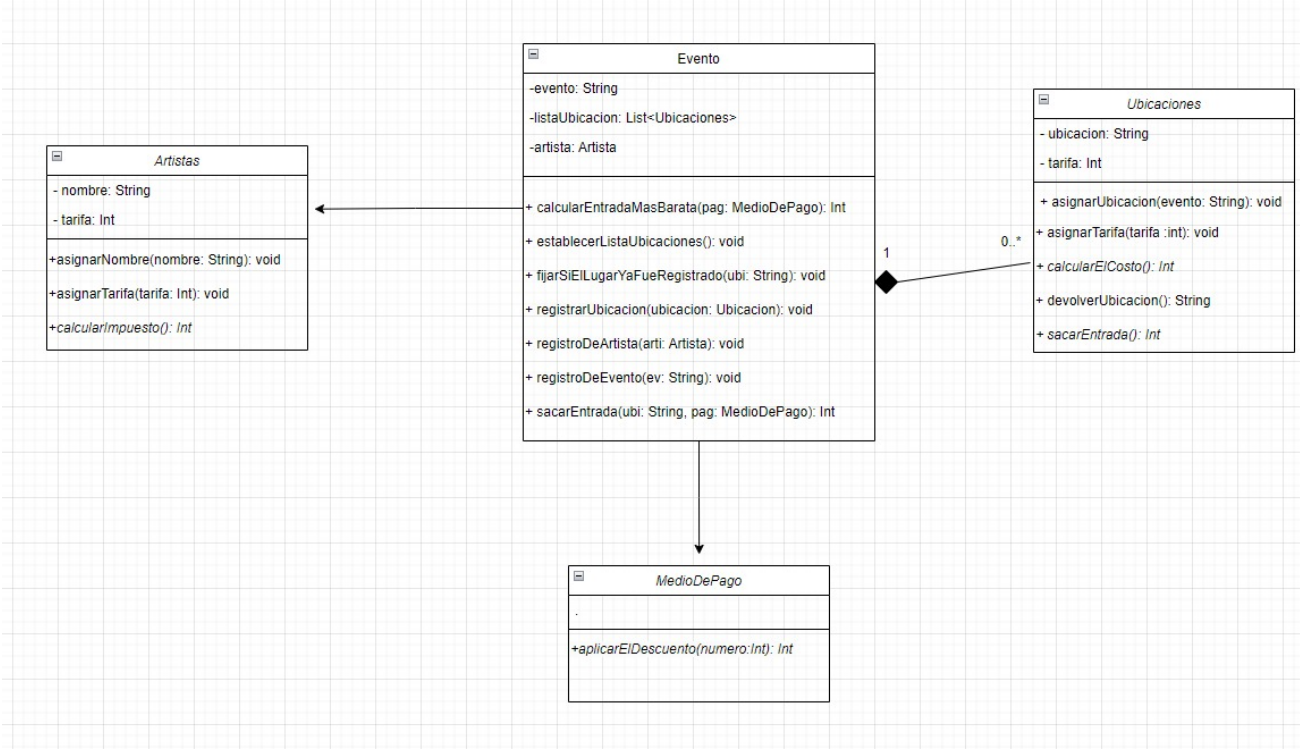
4. Diagrama de clases

En este diagrama muestro como AlgoTek delega todo a la clase Evento

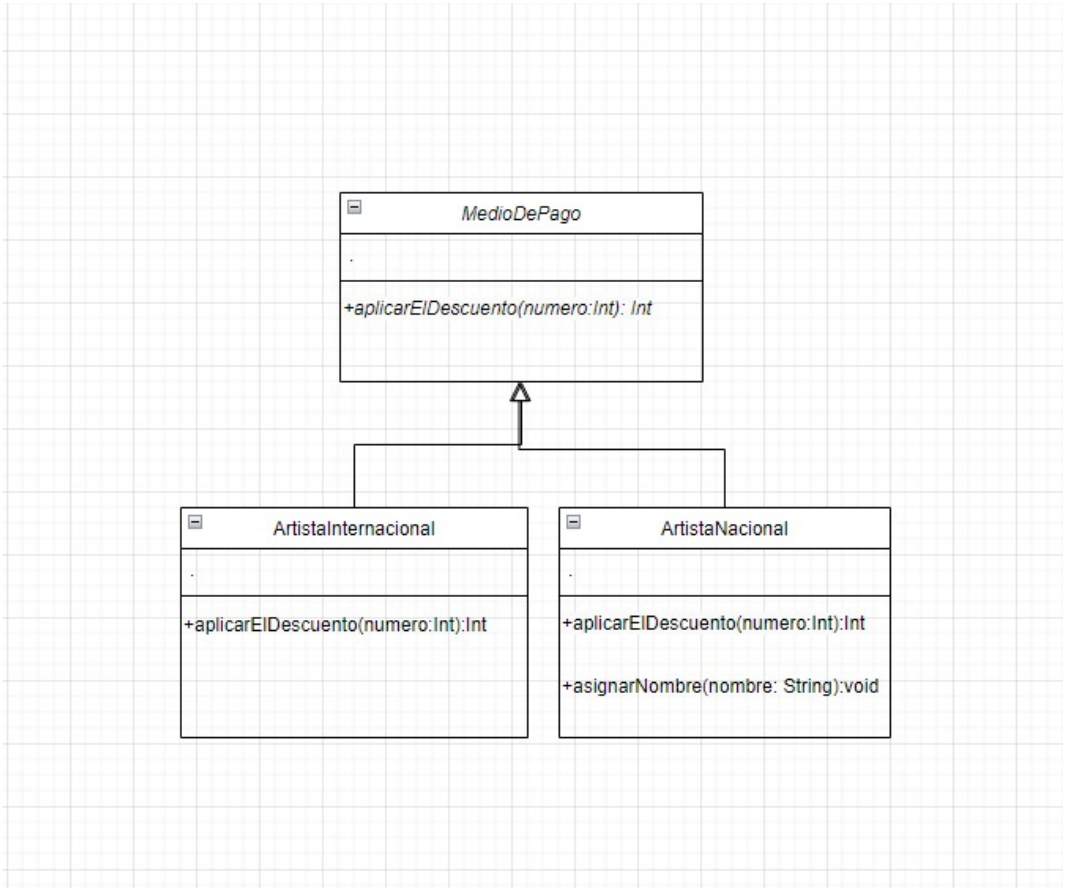


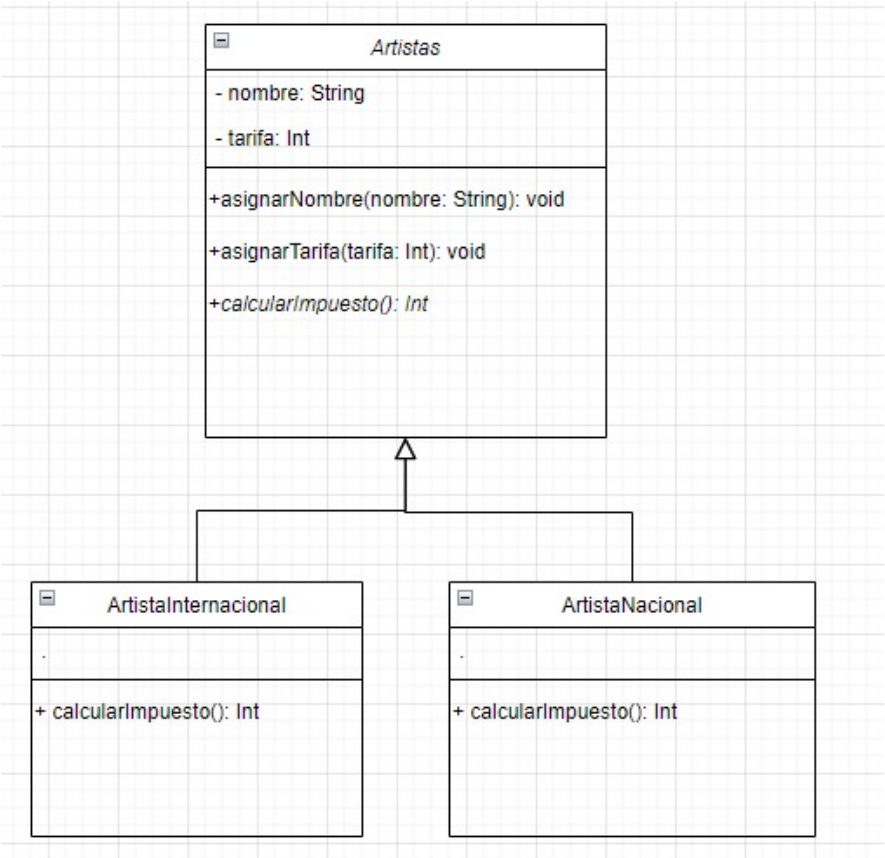
Este para mi es el diagrama más importante, porque acá se visualiza como pensé todo el programa. Quiero aclarar que puse Composición de Ubicaciones / Evento, porque si no hay un Evento yo creo que no tiene sentido que existan unas ubicaciones las cuales están relacionadas con el

