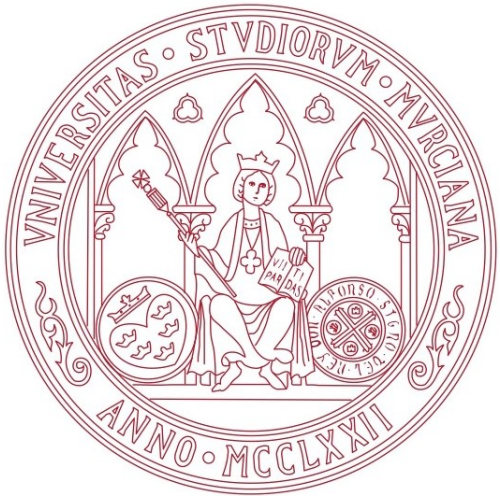


Grado en Ingeniería Informática

**Procesos de Desarrollo de Software**

# Práctica 2: Aplicación de un proceso basado en UML

**Autores**

Francisco David Campuzano Melgarejo
DNI: 48753083Y
Email: franciscodavid.campuzanom@um.es
Grupo: 2.2

Antonio Damián López Tardido
DNI: 23958516Z
Email: ad.lopeztardido@um.es
Grupo: 2.2

**Profesor**

Jose Alberto García Berna

# Índice

**1. Modelo del negocio** **1**

**2. Modelo de casos de uso** **2**

**3. Modelo del dominio** **4**

**4. Especificación Complementaria** **5**

4.1. Requisitos Funcionales . . . . . 5

4.2. Requisitos No Funcionales . . . . . 6

4.3. Restricciones de Diseño . . . . . 6

4.4. Reglas de Negocio . . . . . 7

4.5. Glorario de Definiciones . . . . . 9

**5. Detalles de los CdU, DSS, Contratos y Colaboraciones** **11**

5.1. Caso de Uso 1: Inscribirse en el sistema . . . . . 11

5.1.1. Diagrama de Secuencia . . . . . 12

5.1.2. Contrato 1 . . . . . 12

5.1.3. Diagrama de Colaboración Contrato 1 . . . . . 13

5.2. Caso de Uso 2: Reservar un circuito . . . . . 13

5.2.1. Diagrama de Secuencia . . . . . 15

5.2.2. Contrato 1 . . . . . 15

5.2.3. Diagrama de Colaboración Contrato 1 . . . . . 16

5.3. Caso de Uso 3: Confirmar reserva . . . . . 16

5.3.1. Diagrama de Secuencia . . . . . 17

5.3.2. Contrato 1 . . . . . 18

5.3.3. Diagrama de Colaboración Contrato 1 . . . . . 18

5.3.4. Contrato 2 . . . . . 18

5.3.5. Diagrama de Colaboración Contrato 2 . . . . . 19
5.4. Caso de Uso 4: Designar tripulaciones . . . . . 19
5.4.1. Diagrama de Secuencia . . . . . 21
5.4.2. Contrato 1 . . . . . 21
5.4.3. Diagrama de Colaboración Contrato 1 . . . . . 22
5.5. Caso de Uso 5: Otorgar una calificación . . . . . 22
5.5.1. Diagrama de Secuencia . . . . . 24
5.5.2. Contrato 1 . . . . . 24
5.5.3. Diagrama de Colaboración Contrato 1 . . . . . 25
5.6. Caso de Uso 6: Iniciar proceso de cálculo . . . . . 25
5.6.1. Diagrama de Secuencia . . . . . 27
5.6.2. Contrato 1 . . . . . 27
5.6.3. Diagrama de Colaboración Contrato 1 . . . . . 28
5.7. Caso de Uso 7: Solicitar carrera final . . . . . 29
5.7.1. Diagrama de Secuencia . . . . . 30
5.7.2. Contrato 1 . . . . . 30
5.7.3. Diagrama de Colaboración Contrato 1 . . . . . 31
5.8. Caso de Uso 8: Registrar resultado de la carrera final . . . . . 32
5.8.1. Diagrama de Secuencia . . . . . 33
5.8.2. Contrato 1 . . . . . 33
5.8.3. Diagrama de Colaboración Contrato 1 . . . . . 34

**6. Diagrama de clases del diseño** **35**

6.1. Diagrama de clases del diseño . . . . . 35
6.2. Diagrama de controladores y repositorios . . . . . 36

**7. Diagramas de estado** **37**

7.1. Campeonato . . . . . 37

7.2. Factura . . . . . 37
7.3. Reserva . . . . . 38
7.4. Carrera . . . . . 38
7.5. Comentario . . . . . 39

**8. Persistencia** **40**

8.1. Enumerado Estado . . . . . 40
8.2. Clase Campeonato . . . . . 40
8.3. Clase Carrera . . . . . 42
8.4. Clase Equipo . . . . . 44
8.5. Clase Participacion . . . . . 45
8.6. Clase Reserva . . . . . 47
8.7. Clase Piloto . . . . . 48
8.8. Clase Senior . . . . . 50
8.9. Clase Junior . . . . . 50
8.10. Clase Tripulacion . . . . . 51
8.11. Clase CarreraFinal . . . . . 53

# 1. Modelo del negocio

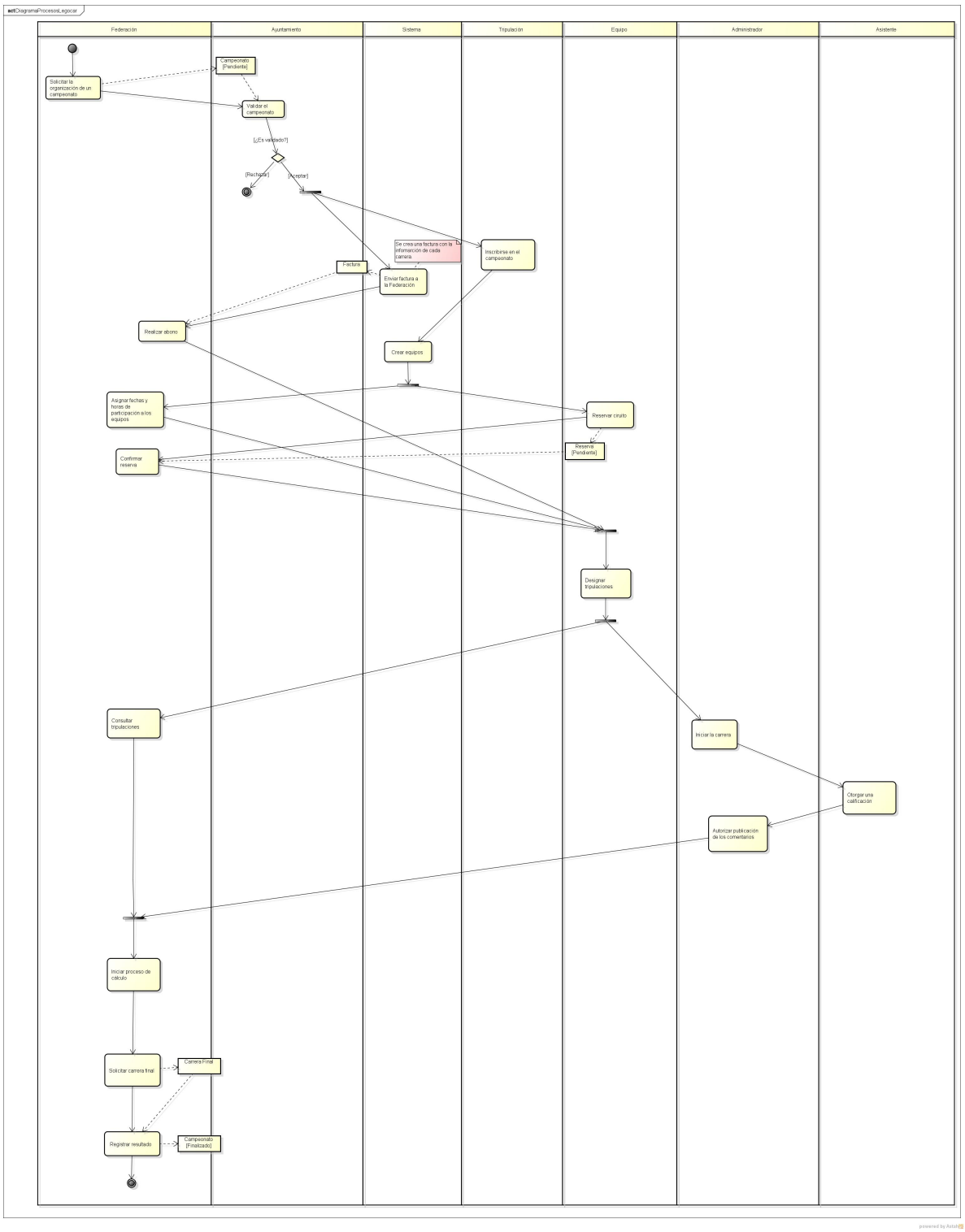


Figura 1: Diagrama de Procesos Legocar

# 2. Modelo de casos de uso

| Actor | CdU | Descripción |
|---|---|---|
| Federación legocars | 1. Solicitar la organización de un campeonato | La federación de legocars del municipio solicita la organización de un campeonato al Ayuntamiento con una antelación mínima de 3 meses. |
| | 4. Realizar abono | La federación de legocars realiza el abono del importe indicado en la factura generada por el Sistema con 10 días de antelación a la celebración del carrera. |
| | 7. Asignar fechas y horas de participación a los equipos | La federación de legocars le asigna a cada equipo las fechas y horas de participación en las distintas carreras del campeonato. |
| | 9. Confirmar reserva | La federación de legocars confirma las reservas de las instalaciones realizadas por los equipos. |
| | 10. Gestión de legocars | La federación de legocars se encarga de gestionar los legocars pudiendo darlos de alta, modificarlos o darlos de baja. |
| | 12. Consultar tripulaciones | La federación de legocars podrá consultar en todo momento las tripulaciones participantes en cada carrera. |
| | 14. Registrar resultado de la carrera | La federación de legocars inicia un proceso para el cálculo automático del resultado de la carrera y para actualizar la puntuación tras finalizar la carrera. |
| | 18. Solicitar carrera final | La federación de legocars solicitará al Sistema una carrera final de exhibición con los dos mejores equipos clasificados tras finalizar el campeonato. |
| | 19. Registrar resultado | La federación de legocars registrará el resultado de la carrera final para que el Sistema actualice el ranking y se concedan los dos premios a los equipos. |

| | | |
|---|---|---|
| Ayuntamiento | 2. Validar el campeonato | El Ayuntamiento valida la solicitud del campeonato realizada por la federación de legocars. |

| | | |
|---|---|---|
| Tripulación | 5. Inscribirse en el Sistema | La tripulación se inscribe en el Sistema indicando la información de la tripulación y la de sus dos pilotos |

| | | |
|---|---|---|
| Equipo | 8. Reservar un circuito | El equipo podrá reservar un circuito para entrenar introduciendo la información necesaria. |
| | 11. Designar tripulaciones | El equipo designará tres tripulaciones que participarán en la carrera tres días antes de su celebración. |

| | | |
|---|---|---|
| Sistema | 3. Enviar factura a la federación | El Sistema se encargará de enviar a la federación una factura por cada carrera un mes antes de la realización de cada carrera. |
| | 6. Crea equipos | El Sistema, una vez cerrado el plazo de inscripción, crea los equipos de manera aleatoria teniendo en cuenta una serie de factores. |

| | | |
|---|---|---|
| Asistente | 15. Entrar en el Sistema | El asistente entra al Sistema leyendo el token de la entrada con su movil. |
| | 16. Otorgar una calificación | El asistente otorga una calificación a cada tripulación durante su participación en la carrera. |

| | | |
|---|---|---|
| Administrador de carrera | 13. Iniciar una carrera | El administrador inicia la carrera en el Sistema. |
| | 17. Autorizar publicación de los comentarios | El administrador autoriza la publicación de los comentarios sobre la carrera en tiempo real en una pantalla gigante. |

# 3. Modelo del dominio

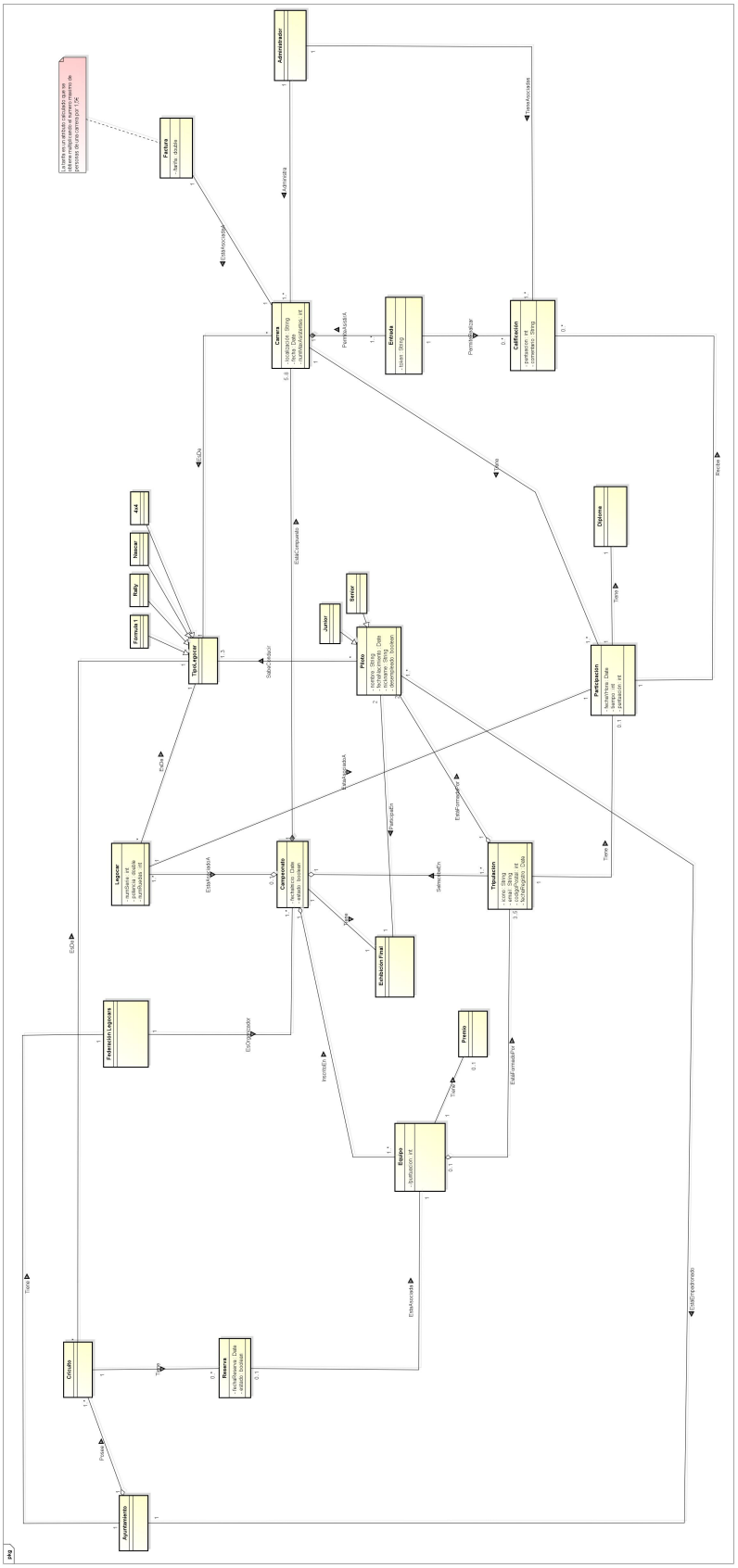


Figura 2: Modelo de Dominio Legocar

# 4. Especificación Complementaria

## 4.1. Requisitos Funcionales

En estas sección se enumerarán todos los requisitos funcionales de la aplicación. Cada requisito tendrá un identificador único, una descripción del propio requisito y una prioridad. Esta prioridad puede ser alta (requisito obligatorio), media (requisito recomendable) o baja (requisito opcional).

