13019 - Diseño de bases de datos *Curso 2003-2004*
Fecha: *9 de julio de 2004*

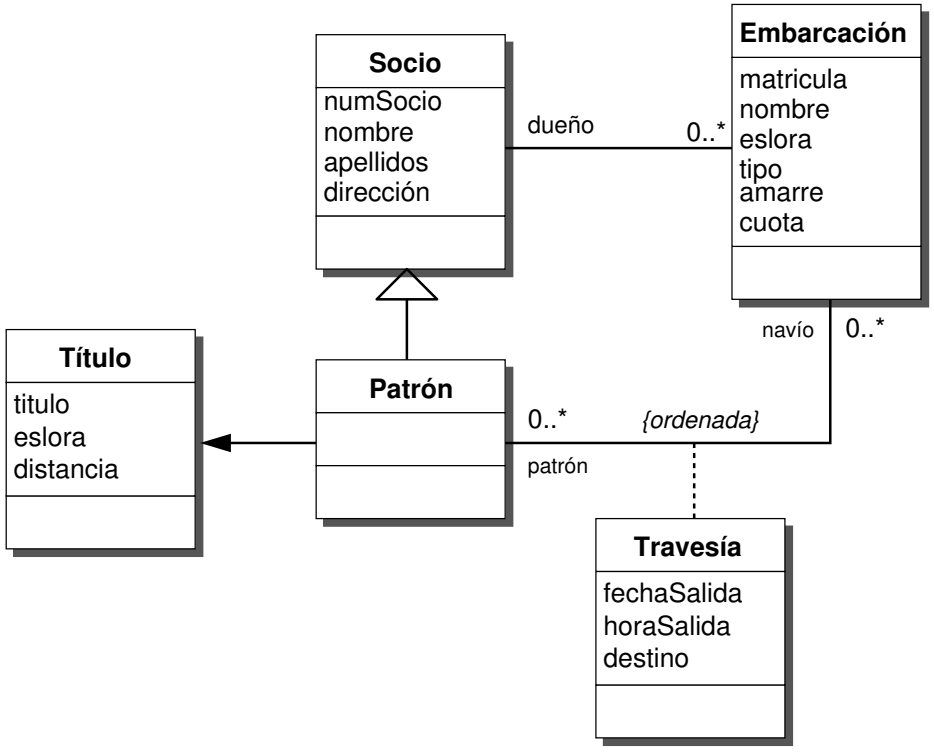
Nombre: _____ Apellidos: _____

1. (*2.00 puntos*) El club naútico **Saplatja** desea implantar un sistema de gestión de sus **socios** y de las **embarcaciones** con las siguientes características:
   - De cada socio se guardan los datos personales: número de socio, nombre y apellidos, dirección, etc...
   - También se deben guardar los datos del barco o barcos que posee cada socio: número de matrícula, nombre, eslora, tipo de embarcación (vela o motor), número del amarre y cuota que paga por el mismo.
   - Además se quiere mantener información (ordenada) sobre las **travesías** realizadas por cada embarcación, con indicación de la fecha y hora de salida, el destino y los datos personales del **patrón**. Este último no tiene por qué ser propietario del barco o socio, pero es necesario que disponga de un título naútico compatible con las características de la embarcación y el destino (ver tabla adjunta).

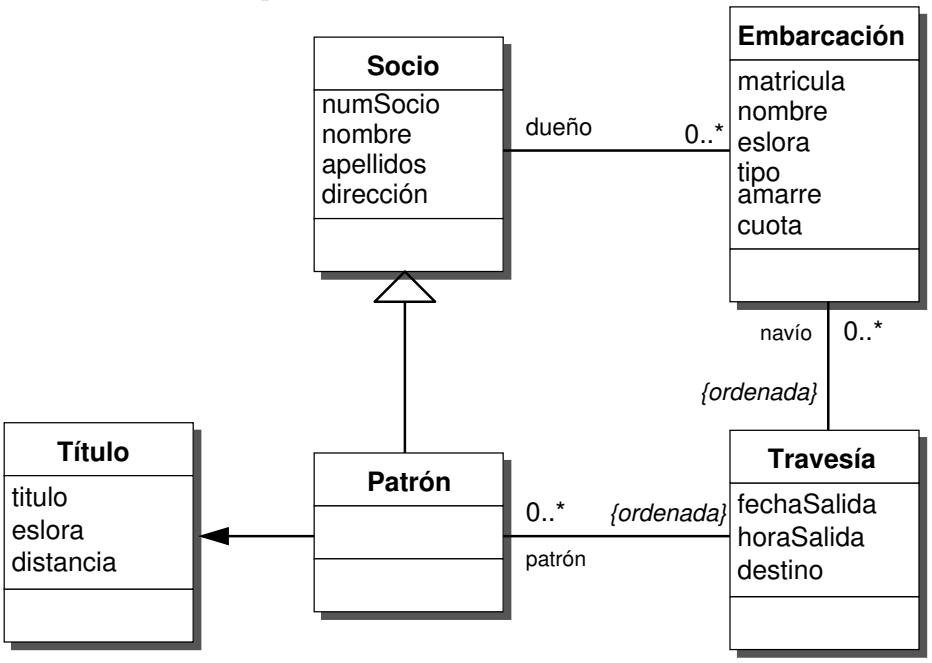
   De acuerdo con el sistema descrito, desarrolle el sistema de clases UML.

