Universitat de Girona
Escola Politècnica Superior

Pràctiques ESII

Curs 2025/26

GEINF- GDDV

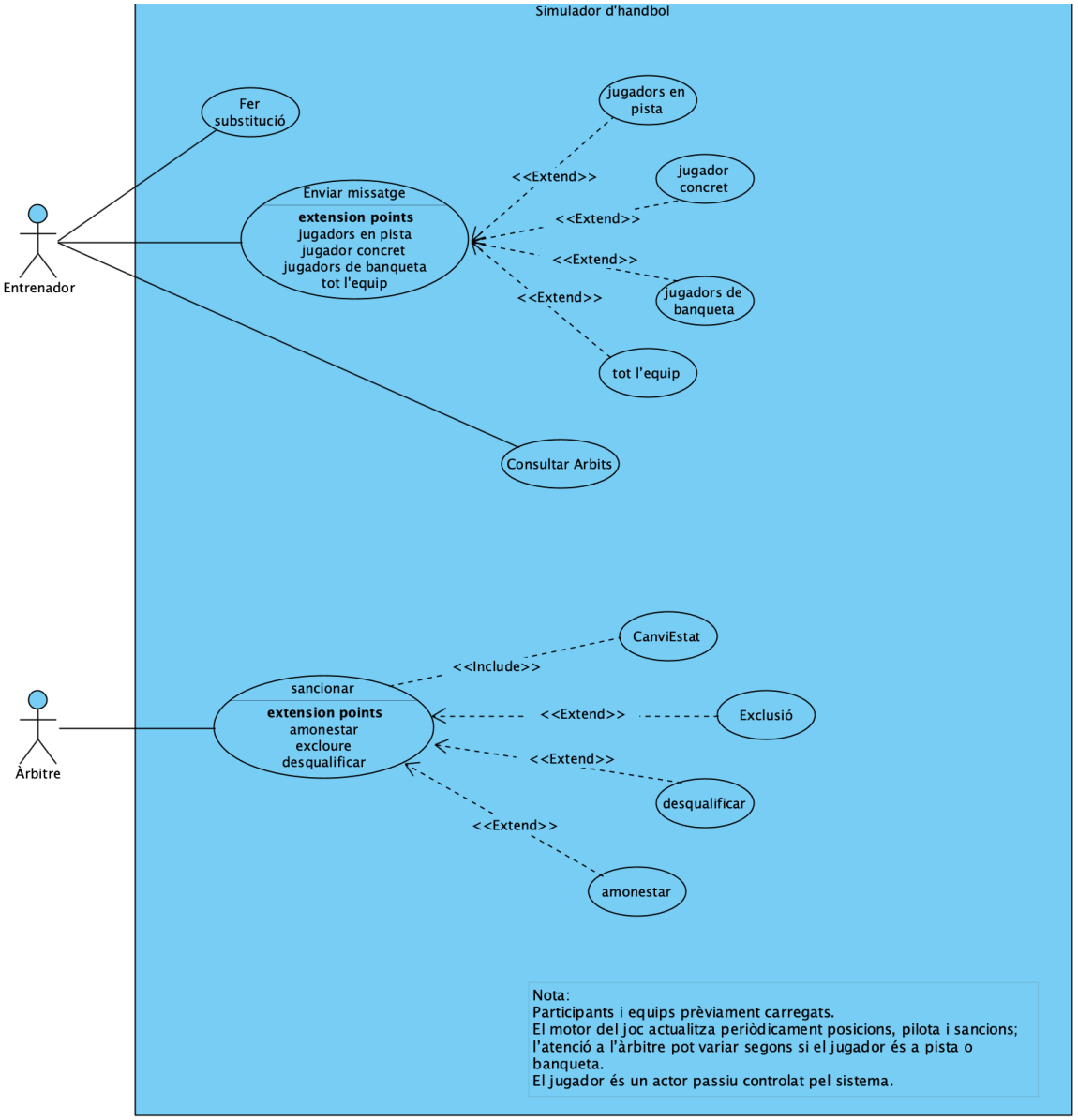
Pràctica 4

Grup O
Erik Nepomnjastsih, Biel Bosch, Matias Segarra

(1 de setembre de 2025)

Índex

1 Diagrama de casos d'ús



Explicació:

Aquest cas d'ús representa la simulació del partit de handbol. El nostre diagrama de casos d'ús té 2 actors, Entrenador i Àrbitre. Cadascun d'aquests té una sèrie d'accions que són úniques per a ell mateix.

L'àrbitre podrà fer una **sanció**, la qual tindrà <<extends>> cap a **amonestar**, **exclusió** i **desqualificar**. Són extends, ja que sempre que l'àrbitre vulgui sancionar haurà d'escollir entre aquestes 3 opcions. Quan es realitza aquesta acció inmediatament haurem d'actualitzar l'**estat del jugador**. Aquest estat està donat per un "string" (<<enumeració>>) d'amonestat, desqualificat o bé exclòs. És per aquest motiu que hem considerat que és un <<include>> de canviar estat del jugador.

Lentrenador tindrà l'opció de **consultar arbitres**. És aquí on actualitzarem el seu **nivell d'atenció** cap a l'àrbitre. L'entrenador tindra l'opcio d'**enviar un missatge** a un jugador en la pista, banqueta,

un jugador en concret o bé a tots alhora. És per això que son <<extends>>, ja que haurà d'escollir a qui envia el missatge.
Finalment, l'entrenador serà l'únic habilitat per realitzar les **substitucions** dels jugadors.