- **RQF001**: La aplicación debe permitir el pago de las facturas por parte de la federación solicitante. (Alta)
- **RQF002**: El sistema deberá asignar, de manera aleatoria, un legocar a cada tripulación elegida para cada carrera. (Alta)
- **RQF003**: El sistema deberá registrar el tiempo de cada vuelta de las tripulaciones en cada carrera. (Alta)
- **RQF004**: El sistema permitirá a la federación de legocars actualizar las puntuaciones de cada equipo en el campeonato y actualizar el ranking. (Alta)
- **RQF005**: El sistema deberá generar un diploma de participación a todas las tripulaciones participantes de la carrera. (Media)
- **RQF006**: El sistema no permitirá a los asistentes modificar una votación ya emitida. (Alta)
- **RQF007**: El sistema deberá actualizar la valoración media de los equipos en tiempo real. (Alta)
- **RQF008**: El sistema deberá filtrar los comentarios de las calificaciones otorgadas a los equipos de palabras malsonantes. (Alta)
- **RQF009**: El sistema intentará corregir la ortografía de los comentarios de las calificaciones de los asistentes. (Baja)
- **RQF010**: La aplicación debe otorgar 3 puntos adicionales a las tres tripulaciones con mejor valoración en cada carrera. (Alta)
- **RQF011**: El sistema seleccionará de forma aleatoria un piloto de cada uno de los dos equipos que participan en la carrera final. (Alta)
- **RQF012**: El sistema deberá conceder dos premios, uno para el equipo campeón y otro para el equipo subcampeón. (Alta)
- **RQF013**: El sistema activará los sensores al iniciar la carrera.

## 4.2. Requisitos No Funcionales

En estas se enumerarán todos los requisitos no funcionales de la aplicación. Al igual que en la subsección anterior, cada requisito tendrá un identificador único, una descripción y una prioridad.

- **RQNF001**: La aplicación web deberá ser adaptativa para poder visualizarse correctamente en todos los dispositivos móviles
- **RQNF002**: El servidor web utilizado debe ser accesible en todo momento, ya que atenderá las peticiones de los usuarios.
- **RQNF003**: La aplicación web deberá permitir la conexión de al menos 500 usuarios simultáneamente.
- **RQNF004**: El sistema debe proporcionar la información en español y en inglés.
- **RQNF005**: La herramienta debe ser una aplicación web.
- **RQNF006**: El ayuntamiento debe tener circuitos habilitados para la práctica con varios tipos de legocars.
- **RQNF007**: Cuando la federación valida una reserva de un circuito, este deja de estar libre.

## 4.3. Restricciones de Diseño

1. Todos los trabajos técnicos, diseños y desarrollos se realizarán de acuerdo a los estándares de trabajo definidos por la Dirección General de Patrimonio, Informática y Telecomunicaciones.
2. El sistema deberá cumplir con la política de seguridad definida en la Arquitectura de Seguridad Avanzada de la CARM (ASA)
3. El sistema deberá estar desarrollado en lenguaje Java (JEE, arquitectura de 3 capas).
4. El servidor web deberá estar implementado en Linux con Apache 2.
5. La base de datos empleada será Oracle 12g en cluster.
6. El servidor de aplicaciones deberá estar implementado en Tomcat 9.0.42 o JBoss.
7. El servidor de integración deberá estar implementado en Apache CFX.
8. El pago de las facturas se realizará de forma segura mediente la plataforma de pago online PayPal.

## 4.4. Reglas de Negocio

- **RN001**: Los legocars son automóviles de Lego a motor.
- **RN002**: Los legocars son de distintos tipos: Fórmula 1, Rally, Nascar, 4x4,...
- **RN003**: Sólo puede haber una federación de legocars por municipio.
- **RN004**: La solicitud de un campeonato se debe realizar con una antelación mínima de tres meses.
- **RN005**: Un campeonato se compone de cinco a ocho carreras.
- **RN006**: Cada carrera está dirigida a un tipo específico de coche.
- **RN007**: Cada carrera tiene una factura que se calcula de la siguiente manera:

$$precioFactura = 1,5€ \times nPersonas \times nDias$$

- **RN008**: El pago de cada factura se debe realizar con al menos diez días de antelación a la celebración de la carrera correspondiente.
- **RN009**: Cuando el ayuntamiento valide la solicitud del campeonato, entonces se debe abrir el proceso de inscripción.
- **RN010**: El plazo de inscripción está abierto hasta diez días antes de la fecha del campeonato.
- **RN011**: Sólo se pueden inscribir en el campeonato las tripulaciones.
- **RN012**: Un equipo de legocars está formado por entre tres y cinco tripulaciones.
- **RN013**: Un equipo de legocars está formado por dos pilotos. Un piloto Junior y otro Senior.
- **RN014**: Una tripulación es capaz de manejar de uno a tres tipos de legocars.
- **RN015**: Todos los pilotos deben estar federados en la federación solicitante del campeonato.
- **RN016**: Las tripulaciones que forman parte de un equipo deben estar próximas según su código postal y orden de registro.
- **RN017**: Las tripulaciones que forman parte de un equipo deben ser expertas en el pilotaje de distintos tipos de legocars.
- **RN018**: Cuando una tripulación no forma parte de un equipo, esta no puede participar en el campeonato.
- **RN019**: Los legocars que se utilizan en el campeonato son gestionados por la federación organizadora.
- **RN020**: En cada carrera participan tres tripulaciones de cada equipo elegidas por el propio equipo.

- **RN021**: Cada tripulación conduce por control remoto un legocar por cada carrera. Este legocar puede ser distinto en distintas carreras.
- **RN022**: Cada carrera tiene una duración de dos vueltas.
- **RN023**: La salida y el paso de cada legocar por línea de meta se registran a través de sensores.
- **RN024**: Cada tripulación que participa en una carrera debe cumplir que en la primera vuelta conduzca el piloto Junior y la segunda el piloto Senior.
- **RN025**: El ranking se debe actualizar al finalizar una carrera.
- **RN026**: Todas las tripulaciones que han participado en la carrera deben recibir un diploma.
- **RN027**: La tripulación ganadora de una carrera recibe diez puntos, la segunda cinco puntos, la tercera tres puntos, el resto pierde un punto y la última pierde cinco puntos.
- **RN028**: La puntuación global de un equipo es la suma de las puntuaciones de las tripulaciones que lo forman.
- **RN029**: Todos los equipos empiezan el campeonato con cero puntos.
- **RN030**: Todas las entradas deben tener un token alfanumérico de doce caracteres.
- **RN031**: Las calificaciones a las tripulaciones tienen una puntuación de entre cero y diez.
- **RN032**: Las calificaciones pueden incluir un comentario de texto de hasta ochenta caracteres.
- **RN033**: Las calificaciones ya emitidas por parte de los asistentes no se pueden modificar.
- **RN034**: Las valoraciones medias de cada equipo se actualizan en tiempo real.
- **RN035**: Los comentarios deben pasar por un filtro de palabras malsonantes y otro de correción de ortografía.
- **RN036**: Las tres tripulaciones mejor calificadas por el público reciben tres puntos adicionales cada una.
- **RN037**: El campeonato debe tener una carrera final con los dos equipos mejor clasificados.
- **RN038**: En la carrera final se enfrentan dos pilotos de cada equipo elegios aleatoriamente.
- **RN039**: La carrera final tiene una duración de cuatro vueltas.
- **RN040**: El ranking se debe actualiar tras la finalización de la carrera final.
- **RN041**: Cada equipo participante en la carrera final recibe un premio.

- **RN042**: Los pilotos premiados que estén desempleados deben recibir una ayuda adicional.

## 4.5. Glorario de Definiciones

| **Término o Acrónimo** | **Sinónimos** | **Definición** |
|---|---|---|
| Legocar | | Son automóviles de Lego a motor de distintos tipos como: Fórmula 1, Rally, Nascar, 4x4,etc |
| Token | Símbolo | Es una clave única que autentica y autoriza el acceso a recursos en un sistema. |
| WCAG 2.1 (Web Content Accessibility Guidelines 2.1) | | Es una serie de pautas establecidas para garantizar que los contenidos en la web sean accesibles para todas las personas, incluyendo aquellas con discapacidades |
| WAI | | Es una iniciativa del World Wide Web Consortium (W3C) que se enfoca en promover la accesibilidad en la web |
| CARM | | Comunidad Autónoma de la Región de Murcia |
| ASA | | Arquitectura de Seguridad Avanzada |
| Java | | Lenguaje de programación de alto nivel y orientado a objetos, diseñado para ser portátil, seguro y robusto |
| JEE (Java Enterprise Edition) | | Es una plataforma de desarrollo de software basada en Java que proporciona un conjunto de especificaciones y API (Application Programming Interfaces) para el desarrollo de aplicaciones empresariales escalables y robustas. |
| API (Application Programming Interfaces) | | Es un conjunto de reglas y protocolos que define cómo se pueden comunicar diferentes componentes de software entre sí. Sirve como una capa de abstracción que permite que dos aplicaciones o sistemas intercambien información y realicen funciones específicas de manera estructurada. |
| Oracle 12g | | Es una base de datos relacional de alta calidad y escalable, diseñada para manejar grandes colúmenes de datos. |
| Cluster | | Es un grupo de computadoras o servidores interconectados que trabajan juntos como si fueran una única entidad. |
| Linux | | Es un sistema operativo de código abierto basado en el núcleo de Linux. A diferencia de otros sistemas operativos comerciales, Linux es conocido por su naturaleza gratuita, su flexibilidad y su amplia comunidad de desarrollo. |

| | | |
|---|---|---|
| Apache 2 | | Es la segunda versión principal del servidor web Apache HTTP Server. Es un software de servidor web de código abierto que proporciona confiabilidad, escalabilidad y flexibilidad para la entrega de contenido web estático y dinámico. |
| Tomcat | | Es un servidor web y contenedor de servlets de código abierto desarrollado por Apache. Se utiliza para ejecutar aplicaciones web basadas en tecnologías Java, como servlets y JSP. Es ampliamente utilizado debido a su facilidad de configuración, autoadministración y soporte para archivos WAR. |
| JBoss | | Es una plataforma de aplicaciones de código abierto desarrollada por Red Hat. Se utiliza para desarrollar, implementar y ejecutar aplicaciones empresariales basadas en Java |
| Apache CFX | | Es un framework de servicios web de código abierto que facilita el desarrollo y consumo de servicios web. Proporciona herramientas y bibliotecas para construir servicios web robustos y escalables, con soporte para tecnologías y protocolos web |
| Responsive | Quick to react | Un diseño o elemento responsive.[es] aquel que se adapta y se presenta de forma óptima en diferentes dispositivos y tamaños de pantalla, brindando una experiencia de usuario consistente y amigable. |

# 5. Detalles de los CdU, DSS, Contratos y Colaboraciones

## 5.1. Caso de Uso 1: Inscribirse en el sistema

**Actor Principal:** Tripulación

**Stakeholders:**

- Tripulación
- Equipo
- Piloto
- Ayuntamiento

**Precondiciones:**

- Se ha validado la realización del campeonato por parte del ayuntamiento.
- El plazo de inscripción se encuentra abierto.
- La inscripción se realiza en el plazo habilitado (10 días antes de la fecha del campeonato).
- Los pilotos deben estar federados en la federación solicitante del campeonato.

**Postcondiciones:**

- Se registró la Tripulación en el Sistema.

**Escenario principal de éxito (o Flujo Básico):**

1. La Tripulación decide inscribirse en el Campeonato.
2. La Tripulación se inscribe en el Campeonato indicando el email de contacto, el icono y el código postal.
3. El Sistema comprueba que no existe otra Tripulación registrada con el mismo email.
4. La Tripulación inscribe a dos Pilotos que son seleccionados de todos los Pilotos federados que están libres.
5. El Sistema registra la Tripulación junto a sus dos Pilotos en el Campeonato y notifica a la Tripulación que la inscripción se ha realizado correctamente.

**Extensiones (o Flujos Alternativos):**

3a El email introducido por la tripulación ya pertenece a otra tripulación registrada anteriormente.

*a*) El Sistema no permite a la tripulación volver a registrarse.

*b*) El Sistema muestra un mensaje de "registro no válido".

2-4a Se produce un fallo en la inscripción de la Tripulación.

*a*) El Sistema cancela el proceso

*b*) El Sistema nofitica a la Tripulación que se ha abortado el proceso de inscripción.