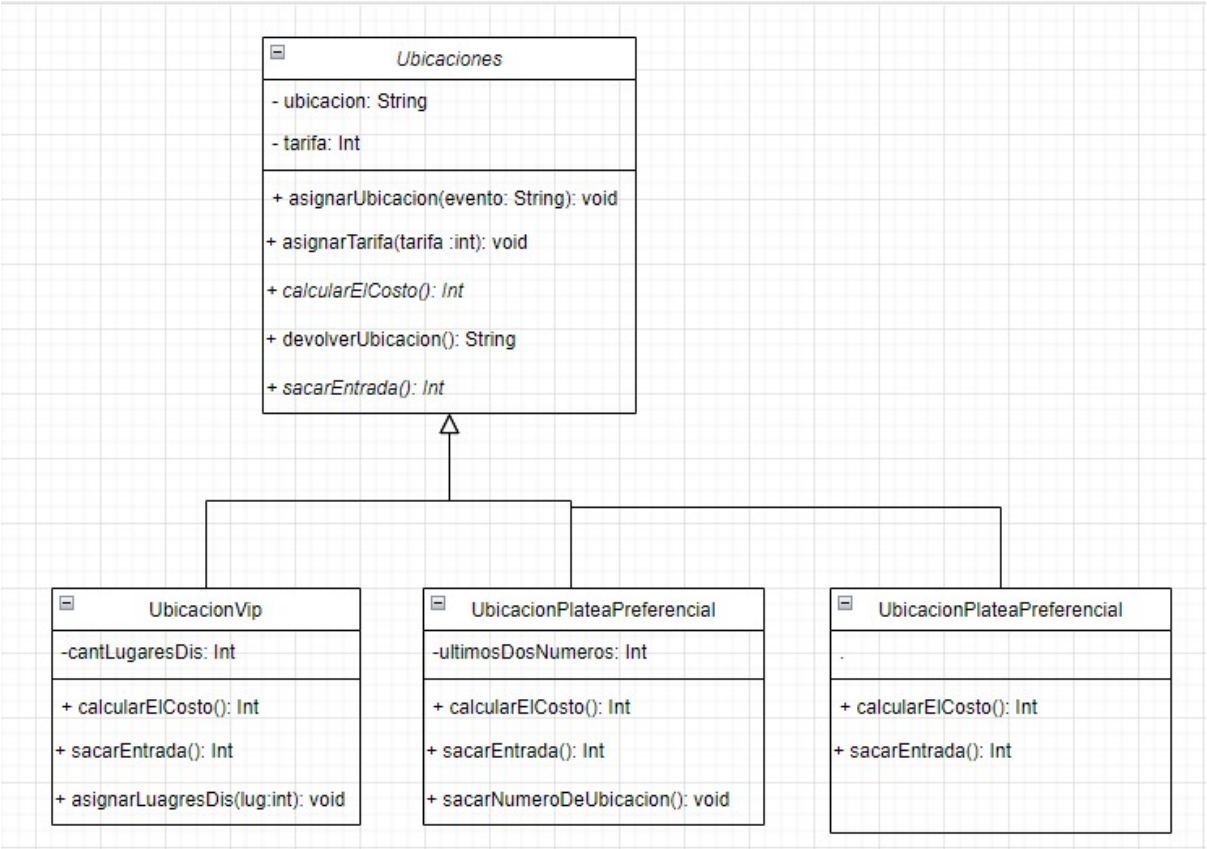
evento, el cual no existe.



