UNIVERSIDAD DE MÁLAGA

Modelado y Diseño de Software

Examen Enero 2023

Nombre: _____

PARTE 1 (15 minutos)

A. Defina los siguientes conceptos [0,-2]
- Ingeniería de software
- Modelo
- Diseño de software
- Patrón de diseño

B. Defina los conceptos "Rol" y "Tipo" en modelado conceptual e indique sus principales semejanzas y diferencias [0.25]

C. Defina los conceptos de "Refactoring" y de "Bad smell" [0.25]

PARTE 2 (3 horas 45 minutos)

P1. Los cursos en las Universidades

Queremos modelar el sistema educativo de universidades que ofrecen distintas titulaciones de grado y de posgrado (másters), de acuerdo a las siguientes especificaciones.

1. Cada titulación está compuesta por un conjunto de asignaturas, cada una con un número de créditos ECTS, que se imparten todos los cursos académicos.
2. Una misma asignatura solo puede impartirse una vez cada curso.
3. Cada asignatura solo pertenece a una Universidad, aunque puede formar parte de varias de sus titulaciones (tanto de grado como de máster).
4. Cada asignatura la imparte un solo profesor, que puede variar de curso a curso.
5. Cada profesor no puede dar más de 4 asignaturas cada curso (independientemente del número de créditos que tengan).
6. Cada curso los profesores evalúan a los alumnos matriculados en las asignaturas que imparten, asignándoles una nota entre 0 y 10 (suponemos que no hay "no presentados", sino que todos los alumnos matriculados obtienen una calificación entre 0 y 10). A partir de una calificación de 5 la asignatura se considera aprobada.
7. Un mismo alumno solo puede matricularse hasta 3 veces de una misma asignatura, en diferentes cursos.
8. Un alumno solo puede aprobar una asignatura una vez, no pudiendo matricularse de asignaturas que ya ha aprobado en cursos anteriores.
9. Por simplicidad, supondremos que cada asignatura tiene solo una convocatoria por curso.
10. Todos los cursos, salvo a lo más el último, deben tener todas sus asignaturas evaluadas.
11. Los profesores no pueden matricularse en aquellas asignaturas que imparten ese curso, aunque sí en otras.
12. Para obtener un título en una Universidad es preciso tener aprobados al menos 240 créditos de las asignaturas que componen una titulación de grado, o los 60 que componen una de posgrado.
13. El título recoge el nombre de la universidad y de la titulación, así como el curso en el que el alumno aprobó la última de las asignaturas con las que completaba los créditos necesarios.
14. A lo largo de su vida, una persona puede obtener tantos títulos como titulaciones oferta una Universidad, si cursa las correspondientes asignaturas.
15. Finalmente, para poder matricularse de una asignatura que se imparta solo en posgrado, el alumno ha de estar en posesión de un título de grado (de esa universidad o de otra).

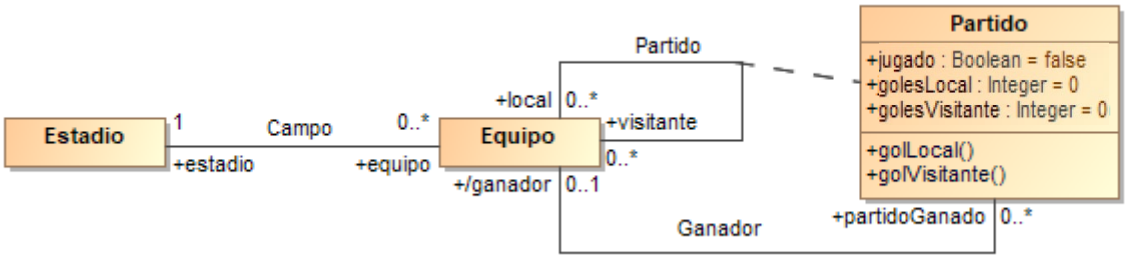
Se pide:

(a) Desarrollar en UML, utilizando la herramienta USE, un modelo conceptual de la estructura de dicho sistema, con los correspondientes elementos, las relaciones entre ellos y todas las restricciones de integridad necesarias.

(b) Modelar el comportamiento del sistema, definiendo las operaciones necesarias para modelar las siguientes acciones: (b1) un alumno se matricula un curso en una asignatura de una titulación; (b2) un profesor califica a un alumno matriculado en una asignatura que imparte un curso; (b3) un alumno solicita la expedición de su título cuando cumple las condiciones requeridas.
- Especificar en OCL, para cada una de dichas operaciones, sus pre- y post-condiciones.
- Especificar en SOIL, para cada una de dichas operaciones, su comportamiento.