### 5.1.1. Diagrama de Secuencia

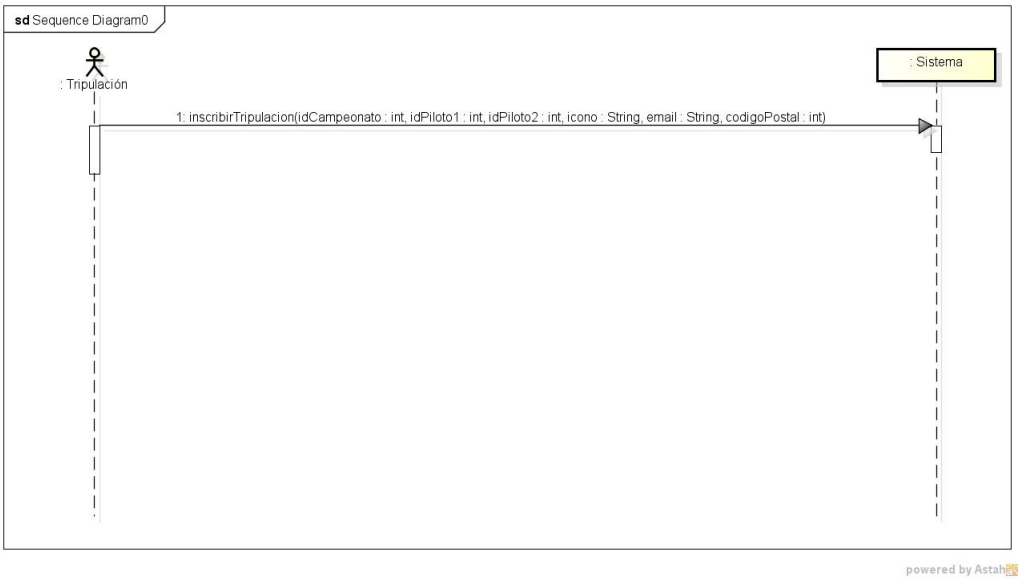


Figura 3: Diagrama de Secuencia del caso de uso 1

### 5.1.2. Contrato 1

- **Operación:** inscribirTripulacion(idCampeonato : int, idPiloto1 : int, idPiloto2 : int, icono : String, email : String, codigoPostal : int)
- **Controlador:** ControladorCampeonato
- **Precondiciones:**
  - Existe un Campeonato c cuyo id es idCampeonato.
  - Existe un Piloto p1 cuyo id es idPiloto1 y está federado en la Federación solicitante

- Existe un Piloto p2 cuyo id es idPiloto2 y está federado en la Federación solicitante.
- La fecha actual de registro debe ser menor en al menos diez días de la fechaInicio del Campeonato c.

▪ **Postcondiciones:**

- Se creó una instancia t de Tripulación.
- Se inicializó la Tripulacion t:
  - t.icono = icono
  - t.email = email
  - t.codigoPostal = codigoPostal
  - t.fechaRegistro = fechaActual
- Se asoció la Tripulacion t con las entidades p1 y p2 de Piloto.
- Se añadió la Tripulación t a la colección de tripulaciones del Campeonato c.

### 5.1.3. Diagrama de Colaboración Contrato 1

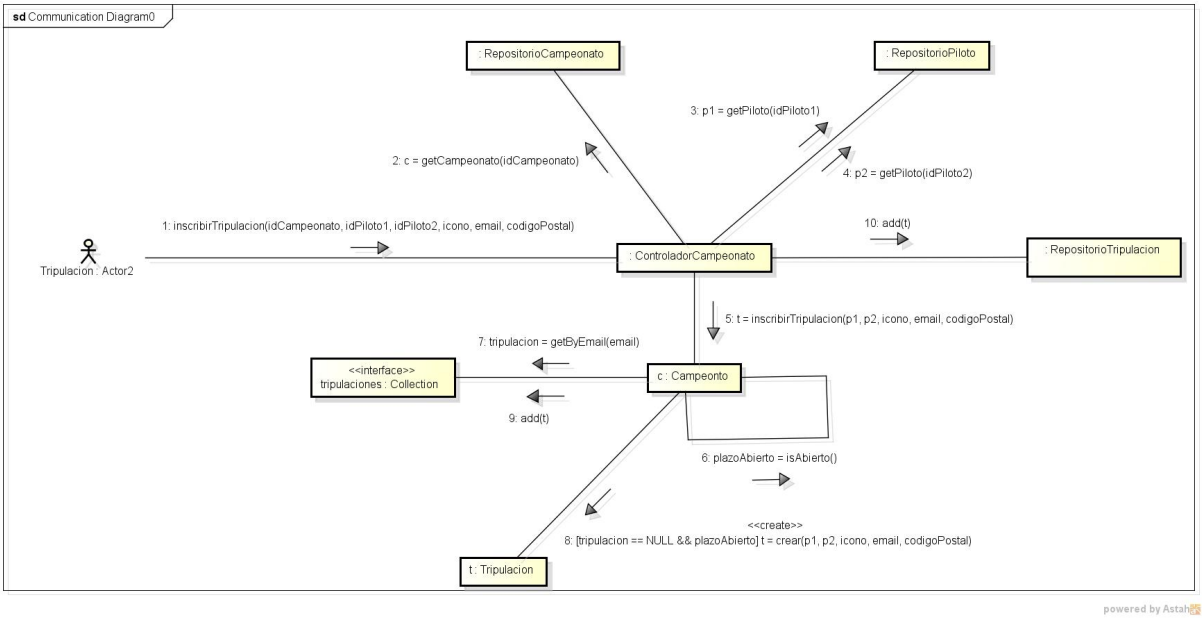


Figura 4: Diagrama Colaboracion CdU 1. Contrato 1

## 5.2. Caso de Uso 2: Reservar un circuito

**Actor Principal:** Equipo

**Stakeholders:**

- Tripulación
- Equipo
- Piloto

**Precondiciones:**

- Se han creado los Equipos formados de 3 a 5 Tripulaciones.
- Faltan varios días para que se celebre la siguiente Carrera.
- El Ayuntamiento dispone de varios Circuitos de Legocars para todos los tipos de legocar.

**Postcondiciones:**

- La solicitud de reserva de un circuito es recibida por la Federación.

**Escenario principal de éxito (o Flujo Básico):**

1. El Equipo quiere realizar la Reserva de un Circuito para entrenar.
2. El Equipo proporciona la información del Equipo, el tipo de legocar con el que va a entrenar y la fecha deseada del entrenamiento para realizar la Reserva.
3. El Sistema buscará el primer Circuito del tipo de legocar indicado por el Equipo y que no tenga una Reserva en la fecha indicada.
4. El Sistema envía la solicitud de reserva a la Federación.

**Extensiones (o Flujos Alternativos):**

2a. El Equipo cancela el proceso de reserva.

*a*) El Sistema aborta el proceso de reserva.

2-3a. Se produce un fallo en el Sistema.

*a*) El Sistema aborta la reserva.
*b*) El Sistema muestra un mensaje de "reserva no realizada".

3a. El Sistema no encuentra ningún circuito libre.

*a*) El Sistema notifica que no hay ningún circuito libre para el tipo de legocar solicitado y fecha solicitada.

### 5.2.1. Diagrama de Secuencia


Figura 5: Diagrama de Secuencia del caso de uso 2

### 5.2.2. Contrato 1

- **Operación:** solicitarReserva(idEquipo : int, idTipoLegocar : int, fechaReserva : Date)
- **Controlador:** ControladorReserva
- **Precondiciones:**
  - Existe un Equipo e cuyo id es idEquipo.
  - Existe un TipoLegocar tl cuyo id es idTipoLegocar.
  - Existe una coleccion de Circuitos circuitos asociados cada uno a un tipo de legocar.
  - La fechaReserva debe ser anterior a la finalización del campeonato (antes de que se inicie o mientras se está celebrando).
- **Postcondiciones:**
  - Si existe un Circuito c del tipo de legocar indicado y que no tenga ninguna Reserva en la fecha indicada
    - Se creó una instancia r de Reserva.
    - Se inicializó la Reserva r:
      - r.estado = “PENDIENTE”

◇ r.fechaReserva = fechaReserva

- Se añadió r a la colección de Reservas del Circuito c.
- Se asoció el Equipo e con la Reserva r.

### 5.2.3. Diagrama de Colaboración Contrato 1

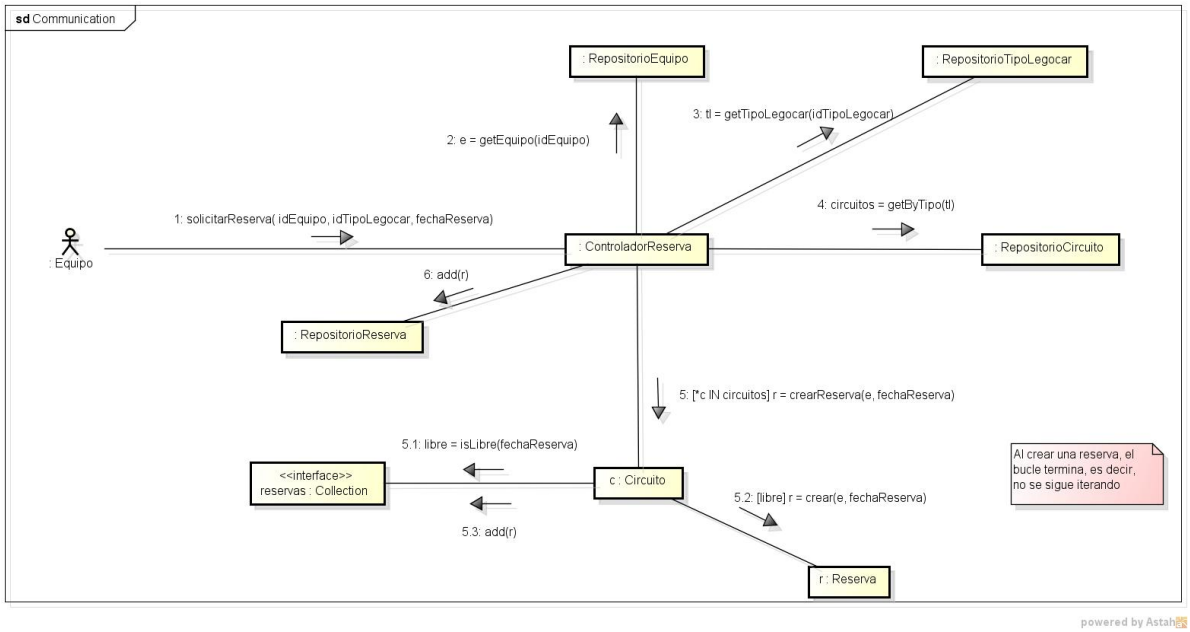


Figura 6: Diagrama de Colaboracion CdU 2. Contrato 1

## 5.3. Caso de Uso 3: Confirmar reserva

**Actor Principal:** Federación de legocars

**Stakeholders:**

- Federación de legocars
- Equipo
- Tripulación
- Pilotos
- Ayuntamiento

**Precondiciones:**

- La federación se ha autenticado en el sistema.

- El equipo ha realizado una solicitud de reserva de un circuito con anterioridad.

**Postcondiciones:**

- Se aprobará/rechazará la reserva del circuito.

**Escenario principal de éxito (o Flujo Básico):**

1. La federación revisa la solicitud de reserva de circuito por parte del equipo.
2. La federación confirma la solicitud de reserva.
3. El Sistema notifica al equipo de que su reserva a sido confirmada.

**Extensiones (o Flujos Alternativos):**

2a La federación no confirma la solicitud de reserva.

*a*) Se cancela la solicitud de reserva del circuito.

*b*) El circuito seleccionado por el Sistema para la reserva vuelve a quedar libre para otra reserva.

*c*) El Sistema notifica al equipo de que su reserva no ha sido confirmada por la federación.

### 5.3.1. Diagrama de Secuencia

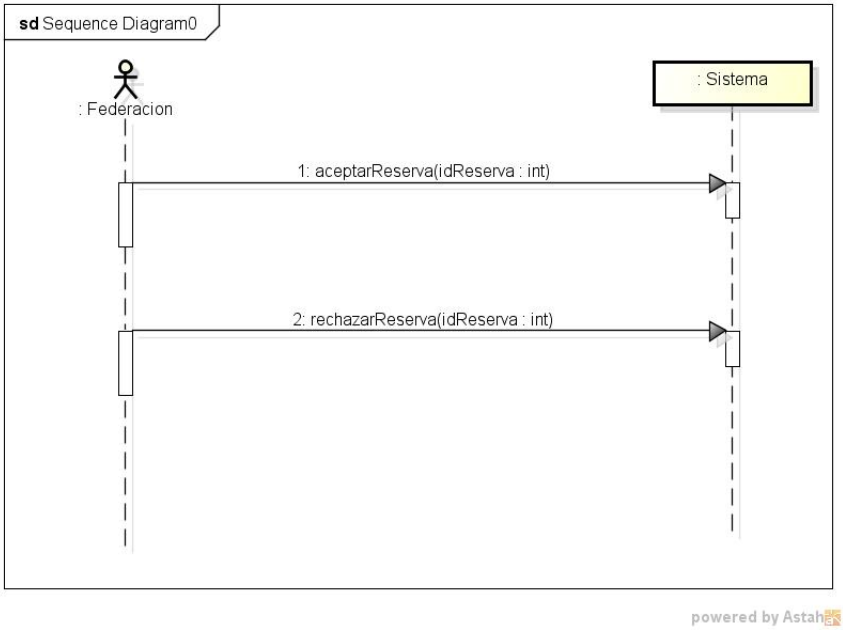


Figura 7: Diagrama de Secuencia del caso de uso 3

### 5.3.2. Contrato 1

- **Operación:** aceptarReserva(idReserva : int)
- **Controlador:** ControladorReserva
- **Precondiciones:**
  - Existe una Reserva r cuyo id es idReserva.
  - Existe una Reserva r en estado "PENDIENTE".
- **Postcondiciones:**
  - r.estado = "CONFIRMADA".