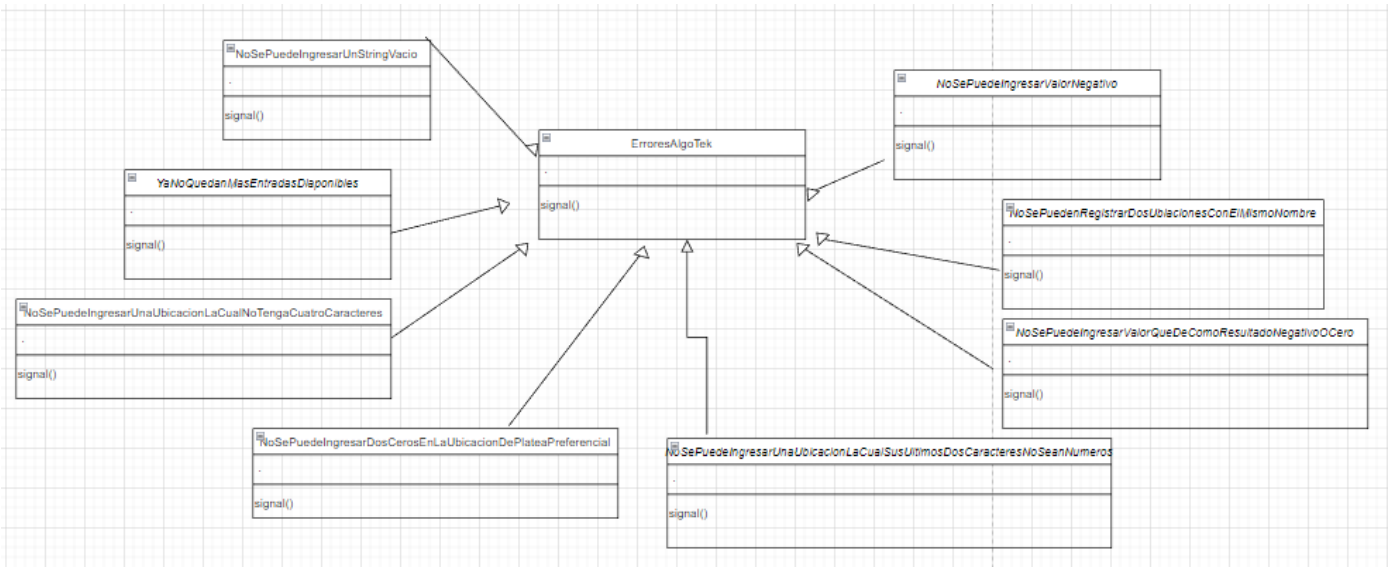
Estos son los diagramas de las clases madres con su hijas, y sus métodos abstractos