P2. Los equipos de futbol

Supongamos el modelo conceptual mostrado a continuación, que representa una liga de futbol en donde los equipos se enfrentan entre ellos en partidos. Cada equipo tiene un estadio, y en cada partido juega el papel de equipo local o de equipo visitante.



Los atributos de la clase de asociación *Partido* indican si el partido se ha terminado de jugar o no (inicialmente su valor es *false*), y los goles que ha marcado cada equipo (incialmente 0). La asociación Ganador es derivada, y se establece cuando un partido ha sido jugado y uno de los dos equipos ha ganado. Las restricciones que debe cumplir tal asociación son las siguientes:

context Partido inv LocalGanador:
self.jugado and self.golesLocal>self.golesVisitante implies self.ganador = self.local
context Partido inv VisitanteGanador:
self.jugado and self.golesLocal<self.golesVisitante implies self.ganador = self.visitante
context Partido inv SinGanador:
not self.jugado or (self.jugado and self.golesLocal=self.golesVisitante) implies self.ganador->isEmpty()

Además, la clase Partido debe cumplir los siguientes requisitos:

context Partido inv EquiposDistintos: self.local<>self.visitante
context Partido inv GolesOK: self.golesLocal>=0 and self.golesVisitante>=0

Los goles que van marcando los equipos pueden solo actualizarse incrementándose de uno en uno mediante las operaciones golLocal() y golVisitante() de la clase Partido. Una vez el atributo jugado de la clase de asociación Partido se ha puesto a true, no se permite incrementar el número de goles de ninguno de los dos equipos en el partido.

Se pide:

a) Convertir el modelo conceptual de la liga de fútbol en un modelo de diseño UML que permita la implementación, en un lenguaje orientado a objetos como Java, de las entidades, asociaciones, restricciones y operaciones correspondientes. Incluir también todas las pre- y post-condiciones de las operaciones. El modelo puede ser realizado con cualquier herramienta de modelado UML, por ejemplo USE o Papyrus. En caso de utilizar USE, que no permite especificar la visibilidad de los atributos, operaciones y extremos de operación, se deberá indicar explícitamente de alguna forma en el texto de la memoria la visibilidad de todos estos elementos del modelo de diseño.
b) Desarrolle una implementación en Java correspondiente a dicho modelo de diseño, incluyendo las operaciones, que incorpore también el código de andamiaje para crear objetos y relaciones entre ellos garantizando en todo momento la coherencia de los objetos, las relaciones de la implementación, y también las restricciones de integridad del modelo y de sus operaciones (incluyendo sus correspondientes pre- y post-condiciones).

- **Examen reducido**: Problemas P1(a) y P2(b).
- **Puntuaciones**: Parte 1: 0.5; Parte 2: P1 (4.5+2) P2 (1+2) .
- **Entrega**: La entrega se hará a través del campus virtual, en un solo archivo **en formato PDF**, que contendrá una memoria explicativa del examen y en donde se describirán las soluciones propuestas para cada problema y se incluirán todas las imágenes con los diagramas UML, así como todos los códigos en Java y USE desarrollados.

SOLUCIONES

A. Defina los siguientes conceptos [0,-2]

- La *ingeniería de software* es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.
- *Modelo*: Una representación o especificación de un sistema, desde un determinado punto de vista y con un objetivo concreto.
- *Diseño*: Conjunto de planes y decisiones para definir un producto con los suficientes detalles como para permitir su realización física de acuerdo a unos requisitos
- *Patrón de Diseño*: Una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de software.

B. Defina los conceptos "Rol" y "Tipo" en modelado conceptual e indique sus principales semejanzas y diferencias [0.25]

Rol: Nombre que se le da a un participante en una relación, o bien, especificación de la estructura y comportamiento de un participante en una relación.

Tipo: predicado lógico que caracteriza un conjunto de entidades

Diferencias:

- Un tipo es en general un predicado lógico, mientras que un rol es o bien un nombre, o bien la especificación de un comportamiento.
- Un rol no puede existir sin una relación, es decir, es un ciudadano de segunda clase, mientras que los tipos son ciudadanos de primera clase que pueden definirse sin depender de ningún otro elemento.