1.1 Cas d’ús “Envia missatge als jugadors de la pista”

1.1.1 Fitxa del cas d’ús

CAS D’ÚS	Envia missatge als jugadors de la pista
Versió	Versió Usuari
Autors	Biel, Erik, Matias
Descripció	L’Entrenador es pot comunicar amb tots els jugadors en pista.
Actors	Entrenador, Jugador
Precondició	**(1)** Participants i equips prèviament carregats. **(2)** Hi ha almenys un jugador del seu equip en pista. **(3)** El sistema permet enviar missatges a subgrups (pista/banqueta/tots/jugador concret).
Flux principal	**1)** L’Entrenador inicia “Enviar missatge”. **2)** Selecciona “jugadors en pista” (qui ha de rebre el missatge) **3)** Introdueix el text del missatge i confirma. **4)** El sistema identifica els receptors. **5)** El sistema lliura el missatge als destinataris.
Subflux	
Fluxos alternatius / Excepcions	**E1)** Jugador no atent temporalment: pot no percebre la notificació; la informació resta disponible via consulta d’esdeveniments del partit.
Postcondició	**(Èxit)** El missatge ha estat enviat correctament; queda registrat (si es contempla l’històric). **(Error)** Cap canvi d’estat.

1.2 Cas d’ús “Substitució jugador”

1.2.1 Fitxa del cas d’ús

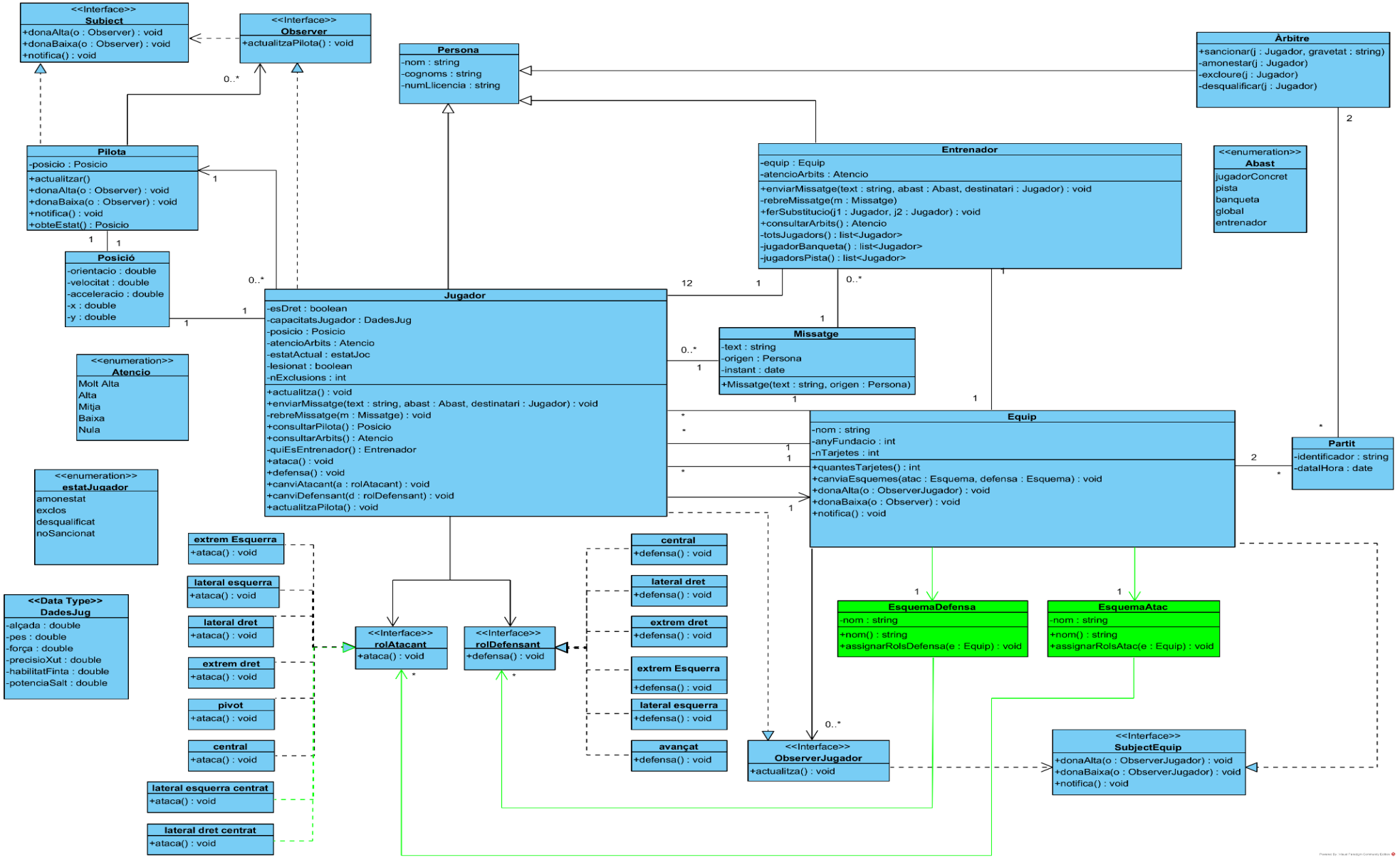
CAS D’ÚS	Substitució jugador
Versió	Versió Usuari
Autors	Biel, Erik, Matias
Descripció	L’Entrenador substitueix un jugador que és a pista per un jugador de la banqueta (inclòs el cas porter, jugador de camp).
Actors	Entrenador
Precondició	**(1)** Partit en curs i estat pista/banqueta actualitzat. **(2)** Jugadors del seu equip donats d’alta. **(3)** El jugador que entra és elegible (no exclòs/desqualificat/lesionat). **(4)** Es pot complir el límit de jugadors en pista després del canvi.
Flux principal	**1)** L’Entrenador inicia “Fer substitució”. **2)** Escull els jugadors que es canviaran entre ells. **3)** Realitza la substitució.
Subflux	Cap
Fluxos alternatius / Excepcions	
Postcondició	**(Èxit)** Estat pista/banqueta actualitzat i canvi registrat. **(Error)** Cap canvi d’estat.