### 5.3.3. Diagrama de Colaboración Contrato 1

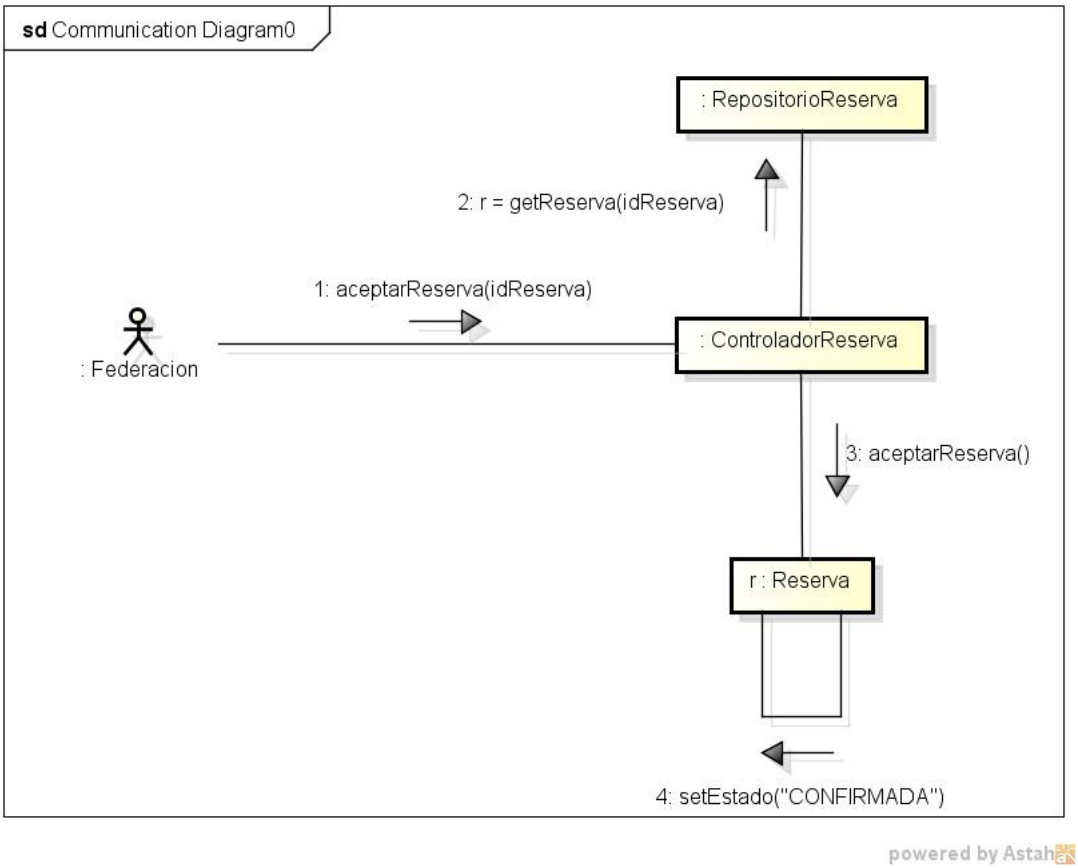


Figura 8: Diagrama de Colaboracion CdU 3. Contrato 1

### 5.3.4. Contrato 2

- **Operación:** rechazarReserva(idReserva : int)
- **Controlador:** ControladorReserva

- **Precondiciones:**
  - Existe una Reserva r cuyo id es idReserva.
  - Existe una Reserva r en estado “PENDIENTE”.
- **Postcondiciones:**
  - r.estado = “RECHAZADA”.

### 5.3.5. Diagrama de Colaboración Contrato 2

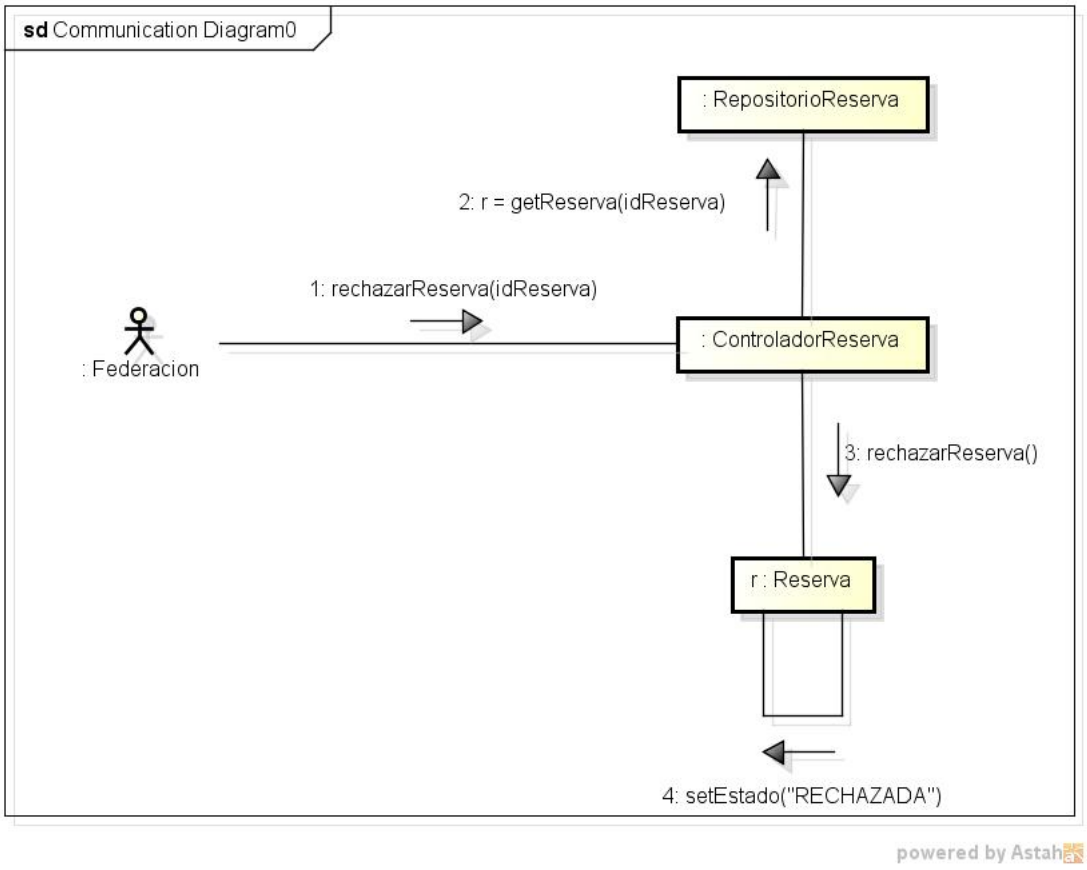


Figura 9: Diagrama de Colaboracion CdU 3. Contrato 2

## 5.4. Caso de Uso 4: Designar tripulaciones

**Actor Principal:** Equipo

**Stakeholders:**

- Equipo
- Tripulación

- Pilotos

**Precondiciones:**

- Se les asignó a los Equipos la fecha y hora de participación en la Carrera.
- Quedan tres días para la Carrera.

**Postcondiciones:**

- Cada Equipo elegirá 3 Tripulaciones que participarán en la Carrera.
- Se les asociará a las Tripulaciones un Legocar para la Carrera

**Escenario principal de éxito (o Flujo Básico):**

1. El Equipo se dispone a designar las Tripulaciones que participarán en la carrera.
2. El Sistema comprueba las Tripulaciones cuyos Pilotos sepan conducir Legocars del mismo tipo que el de la Carrera.
3. El Sistema selecciona tres Tripulaciones para participar en la Carrera.
4. El Sistema asigna un Legocar para cada una de las Tripulaciones designadas.

**Extensiones (o Flujos Alternativos):**

3a. El Sistema no encuentra tres Tripulaciones cuyos Pilotos sepan conducir el tipo de legocar de la Carrera.

   *a*) El Sistema saca al equipo de la carrera.
   *b*) El Sistema notifica al equipo de su inactividad en la carrera.

### 5.4.1. Diagrama de Secuencia

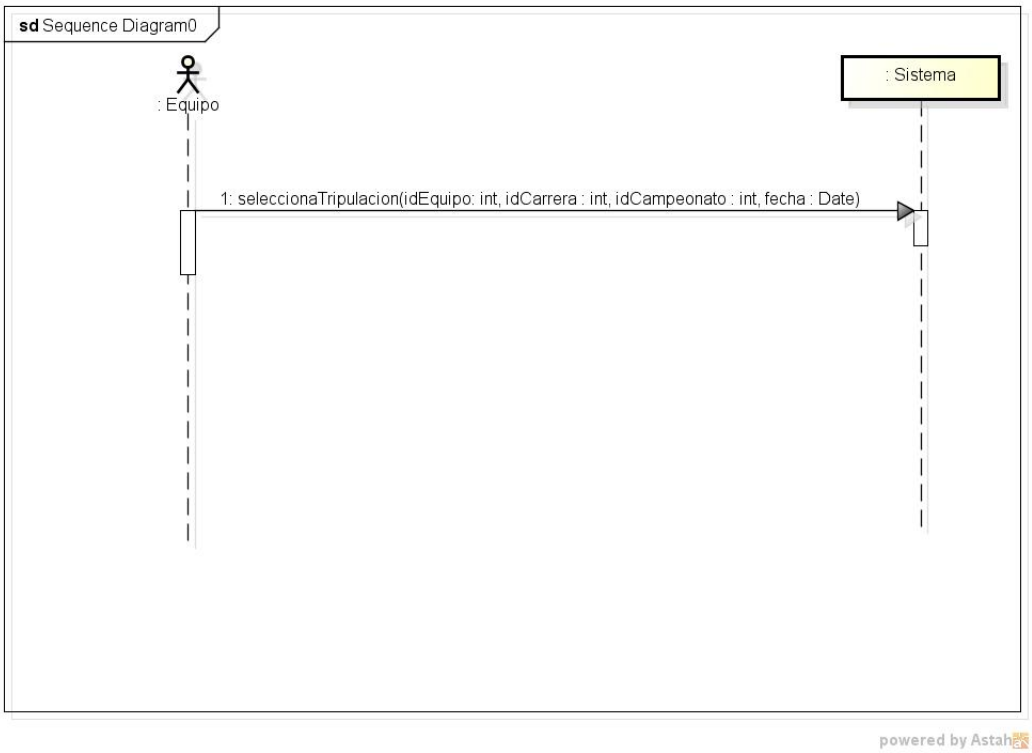


Figura 10: Diagrama de Secuencia del caso de uso 4

### 5.4.2. Contrato 1

- **Operación:** seleccionarTripulacion(idEquipo : int, idCarrera : int, idCampeonato : int, fecha : Date)
- **Controlador:** ControladorCarrera
- **Precondiciones:**
  - Existe un Equipo e cuyo id es idEquipo.
  - Exite una Carrera c cuyo id es idCarrera.
  - Exixte un Campeonato c cuyo id es idCampeonato.
  - La fecha es posterior a la fechaActual.
- **Postcondiciones:**
  - Por cada Tripulacion t del Equipo e
    - Si los dos Pilotos de la Tripulacion t saben conducir el Legocars del mismo tipo que el de la Carrera c
      - ◇ Se creó una instancia p de Participacion.
      - ◇ Se inicializó la Participacion p:
      - ◇ p.fecha = fecha

- ◇ Se asoció la Participación p con la Tripulación t.
- ◇ Se asoció la Participación p con un Legocar l del mismo tipo que la Carrera c.
- ◇ Se añadió la Participación p a la colección de Participaciones de la Carrera c.

### 5.4.3. Diagrama de Colaboración Contrato 1

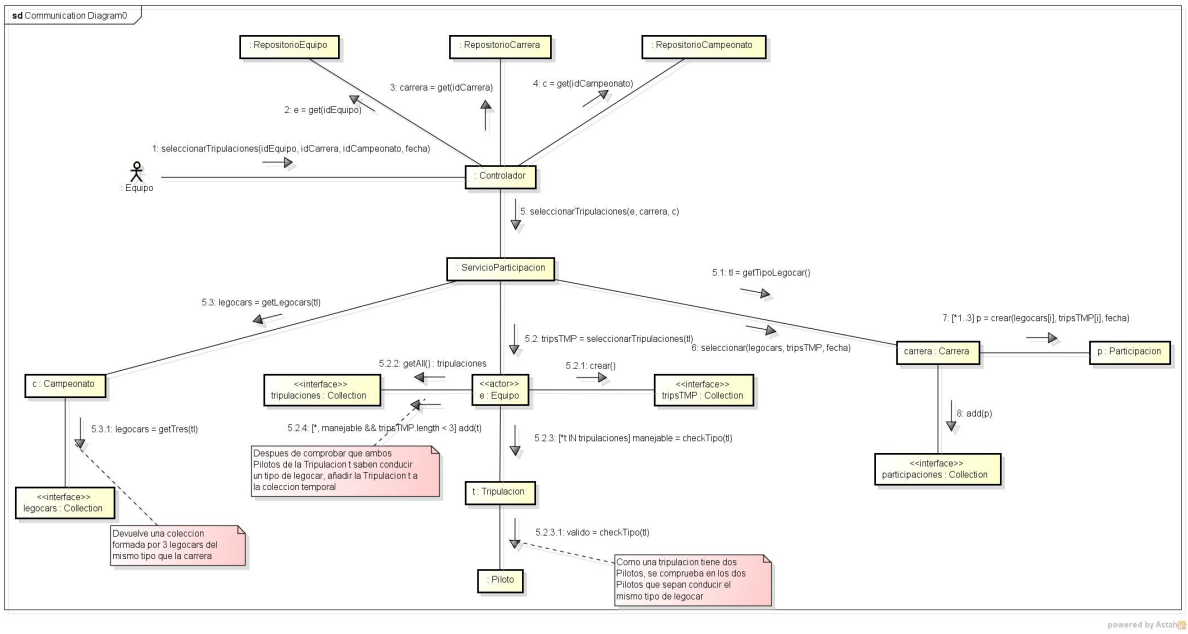


Figura 11: Diagrama de Colaboracion CdU 4. Contrato 1

## 5.5. Caso de Uso 5: Otorgar una calificación

**Actor Principal:** Asistente

**Stakeholders:**

- Asistente
- Tripulación
- Equipo
- Administrador de carrera

**Precondiciones:**

- Una Carrera del Campeonato se está realizando en estos momentos.

- El Asistente se registra en el Sistema a través del token de su Entrada.
- La Tripulación que recibe una Calificación ha terminado la Carrera.

**Postcondiciones:**

- Se registra la Calificación otorgada a una Tripulación.
- Se filtran los mensajes de cada Calificación (si se tienen).
- Se actualiza en tiempo real la valoración media de cada Equipo.

**Escenario principal de éxito (o Flujo Básico):**

1. El Asistente se dispone a calificar la actuación de una Tripulación en la Carrera.
2. El Sistema comprueba que la Tripulación ha terminado la carrera y que el Asistente no ha calificado aún la Tripulación seleccionada.
3. El Asistente elige una puntuación de 0 a 10.
4. El Asistente escribe un comentario opcional (máximo 80 caracteres).
5. El Asistente ratifica la calificación y la envía al Sistema.
6. El Sistema registra la nueva calificación asociándola a la Tripulación a la que se le ha otorgado.
7. El Sistema calcula la valoración media de las calificaciones otorgadas por los asistentes a cada Tripulación.
8. El Sistema aplica un filtro de palabras malsonantes y otro filtro de ortografía a los comentarios de las calificaciones.
9. El Sistema introduce los comentarios filtrados en una cola para su posterior autorización por parte del Administrador de carrera.

**Extensiones (o Flujos Alternativos):**

2a. El Sistema comprueba que la Tripulación no ha terminado la carrera.

  *a*) El Sistema no permite al Asistente calificar esa Tripulación.

2b. El Sistema comprueba que la Tripulación ya ha sido calificada por el Asistente 'actual'.

