Primer entregable: Documentos de diseño

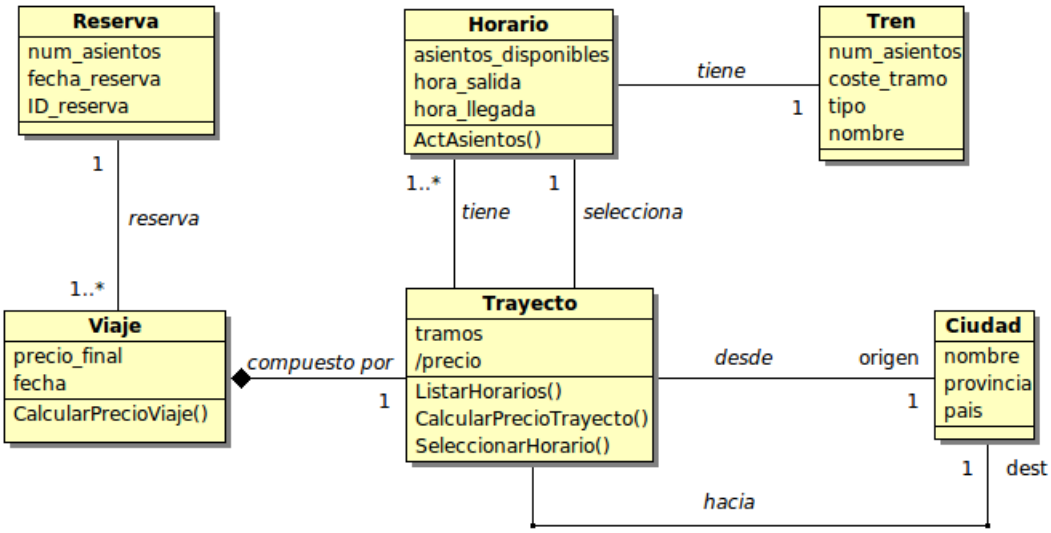
Manuel Jesús de la Calle Brihuega
Miguel Ángel Pérez Montero