Título	Eslora	Distancia
Patrón para navegación básica	Vela: 8m, Motor: 6m.	4 millas de abrigo o playa accesible
Patrón de embarcaciones de recreo	12 m	12 millas e interinsular
Patrón de Yate	20 m.	60 millas
Capitán de Yate	Sin límite	Sin límite

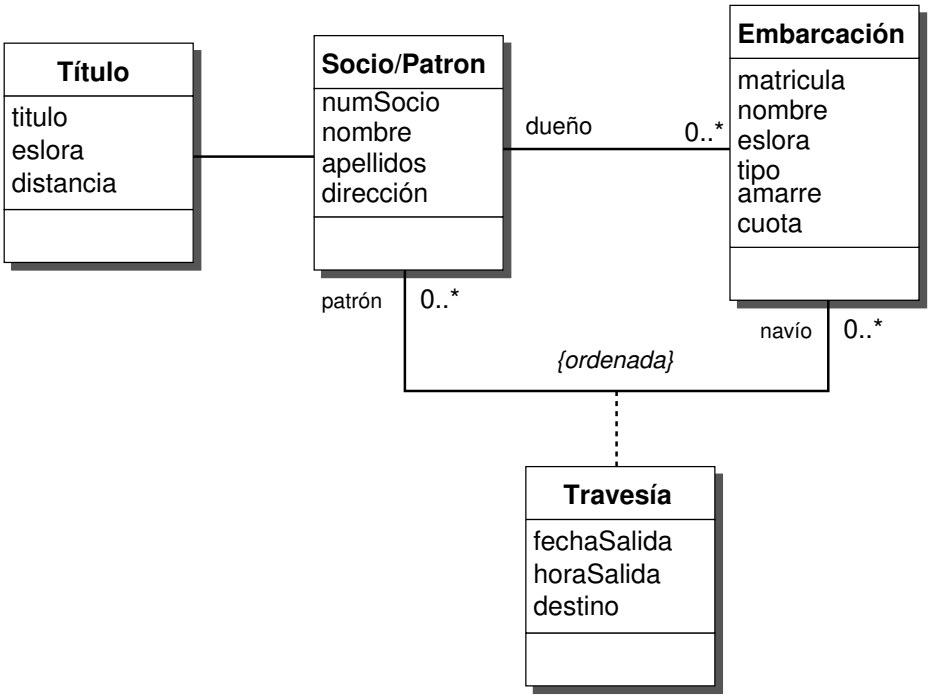
- Un posible diagrama UML que se ajusta a la descripción es:



- Para la implementación orientada a objetos, es más adecuado este diagrama, en donde la clase de asociación se ha promovido a clase.



- En la implementación relacional es más adecuado eliminar la herencia, como se muestra en esta variante del diagrama:



13019 - Diseño de bases de datos *Curso 2003-2004*
Fecha: *9 de julio de 2004*

Nombre: _____ Apellidos: _____

2. (*3.50 puntos*) Partiendo del sistema descrito en la pregunta anterior y de su digrama UML, conteste a las siguientes cuestiones:

   (a) (*1.00 puntos*) Escriba, de acuerdo con el estándar ODMG/C++, la declaración de los atributos y relaciones.

   (b) (*0.25 puntos*) Implemente el método void Socio::ficha(void) const que escriba en la salida estándar una ficha con todos los datos relativos a un socio y la embarcación o embarcaciones con las que cuenta.

   (c) (*0.25 puntos*) Implemente el método void Embarcacion::ficha(void) const que muestre en la salida estándar todos los datos relativos a una embarcación.

   (d) (*0.50 puntos*) Implemente el método string Embarcacion::titulo(void) const que devuelva la titulación mínima necesaria para patronear una determinada embarcación.

   (e) (*0.50 puntos*) Implemente el método bool Embarcacion::autorizada(const string titulo, const float distancia) const que, dados un patron (caracterizado por su título) y una travesía (caracterizada por su distancia al club), determine si puede o no autorizarse la salida de puerto.