  *a*) El Sistema muestra la calificación otorgada por el Asistente.
  *b*) El Sistema no permite que vuelva a calificar a la misma Tripulación.

3-5a. El Asistente cancela la calificación.

*a*) El Sistema aborta el proceso de calificación.

3-5b. Se produce un fallo en el Sistema.

*a*) El Sistema aborta el proceso de calificación

### 5.5.1. Diagrama de Secuencia


Figura 12: Diagrama de Secuencia del caso de uso 5

### 5.5.2. Contrato 1

- **Operación:** otorgarCalificacion(idTripulacion : int, idEntrada : int, puntuacion : int, comentario : String)
- **Controlador:** ControladorCarrera
- **Precondiciones:**
  - Existe una Tripulacion t cuyo id es idTripulacion.
  - Existe una Entrada ent cuyo id es idEntrada.
  - La puntuación es un valor entre 0 y 10.
  - El comentario es opcional y tiene un máximo de 80 caracteres.
- **Postcondiciones:**
  - Si la Participación p de la Tripulación t no tiene ninguna Calificación realizada por la Entrada ent
    - Se creó una Calificación cal.
    - Se inicializó la Calificación cal:

- ◇ cal.puntuacion = puntuacion
- ◇ cal.comentario = comentario

- ○ Se asoció la Calificación cal con la Entrada ent.
- ○ Se añadió la Calificación cal a la colección de Calificaciones de la Participación p.

### 5.5.3. Diagrama de Colaboración Contrato 1

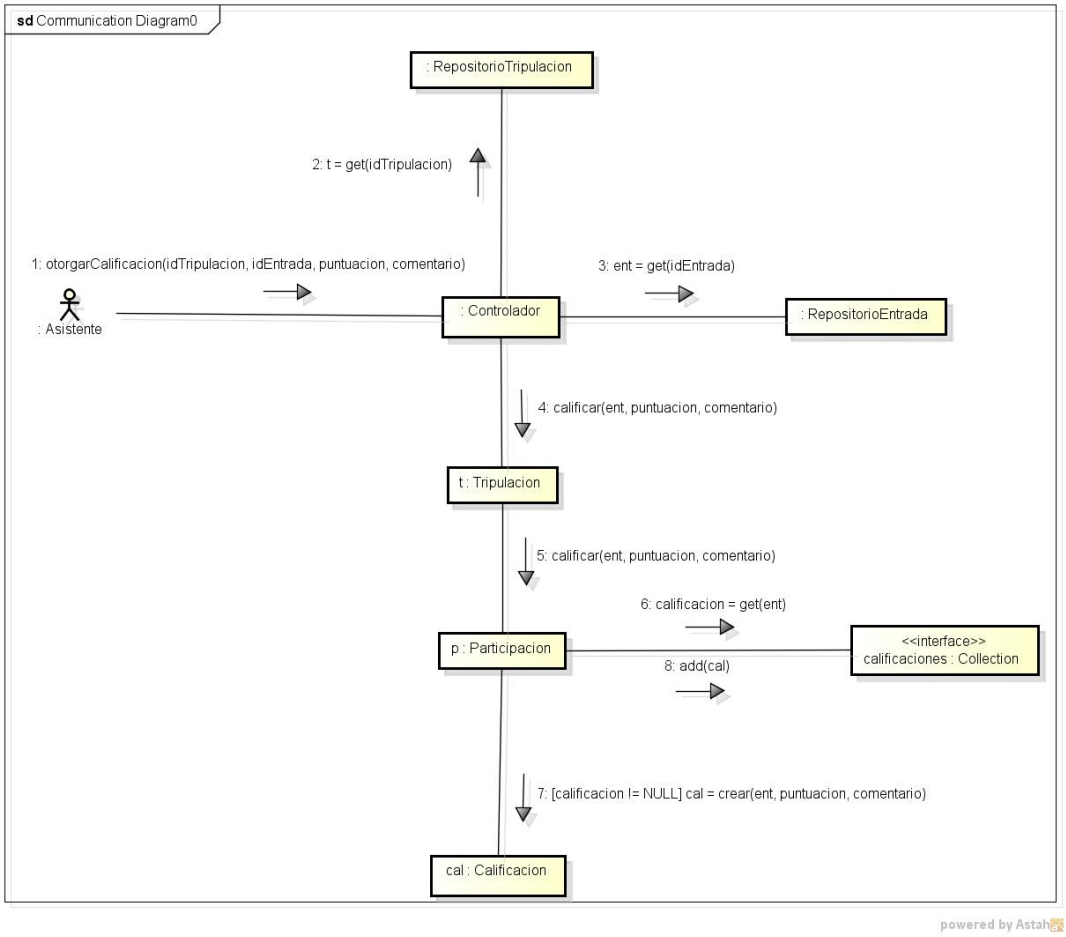


Figura 13: Diagrama de Colaboracion CdU 5. Contrato 1

## 5.6. Caso de Uso 6: Iniciar proceso de cálculo

**Actor Principal:** Federación.

**Stakeholders:**

- Federación
- Tripulación

- Equipo

**Precondiciones:**

- La carrera que se está celebrando ha concluido.
- Se han registrado los tiempos de todas las Tripulaciones participantes.
- Se han registrado todas las calificaciones de los Asistentes.

**Postcondiciones:**

- Los Equipos y Tripulaciones tiene sus puntuaciones actualizadas.
- Las Tripulaciones participantes reciben un diploma de participación.

**Escenario Principal de Éxito (o Flujo Básico):**

1. La Federación inicia el proceso del cálculo automático del resultado de la carrera.
2. El Sistema obtiene todas las Tripulaciones participantes en la carrera.
3. El Sistema calcula para cada Tripulación la suma de los tiempos en cada vuelta.
4. El Sistema ordena de menor a mayor la suma de los tiempos de cada Tripulación.
5. El Sistema otorga 10 puntos a la primera Tripulación.
6. El Sistema otorga 5 puntos a la segunda Tripulación.
7. El Sistema otorga 3 puntos a la tercera Tripulación.
8. El Sistema resta 1 punto a todas las Tripulaciones que estén entre la cuarta y la penúltima posición.
9. El Sistema resta 5 puntos a la última Tripulación.
10. El Sistema ordena de mayor a menor la media de las calificaciones de las Tripulaciones.
11. El Sistema otorga 3 puntos a las 3 primeras Tripulaciones.
12. El Sistema calcula la puntuación de cada Equipo como la suma de las puntuaciones de las Tripulaciones del Equipo.
13. El Sistema concede un diploma a todas las Tripulaciones participantes en la carrera.

**Extensiones (o Flujos Alternativos):**

1-9a. Se produce un fallo en el Sistema.

*a*) El Sistema aborta el proceso de cálculo.

*b*) El Sistema no guarda los cambios realizados.

10-11a. Se produce un fallo en el Sistema.

*a*) El Sistema aborta el proceso de cálculo.

*b*) El Sistema no guarda los cambios realizados.

### 5.6.1. Diagrama de Secuencia

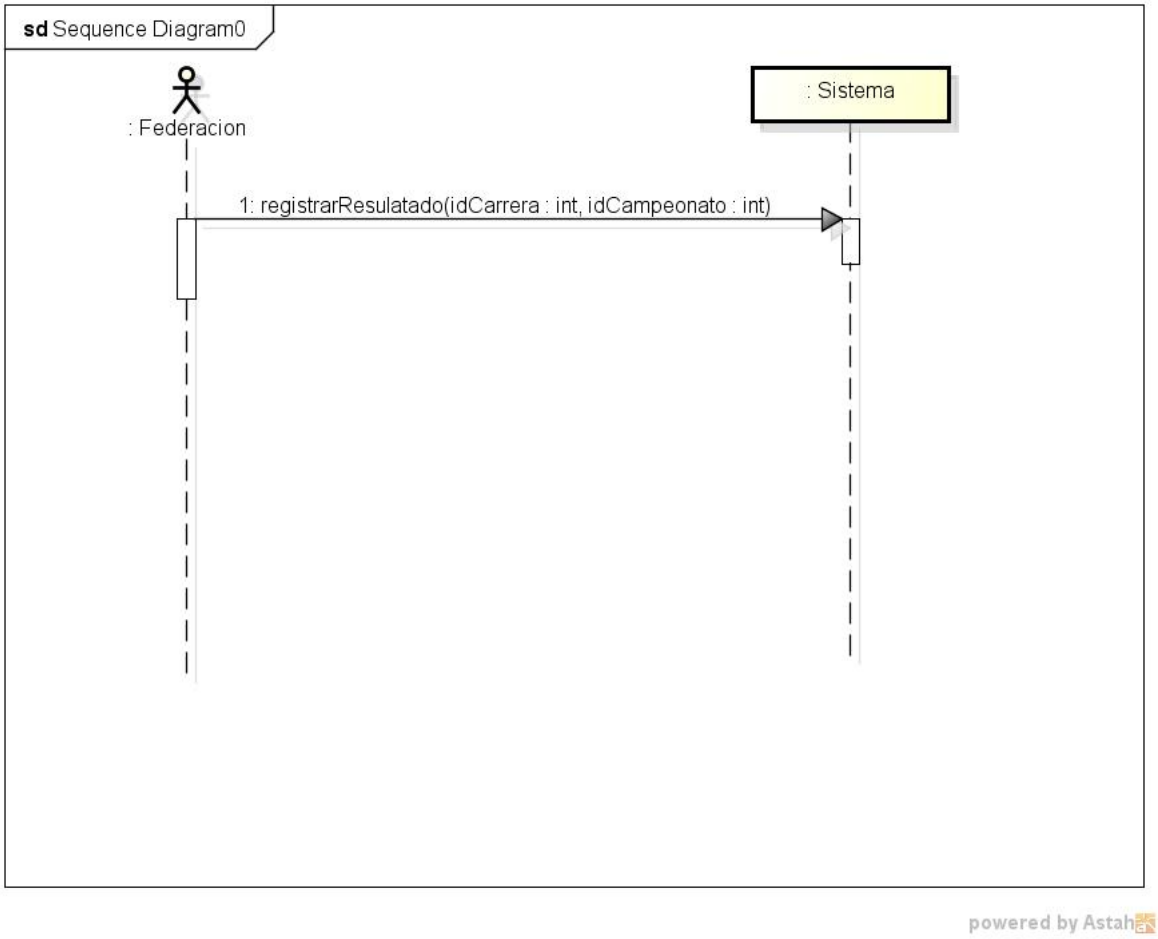


Figura 14: Diagrama de Secuencia del caso de uso 6

### 5.6.2. Contrato 1

- **Operación:** registrarResulatado(idCarrera : int, idCampeonato : int)
- **Controlador:**
- **Precondiciones:**
  - Existe una Carrera carrera con id idCarrera

- Existe un Campeonato c con id idCampeonato

- **Postcondiciones:**

  - Por cada Participación p de la Carrera carrera
    - Se creó un Diploma d
    - Se asoció el Diploma d con la Participacion p
    - Se estableció la puntuación de la Participación p como la suma de los puntos por tiempo más la suma de las puntuaciones de las Calificaciones de la Participación p
  - Se estableció la puntuación de cada Equipo del Campeonato c.

### 5.6.3. Diagrama de Colaboración Contrato 1

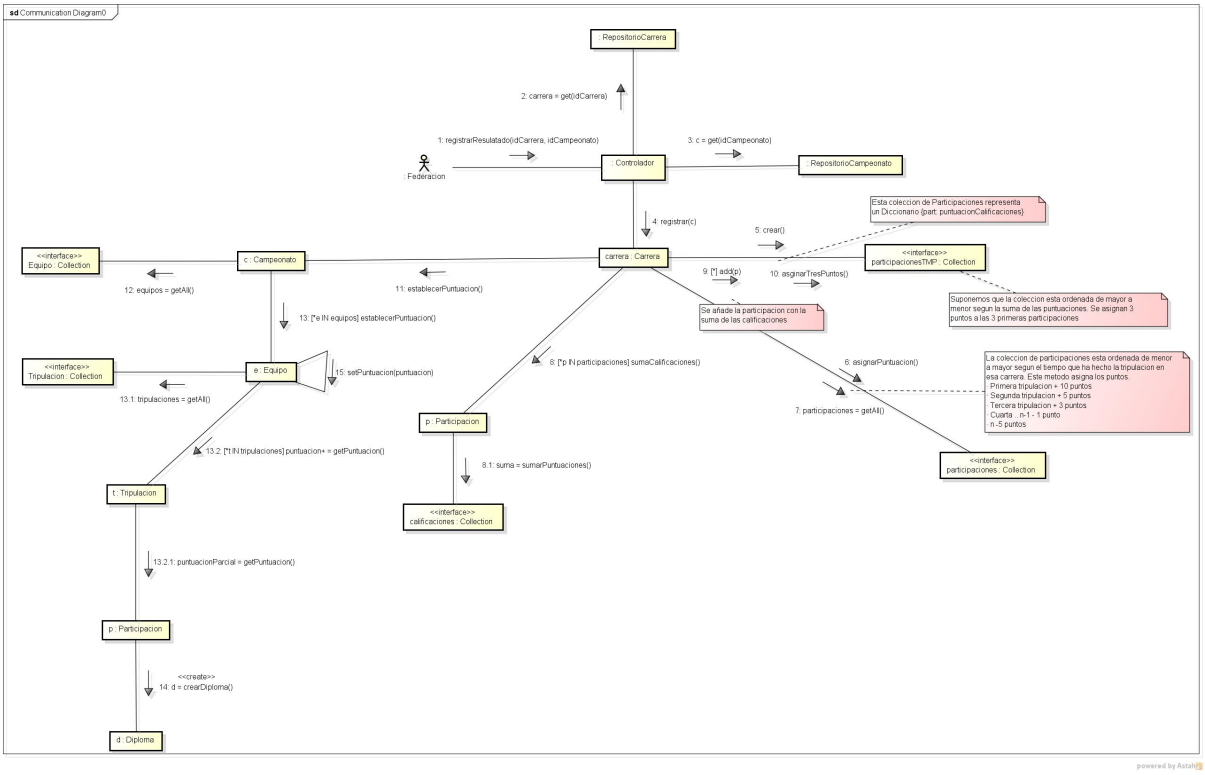


Figura 15: Diagrama de Colaboracion CdU 6. Contrato 1

## 5.7. Caso de Uso 7: Solicitar carrera final

**Actor Principal:** Federación

**Stakeholders:**

- Federación
- Tripulación
- Equipo

**Precondiciones:**

- Todas las carreras del campeonato de legocars han concluido.
- Las puntuaciones de los equipos están completamente actualizadas.