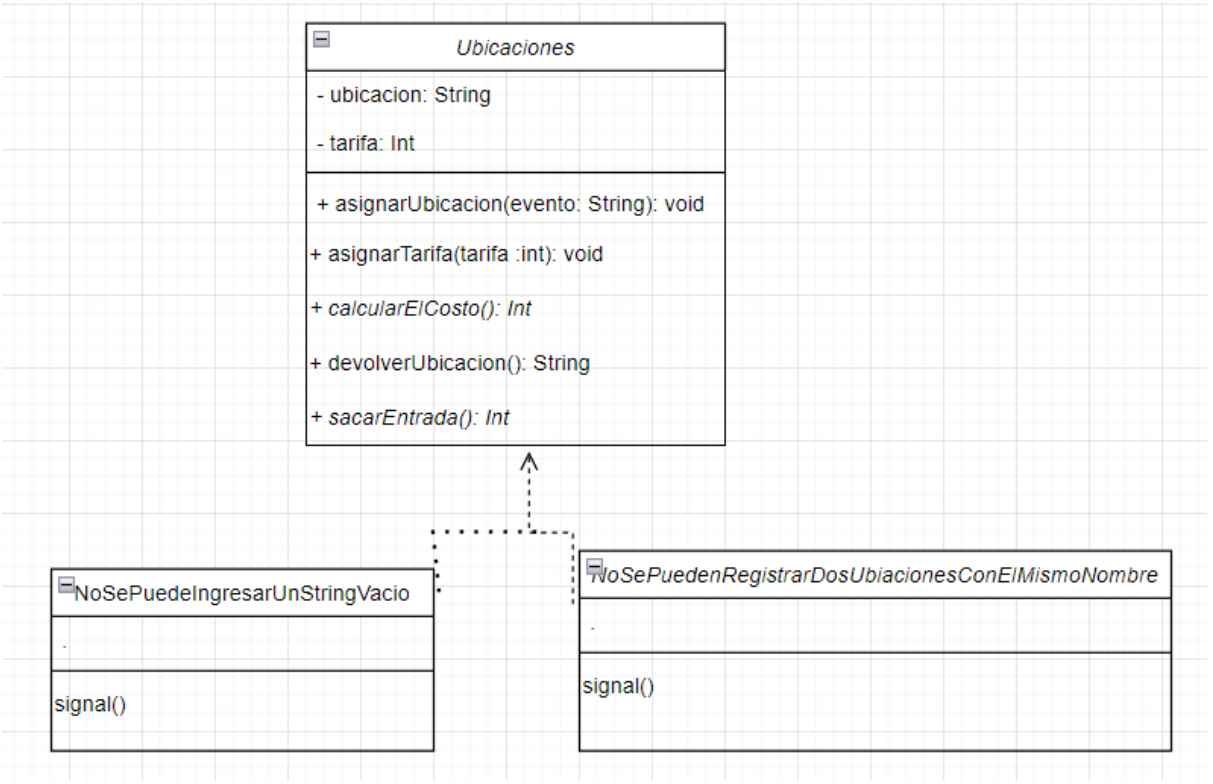

Artistas
- nombre: String
- tarifa: Int
+asignarNombre(nombre: String): void
+asignarTarifa(tarifa: Int): void
+calcularImpuesto(): Int
ArtistaInternacional
.
+ calcularImpuesto(): Int
ArtistaNacional
.
+ calcularImpuesto(): Int