Semejanzas:

- En UML, ambos pueden representantes mediante clases.
- Ambos se pueden utilizar para identificar la estructura y/o el comportamiento de entidades

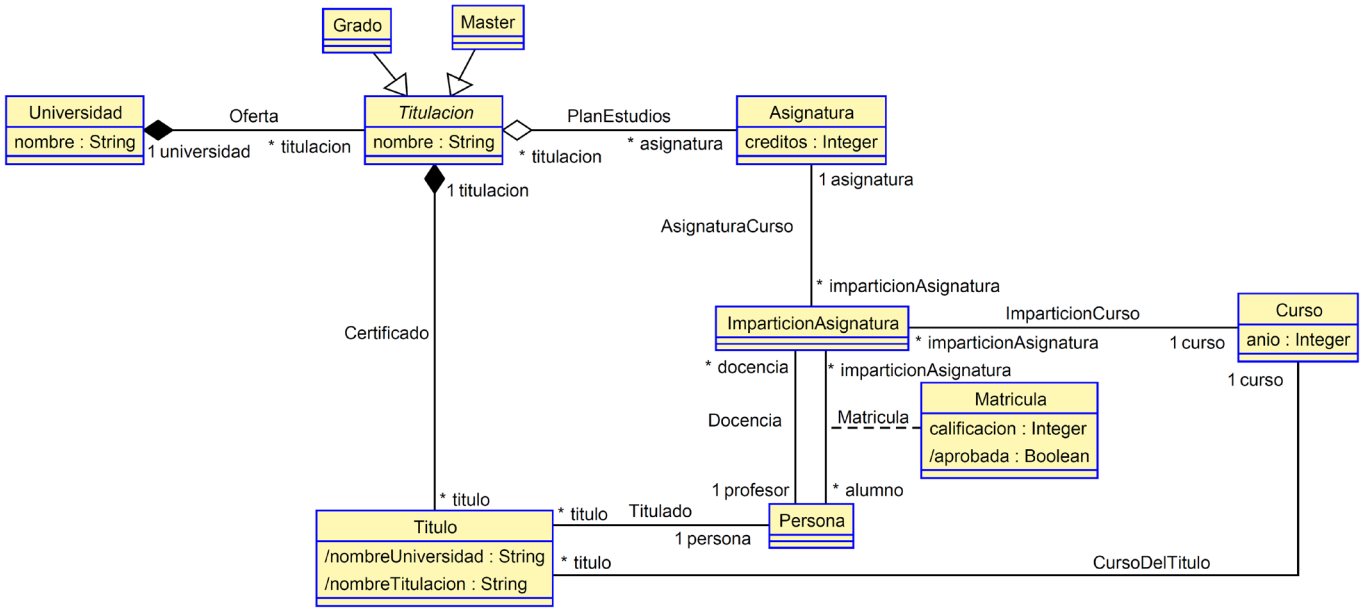
C. Defina los conceptos de "Refactoring" y de "Bad smell" [0.25]

Refactoring es el proceso que consiste en cambiar un sistema software sin alterar su comportamiento externo pero mejorando su estructura interna (the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure [Fowler09]).

Bad smell: Fragmentos de código que contienen algún tipo de error de diseño. O bien fragmentos de código que estan mal en algún sentido, y que parecen feos. (Fragments of code that contain some design mistake. Pieces of code that are wrong (in some sense) and that are ugly to see. [Fowler09])

P1. Las Universidades

Un posible modelo del sistema pedido viene representado por el siguiente diagrama de clases en UML



El correspondiente código USE, que también contiene las 15 restricciones de integridad expresadas en el enunciado, es el siguiente:

```
model Unversidad

class Universidad
  attributes nombre:String
end
abstract class Titulacion
  attributes nombre:String
end
class Grado < Titulacion
end
class Master < Titulacion
end
composition Oferta between -- Requisito #3
    Universidad [1]
    Titulacion [*]
end
aggregation PlanEstudios between -- Requisito #1
    Titulacion [*]
    Asignatura [*]
end
class Asignatura -- Requisito #1
  attributes
    creditos:Integer init: 6
end
class ImparticionAsignatura
end
association ImparticionCurso between -- Requisito #2
    ImparticionAsignatura [*]
    Curso [1]
end
```