1.3 Cas d'ús "Amonestar un jugador"

1.3.1 Fitxa del cas d'ús

CAS D'ÚS	Amonestar un jugador
Versió	Versió Usuari
Autors	Biel, Erik, Matias
Descripció	L'Àrbitre aplica targeta groga a un jugador.
Actors	Àrbitre, Jugador
Precondició	**(1)** Partit en curs; jugadors i equips registrats. **(2)** Àrbitre autoritzat per sancionar. **(3)** S'identifica el jugador a sancionar.
Flux principal	**1)** L'Àrbitre inicia "Amonestar un jugador". **2)** Selecciona el jugador que reb l'amonestació. **3)** El sistema consulta l'històric de sancions del jugador i de l'equip. **4)** El sistema verifica que el jugador no tingui ja groga i que l'equip no tingui 3 grogues. **5)** El sistema assigna la targeta groga i actualitza els recomptes. **6)** El sistema registra la sanció.
Subflux	Cap
Fluxos alternatius / Excepcions	**E1)** Jugador ja amonestat (2a groga): a (4) el sistema no aplica groga i inicia exclusió (2 min); registra i informa. **E2)** Quarta groga d'equip: a (4) el sistema no aplica groga i inicia exclusió (2 min) al jugador seleccionat; registra i informa.
Postcondició	**(Èxit)** Groga aplicada i registrada; recomptes actualitzats.(Alternativa E1/E2) Exclusió (2') aplicada i registrada. **(Error)** Cap canvi d'estat.

2 Diagrama de classes



2.1. Explicació del diagrama

El diagrama de la imatge descriu com es comporta l'aplicació descrita imprecisament per la Fàtima. A continuació una explicació del diagrama:

Persona: Es declara una classe abstracta "Persona" que compta amb atributs comuns que hereten la resta de persones del diagrama. En el punt 9 es fa referència a la seva necessitat.

Àrbitre: Es tracta d'una classe que estén de Persona, pel que hereta tots els seus atributs. Aquest implementa els següents mètodes:

- Públics:
  - **sancionar (j : jugador, gravetat : string):** Aquesta serà la funció que es cridarà sempre que s'ha de "castigar" un jugador, els paràmetres que rep son: un jugador (qui rep el càstig) i un string anomenat gravetat que servirà per especificar quin dels mètodes privats haurà de cridar en funció del grau d'aquesta.
- Privats:
  - **amonestar (j : jugador):** Aquesta funció rep per paràmetre el jugador que ha de ser amonestat, aquest rebrà una targeta groga i posteriorment es comprovarà si ja és la seva segona tarjeta d'aquest color o si és la tercera de l'equip. En cas afirmatiu haurà de fer la crida a excloure, pel contrari es modifica l'estat del jugador a amonestat i es suma una tarja al nombre total d'aquestes rebudes per l'equip.

    S'entra en aquest mètode donada una infracció lleu (gravetat baixa).

  - **excloure (j : jugador):** Aquesta funció rep per paràmetre el jugador que ha de ser exclòs, es retira el jugador durant 2 minuts i durant aquest temps l'equip juga amb un jugador menys. També farà la comprovació per saber si és el tercer cop que s'esclou aquell jugador, en cas afirmatiu aquest és desqualificat, pel contrari, s'actualitza l'estat i suma u al numero d'exclusions del jugador.

    S'entra en aquest mètode donada una infracció "menys greu" (gravetat mitja), per acumulació de 2 amonestacions individuals o 3 de col·lectives.

  - **desqualificar (j : jugador):** Aquesta funció rep per paràmetre el jugador que ha de ser desqualificat. S'expulsa al jugador de manera definitiva del partit i l'equip haurà de jugar amb un jugador menys durant 2 minuts.

    S'entra en aquest mètode donada una infracció molt greu (gravetat alta), per acumulació de 3 exclusions.

La classe amb els seus mètodes s'expliquen al punt 5.

Entrenador: Aquesta classe estén de Persona i a més a més implementa els atributs privats següents:

- **equip : Equip:** Atribut de tipus Equip, dessigna l'equip al que entrena.
- **atencioArbits : Atencio:** Atribut de tipus Atencio que descriu el nivell d'atenció que està parant al arbitre.

Aquesta classe també compta amb els mètodes següents:

- Públics:
  - **enviarMissatge(text : string, abast : Abast, destinatari : Jugador) :** void: Rep per paràmetre qui envia el missatge, quin és aquest missatge, a qui va dirigit (un jugador concret, els jugadors de pista, els de la banqueta o tots) i el jugador que ha de rebre el missatge en cas que sigui concret, null en cas que no sigui d'abast individual.

    Aquesta funció s'encarregarà de crear el missatge i que l'intenti rebre el destinatari.