**Postcondiciones:**

- Se crea una última carrera
- Los equipos mejor clasificados disputan la última carrera del campeonato

**Escenario Principal de Éxito (o Flujo Básico):**

1. La Federación solicita al Sistema la celebración de una carrera final.
2. El Sistema crea una nueva carrera a 4 vueltas.
3. El Sistema obtiene los dos equipos mejor clasificados del campeonato.
4. El Sistema selecciona aleatoriamente un piloto de cada Equipo para disputar la última carrera del campeonato.

**Extensiones (o Flujos Alternativos):**

3a. Existen más de dos equipos empatados en las dos primeras posiciones de la clasificación

   *a*) El Sistema selecciona el equipo que tiene más Tripulaciones ganadoras

### 5.7.1. Diagrama de Secuencia

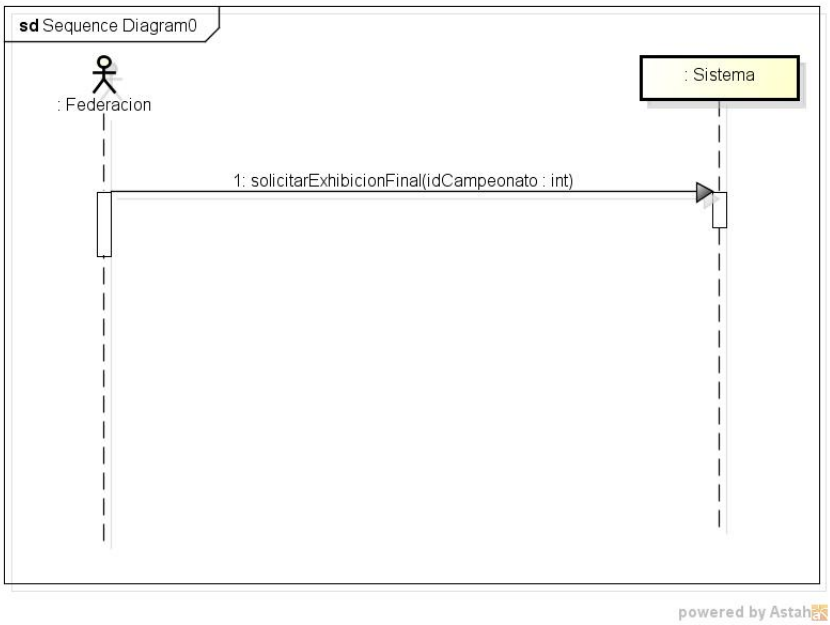


Figura 16: Diagrama de Secuencia del caso de uso 7

### 5.7.2. Contrato 1

- **Operación:** solicitarExhibicionFinal(idCampeonato : int)
- **Controlador:** ControladorExhibicionFinal
- **Precondiciones:**
  - Existe un Campeonato c cuyo identificador es idCampeonato.
- **Postcondiciones:**
  - Se creó una ExhibicionFinal cFinal
  - Se asoció cFinal con un Piloto p1 random del primer equipo en la clasificación del Campeonato c.
  - Se asoció cFinal con un Piloto p2 random del segundo equipo en la clasificación del Campeonato c.

### 5.7.3. Diagrama de Colaboración Contrato 1

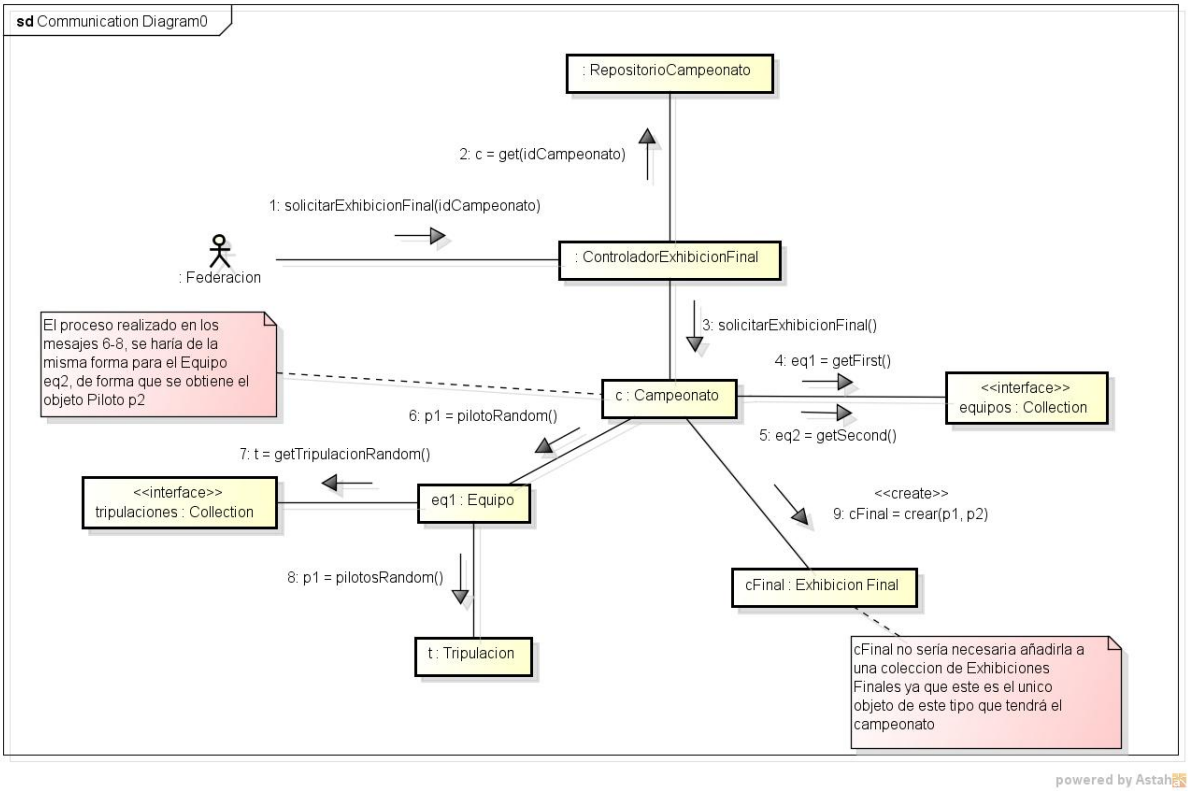


Figura 17: Diagrama de Colaboracion CdU 7. Contrato 1

## 5.8. Caso de Uso 8: Registrar resultado de la carrera final

**Actores Principal:** Federación

**Stakeholders:**

- Federación
- Equipo
- Tripulación
- Piloto

**Precondiciones:**

- La carrera final ha concluido.
- Se han obtenido los tiempos de cada piloto participante.

**Postcondiciones:**

- Los Equipos y Tripulaciones tienen sus puntuaciones actualizadas
- Los pilotos de los Equipos han recibido sus premios.

**Escenario Principal de Éxito (o Flujo Básico):**

1. La Federación empieza el registro del resultado de la carrera final.
2. El Sistema calcula el tiempo total de cada Piloto en la carrera como la suma de los tiempos en cada una de las cuatro vueltas.
3. El Sistema otorga 10 puntos a la Tripulación cuyo Piloto ha obtenido el menor tiempo en la carrera final.
4. El Sistema otorga 5 puntos a la Tripulación cuyo Piloto ha obtenido el mayor tiempo en la carrera final.
5. El Sistema calcula las puntuaciones totales de los Equipos como la suma de las puntuaciones de sus Tripulaciones.
6. El Sistema ordena los Equipos de mayor a menor puntuación.
7. El Sistema concede el primer premio al Equipo campeón.
8. El Sistema concede el segundo premio al Equipo subcampeón.
9. El Sistema comprueba la existencia de Pilotos desempleados en los dos Equipos finalistas y les proporciona una ayuda económica.

**Extensiones (o Flujo Alternativo):**

7-8a. El primer y el segundo Equipo están empatados a puntos.

*a*) El Equipo que gana la carrera final es el campeón de la competición.

9a. No hay ningún Piloto desempleado en ambos Equipos.

*a*) El Sistema no proporciona ninguna ayuda económica adicional a ningún Piloto.

### 5.8.1. Diagrama de Secuencia

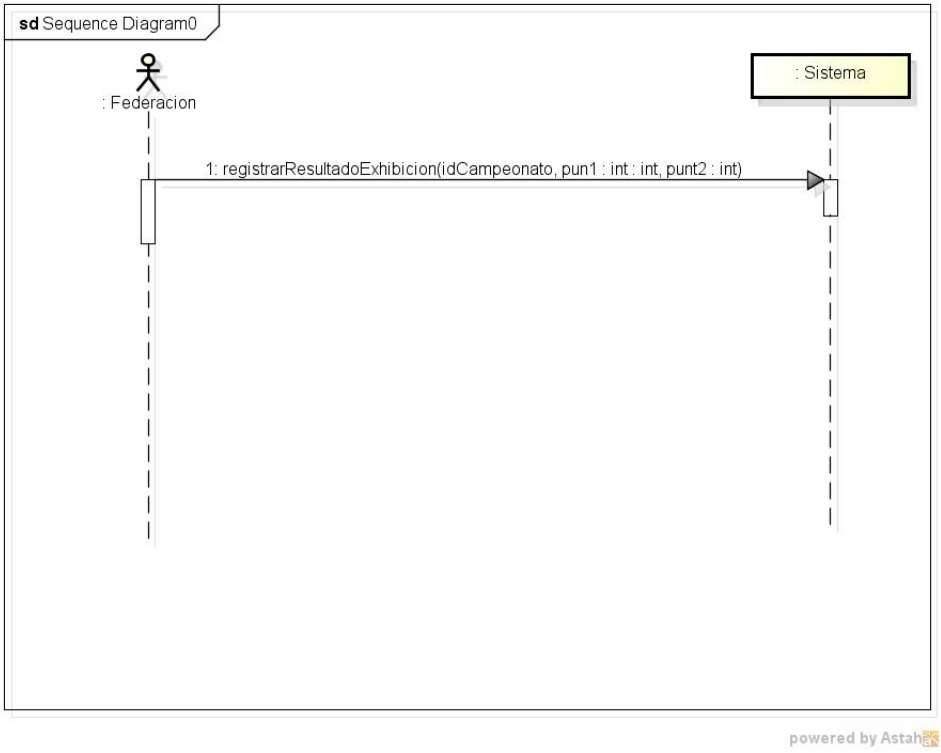


Figura 18: Diagrama de Secuencia del caso de uso 8

### 5.8.2. Contrato 1

- **Operación:** registrarResultadoExhibicion(idCampeonato, punt1 : int : int, punt2 : int )
- **Controlador:** ControladorExhibicionFinal
- **Precondiciones:**
  - Existe un Campeonato c cuyo identificador es idCampeonato.
- **Postcondiciones:**
  - Para los dos primeros equipos en la clasificación del Campeonato c:

- Se creó un Premio p1 y un Premio p2.
- Se asoció en Premio p1 con el Equipo eq1.
- Se asoció en Premio p2 con el Equipo eq2.