   (f) (*0.50 puntos*) Implemente el método void Embarcacion::travesias(void) const que muestre la lista ordenada de las travesías realizadas por una embarcación junto con los datos del patrón (nombre, apellidos y título).

   (g) (*0.50 puntos*) Implemente el método void Patron::embarcaciones(void) const que muestre la lista de todas la embarcaciones con las que el patrón ha realizado al menos una travesía. Note que el método **no** debe mostrar embarcaciones duplicadas.

   (h) (*0.00 puntos*) Describa las diferencias que existen entre virar y trasluchar.

(a) Declaración de atributos y métodos

```
#include <iostream>
#include <odmg.h>
class Socio : public d_Object {
  public:
  string numSocio;
  string nombre;
  string apellidos;
  string direccion;
  d_Rel_Set<Embarcacion,"dueno"> embarcaciones;
  void ficha(void) const;
}
class Patron : public Socio {
  public:
  d_Ref<Titulo> titulo;
  d_Rel_List<Travesia,"patron"> travesias;
  void embarcaciones(void) const;
}
class Titulo : public d_Object {
  public:
  string titulo;
  double slora;
  double distancia;
}
class Embarcacion : public d_Object {
  public:
  string matricula;
```

```
  string nombre;
  double slora;
  double tipo;
  double amarre;
  double cuota;
  d_Rel_List<Travesia,"navio"> travesias;
  d_Rel_Ref<Socio,"embarcaciones"> dueno
  void ficha(void) const
  string titulo(void) const
  bool autorizada(const string titulo, const float distancia) const
  void travesias(void) const
}
class Travesia : public d_Object {
  public:
  d_Date fechaSalida;
  d_Time horaSalida;
  double destino;
  d_Rel_Ref<Embarcacion,"travesias"> navio;
  d_Rel_Ref<Patron,"travesias"> patron;
}
```

(b) void Socio::ficha(void) const.

```
void Socio::ficha(void) const {
  d_Iterator<d_Ref<Embarcacion>> ie;
  d_Ref<Embarcacion> unBarco;

  cout << "Socio numero " << numSocio << endl;
  cout << "Nombre: " << apellidos << ", " << nombre << endl;
  cout << "Dirección: " << direccion << endl;
  cout << "Embarcaciones:" << endl;
  cout << "Matricula  Nombre          Amarre  Cuota\n";
  cout << "---------- -------------- ------- ------\n";
  for (ie = embarcaciones.create_iterator();
       ie.not_done();
       ie.advance()) {
         unBarco = ie.get_element();
         cout << unBarco->matricula << unBarco->nombre <<
                 unBarco->amarre << unBarco->cuota << endl;
  }
  return;
}
```

(c) void Embarcacion::ficha(void) const.

```
void Embarcacion::ficha(void) const {
  cout << "Matricula: " << matricula << endl;
  cout << "Nombre: " << nombre << endl;
  cout << "Dueño: " << dueno->nombre << " " << dueno->apellidos
       << " (" << dueno->numSocio << ")" << endl;
  cout << "Eslora: " << eslora << ", Tipo: " << tipo << endl;
  cout << "Amarre: " << amarre << ", Cuota: " << cuota << endl;
}
```

(d) string Embarcacion::titulo(void) const.

```
string Embarcacion::titulo(void) const {
  if (eslora > 20.0)
     return "Capitán de Yate";
  else if (eslora > 12.0)
     return "Patrón de Yate";
  else if ( (eslora > 8.00) || (eslora > 6.0 && tipo == "Motor") )
     return "Patrón de embarcaciones de recreo";
  else
     return "Patrón para navegación básica";
}
```

(e) bool Embarcacion::autorizada(const string titulo, const float distancia) const.

(f) void Embarcacion::travesias(void) const.

```
void Embarcacion::travesias(void) const {
  d_Iterator<d_Ref<Travesia>> it;
  d_Ref<Travesia> unaTravesia;

  for (it = travesias.create_iterator();
       it.not_done();
       it.advance()) {
         unaTravesia = it.get_element();
         cout << "Destino: " << unaTravesia->destino << endl;
         cout << unaTravesia->fechaSalida << ", "
              << unaTravesia->horaSalida << endl;
         cout << "Destino: " << unaTravesia->destino << endl;
         cout << "Patrón: " << unaTravesia->patron->nombre << ","
              << unaTravesia->patron->apellidos << "("
              << unaTravesia->patron->titulo << ")" << endl:
  }
  return;
}
```