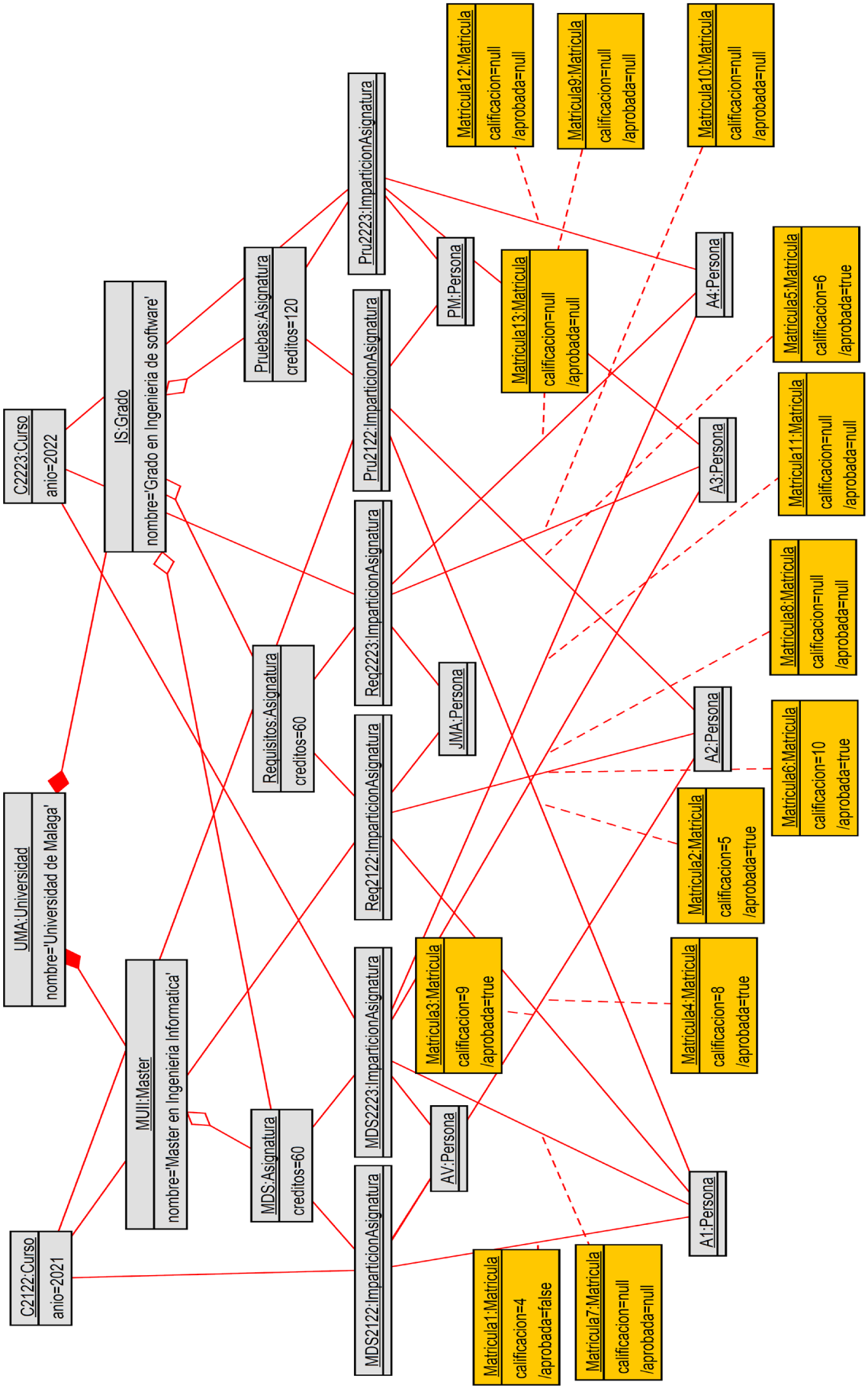
```
association AsignaturaCurso between -- Requisito #1
    Asignatura [1] -- Materializacion
    ImparticionAsignatura [*]
end
class Curso
  attributes
   anio:Integer
end
class Titulo
  attributes
    nombreUniversidad:String derive: self.titulacion.universidad.nombre
    nombreTitulacion:String derive: self.titulacion.nombre
end
composition Certificado between
    Titulacion [1]
    Titulo [*]
end
association Titulado between -- Requisito #14
    Titulo [*]
    Persona [1]
end
association CursoDelTitulo between
    Titulo [*]
    Curso [1]
end
associationclass Matricula between -- Requisito #6
    ImparticionAsignatura [*]
    Persona [*] role alumno
  attributes
    calificacion : Integer
    aprobada : Boolean derive: self.calificacion>=5
end
association Docencia between -- Requisito #4
    ImparticionAsignatura [*] role docencia
    Persona [1] role profesor
end
class Persona
end

constraints
-- Requisito #5. Cada profesor no puede dar más de 4 asignaturas cada curso (independientemente del
número de créditos que tengan).
context Persona inv DocenciaLimitada:
    Curso.allInstances->forAll (c |
       Persona.allInstances->forAll(p|p.docencia->select(ia|ia.curso=c)->size() <= 4))
-- Requisito #6. Las notas son o null o un entero entre 0 y 10
context Matricula inv ValorCorrecto:
    self.calificacion<>null implies (self.calificacion>=0 and self.calificacion<=10)
-- Requisito #7. Un alumno solo puede matricularse hasta 3 veces de una misma asignatura en cursos
distintos
```

```
context Persona inv Solo3Convocatorias:
    let asigs:Bag(Asignatura) = self.imparticionAsignatura.asignatura in
    asigs->forAll(a1|asigs->select(a|a=a1)->size()<=3)
-- Requisito #8. Un alumno solo puede aprobar una asignatura una vez, no pudiendo matricularse de
asignaturas que ya ha aprobado en cursos anteriores.
context Persona inv SoloUnaConvocatoriaAprobada:
    let asigs:Bag(Asignatura) = self.imparticionAsignatura.asignatura in
        asigs->forAll(a|self.matricula->select(m
              |m.aprobada and m.imparticionAsignatura.asignatura=a)->size()<=1)
    and
    self.matricula->forAll(m1,m2|m1<>m2 implies
        ((m1.imparticionAsignatura.asignatura=m2.imparticionAsignatura.asignatura and m1.aprobada)
            implies (not m2.aprobada and m1.imparticionAsignatura.curso.anio >
                                        m2.imparticionAsignatura.curso.anio)))
-- Requisito #9. dos imparticiones de una asignatura no pueden coincidir en un mismo curso
context Asignatura inv UnaImparticionPorCurso:
    self.imparticionAsignatura->forAll(i1,i2|i1<>i2 implies i1.curso.anio<>i2.curso.anio)
-- Requisito #10. Todos los cursos, salvo a lo más el último, deben tener todas sus asignaturas
evaluadas.
context Curso inv TodasLasAsignaturasEvaluadas:
    let maxAnio : Integer = Curso.allInstances->collect(anio)->max() in
    self.anio<>maxAnio implies self.imparticionAsignatura.matricula->forAll(m|m.calificacion<>null)
-- Requisito #11. Los profesores no pueden matricularse en aquellas asignaturas que imparten ese curso,
aunque sí en otras.
context Persona inv NoSeDaClaseASiMismo:
    self.docencia->forAll(i|i.alumno->excludes(self))
-- los cursos comienzan en años distintos
context Curso inv AniosDistintos:
    Curso.allInstances->forAll(c1,c2|c1<>c2 implies c1.anio<>c2.anio)
-- Requisito #12. El titulo tiene que tener los creditos aprobados. Junto con:
-- Requisito #13. La fecha del titulo tiene que ser posterior o igual a las de aprobar las asignaturas
correspondientes.
-- Para eso comprobamos que el curso expresado en el titulo el alumno tiene aprobados mas de los
creditos correspondientes.
-- Ojo que la fecha no puede ser derivada porque el alumno puede matricularse de mas asignaturas de esa
titulacion
-- posteriormente a la obtencion del titulo
context Titulo inv FechaCorrecta:
    let imparticionAsignaturasAprobadas:Set(ImparticionAsignatura) =
      self.persona.matricula->select(aprobada)->collect(imparticionAsignatura)
      ->select(ia|ia.curso.anio<=self.curso.anio and
                  self.titulacion.asignatura->includes(ia.asignatura))->asSet() in
    imparticionAsignaturasAprobadas->collect(asignatura.creditos)->sum() >=
       if self.titulacion.oclIsTypeOf(Master) then 60 else 240 endif

-- Requisito #15. si esta matriculado de una asignatura de master, tiene que tener un titulo de grado
-- Ojo que puede haber asignaturas de grado y de master
context Persona inv MatriculaMasterConTituloGrado:
    self.imparticionAsignatura.asignatura->select(a|
       a.titulacion->forAll(t|t.oclIsTypeOf(Master)))->notEmpty() implies
              self.titulo->select(t|t.titulacion.oclIsTypeOf(Grado))->notEmpty()
```