### 5.8.3. Diagrama de Colaboración Contrato 1

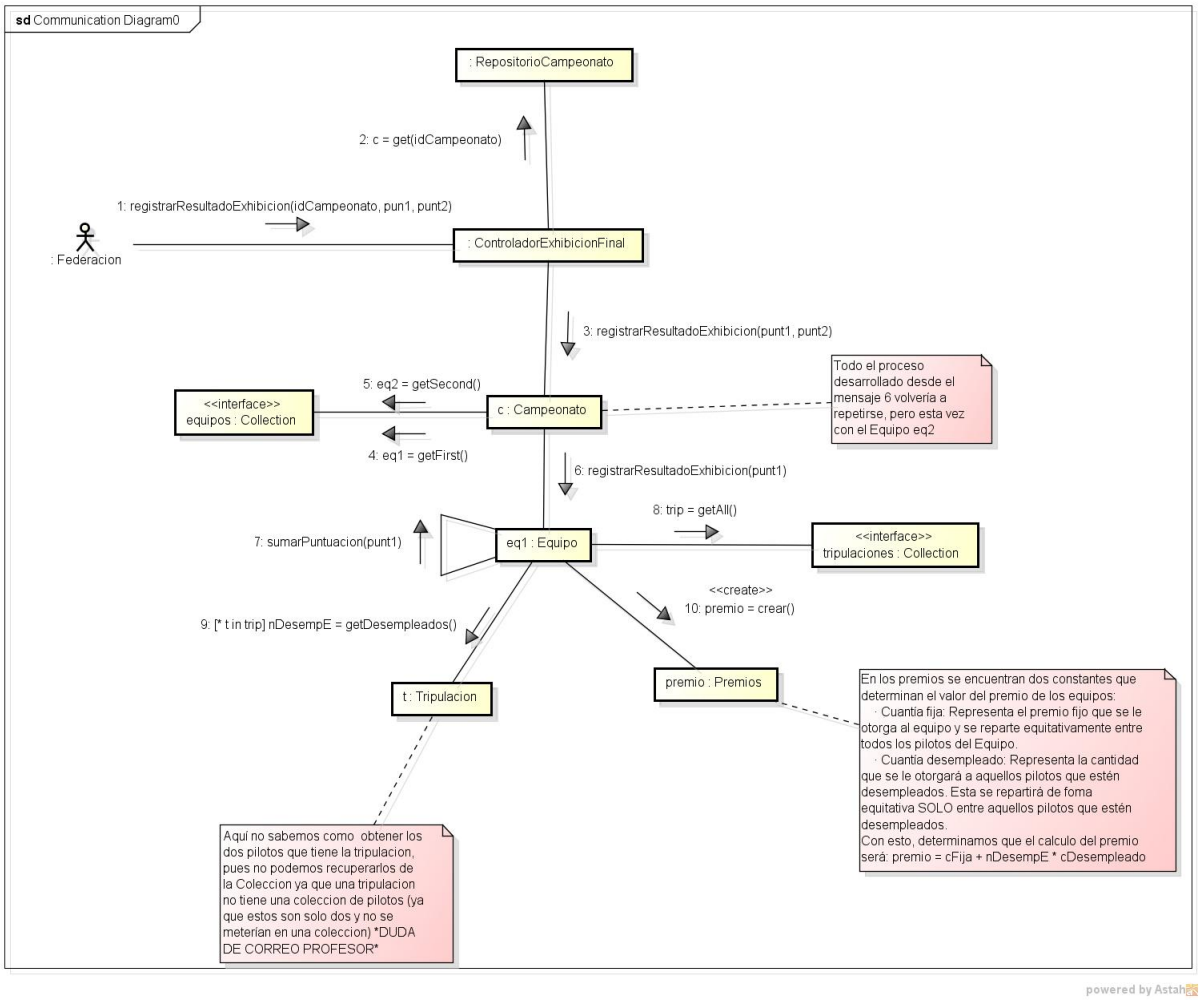


Figura 19: Diagrama de Colaboracion CdU 8. Contrato 1

# 6. Diagrama de clases del diseño

## 6.1. Diagrama de clases del diseño

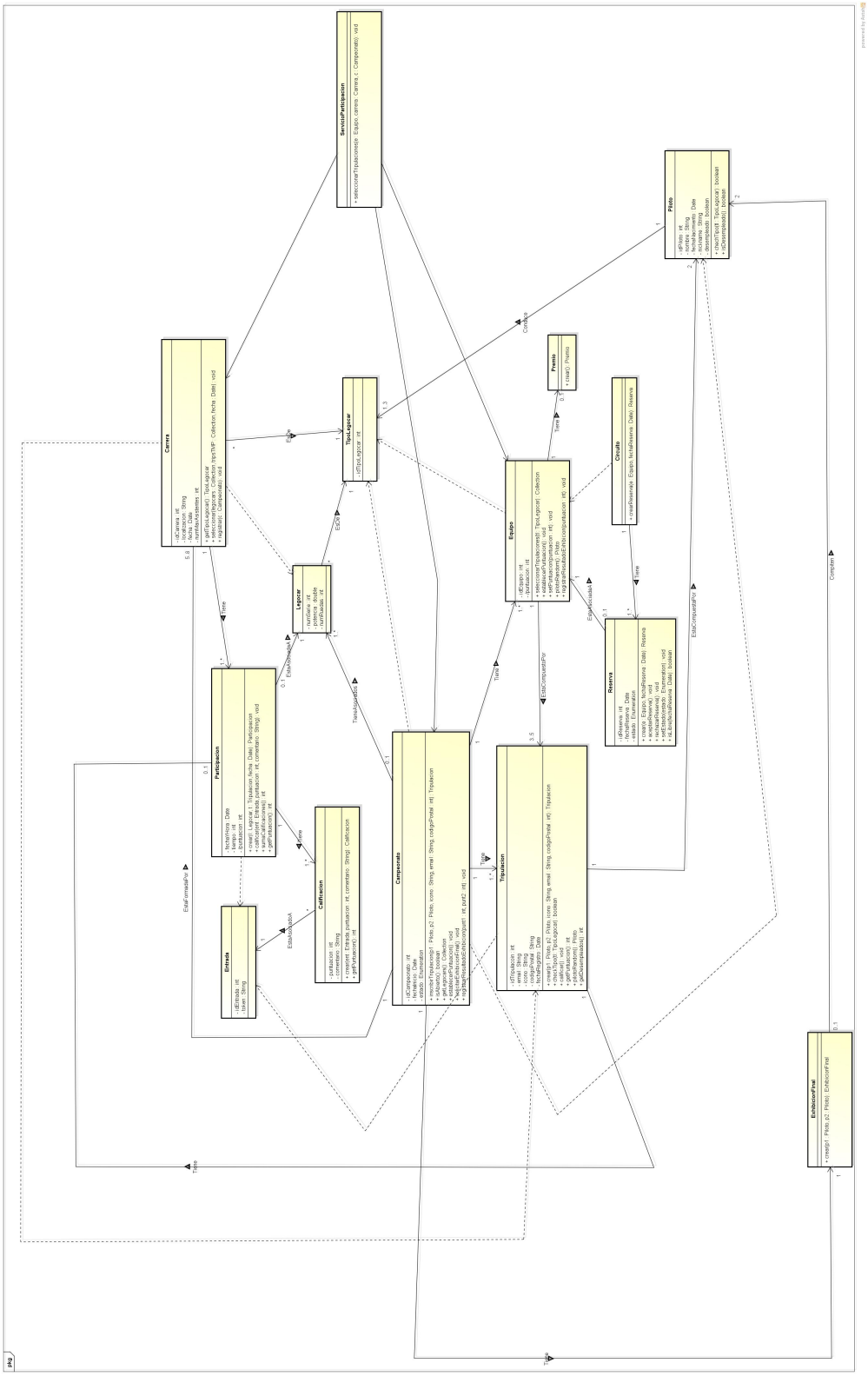


Figura 20: Diagrama de Clases del Diseño

## 6.2. Diagrama de controladores y repositorios

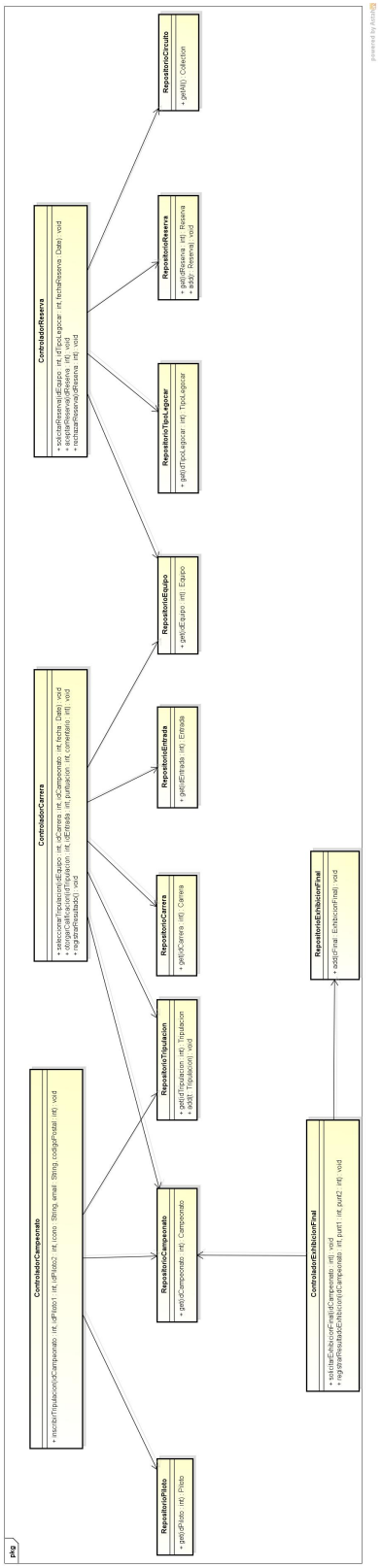


Figura 21: Diagrama de Clases del Diseño. Repositorios y Controladores

# 7. Diagramas de estado

## 7.1. Campeonato

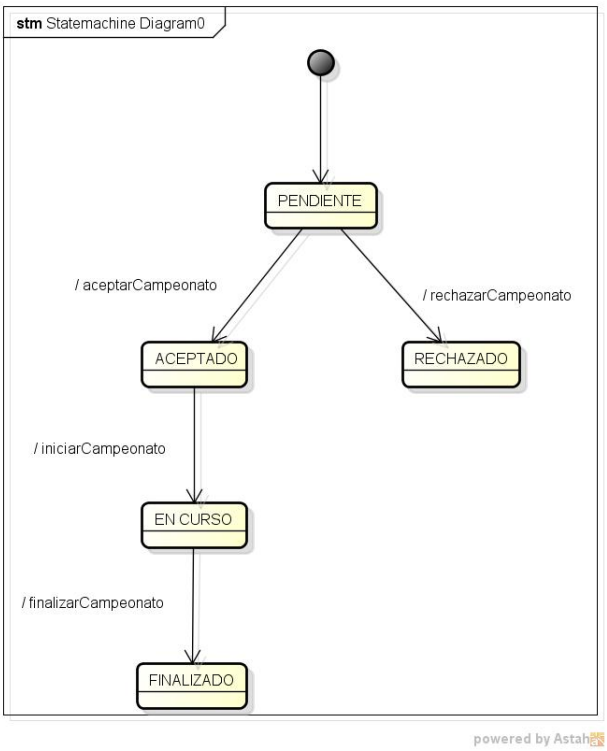


Figura 22: Diagrama de estados de Campeonato

## 7.2. Factura

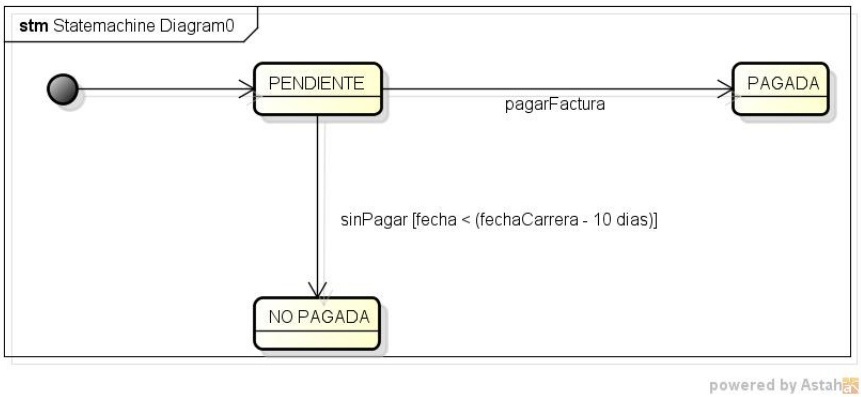


Figura 23: Diagrama de estados de Factura

## 7.3. Reserva

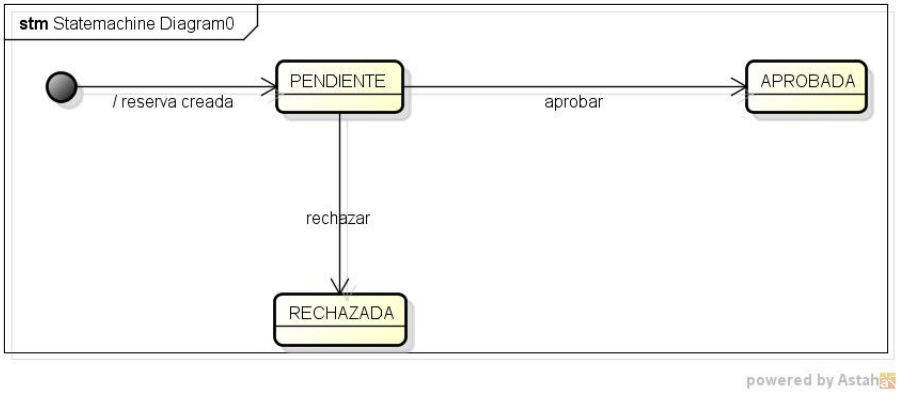


Figura 24: Diagrama de estados de Reserva

## 7.4. Carrera

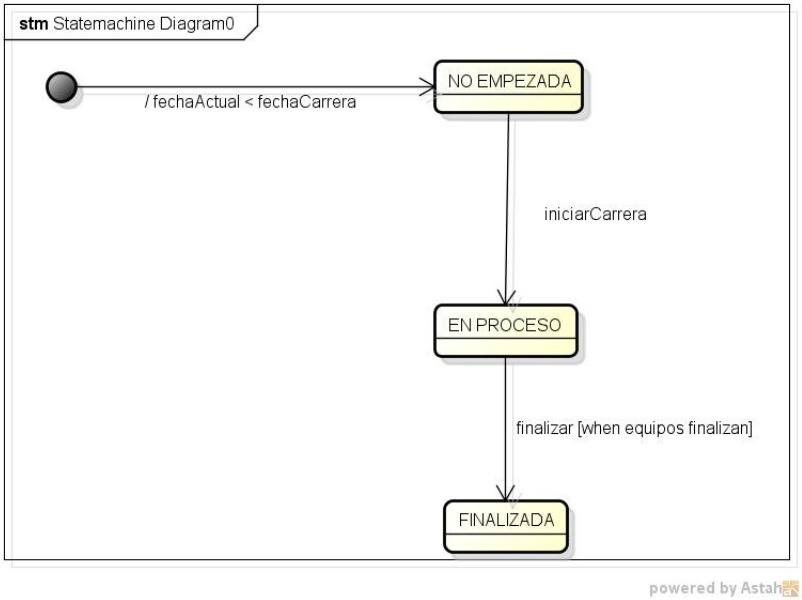


Figura 25: Diagrama de estados de Carrera

## 7.5. Comentario

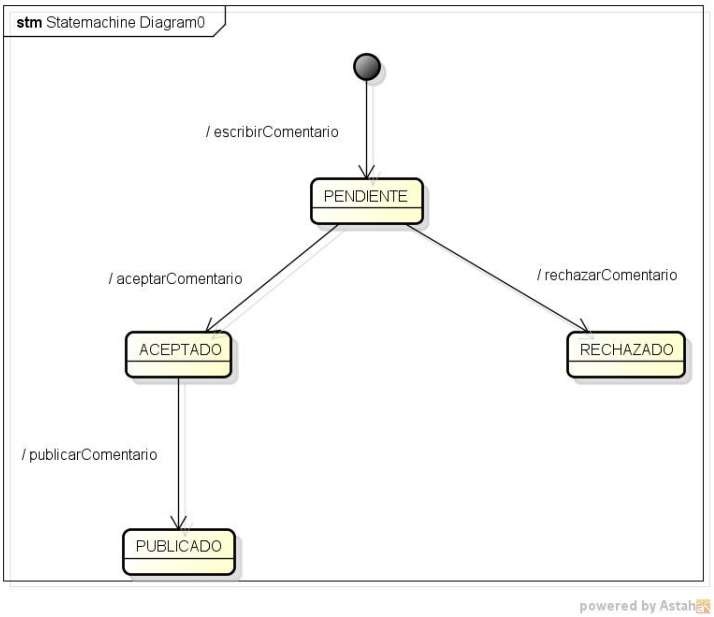


Figura 26: Diagrama de estados de Comentario

# 8. Persistencia

## 8.1. Enumerado Estado

```
package umu.pds.entidades;

public enum Estado {
    PENDIENTE, ACEPTADO, RECHAZADO

}
```

## 8.2. Clase Campeonato

```
package umu.pds.entidades;

import java.io.Serializable;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name = "campeonato")
public class Campeonato implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "fecha")
    private LocalDate fecha;