  - **ferSubstitucio(j1 : Jugador, j2 : Jugador): void:** Aquest mètode rep els 2 jugadors per a fer l'intercanvi i en cas que es pugui fer, es realitza.
  - **consultarArbits(): Atencio:** Amb la crida a aquesta funció, s'obté el nivell d'atenció que està parant qui la realitza a els Àrbitres.
- Privat:
  - **rebreMissatge(m : Missatge) : void:** Aquesta funció fa que aquell que la crida intenti rebre el missatge enviat per paràmetre.
  - **totsJugadors() : list<Jugador>:** Retorna una llista de tots els jugadors de la plantilla.
  - **jugadorBanqueta() : list<Jugador>:** Retorna una llista de tots els jugadors de la banqueta.
  - **jugadorPista() : list<Jugador>:** Retorna una llista de tots els jugadors de la pista.

<<DataType>> DadesJug: S'ha fet un datatype que englobi les característiques concretes d'un jugador, amb una única dada s'engloba tot.

Aquest datatype engloba:

- **pes : double:** pes corporal del jugador en quilograms.
- **alçada : double:** l'alçada del jugador en metres.
- **força : double:** valor numèric que representa la potencia física del jugador.
- **precisióXut : double:** grau d'encert en el llançament a porteria.
- **habilitatFinta : double:** mesura la capacitat del jugador per fer driblings.
- **potenciaSalt : double:** indica la potència del sant vertical del jugador.
- **esDret : boolean:** indica si el jugador es dretà (true) o esquerrà (false).

Jugador: Es tracta d'un extend de Persona, representen els jugadors d'handbol, aquests implementen els atributs privats següents:

- **posicio : Posicio:** atribut de tipus Posició que guarda la ubicació actual del jugador dins del camp.
- **atencioArbits : Atencio:** defineix el nivell d'atenció que el jugador està prestant als àrbitres
- **lesionat : boolean :** indica si el jugador està lesionat (true) i per tant no podrà participar en el joc.
- **nExclusions : int:** comptador d'exclusions acumulades pel jugador. Un cop arriba a tres implica la desqualificació automàtica.
- **caracteristiquesJugador: DadesJug:** Totes les característiques d'un jugador.

També està compost dels següents mètodes:

- Públics:
  - **actualitza() : void:** Aquest mètode s'utilitzarà per a actualitzar el jugador al haver estat notificat pel patró observer.
  - **enviarMissatge(text : string, abast : Abast, destinatari : Jugador) :** void: Rep per paràmetre qui envia el missatge, quin és aquest missatge, a qui va dirigit (un jugador concret o l'entrenador) i el jugador que ha de rebre el missatge en cas que sigui la opció del company, null en cas que no.

    Aquesta funció s'encarregarà de crear el missatge i que l'intenti rebre el destinatari.

  - **consultarArbits() : Atencio:** Retorna una Atenció que representa el cas que s'està parant als àrbitres.

  - **canviAtacant(a: Rolatacant)t**: Usa el patró strategy per canviar a un rol atacant.
  - **canviDefensant(d: Roldefensant):** Usa el patró strategy per canviar a un rol defensor.
  - **actualitzaPilota(): void** Actualitza la posició de la pilota amb el patró observer pull.
- Privat:
  - **rebreMissatge(m : Missatge) : void:**

EsquemaAtac: Classe que representa l'esquema d' atac de l'equip. Compta amb un atribut privat:

- **nom : string:** Nom de l'esquema

Mètodes públics:

- **assignaRolsAtac(e: Equip): void:** Assigna els rols corresponents als jugadors de l'equip un cop es crida canviaEsquemes().
- **nom() : string:** retorna el nom.

EsquemaDefensa: Classe que representa l'esquema de defensa de l'equip. Compta amb un atribut privat:

- **nom : string:** Nom de l'esquema

Mètodes públics:

- **assignaRolsDefensa(e: Equip): void:** Assigna els rols corresponents als jugadors de l'equip un cop es crida canviaEsquemes().
- **nom() : string:** retorna el nom.

Les classes d'esquemes estan connectades amb les interfícies dels rols de forma que les funcions d'assignarRolsX s'ajuden d'aquestes per fer els canvis pertinents.

Equip: Aquesta classe representa cada equip que participa.

- **nTarjetes : int:** Controla les targetes acumulades per l'equip.
- **anyFundacio : int:** Informació sobre quin any es va fer l'equip.
- **nom : string:** Nom de l'equip.

S'ha eliminat els atributs d'esquemes i s'ha afegit la relació amb les classes anteriorment descrites que ja s' en ocupen

- Mètodes públics:
  - **quantesTarjetes(): int:** getter de nTarjetes per a que l'Àrbitre el consulti. En el punt 1 es fa referència a la seva necessitat.
  - **canviaEsquemes (atac: Esquema, defensa: Esquema) : void:** Canvia l'esquema actual (atac o defensa) de l'equip per un altre. Serà invocada per l'entrenador.

Partit: Representa una instancia concreta d'un partit.

- Atributs privats:
  - **id : int:** Identificador individual de cada partit
  - **dataIHora:** Dia i hora del partit.

En el punt 1 es fa referència a la seva necessitat.

JugadorPista/JugadorBanqueta: Són una relació Jugador-Equip.

Posició: Aquesta classe té els atributs que permeten saber la posició d'un objecte en el camp de handbol. La classe posició estarà associada amb pilota i jugador.

- **orientacio : double :** determina cap on "mira" un objecte.
- **velocitat : double :** defineix la velocitat d'un objecte.
- **acceleracio : double :** defineix l'acceleració d'un objecte.
- **x : double :** coordenada horitzontal al camp.
- **y : double :** coordenada vertical al camp.

En el punt 4 es fa referència a la seva necessitat.