A modo de ejemplo, el siguiente fichero SOIL describe una serie de acciones para crear un modelo del sistema como se muestra en el diagrama de objetos:



```
reset
!new Universidad('UMA')
!UMA.nombre := 'Universidad de Malaga'
!new Grado('IS')
!IS.nombre := 'Grado en Ingenieria de software'
!new Master('MUII')
!MUII.nombre := 'Master en Ingenieria Informatica'
!new Asignatura('MDS')
!MDS.nombre := 'Modelado y Diseño de Software'
!MDS.creditos:=60
!new Asignatura('Pruebas')
!Pruebas.creditos:=120
!new Asignatura('Requisitos')
!Requisitos.creditos:=60
!insert (UMA,IS) into Oferta
!insert (UMA,MUII) into Oferta
!insert (MUII,MDS) into PlanEstudios
!insert (IS,MDS) into PlanEstudios
!insert (IS,Pruebas) into PlanEstudios
!insert (IS,Requisitos) into PlanEstudios
!new ImparticionAsignatura('MDS2122')
!new ImparticionAsignatura('MDS2223')
!new ImparticionAsignatura('Pru2122')
!new ImparticionAsignatura('Pru2223')
!new ImparticionAsignatura('Req2122')
!new ImparticionAsignatura('Req2223')
!insert (MDS,MDS2122) into AsignaturaCurso
!insert (MDS,MDS2223) into AsignaturaCurso
!insert (Pruebas,Pru2122) into AsignaturaCurso
!insert (Pruebas,Pru2223) into AsignaturaCurso
!insert (Requisitos,Req2122) into AsignaturaCurso
!insert (Requisitos,Req2223) into AsignaturaCurso
!new Curso('C2122')
!C2122.anio:=2021
!new Curso('C2223')
!C2223.anio:=2022
!insert (MDS2122,C2122) into ImparticionCurso
!insert (Pru2122,C2122) into ImparticionCurso
!insert (Req2122,C2122) into ImparticionCurso
!insert (MDS2223,C2223) into ImparticionCurso
!insert (Pru2223,C2223) into ImparticionCurso
!insert (Req2223,C2223) into ImparticionCurso
!new Persona('AV')
!new Persona('JMA')
!new Persona('PM')
!new Persona('A1')
!new Persona('A2')
!new Persona('A3')
!new Persona('A4')
!insert (MDS2122,AV) into Docencia
!insert (MDS2223,AV) into Docencia
!insert (Pru2122,PM) into Docencia
!insert (Pru2223,PM) into Docencia
!insert (Req2223,JMA) into Docencia
!insert (Req2122,JMA) into Docencia
!insert (MDS2122,A1) into Matricula
!Matricula1.calificacion:=4
!insert (Pru2122,A1) into Matricula
!Matricula2.calificacion:=5
!insert (Req2122,A1) into Matricula
!Matricula3.calificacion:=9
!insert (MDS2122,A2) into Matricula
!Matricula4.calificacion:=8
!insert (Pru2122,A2) into Matricula
!Matricula5.calificacion:=6
!insert (Req2122,A2) into Matricula
!Matricula6.calificacion:=10
!insert (MDS2223,A1) into Matricula
!insert (MDS2223,A3) into Matricula
!insert (Pru2223,A3) into Matricula
!insert (Req2223,A3) into Matricula
```

```
!insert (MDS2223,A4) into Matricula
!insert (Pru2223,A4) into Matricula
!insert (Req2223,A4) into Matricula
check
```