```

```
40      @Enumerated(EnumType.STRING)
41      @Column(name = "estado")
42      private Estado estado;
43
44      // Campeonato relacion 1 a Muchos con Carrera
45      @OneToMany(mappedBy = "campeonato", cascade = { CascadeType.ALL })
46      private List<Carrera> carreras = new ArrayList<Carrera>();
47
48      // Campeonato relacion 1 a Muchos con Tripulacion
49      @OneToMany(cascade = CascadeType.ALL)
50      @JoinColumn(name = "id_campeonato")
51      private List<Tripulacion> tripulaciones = new LinkedList<
            Tripulacion>();
52
53      // Campeonato relacion 1 a Muchos con Equipo
54      @OneToMany(cascade = CascadeType.ALL)
55      @JoinColumn(name = "id_campeonato")
56      private List<Equipo> equipos = new LinkedList<Equipo>();
57
58      // Relacion 1 a 1 con CarreraFinal
59      @OneToOne(cascade = CascadeType.ALL)
60      @JoinColumn(name = "id_carrera_final", referencedColumnName = "id")
61      private CarreraFinal carreraFinal;
62
63      // Constructores
64      public Campeonato() {
65
66      }
67
68      public Campeonato(LocalDate fecha, Estado estado) {
69          this.fecha = fecha;
70          this.estado = estado;
71      }
72
73      // Getters y Setters
74      public long getId() {
75          return id;
76      }
77
78      public void setId(long id) {
79          this.id = id;
80      }
81
82      public LocalDate getFecha() {
83          return fecha;
84      }
85
86      public void setFecha(LocalDate fecha) {
87          this.fecha = fecha;
88      }
89
90      public Estado getEstado() {
91          return estado;
92      }
93
94      public void setEstado(Estado estado) {
95          this.estado = estado;
96      }
```

```
97
98      public List<Carrera> getCarreras() {
99          return carreras;
100     }
101
102     public void setCarreras(List<Carrera> carreras) {
103         this.carreras = carreras;
104     }
105
106     public List<Tripulacion> getTripulaciones() {
107         return tripulaciones;
108     }
109
110     public void setTripulaciones(List<Tripulacion> tripulaciones) {
111         this.tripulaciones = tripulaciones;
112     }
113
114     public List<Equipo> getEquipos() {
115         return equipos;
116     }
117
118     public void setEquipos(List<Equipo> equipos) {
119         this.equipos = equipos;
120     }
121
122     public CarreraFinal getCarreraFinal() {
123         return carreraFinal;
124     }
125
126     public void setCarreraFinal(CarreraFinal carreraFinal) {
127         this.carreraFinal = carreraFinal;
128     }
129 }
```

## 8.3. Clase Carrera

```
1  package umu.pds.entidades;
2
3  import java.io.Serializable;
4  import java.time.LocalDate;
5  import java.util.LinkedList;
6  import java.util.List;
7
8  import javax.persistence.CascadeType;
9  import javax.persistence.Column;
10 import javax.persistence.Entity;
11 import javax.persistence.GeneratedValue;
12 import javax.persistence.GenerationType;
13 import javax.persistence.Id;
14 import javax.persistence.JoinColumn;
15 import javax.persistence.ManyToOne;
16 import javax.persistence.OneToMany;
17 import javax.persistence.Table;
18
```

```java
@Entity
@Table(name = "carrera")
public class Carrera implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "localizacion")
    private String localizacion;

    @Column(name = "fecha")
    private LocalDate fecha;

    @Column(name = "max_asistentes")
    private long numMaxAsistentes;

    // Relacion Muchos a 1 con Campeoanto
    @ManyToOne
    @JoinColumn(name = "id_campeonato")
    private Campeonato campeonato;

    // Relacion 1 a Muchos con Participacion
    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "id_carrera")
    private List<Participacion> participaciones = new LinkedList<
        Participacion>();

    public Carrera() {

    }

    // Getters y Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getLocalizacion() {
        return localizacion;
    }

    public void setLocalizacion(String localizacion) {
        this.localizacion = localizacion;
    }

    public LocalDate getFecha() {
        return fecha;
    }
```

```
76
77      public void setFecha(LocalDate fecha) {
78          this.fecha = fecha;
79      }
80
81      public long getNumMaxAsistentes() {
82          return numMaxAsistentes;
83      }
84
85      public void setNumMaxAsistentes(long numMaxAsistentes) {
86          this.numMaxAsistentes = numMaxAsistentes;
87      }
88
89      public Campeonato getCampeonato() {
90          return campeonato;
91      }
92
93      public void setCampeonato(Campeonato campeonato) {
94          this.campeonato = campeonato;
95      }
96
97      public List<Participacion> getParticipaciones() {
98          return participaciones;
99      }
100
101     public void setParticipaciones(List<Participacion> participaciones)
            {
102         this.participaciones = participaciones;
103     }
104
105 }
```

## 8.4. Clase Equipo

```
1  package umu.pds.entidades;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  import javax.persistence.*;
8
9
10 /**
11  * Entity implementation class for Entity: Equipo
12  *
13  */
14 @Entity
15 @Table(name = "equipo")
16 public class Equipo implements Serializable {
17
18     private static final long serialVersionUID = 1L;
19
20
```

```java

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "puntuacion")
    private long puntuacion;

    // Relacion 1 a Muchos con Tripulacion
    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "id_equipo")
    private List<Tripulacion> tripulaciones = new ArrayList<Tripulacion
        >(5);

    public Equipo() {
        super();
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public long getPuntuacion() {
        return puntuacion;
    }

    public void setPuntuacion(long puntuacion) {
        this.puntuacion = puntuacion;
    }

    public List<Tripulacion> getTripulaciones() {
        return tripulaciones;
    }

    public void setTripulaciones(List<Tripulacion> tripulaciones) {
        this.tripulaciones = tripulaciones;
    }

}
```

## 8.5. Clase Participacion

```java
package umu.pds.entidades;

import java.io.Serializable;
import java.time.LocalDateTime;

import javax.persistence.*;

```

```java
/**
 * Entity implementation class for Entity: Participacion
 *
 */
@Entity
@Table(name = "participacion")
public class Participacion implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "fecha_hora")
    private LocalDateTime fechaYHora;

    @Column(name = "tiempo")
    private long tiempo;

    @Column(name = "puntuacion")
    private long puntuacion;

    // Relacion 1 a 1 con Tripulacion
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "id_tripulacion", referencedColumnName = "id")
    private Tripulacion tripulacion;

    public Participacion() {
        super();
    }

    // Getters y Setters
    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public LocalDateTime getFechaYHora() {
        return fechaYHora;
    }

    public void setFechaYHora(LocalDateTime fechaYHora) {
        this.fechaYHora = fechaYHora;
    }

    public long getTiempo() {
        return tiempo;
    }

    public void setTiempo(long tiempo) {
        this.tiempo = tiempo;
    }

```

```
66      public long getPuntuacion() {
67          return puntuacion;
68      }
69
70      public void setPuntuacion(long puntuacion) {
71          this.puntuacion = puntuacion;
72      }
73
74      public Tripulacion getTripulacion() {
75          return tripulacion;
76      }
77
78      public void setTripulacion(Tripulacion tripulacion) {
79          this.tripulacion = tripulacion;
80      }
81
82  }
```

## 8.6. Clase Reserva

```
1   package umu.pds.entidades;
2
3   import java.io.Serializable;
4   import java.time.LocalDate;
5
6   import javax.persistence.*;
7
8   /**
9    * Entity implementation class for Entity: Reserva
10   *
11   */
12  @Entity
13  @Table(name = "reserva")
14  public class Reserva implements Serializable {
15
16      private static final long serialVersionUID = 1L;
17
18      @Id
19      @GeneratedValue(strategy = GenerationType.IDENTITY)
20      @Column(name = "id")
21      private long id;
22
23      @Column(name = "fecha")
24      private LocalDate fecha;
25
26      @Column(name = "estado")
27      @Enumerated(EnumType.STRING)
28      private Estado estado;
29
30      // Relacion unidireccional 1 a 1 con Equipo
31      @OneToOne(cascade = CascadeType.ALL)
32      @JoinColumn(name = "id_equipo", referencedColumnName = "id")
33      private Equipo equipo;
34
```

```
35      public Reserva() {
36          super();
37      }
38
39      // Getters y Setters
40      public long getId() {
41          return id;
42      }
43
44      public void setId(long id) {
45          this.id = id;
46      }
47
48      public LocalDate getFecha() {
49          return fecha;
50      }
51
52      public void setFecha(LocalDate fecha) {
53          this.fecha = fecha;
54      }
55
56      public Estado getEstado() {
57          return estado;
58      }
59
60      public void setEstado(Estado estado) {
61          this.estado = estado;
62      }
63
64  }
```

## 8.7. Clase Piloto

```
1   package umu.pds.entidades;
2
3   import java.io.Serializable;
4   import java.time.LocalDate;
5
6   import javax.persistence.*;
7
8   /**
9    * Entity implementation class for Entity: Piloto
10   *
11   */
12  @Entity
13  @Table(name = "piloto")
14  @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
15  @DiscriminatorColumn(name = "tipo_piloto", discriminatorType =
        DiscriminatorType.STRING)
16  public abstract class Piloto implements Serializable {
17
18      private static final long serialVersionUID = 1L;
19
20
```

```
21      @Id
22      @GeneratedValue(strategy = GenerationType.IDENTITY)
23      @Column(name = "id")
24      private long id;
25
26      @Column(name = "nombre")
27      private String nombre;
28
29      @Column(name = "fecha_nacimiento")
30      private LocalDate fechaNacimiento;
31
32      @Column(name = "nickname")
33      private String nickname;
34
35      @Column(name = "desempleado")
36      private Boolean desempleado;
37
38      public Piloto() {
39          super();
40      }
41
42      // Getters y Setters
43
44      public long getId() {
45          return id;
46      }
47
48      public void setId(long id) {
49          this.id = id;
50      }
51
52      public String getNombre() {
53          return nombre;
54      }
55
56      public void setNombre(String nombre) {
57          this.nombre = nombre;
58      }
59
60      public LocalDate getFechaNacimiento() {
61          return fechaNacimiento;
62      }
63
64      public void setFechaNacimiento(LocalDate fechaNacimiento) {
65          this.fechaNacimiento = fechaNacimiento;
66      }
67
68      public String getNickname() {
69          return nickname;
70      }
71
72      public void setNickname(String nickname) {
73          this.nickname = nickname;
74      }
75
76      public Boolean getDesempleado() {
77          return desempleado;
78      }
```

```
79
80      public void setDesempleado(Boolean desempleado) {
81          this.desempleado = desempleado;
82      }
83
84  }
```

## 8.8. Clase Senior

```
1   package umu.pds.entidades;
2
3   import javax.persistence.*;
4
5   /**
6    * Entity implementation class for Entity: Senior
7    *
8    */
9   @Entity
10  @DiscriminatorValue("senior")
11  public class Senior extends Piloto {
12
13      private static final long serialVersionUID = 1L;
14
15      public Senior() {
16          super();
17      }
18  }
```

## 8.9. Clase Junior

```
1   package umu.pds.entidades;
2
3   import javax.persistence.DiscriminatorValue;
4   import javax.persistence.Entity;
5
6   @Entity
7   @DiscriminatorValue("junior")
8   public class Junior extends Piloto {
9
10      /**
11       *
12       */
13      private static final long serialVersionUID = 1L;
14
15      public Junior() {
16          super();
17      }
18  }
```

## 8.10. Clase Tripulacion

```java
package umu.pds.entidades;

import java.io.Serializable;
import java.time.LocalDate;

import javax.persistence.*;

/**
 * Entity implementation class for Entity: Tripulacion
 *
 */
@Entity
@Table(name = "tripulacion")
public class Tripulacion implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "icono")
    private String icono;

    @Column(name = "email")
    private String email;

    @Column(name = "codigo_postal")
    private Integer codigoPostal;

    @Column(name = "fecha_registro")
    private LocalDate fechaRegistro;

    // 1 a Muchos Piloto

    // Relacion Uno a Uno bidireccional con Participacion
    @OneToOne(mappedBy = "tripulacion", cascade = CascadeType.ALL)
    private Participacion participacion;

    // Relacion Uno a Uno unidireccional con Piloto
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "id_piloto1", referencedColumnName = "id")
    private Piloto piloto1;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "id_piloto2", referencedColumnName = "id")
    private Piloto piloto2;

    public Tripulacion() {
        super();
    }

    // Getters y Setters


```

```
57      public long getId() {
58          return id;
59      }
60
61      public void setId(long id) {
62          this.id = id;
63      }
64
65      public String getIcono() {
66          return icono;
67      }
68
69      public void setIcono(String icono) {
70          this.icono = icono;
71      }
72
73      public String getEmail() {
74          return email;
75      }
76
77      public void setEmail(String email) {
78          this.email = email;
79      }
80
81      public Integer getCodigoPostal() {
82          return codigoPostal;
83      }
84
85      public void setCodigoPostal(Integer codigoPostal) {
86          this.codigoPostal = codigoPostal;
87      }
88
89      public LocalDate getFechaRegistro() {
90          return fechaRegistro;
91      }
92
93      public void setFechaRegistro(LocalDate fechaRegistro) {
94          this.fechaRegistro = fechaRegistro;
95      }
96
97      public Participacion getParticipacion() {
98          return participacion;
99      }
100
101     public void setParticipacion(Participacion participacion) {
102         this.participacion = participacion;
103     }
104
105     public Piloto getPiloto1() {
106         return piloto1;
107     }
108
109     public void setPiloto1(Piloto piloto1) {
110         this.piloto1 = piloto1;
111     }
112
113
114
```

```
115     public Piloto getPiloto2() {
116         return piloto2;
117     }
118 
119     public void setPiloto2(Piloto piloto2) {
120         this.piloto2 = piloto2;
121     }
122 
123 }
```

## 8.11. Clase CarreraFinal

```
1  package umu.pds.entidades;
2  
3  import java.io.Serializable;
4  import javax.persistence.*;
5  
6  /**
7   * Entity implementation class for Entity: CarreraFinal
8   *
9   */
10 @Entity
11 @Table(name = "carrera_final")
12 public class CarreraFinal implements Serializable {
13 
14     private static final long serialVersionUID = 1L;
15 
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     @Column(name = "id")
19     private long id;
20 
21     // Relacion Uno a Uno unidireccional con Piloto
22     @OneToOne(cascade = CascadeType.ALL)
23     @JoinColumn(name = "id_piloto1", referencedColumnName = "id")
24     private Piloto piloto1;
25 
26     @OneToOne(cascade = CascadeType.ALL)
27     @JoinColumn(name = "id_piloto2", referencedColumnName = "id")
28     private Piloto piloto2;
29 
30     public CarreraFinal() {
31         super();
32     }
33 
34     public long getId() {
35         return id;
36     }
37 
38     public void setId(long id) {
39         this.id = id;
40     }
41 
42 
```

```
43      public Piloto getPiloto1() {
44          return piloto1;
45      }
46
47      public void setPiloto1(Piloto piloto1) {
48          this.piloto1 = piloto1;
49      }
50
51      public Piloto getPiloto2() {
52          return piloto2;
53      }
54
55      public void setPiloto2(Piloto piloto2) {
56          this.piloto2 = piloto2;
57      }
58
59  }
```