Diseño de Sistemas Software

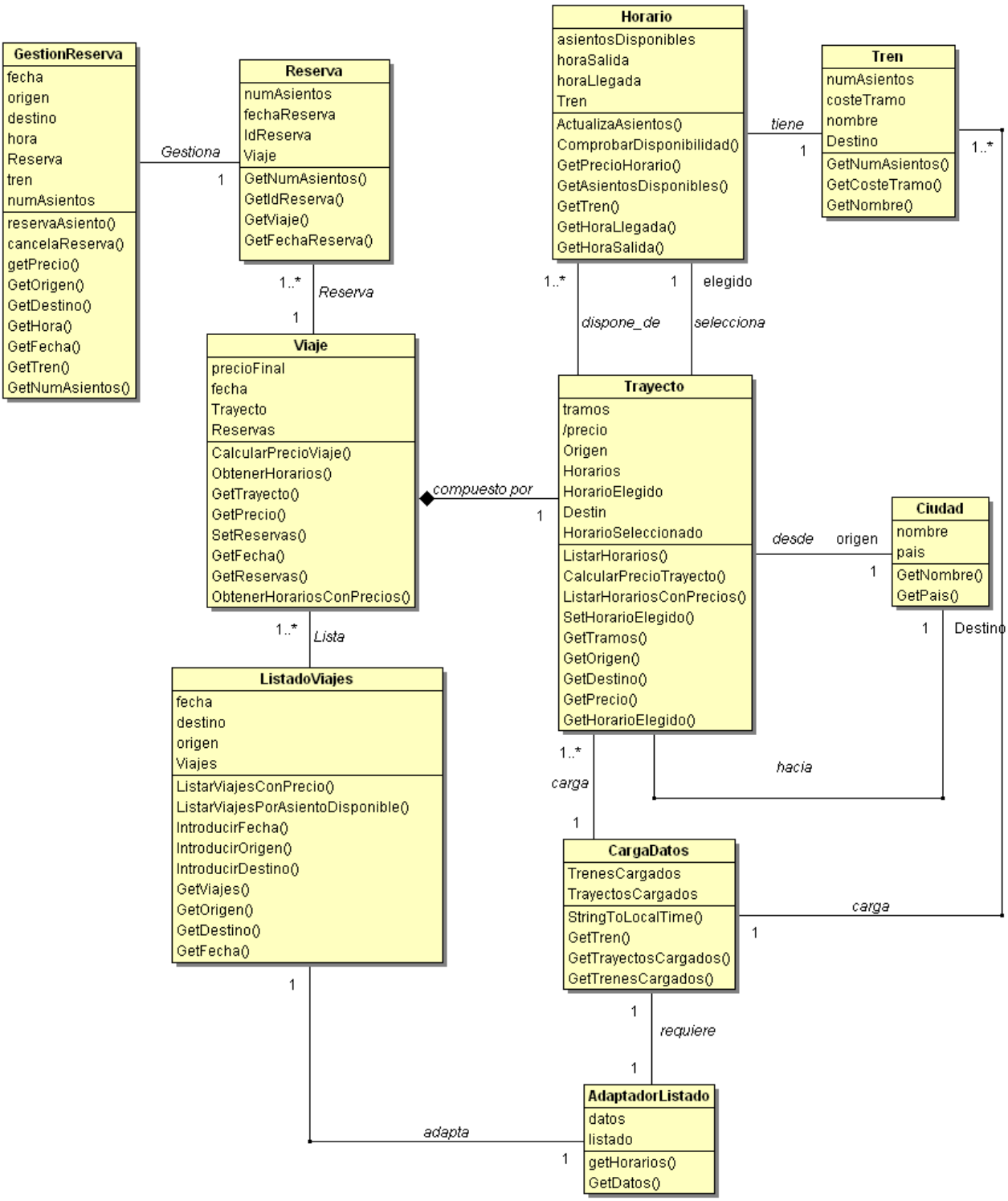
Curso 2010/2011

1. Diagramas UML de clase del sistema general

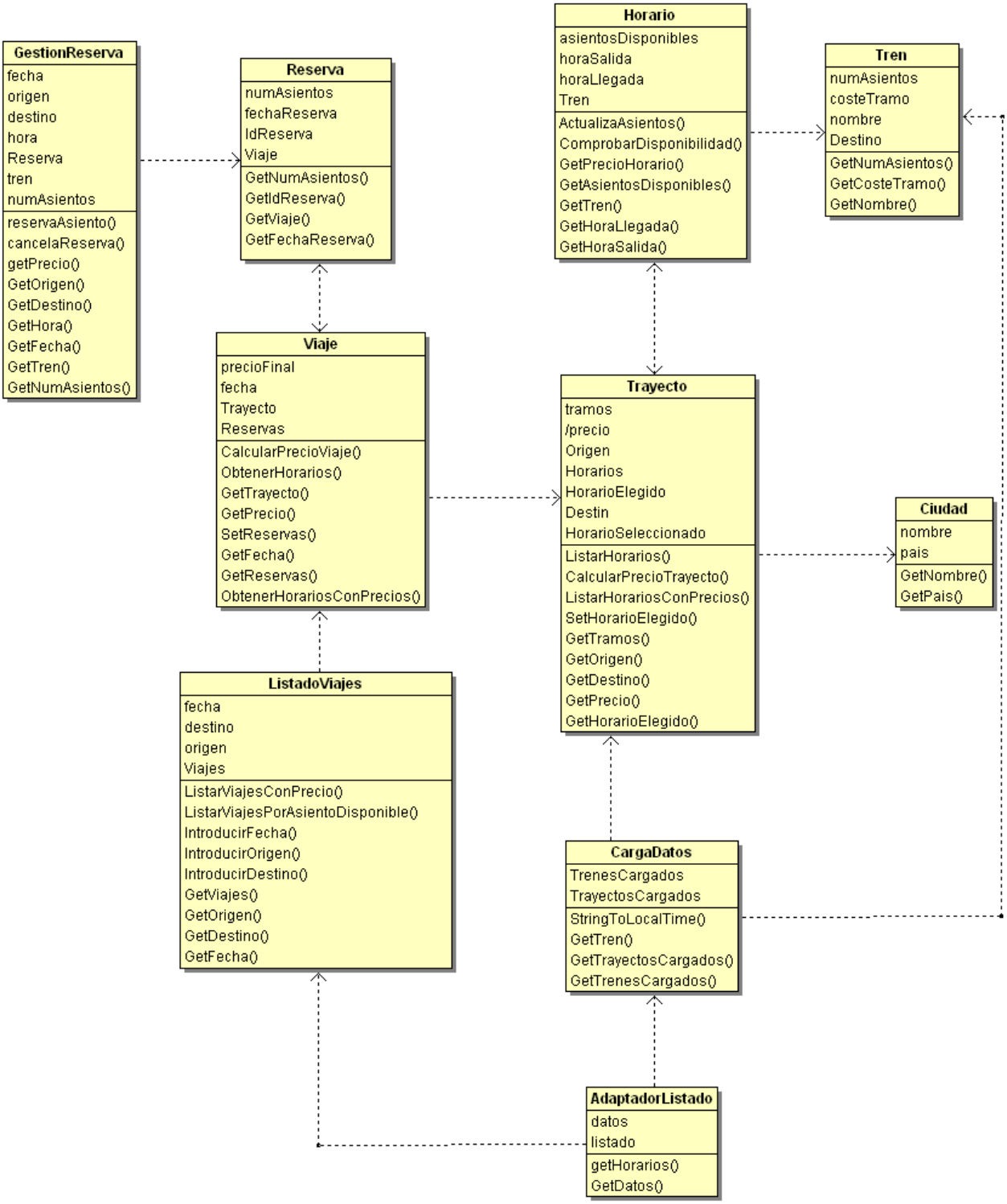
1.1. Diagrama de clases, resultado del análisis:



1.2. Diagrama de clases de diseño:



1.3. Diagrama de dependencias:



2. Explicación del significado de las distintas clases

2.1. Clase *Ciudad*:

Representa la ciudad origen o destino del viaje.

Descripción:

Clase encargada de representar una ciudad en el sistema de viajes de tren, identificándola en el mismo mediante su nombre. También guarda la provincia y el país al que pertenece.

Atributos:

- nombre: Contiene el nombre de la ciudad, que sirve a la vez, como identificador de la misma en el sistema. Declaración:
  - Uml: - nombre : string

- país: Contiene el país al que pertenece la ciudad. Declaración:
  - Uml: - pais : string

Métodos:

- GetNombre: Método consultor de nombre. Devuelve un string con el nombre de la ciudad. Declaración:
  - Uml: + GetNombre() : string

- GetPais: Método consultor de país. Devuelve un string con el nombre del país al que pertenece la ciudad. Declaración:
  - Uml: + GetPais() : String

2.2. Clase *Tren*:

Representa al tren que realiza el viaje.

Descripción:

Clase encargada de representar al tren que realizará el viaje determinado en el sistema. En ella se describen aspectos fundamentales del mismo, como puede ser su nombre, que lo identificará, su número de asientos, el tipo o el coste por tramo viajado.

Atributos:

- nombre: Contiene el nombre del tren, que sirve a la vez, como identificador del mismo en el sistema. Declaración:
  - Uml: - nombre : string

- numAsientos: Contiene el número de asientos del tren. Declaración:
  - Uml: - numAsientos : int

- costeTramo: Contiene el precio por tramo viajado. Declaración:
  - Uml: - coste_tramo : float

Métodos:

- GetNumAsientos: Método consultor de asientos. Devuelve el número de asientos del tren. Declaración:
    - Uml: + GetNumAsientos() : int

- GetCosteTramo: Método consultor de precio. Devuelve el precio por tramo viajado en el tren. Declaración:
    - Uml: + GetCosteTramo() : float

- GetNombre: Método consultor del nombre. Devuelve el nombre del tren. Declaración:
    - Uml: + GetNombre() : String

2.3. Clase *Horario*:

Representa el horario previsto para un viaje en el sistema.