Pilota: La pilota només caldrà tenir la posició com a atribut. **Patró observer pull aplicat**

- **posicio : Posicio :** indica la posició actual de la pilota en un partit.
- **actualitza() : void:** Aquest mètode s'executa periòdicament per part del motor del joc i s'encarrega de refrescar l'estat de la pilota **amb l'ajuda del patro observer**.

En el punt 7 es fa referència a la seva necessitat.

estatJugador: Representa la situació d'un jugador dins del partit. Ens especifica si un jugador pot formar part del joc, si està temporalment fora del joc o definitivament a causa d'una sanció

Atenció: Aquesta enumeració serveix per representar el nivell d'atenció que un jugador o entrenador està prestant als àrbitres en un moment determinat**.**

Abast: Defineix els diferents destinataris d'un missatge dins del partit.

2.1.2. Strategy

Strategy: En el nostre cas, es tracta d'una situació on tenim la classe Jugador, que pot assumir diferents rols dins el camp segons si està atacant o defensant. Cada jugador pot tenir assignades dues estratègies: una d'atac i una de defensa, que representen comportaments diferents dins del joc.

Jugador actua com a context, ja que és qui manté una referència a les estratègies i utilitza aquestes per dur a terme les accions específiques, com atacar o defensar. Aquesta estructura permet que el jugador pugui canviar el seu comportament durant l'execució simplement substituint l'objecte estratègia corresponent, cosa que fa el sistema més flexible i fàcil d'ampliar.

Les interfícies **rolAtacant** i **rolDefensant** representen les interfícies Strategy, ja que defineixen les operacions generals (ataca() i defensa()) que cada rol haurà d'implementar. Les classes com extrem esquerra, lateral dret, pivot, central, etc., són les estratègies concretes, cadascuna amb la seva manera específica d'executar l'atac o la defensa segons el tipus de jugador.

D'aquesta manera, l'ús del patró Strategy ens permet encapsular el comportament dels rols, evitar condicions dins de Jugador i facilitar l'extensió del sistema quan es vulgui afegir un nou rol o modificar-ne un d'existent.

Exemple patró Strategy

Es tracta d'una situació on tenim una classe abstracta Programador i diferents especialistes en algún llenguatge de programació. Aquests programadors podran dissenyar i també podran programar (seran les nostres interficies). Quan dissenyin podran fer-ho a mà o bé amb el visual paradigm. Quan programin ho faran amb chat gpt o bé fent ús de les seves capacitats intel·lectuals.

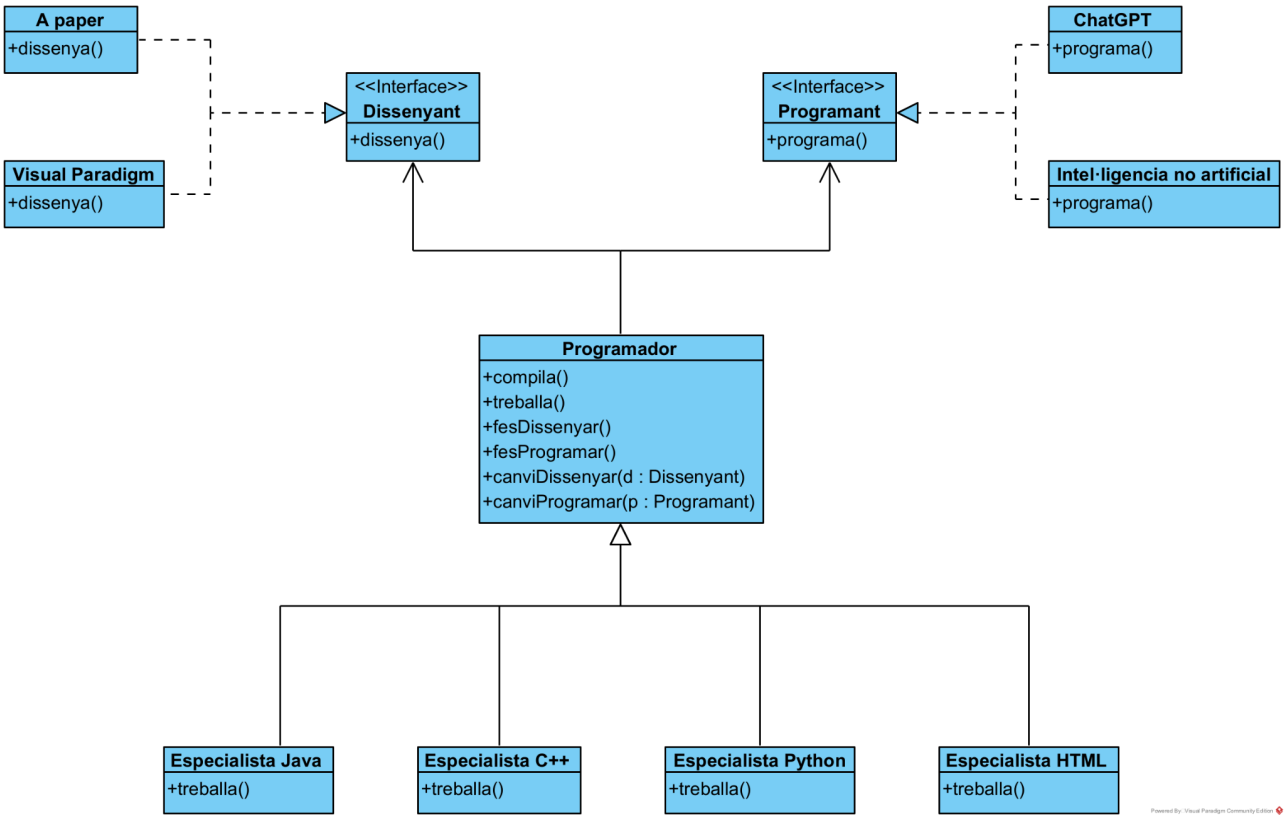
Programador = Context: És una classe abstracta que té l'estratègia de dissenyar i una de programar. Implementa mètodes per fer les accions i té l'opció d'executar-les canviant el paràmetre corresponent. El canvi de paràmetres ajuda a l'encapsulació i a més a més permet que es pugui canviar el mètode a executar en temps d'execució, cosa que el fa molt versàtil.