Apartado b) La especificación de las operaciones que se piden es la siguiente

(b1) un alumno se matricula un curso en una asignatura de una titulación. Para ello se define la siguiente operación en la clase Universidad:

```
matricular(t:Titulacion, a: Asignatura, c:Curso, p:Persona)
begin
    declare i:ImparticionAsignatura;
    i:=c.imparticionAsignatura->select(i|i.asignatura = a)->any(true);
    insert (i,p) into Matricula;
end
pre TitulacionOK: self.titulacion->includes(t)
pre AsignaturaOK: t.asignatura->includes(a)
pre CursoOK: c.imparticionAsignatura->select(i|i.asignatura = a)->notEmpty()
pre PersonaOK1: -- no es ni profesor ni alumno de esa impartición de asignatura
    let ia:ImparticionAsignatura = c.imparticionAsignatura->select(i|i.asignatura = a)->any(true) in
    ia.profesor<>p and ia.alumno->excludes(p)
pre PersonaOK2: -- no la tiene aprobada antes
    a.imparticionAsignatura.matricula->select(m|m.alumno=p and m.aprobada)->isEmpty()
pre TieneTituloGradoSiPosgrado:
    t.oclIsTypeOf(Master) implies p.titulo->select(t|t.titulacion.oclIsTypeOf(Grado))->notEmpty()
post Matriculado:
    let ia:ImparticionAsignatura = c.imparticionAsignatura->select(i|i.asignatura = a)->any(true) in
    ia.alumno->includes(p)
```