Descripción:

Clase encargada de representar el horario de viaje previsto para un tren en el sistema. Describe tanto la hora de salida como la hora de llegada del mismo.

Atributos:

- asientosDisponibles: Refleja los asientos disponibles para viajar en el horario de un tren determinado. Declaración:
    - Uml: - asientosDisponibles : int

- horaLlegada: Contiene la hora de llegada prevista para un tren que realizará un trayecto perteneciente a un viaje. Declaración:
    - Uml: - horaLlegada : string

- horaSalida: Contiene la hora de salida prevista para un tren que realizará un trayecto perteneciente a un viaje. Declaración:
    - Uml: - horaSalida : string

- tren: Contiene una referencia al objeto tren que cubre los viajes en dicho horario. Declaración:
    - Uml: - tren : Tren

Métodos:

- GetAsientosDisponibles: Método consultor de disponibilidad. Devuelve el número de asientos disponibles para el horario. Declaración:
    - Uml: + GetAsientosDisponibles() : int

- GetTren: Método consultor de tren. Devuelve el tren que realiza el trayecto para este horario. Declaración:
    - Uml: + GetTren() : Tren

- GetHoraLlegada: Método consultor de hora. Devuelve la hora de llegada prevista para el tren en este horario. Declaración:
    - Uml: + GetHoraLlegada() : LocalTime

- GetHoraSalida: Método consultor de hora. Devuelve la hora de salida prevista para el tren en este horario. Declaración:
    - Uml: + GetHoraSalida() : LocalTime

- ComprobarDisponibilidad: Comprueba si existen asientos disponibles para el horario. Devuelve un booleano indicando su resultado. Declaración:
    - Uml: + ComprobarDisponibilidad() : boolean

- ActualizaAsientos: Actualiza el número de asientos libres para el horario. Decrementa o incrementa el atributo asientos_disponibles con el número de asientos reservados o cancelados('asientosReservados'). Declaración:
    - Uml: - ActualizaAsientos(in asientosReservados : int) : void

- GetPrecioHorario: Devuelve el precio de un horario, que no es más que el precio por tramo del tren que cubre. No devuelve un precio final, solo el precio del tramo viajado. Declaración:
    - Uml: + GetPrecioHorario() : float

2.4. Clase *Trayecto*:

Representa un trayecto realizado en el viaje.

Descripción:

Clase encargada de representar un trayecto a realizar por un tren dentro de un viaje. Se identifica con la ciudad origen y destino del mismo, y forma parte de un viaje. Su distancia se mide mediante tramos.

Atributos:

- tramos: Valor numérico que refleja la distancia a recorrer entre las dos ciudades, origen y destino. Declaración:
    - Uml: - tramos : int

- Origen: Contiene una referencia al objeto ciudad que es la ciudad origen del trayecto. Declaración:
    - Uml: - Origen : Ciudad

- Destino: Contiene una referencia al objeto ciudad que es la ciudad destino del trayecto. Declaración:
    - Uml: - Destino : Ciudad

- Horarios: Contiene un conjunto de referencias a objetos horarios en los que se realiza el trayecto. Declaración:
    - Uml: - Horarios : ArrayList<Horario>

- HorarioElegido: Contiene una referencia al objeto horario, elegido por el usuario para realizar su viaje. Declaración:
    - Uml: - HorarioElegido : Horario

- HorarioSeleccionado: Contiene un booleano que indica si el trayecto tiene ya un horario elegido o no. Declaración:
    - Uml: - HorarioSeleccionado : boolean

- /precio: Atributo derivado resultante de la multiplicación del precio por tramo para un tren que cubre un determinado horario elegido por el número de tramos que mide el trayecto. Declaración:
    - Uml: - precio : float

Métodos:

- ListarHorarios: Lista y devuelve un conjunto de horarios, que son los previstos para el trayecto. Declaración:
    - Uml: + ListarHorarios() : ArrayList<Horario>

- GetTramos: Método consultor de tramos. Devuelve el número de tramos que componen el trayecto. Declaración:
    - Uml: + GetTramos() : int