Dissenyant i Programant = Interfícies Strategy. Aquestes interficies fan una operació cadascuna, dissenyar i programar respectivament. Com a l'exemple del nedadors/pedaladors fet a classe.

Amb paper / Visual Paradigm = Representen les diferents formes que pot adoptar el procés de disseny dins de la interfície Dissenyant.

Chat GPT / Intel·ligència no artificial = Representen les diferents maneres de programar dins de la interfície Programant.

Especialista en Python/Java/C++/HTML = Subclasses de Programador. Poden establir valors per defecte de les estratègies.


A paper
+dissenya()
Visual Paradigm
+dissenya()
<<Interface>>
Dissenyant
+dissenya()
<<Interface>>
Programant
+programa()
ChatGPT
+programa()
Intel·ligencia no artificial
+programa()
Programador
+compila()
+treballa()
+fesDissenyar()
+fesProgramar()
+canviDissenyar(d : Dissenyant)
+canviProgramar(p : Programant)
Especialista Java
+treballa()
Especialista C++
+treballa()
Especialista Python
+treballa()
Especialista HTML
+treballa()

2.1.3. Observer

Es tracta d'una situació on tenim la Pilota, la qual es mou contínuament pel camp i de la qual tots els jugadors necessiten conèixer la seva posició.
Cada vegada que la Pilota actualitza la seva posició, aquesta notifica automàticament tots els jugadors que l'estan observant.
D'aquesta manera, cada jugador pot reaccionar al moviment de la pilota segons el seu rol o fins i tot ignorar-la temporalment si ha perdut la visió de la jugada.
Aquesta estructura correspon a un patró Observer en versió pull; la Pilota només avisa que alguna cosa ha canviat i els jugadors, si ho decideixen, consulten la nova posició mitjançant obteEstat().

Subject
En el nostre cas, la Pilota és el Subject concret: manté la llista d'observadors i és qui fa notifica() quan la seva posició canvia.
També donaAlta(o : Observer) quan un jugador entra a pista, la pilota l'hi dona d'alta i donaBaixa(o : Observer) quan és substituït/exclòs, la pilota el dona de baixa.

Observer
Defineix l'operació actualitzaPilota().
En la nostra aplicació, els Jugadors implementen Observer i reaccionen al moviment de la pilota quan reben l'avís.

Subject concret = Pilota
La Pilota implementa Subject i és qui:

- Guarda la seva posició
- S'actualitza quan el motor del joc la modifica
- Avisa tots els observadors amb notifica() perquè decideixin si volen consultar la nova posició.

És l'element central que genera els canvis als quals els jugadors han de reaccionar.

Observer concret = Jugadors
Els jugadors implementen l'Observer i reben l'avís quan la pilota canvia de posició.
Cada jugador decideix si consulta la nova posició (obteEstat()) segons el seu estat, rol o nivell d'atenció.
Això permet simular que alguns jugadors poden "perdre de vista" la pilota.

Exemple patró Observer

Es tracta d'una situació on tenim un sistema central que recull notícies i diversos programes informatius que volen actualitzar-se automàticament quan arriba una notícia nova.
Quan el RecullNoticies rep una notícia de la fontNoticies (el fitxer d'entrada), aquesta és enviada als seus observadors perquè cadascun d'ells l'actualitzi o la mostri segons les seves necessitats.
Aquesta estructura correspon al patró Observer en versió push.

Subject = Subject .
Exactament la mateixa estructura que a l'exemple de classe.

Observer = Observer
Exactament la mateixa estructura que a l'exemple de classe.

