FUNCIONES AUXILIARES

Context Evento::solaparHorarios(tiempo: Integer):Boolean

derive: self.horaIni < tiempo and self.horaFin > tiempo

Context Evento::solapaCon(otro:Evento):Boolean

derive: result = self.solaparHorarios(otro.horaIni)

Context Emplazamiento::dameMaximosSolapantes(eventos : Collection(Evento)) : Integer

derive:

result = eventos->asSet()->iterate(

e1 : Evento ;

max : Integer = 0 |

let

newMax : Integer = eventos->select(e2 | e1.solapaCon(e2))

in

if(newMax > max)

max = newMax

else

max = max

endif

)

Context Emplazamiento::damePiscinasLibres(e1 : Evento) :: Integer

derive:

let eventos : Collection(Eventos) =

self.misEventos->select(e2 |

e2.solapaCon(e1) and

e2.miCompeticion.requierePiscina

)

in

result = numPiscinas - dameMaximosSolapantes(eventos)

Context Emplazamiento::damePistasAtletismoLibres(e1 : Evento) :: Integer

derive:

let eventos : Collection(Eventos) =

self.misEventos->select(e2 |

e2.solapaCon(e1) and

e2.miCompeticion.requierePistaAtletismo

)

in

result = numPistasAtletismo - dameMaximosSolapantes(eventos)

Context Emplazamiento::dameCircuitosLibres(e1 : Evento) :: Integer

derive:

let eventos : Collection(Eventos) =

self.misEventos->select(e2 |

e2.solapaCon(e1) and

e2.miCompeticion.requiereCircuito

)

in

result = numCircuitos – dameMaximosSolapantes(eventos)

INVARIANTES

El numero de jugadores de un equipo estará comprendido entre un nº minimo y un nº máximo

context Equipo inv:

self.misDeportistas->size() >= self.miCompeticion.minJugadoresPorEquipo and

self.misDeportistas->size() <= self.miCompeticion.maxJugadoresPorEquipo

Si una competición tiene un único árbitro la nacionalidad del arbitro debe ser distinta de la de los equipos participantes

Context Evento inv:

let nacionArbitro : Collection(String) = self.misArbitros.nacionalidad in

Self.miCompeticion.nArbitrosPorEvento = 1 implies

self.misEquipos->allInstances(e | e.misDeportistas.nacionalidad->intersection(nacionArbitro)->isEmpty())

Si una competición tiene varios árbitros, no pueden compartir nacionalidad entre ellos (esto implica que para cada equipo, a lo sumo solo hay un arbitro con la misma nacionalidad)

Context Evento inv:

Self.miCompeticion.tipoArbitraje = self.nArbitros > 1 implies

self.misArbitros->forAll(a1,a2 | a1.nacionalidad <> a2.nacionalidad)

Todos los miembros de un equipo tienen que ser de la misma nacionalidad

Context Equipo inv:

Self.misDeportistas->forAll(d1,d2 | d1.nacionalidad = d2.nacionalidad)

Los equipos de un evento tienen que estar inscritos en la competición a la que pertenece el evento

Context Evento inv:

Self.misEquipos->forAll(e| e.miCompeticion = self.miCompeticion)

Los árbitros de un evento tienen que pertenecer a la competición a la que pertenece el evento

Context Evento inv:

Self.misArbitros->forAll(a| a.misCompeticiones->includes(self.miCompeticion))

Un mismo equipo no pueden estar en dos eventos que se solapen

Context Equipo inv:

Self.misEventos->forAll(e1, e2 | e1.solapaCon(e2) = false)

Un mismo árbitro no pueden estar en dos eventos que se solapen

Context Arbitros inv:

Self.misEventos->forAll(e1, e2 | e1.solapaCon(e2) = false)

Un jugador no puede estar en dos equipos con eventos que se solapen

Context Deportista inv:

Self.misEquipos->forAll(eq1,eq2|eq1.misEventos->forAll(ev1 | eq2.misEventos->forAll(ev2 | ev1.solapaCon(ev2) = false)))