(b2) un profesor califica a un alumno matriculado en una asignatura que imparte un curso. Para ello se define la siguiente operación en la clase Matricula:

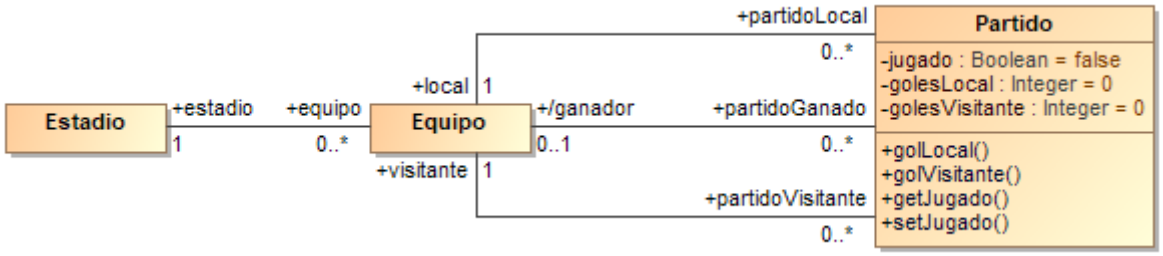
```
calificar(p:Persona,a:Persona, nota:Integer)
begin
    self.calificacion:=nota;
end
pre Matriculado: self.imparticionAsignatura.profesor=p and self.alumno->includes(a)
pre NotaOK: nota>=0 and nota<=10
pre NoCalificadoYa: self.calificacion=null
post Calificado: self.calificacion=nota
```

(b3) un alumno solicita la expedición de su título cuando cumple las condiciones requeridas. Para ello se define la siguiente operación en la clase Universidad:

```
expedirTitulo(t:Titulacion, p:Persona, c:Curso)
begin
    declare ti:Titulo;
    ti:=new Titulo();
    insert(t,ti) into Certificado;
    insert(ti,p) into Titulado;
    insert(ti,c) into CursoDelTitulo;
end
pre TitulacionOK: self.titulacion->includes(t)
pre NolotieneYa: p.titulo->select(ti|ti.titulacion=t)->isEmpty()
pre TieneCreditos:
    let imparticionAsignaturasAprobadas:Set(ImparticionAsignatura) =
        p.matricula->select(m|m.calificacion>=5)->collect(imparticionAsignatura)->
          select(ia|ia.curso.anio<=c.anio and t.asignatura->includes(ia.asignatura))->asSet() in
    imparticionAsignaturasAprobadas->collect(asignatura.creditos)->sum()
          >= if t.oclIsTypeOf(Master) then 60 else 240 endif
post: self.titulacion.titulo->one(ti|ti.curso=c and ti.persona=p)
```

P2. Los equipos de futbol

a) El modelo de diseño es el siguiente:



Las expresiones OCL que definen las restricciones de integridad, además de las expresadas por las multiplicidades del modelo, son:

Context Partido inv NoDuplicados: -- semántica de la clase de asociación frente a la de una clase normal de relación
Partido.allInstances->forAll(p1,p2 | p1<>p2 implies (p1.local<>p2.local or p1.visitante<>p2.visitante)

context Partido inv EquiposDistintos: self.local<>self.visitante

context Partido inv GolesOK: self.golesLocal>=0 and self.golesVisitante>=0

context Partido::ganador : Equipo derive:
(self.jugado and self.golesLocal>self.golesVisitante implies self.local) and
(self.jugado and self.golesLocal<self.golesVisitante implies self.visitante)

context Partido::golLocal()
pre: not self.jugado
post: self.golesLocales = self.golesLocales@pre + 1

context Partido::golVisitante()
pre: not self.jugado
post: self.golesVisitante = self.golesVisitante@pre + 1