(g) void Patron::embarcaciones(void) const.

```
void Patron::embarcaciones(void) const {
  d_Iterator<d_Ref<Travesia>> it;
  d_Ref<Travesia> unaTravesia;
  d_Set<d_Ref<Embarcaciones>> misBarcos;
  d_Iterator<d_Ref<Embarcaciones>> ie;
  d_Ref<Embarcacion> unBarco;

  for (it = travesias.create_iterator();
       it.not_done();
       it.advance()) {
         unaTravesia = it.get_element();
         unBarco = unaTravesia->navio;
         if (! misBarcos.contains_element(unBarco))
            misBarcos.insert_element(unBarco);
  }
  for (ie = misBarcos.create_iterator();
       ie.not_done();
       ie.advance()) {
         unBarco = ie.get_element();
         cout << unBarco->matricula << " " << unBarco->nombre << endl;
  }
  return;
}
```

13019 - Diseño de bases de datos *Curso 2003-2004*
Fecha: *9 de julio de 2004*

Nombre: _____ Apellidos: _____

3. (*3.50 puntos*) Partiendo del sistema descrito en la primera pregunta y de su digrama UML, conteste a las siguientes cuestiones:
   (a) (*1.50 puntos*) Proyecte el diagrama de objetos en el modelo relacional (u objeto–relacional).
   (b) (*0.50 puntos*) Escriba una consulta SQL que muestre los datos relativos a un socio y la embarcación o embarcaciones (matrícula y nombre) con las que cuenta.
   (c) (*0.50 puntos*) Escriba una consulta SQL que, dada una embarcación caracterizada por su número de matrícula, muestre los datos relativos al propietario (socio) y el amarre.
   (d) (*0.50 puntos*) Escriba una consulta SQL que muestre todos las travesías realizadas por una embarcación (fecha y hora de salida y destino) junto con los datos del patrón asociados a dicha travesía (nombre y apellidos y título naútico).
   (e) (*0.50 puntos*) Escriba una consulta SQL que muestre la lista sin duplicados de todas la embarcaciones con las que un determinado patrón ha realizado alguna travesía.
   (f) (*0.00 puntos*) Comente la frase *"la única cuerda que hay en un barco es la del reloj"*.

(a) Implementación relacional.

```
CREATE TABLE Titulo (
  idTitulo  INTEGER PRIMARY KEY,
  tit       VARCHAR2(20),
  eslora    REAL,
  distancia REAL
);
CREATE TABLE Persona (
  idPersona INTEGER PRIMARY KEY,
  numSocio  VARCHAR2(20),
  nombre    VARCHAR2(100),
  apellidos VARCHAR2(100),
  direccion VARCHAR2(100)
  idTitulo INTEGER,
  FOREIGN KEY (idTitulo) REFERENCES Titulo
);
CREATE TABLE Embarcacion (
  idBarco   INTEGER PRIMARY KEY,
  matricula VARCHAR2(20),
  nombre    VARCHAR2(100),
  eslora    REAL,
  tipo      VARCHAR2(10),
  amarre    VARCHAR2(10),
  cuota     REAL,
  idDueno   INTEGER,
  FOREIGN KEY (idDueno) REFERENCES Persona
        ON DELETE CASCADE
);
CREATE TABLE Travesia (
  idTravesia  INTEGER PRIMARY KEY,
  fechaSalida DATE,
  horaSalida  DATE,
  destino     REAL,
  idPatron    INTEGER,
  idBarco     INTEGER,
  FOREIGN KEY (idPatron) REFERENCES Persona,
  FOREIGN KEY (idBarco) REFERENCES Embarcacion
);
CREATE SEQUENCE seqTravesia;
```

(b) Datos relativos a un socio.

```
SELECT s.numSocio, s.nombre, s.apellidos, e.matricula,
       e.nombre, e.slora, e.amarre, e.quota
  FROM Persona s, Embarcacion e
 WHERE s.idPersona = _unSocio_
   AND s.idPersona = e.idDueno
```

(c) Datos de una embarcación.

```
SELECT s.numSocio, s.nombre, s.apellidos, e.amarre, e.quota
  FROM Embarcacion e, Persona s
 WHERE e.matricula = _unaMatricula_
   AND e.idDueno = s.idPersona
```

(d) Datos de una travesía:

```
SELECT t.fechaSalida, t.horaSalida, t.destino, p.nombre,
       p.apellidos, ti.tit
  FROM Travesia t, Embarcacion e, Persona p, Titulo ti
 WHERE e.matricula = _unaMatricula_
   AND e.idBarco = t.idBarco
   AND t.idPatron = p.idPersona
   AND p.idTitulo = ti.idTitulo
```

(e) Embarcaciones manejadas por un patrón:

```
SELECT DISTINCT e.matricula, e.nombre
  FROM Persona p, Travesia t, Embarcacion e
 WHERE p.idPersona = _unPatron_
   AND p.idPersona = t.idPatron
   AND t.idBarco = e.idBarco
```