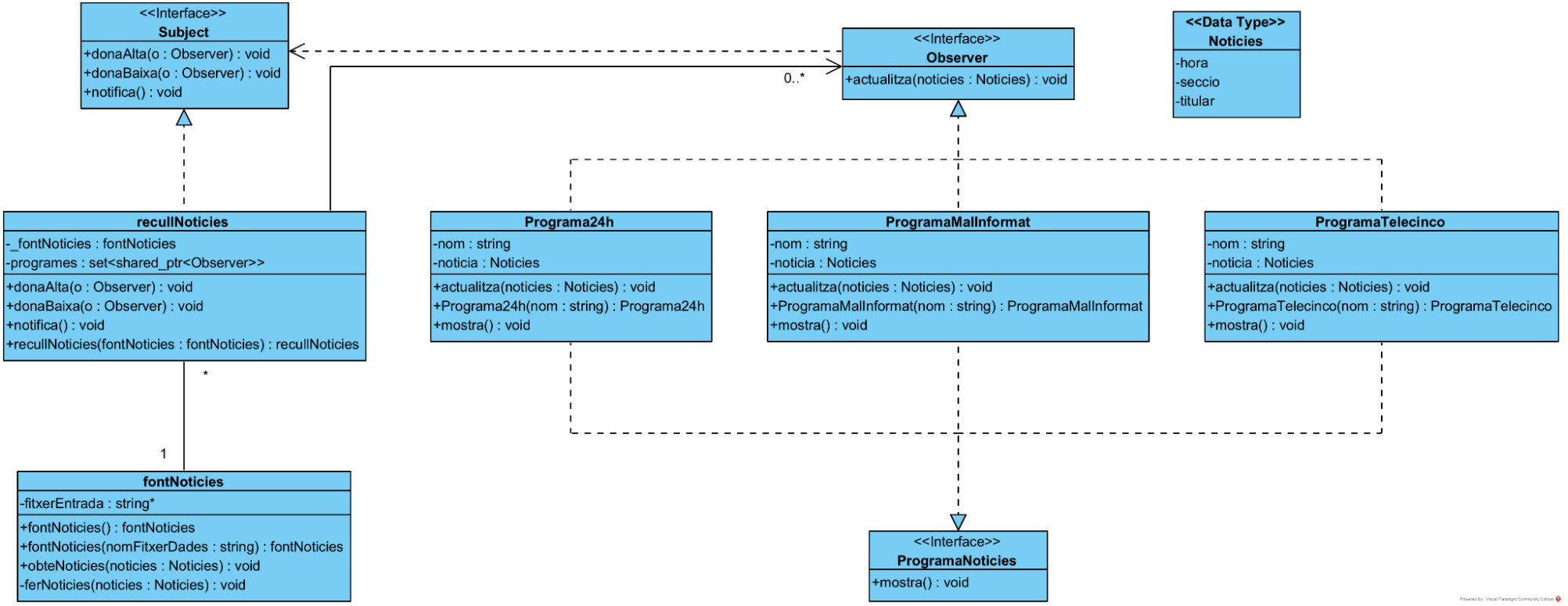
Rastrejador = recullNoticies.
Al igual que la original, rep les dades de fontNoticies i les envia als observadors amb el notifica.

RellotgeEspavilat = fontNoticies
Es la font de les notícies, les llegeix, les crea i proporciona a recullNoticies on seran gestionades.

Visualitzadors diversos = Programa24h, ProgramaTelecinco, ProgramaMalInformat
Simulen diferents programes de Notícies, un que està atent les 24h del dia i les diu totes, un de prensa rosa i un últim que sempre s'informa malament (confón hores o directament no sap la notícia).

Dades = Noticies
S'ha canviat les dades per notícies.

Visualitzador = ProgramaNoticies
No és part del patró, però queda més clar si es posa. (es pot ometre).


<<Interface>>
Subject
+donaAlta(o : Observer) : void
+donaBaixa(o : Observer) : void
+notifica() : void
<<Interface>>
Observer
+actualitza(noticies : Noticies) : void
0..*
<<Data Type>>
Noticies
-hora
-seccio
-titular
recullNoticies
-_fontNoticies : fontNoticies
-programes : set<shared_ptr<Observer>>
+donaAlta(o : Observer) : void
+donaBaixa(o : Observer) : void
+notifica() : void
+recullNoticies(fontNoticies : fontNoticies) : recullNoticies
*
1
fontNoticies
-fitxerEntrada : string*
+fontNoticies() : fontNoticies
+fontNoticies(nomFitxerDades : string) : fontNoticies
+obteNoticies(noticies : Noticies) : void
-ferNoticies(noticies : Noticies) : void
Programa24h
-nom : string
-noticia : Noticies
+actualitza(noticies : Noticies) : void
+Programa24h(nom : string) : Programa24h
+mostra() : void
ProgramaMalInformat
-nom : string
-noticia : Noticies
+actualitza(noticies : Noticies) : void
+ProgramaMalInformat(nom : string) : ProgramaMalInformat
+mostra() : void
ProgramaTelecinco
-nom : string
-noticia : Noticies
+actualitza(noticies : Noticies) : void
+ProgramaTelecinco(nom : string) : ProgramaTelecinco
+mostra() : void
<<Interface>>
ProgramaNoticies
+mostra() : void

2.1.3. Decorator

A la nostra pràctica hem vist que el patró Decorator no encaixa gaire bé amb el que volem modelar. El motiu principal és que les coses que canvien en un jugador no s'acumulen, sinó que simplement canvien d'estat o de rol. Per exemple, un jugador pot ser extrem o pivot, però no té sentit que sigui "extrem decorat amb pivot". El que fem és substituir un rol per un altre, no afegir capes de comportament com faria un Decorator.

El mateix passa amb les sancions: un jugador pot estar amonestat, exclòs, etc., però són estats que van canviant amb el temps, no funcionalitats que s'hi puguin enganxar una sobre l'altra.
A més, el Decorator té sentit quan pots envoltar un objecte amb altres capes petites sense trencar res. En el nostre cas, Jugador està connectat amb Equip, Àrbitre, missatgeria, rols… i embolicar-lo amb decoradors faria que la identitat fos confusa i complicaria totes aquestes relacions.

Per tot això, creiem que en aquesta pràctica no és pràctic ni natural fer servir Decorator. Els canvis que volem representar es resolen millor amb Strategy i Observer, que encaixen molt més bé amb el comportament real del sistema.

Exemple patró Decorator

Aquest cas modela un sistema de reserves d'hotel on els clients poden triar un pla base (Basic, Plus o Gold) i afegir-hi extres opcionals com tot inclòs, pàrquing privat, accés a spa o vista al mar. S'utilitza el patró Decorator perquè cada extra envolta un pla existent i n'amplia el comportament (preu i descripció) sense crear subclasses per a totes les combinacions possibles. Això permet combinar lliurement serveis addicionals mantenint una estructura flexible i ampliable.