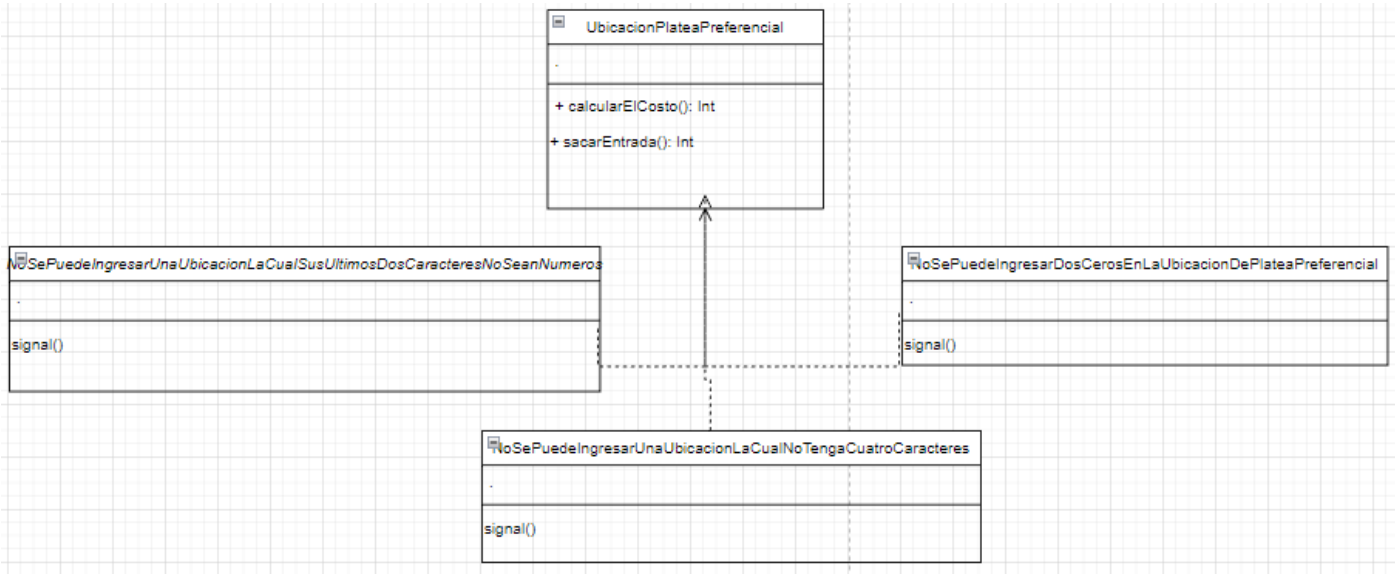

Ubicaciones
- ubicacion: String
- tarifa: Int
+ asignarUbicacion(evento: String): void
+ asignarTarifa(tarifa :int): void
+ calcularElCosto(): Int
+ devolverUbicacion(): String
+ sacarEntrada(): Int
UbicacionVip
-cantLugaresDis: Int
+ calcularElCosto(): Int
+ sacarEntrada(): Int
+ asignarLuagresDis(lug:int): void
UbicacionPlateaPreferencial
-ultimosDosNumeros: Int
+ calcularElCosto(): Int
+ sacarEntrada(): Int
+ sacarNumeroDeUbicacion(): void
UbicacionPlateaPreferencial
.
+ calcularElCosto(): Int
+ sacarEntrada(): Int