Aunque podríamos haber dejado el atributo jugado con una visibilidad “públic”, para indicar que su traducción a Java consistía en convertirlo a privado e incluir sus correspondientes getter y setter, lo hemos indicado explícitamente en el modelo de diseño para poder incluir una precondición en el setter (el atributo jugado solo puede cambiarse una vez).

context Partido::getJugado() = self.jugado

context Partido::setJugado()
pre: not self.jugado
post: self.jugado

b) La implementación en Java correspondiente al modelo de diseño anterior, es la siguiente.

```
//-------------------
// CLASS Estadio
//-------------------

import java.util.LinkedList;
import java.util.List;

public class Estadio {

	private List<Equipo> equipo;

	public Estadio() {
		equipo = new LinkedList<Equipo>();
	}
```

```
	void addEquipo(Equipo e) {
		assert (e!=null);
		equipo.add(e);
	}
	void rmEquipo(Equipo e) {
		assert (e!=null);
		equipo.remove(e);
	}
}

//-------------------
// CLASS Equipo
//-------------------

import java.util.Enumeration;
import java.util.LinkedList;
import java.util.List;
import java.util.Collection;

public class Equipo {

	private Estadio estadio;
	private List<Partido> partidoGanado = new LinkedList<Partido>();
	private List<Partido> partidoLocal = new LinkedList<Partido>();
	private List<Partido> partidoVisitante = new LinkedList<Partido>();

	public Estadio getEstadio() {
		return this.estadio;
	}

	public void setEstadio(Estadio e) {
		assert (e!=null);
		if (this. estadio!= null)
			this.estadio.rmEquipo(this);
		this.estadio = e;
		e.addEquipo(this);
	}
	public Equipo(Estadio e) {
		this.setEstadio(e);
	}
	void addPartidoLocal(Partido p) {
		assert (p!=null):
		this.partidoLocal.add(p);
	}
	void addPartidoVisitante(Partido p) {
		assert (p!=null):
		this.partidoVisitante.add(p);
	}
	void addPartidoGanado(Partido p) {
		assert (p!=null):
		this.partidoGanado.add(p);
	}
	public Enumeration<Partido> getPartidoLocal() {
		return java.util.Collections.enumeration(this.partidoLocal);
	}
	public Enumeration<Partido> getPartidoVisitante() {
		return java.util.Collections.enumeration(this.partidoVisitante);
	}
	public Enumeration<Partido> getPartidoGanado() {
		return java.util.Collections.enumeration(this.partidoGanado);
	}
}

//-------------------
// CLASS Partido
//-------------------

import java.util.Enumeration;

public class Partido {
```

```
	private boolean jugado = false;
	private int golesLocal = 0;
	private int golesVisitante = 0;
	private Equipo local;
	private Equipo visitante;
	private Equipo ganador;

	// Constructors
	public Partido(Equipo local, Equipo visitante) {
		assert (local!=null);
		assert (visitante!=null);
		assert (this.noDups(local,visitante));
		assert (local!=visitante);

		this.setLocal(local);
		this.setVisitante(visitante);
		local.addPartidoLocal(this);
		visitante.addPartidoVisitante(this);
	}

	// Accessors for attribute "local"
	public Equipo getLocal() { return this.local; }

	private void setLocal(Equipo e) {
		assert (e!=null);
		this.local=e;
	}

	// Accessors for attribute "visitante"
	public Equipo getVisitante() { return this.visitante; }

	private void setVisitante(Equipo e) {
		assert (e!=null);
		this.visitante=e;
	}

	// Accessors for attribute "jugado"
	public boolean getJugado() {return this.jugado; }

	public void setJugado() {
		assert (!this.jugado);
		this.jugado=true;
		if (this.golesLocal > this.golesVisitante) {
			this.ganador = this.getLocal();
			this.ganador.addPartidoGanado(this);
		}
		if (this.golesLocal < this.golesVisitante) {
			this.ganador = this.getVisitante();
			this.ganador.addPartidoGanado(this);
		}
	}
	// methods
	public void golLocal() {
		assert (!this.jugado);
		this.golesLocal++;
	}
	public void golVisitante() {
		assert (!this.jugado);
		this.golesVisitante++;
	}
	// auxiliary private methods
	private boolean noDups(Equipo l, Equipo v) {
		Enumeration<Partido> a = l.getPartidoLocal();
		while (a.hasMoreElements()) {
			if (a.nextElement().getVisitante() == v) return false;
		};
		return true;
	}

}
```