El emplazamiento de todo evento cumple las restricciones de la competicion

Context Evento inv:

Self.miCompeticion.necesitaPiscina implies self.miEmplazamiento.numeroPiscina > 0

Context Evento inv:

Self.miCompeticion.necesitaPistaAtletismo implies self.miEmplazamiento.numeroPistaAtletismo > 0

Context Evento inv:

Self.miCompeticion.necesitaCircuito implies self.miEmplazamiento.numeroCircuito > 0

Dado un emplazamiento no hay mas eventos asignados que necesiten recurso que recursos tiene el emplazamiento

Context evento inv:

self.necesitaPiscina implies self.miEmplazamiento.misEventos->select(e|

self.solapaCon(e) and

e.necesitaPiscina

)->size() <= self.miEmplazamiento.numeroPiscinas

Context evento inv:

self.necesitaPistaAtletismo implies self.miEmplazamiento.misEventos->select(e|

self.solapaCon(e) and

e.necesitaPistaAtletismo

)->size() <= self.miEmplazamiento.numeroPistaAtletismo

Context evento inv:

self.necesitaCircuito implies self.miEmplazamiento.misEventos->select(e|

self.solapaCon(e) and

e.necesitaCircuito

)->size() <= self.miEmplazamiento.numeroCircuito

Context Entrada

inv entradaTieneEvento:

self.asiento.miEmplazamiento.misEventos->exist(e| e.solapaCon(self.instante))

inv entradaDisponibleSiYSoloSiNoTieneComprador

self.estado = EstadoReserva::disponible implies self.miComprador->isEmpty() and

self.miComprador->isEmpty() implies self.estado = EstadoReserva::disponible

Ningun asiento tiene dos reservas en el mismo momento

Context Asiento inv:

self.misReservas->forAll(r1,r2|r1 <> r2 and r1.instante <>r2.instante)

Para cada evento hay tantos equipos participantes como requiere la competición

Context Evento inv:

self.misEquipos->size() = self.miCompeticion.nEquiposPorEvento

Para cada evento hay tantos arbitros como requiere la competición

Context Evento inv:

self.misArbitros->size() = self.miCompeticion.nArbitrosPorEvento

Un evento esta finalizado si y solo si tiene un acta asignada

Context Evento inv:

self.miActa->notEmpty() implies self.isFinalizado() and

self.isFinalizado() implies self.miActa->notEmpty()

El saldo de un comprador no puede ser negativo

Context Comprador inv:

self.saldo >= 0

Los equipos que aperecen en los resultados de un acta son los mismos que los que los equipos del evento de acta

Context Acta inv:

self.miEvento->notEmpty() implies self.resultado.equipos = self.miEvento.misEquipos

Las lesiones que aparecen en un acta pueden hablar de un deportista o de un arbitro del evento

Context Lesión inv:

self.miEvento->notEmpty() implies (

self.miActa.miEvento.misEquipos.misDeportistas->includes(self.persona) or

self.miActa.miEvento.misArbitros->includes(self.persona)

)

Las sanciones que aparecen en un acta pueden hablar de un arbitro del evento

Context Sanción inv:

self.miEvento->notEmpty() implies (

self.miActa.miEvento.misEquipos.misDeportistas->includes(self.deportista)

)

Las competiciones de dos equipos por evento tienen una final y una de consolacion

(respectivamente el ultimo y penultimo evento de esa competicion). En estos dos

eventos los equipos participantes han de ser distintos

Context Competicion inv

let

final : Evento = self.getEventosOrdenados()->last(),

consolacion : Evento = self.getEventosOrdenados()->at(self.getEventosOrdenados()->size()-2)

in

Competicion.nEquiposPorEvento = 2 implies

final.misEquipos->intersection(consolacion->misEquipos)->isEmpty()