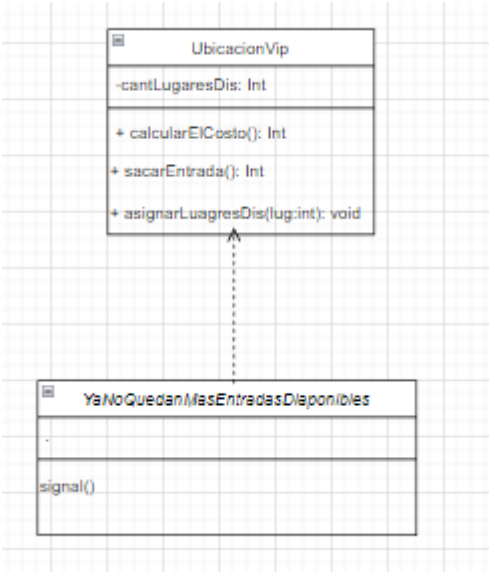
Relacion Entre errores y la clase madre



Uso de las excepciones





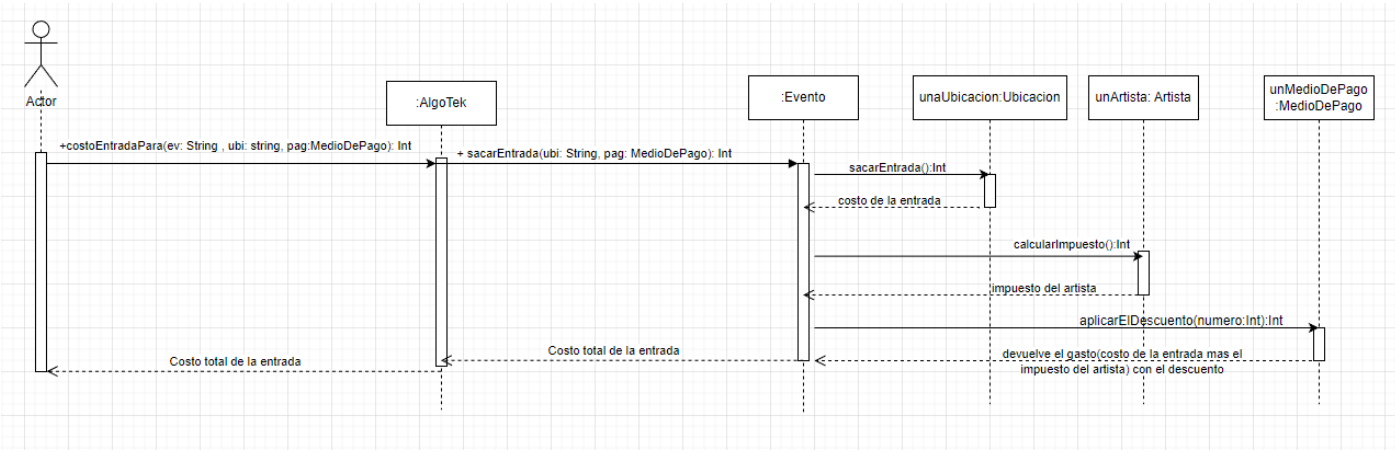


Después las demás excepciones se aplican para todas las demás clases, como poner números negativos, o String vacíos. Por eso pongo estos 3 diagramas que son de las excepciones enfocadas hacia un error en específico.

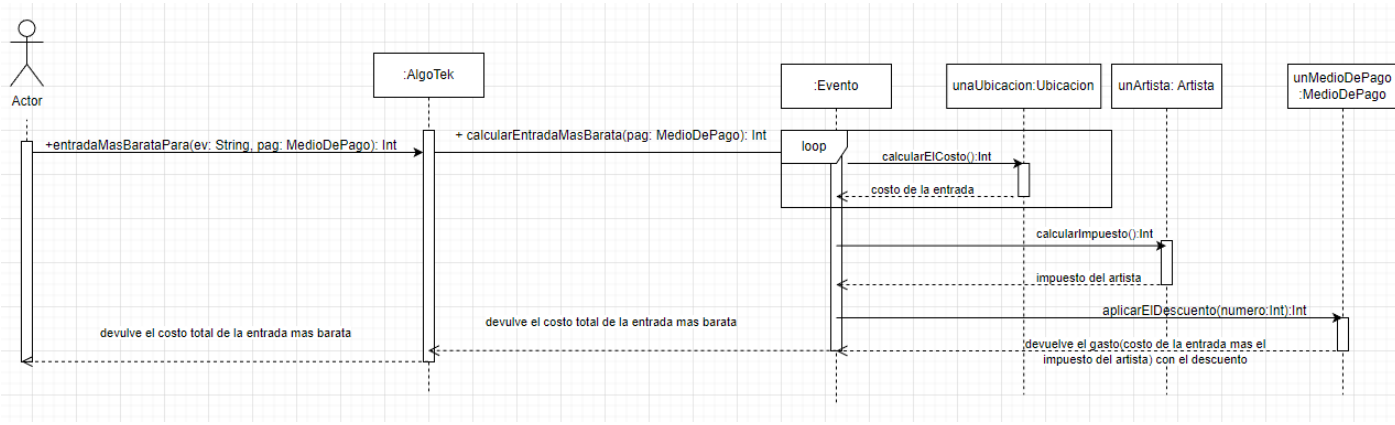
5. Diagrama de secuencia:

1) Sacar una entrada (ya están registradas las ubicaciones)
2) Saber cual es la entrada mas barata (ya están registradas las ubicaciones)
3) Quiero sacar una entrada en ubicacionVip cuando ya no hay mas lugares disponibles, lanza error

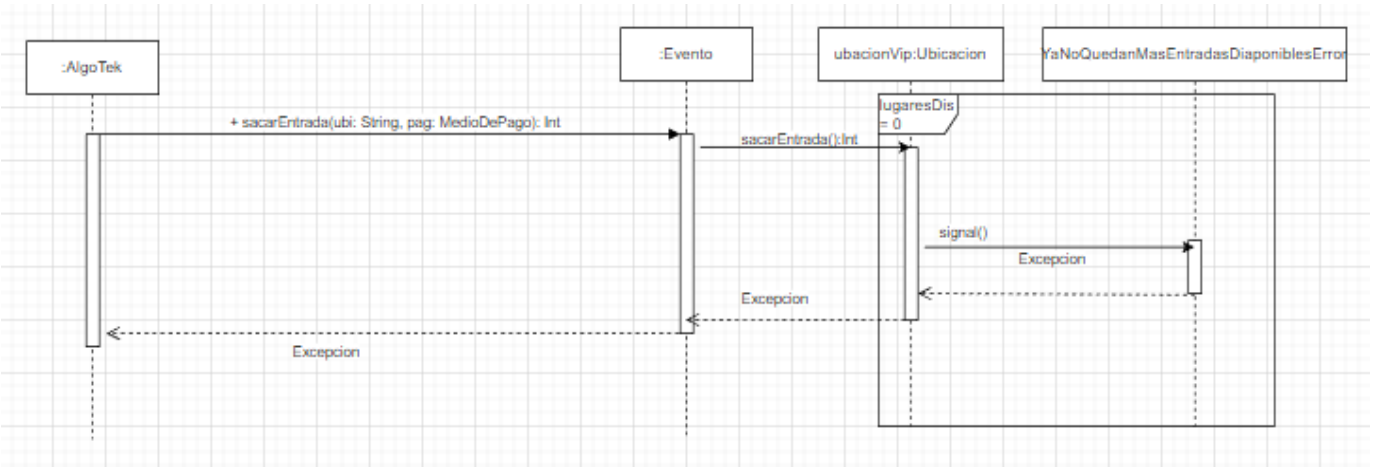
1)


Actor
:AlgoTek
:Evento
unaUbicacion:Ubicacion
unArtista: Artista
unMedioDePago
:MedioDePago
+costoEntradaPara(ev: String , ubi: string, pag:MedioDePago): Int
+ sacarEntrada(ubi: String, pag: MedioDePago): Int
sacarEntrada():Int
costo de la entrada
calcularImpuesto():Int
impuesto del artista
aplicarElDescuento(numero:Int):Int
devuelve el gasto(costo de la entrada mas el impuesto del artista) con el descuento
Costo total de la entrada
Costo total de la entrada

2)


Actor
:AlgoTek
:Evento
unaUbicacion:Ubicacion
unArtista: Artista
unMedioDePago
:MedioDePago
+entradaMasBarataPara(ev: String, pag: MedioDePago): Int
+ calcularEntradaMasBarata(pag: MedioDePago): Int
loop
calcularElCosto():Int
costo de la entrada
calcularImpuesto():Int
impuesto del artista
aplicarElDescuento(numero:Int):Int
devuelve el gasto(costo de la entrada mas el impuesto del artista) con el descuento
devulve el costo total de la entrada mas barata
devulve el costo total de la entrada mas barata

3)


:AlgoTek
:Evento
ubacionVip:Ubicacion
YaNoQuedanMasEntradasDiaponiblesError
lugaresDis
= 0
+ sacarEntrada(ubi: String, pag: MedioDePago): Int
sacarEntrada():Int
signal()
Excepcion
Excepcion
Excepcion