PlaBasic, PlaPlus i PlaGold són les versions base dels plans d'hotel.
Cada un defineix un nivell diferent de confort i serveis (bàsic, intermig i premium) amb el seu preu i descripció corresponents.

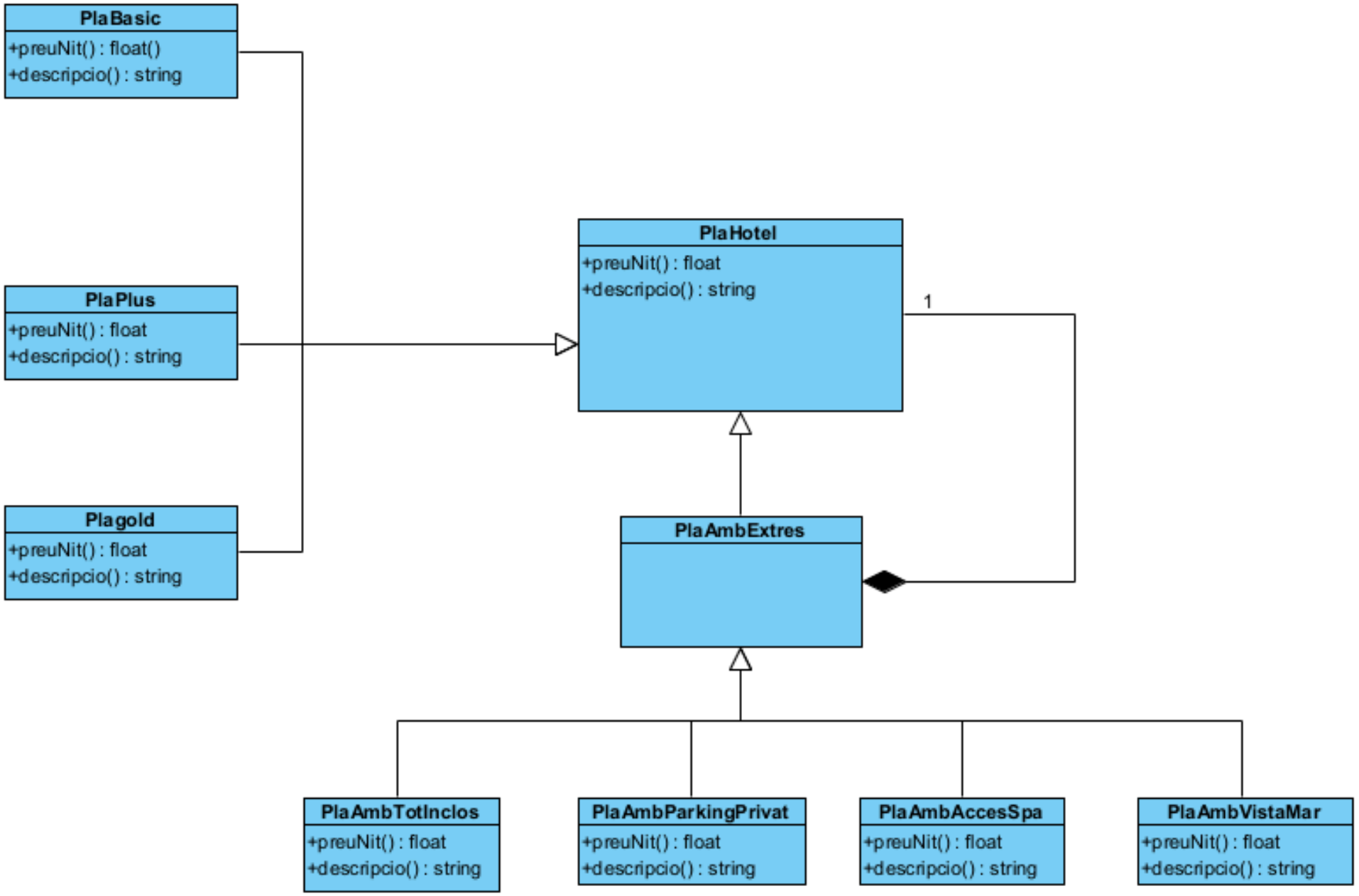
PlaAmbExtres
Decorador abstracte que també hereta de PlaHotel i conté internament un altre PlaHotel. No afegeix funcionalitat per si sol: serveix de base perquè els extres concrets puguin envoltar un pla.

PlaAmbTotInclos
Decorador concret que afegeix servei de "tot inclòs" al pla intern. Incrementa el preuPerNit() i afegeix la informació corresponent a la descripcio().

PlaAmbParkingPrivat
Decorador que suma el cost del pàrquing privat al preu del pla intern i afegeix a la descripció que el client disposa de pàrquing reservat.

PlaAmbAccesSpa
Decorador que millora o afegeix accés a spa (per exemple, zona premium). Augmenta el preu per nit i actualitza la descripció indicant aquest servei extra.

PlaAmbVistaMar
Decorador que representa que l'habitació té vista al mar. Suma un suplement al preu i modifica la descripció per reflectir-ho.


PlaBasic
+preuNit() : float()
+descripcio() : string
PlaPlus
+preuNit() : float
+descripcio() : string
PlaGold
+preuNit() : float
+descripcio() : string
PlaHotel
+preuNit() : float
+descripcio() : string
1
PlaAmbExtres
PlaAmbTotInclos
+preuNit() : float
+descripcio() : string
PlaAmbParkingPrivat
+preuNit() : float
+descripcio() : string
PlaAmbAccesSpa
+preuNit() : float
+descripcio() : string
PlaAmbVistaMar
+preuNit() : float
+descripcio() : string