PRE Y POST CONDICIONES

context Asiento::reservar(c : Comprador, fecha : Integer) : Entrada

pre:

self.misEntradas->select(e | a.fecha = fecha)->isEmpty()

precio >= 0

post:

result.oclIsNew() and

result.asiento = self and

result.fecha = fecha and

result.precio = damePrecio(fecha) and

result.estado = EstadoReserva::reservada and

result.miComprador->notEmpty() and

self.misEntradas->includes(result)

context Asiento::vender(e: Entrada, c:Comprador) : Boolean

post:

if

e.estado@pre = EstadoReserva::reservada and

c = e.miComprador and

c.saldo@pre > e.precio and

self.includes(e)

then

result = true and

e.estado = EstadoReserva::comprada and

c^ajustarSaldo(-e.precio)

else

result = false

endif

Context Asiento::anular(e: Entrada, c : Comprador) : Boolean

post:

if

e.estado@pre = EstadoReserva::reservada and

c = e.miComprador and

self.includes(e)

then

result = true and

e.estado = EstadoReserva::disponible and

e.miComprador->isEmpty() and

self.misEntradas->excludes(e)

else

result = false

endif

context Comprador::ajustarSaldo(incremento :: Integer) : Integer

pre:

self.saldo + incremento >= 0

post:

self.saldo = self.saldo@pre + incremento

context Entrada::isCaducada() : Boolean

body:

if self.estado = EstadoEntrada::reservada

then time() < momentoReserva + caducidadReserva

else false

endif

Context Emplazamiento::quitarEvento(e : Evento) : Boolean

post:

self.misEventos = self.misEventos@pre->excluding(e)

Context Emplazamiento::puedoAsignarEvento(e : Evento) : Boolean

post:

if

self.misEventos->includes(e)

then

result = true

else

result =

e.miCompeticion.requierePiscinas implies

self.damePiscinasLibres(e) > 1 and

e.miCompeticion.requierePiscinas implies

self.damePistasAtletismoLibres(e) > 1 and

e.miCompeticion.requierePiscinas implies

self.dameCircuitosLibres(e) > 1

endif

Context Emplazamiento::asignarEvento(e : Evento) : Boolean

post:

if self.puedoAsignarEvento(e)

then

self.misEventos = self.misEventos@pre->including(e)

result = true

else

result = false

Context Evento::reservarEmplazamiento(e : Emplazamiento) : Boolean

post:

if e = self.miEmplazamieto@pre then result = true

else

if e.puedoAsignarEvento(self)

then

(self.miEmplazamieto@pre->notEmpty() implies

self.miEmplazamiento@pre^quitarEvento(self)) and

self.miEmplazamiento = e and

e^asignarEvento(self) and

result = true

else

result = false

endif

endif

Context Acta::getResultado(e:Equipo) : Integer

pre:

self.miEvento.misEquipos->includes(e)

post:

result = self.resultado->at(self.miEvento.misEquipos->indexOf(e))

Context Evento::getClasificacion() : OrderedSet(Equipo)

pre:

self.finalizado()

post:

if self.miCompeticion.ganaElMayorResultado

then

result = self.miActa.resultado->sortedBy(resultado).equipo

else

result = self.miActa.resultado->sortedBy(-resultado).equipo

endif

Context Competicion::getEventosOrdenados() : OrderedSet(Evento)

derive:

self.misEventos->asOrderedSet()->sortBy(horaInicio)

Context Competicion::getMedallero() : OrderedSet(Equipo)

pre:

self.misEventos->select(e | not e.finalizado())->isEmpty()

post:

let eventosOrdenados : OrderedSet(Evento) = self.getEventosOrdenados() in

if eventosOrdenados->last().getClasificacion()->size() >= 3 or misEquipos->size() < 3

then

eventosOrdenados->last().getClasificacion()

else

eventosOrdenados->last().getClasificacion().prepend(

eventosOrdenados->at(eventosOrdenados->size()-2).getClasificacion()

->first())

endif

Context Evento::finalizar(a : Acta) : Boolean

post:

if self.miActa->isEmpty() then

self.miActa = a and

a.miEvento = self and

result = true and

else

result = false

endif