- GetOrigen: Método consultor de ciudad origen. Devuelve la ciudad origen del trayecto. Declaración:
    - Uml: + GetOrigen() : Ciudad

- GetDestino: Método consultor de ciudad destino. Devuelve la ciudad destino del trayecto. Declaración:
    - Uml: + GetDestino() : Ciudad

- CalcularPrecioTrayecto: Calcula y devuelve el precio del trayecto, en función del coste por tramo del tren asociado al horario que recibe por parámetro. Declaración:
    - Uml: + CalcularPrecioTrayecto(in HorarioElegido : Horario) : float

- SetHorarioElegido: Selecciona el horario que el usuario ha elegido de entre todos, para realizar el trayecto. Declaración:
    - Uml: - SetHorarioElegido(in valor : Horario) : void

- GetPrecio: Método consultor del precio. Devuelve el precio que tiene el trayecto. Declaración:
    - Uml: + GetPrecio() : Float

- ListarHorariosConPrecios: Lista y devuelve un conjunto de pares horario-precio, es decir, la lista de horarios disponibles para el trayecto y su precio asociado. Declaración:
    - Uml: + ListarHorarioConPrecios() : Map<Horario, float>

- GetHorarioElegido: Devuelve el horario elegido por el usuario para viajar. Declaración:
    - Uml: + GetHorarioElegido() : Horario

2.5. Clase *Viaje*:

Representa un viaje de tren.

Descripción:

Clase encargada de representar un viaje en el sistema. En ella se describe el trayecto a realizar por un determinado tren, en una fecha y hora concretas, además de su precio.

Atributos:

- precioFinal: Contiene el precio final del viaje, establecido por el método 'CalcularPrecioViaje'. Declaración:
  - Uml: - precioFinal : float

- fecha: Contiene la fecha en la que se realizará el viaje. Declaración:
  - Uml: - fecha : LocalDate

- Reservas: Contiene un conjunto de referencias a objetos Reserva, que serán las distintas reservas realizadas por los usuarios para ese determinado viaje. Declaración:
  - Uml: - Reservas : ArrayList<Reserva>

- Trayecto: Contiene la referencia al objeto Trayecto del cual se compone el viaje. Declaración:
  - Uml: - trayecto : Trayecto

Métodos:

- CalcularPrecioViaje: Calcula el precio final del viaje en función del precio del trayecto, y el número de trayectos que componen el viaje. Establece dicho precio en el atributo 'precioFinal'. Declaración:
  - Uml: - CalcularPrecioViaje() : void

- SetReservas: Método modificador. Establece la reserva, o conjunto de reservas realizadas previamente para el viaje. Declaración:
  - Uml: - SetReservas(in valor : ArrayList<Reserva>) : void

- GetFecha: Método consultor de la fecha. Devuelve la fecha en la que se realizará el viaje. Declaración:
  - Uml: + GetFecha() : LocalDate

- GetPrecio: Devuelve el precio del viaje. Declaración:
  - Uml: + GetPrecio() : float

- GetReservas: Método consultor. Devuelve el conjunto de reservas realizadas para el viaje. Declaración:
  - Uml: + GetReservas() : ArrayList<Reserva>

- ObtenerHorarios: Lista los horarios disponibles para realizar el viaje. Declaración:
  - Uml: + ObtenerHorarios() : ArrayList<Horario>

- ObtenerHorariosConPrecios: Lista los horarios disponibles para realizar el viaje, con sus precios correspondientes. Declaración:
    - Uml: + ObtenerHorariosConPrecios() : Map<Horario, Float>

- GetTrayecto: Devuelve el trayecto que compondrá el viaje. Declaración:
    - Uml: + GetTrayecto() : Trayecto

2.6. Clase *Reserva*:

Representa la reserva de un viaje en el sistema.

Descripción:

Clase encargada de representar una reserva de un número de asientos determinado para un viaje concreto. La reserva se identificará por un código 'ID_reserva', y también se almacenará la fecha en la que se realice la misma.

Atributos:

- numAsientos: Contiene el número de asientos a reservar para el viaje. Declaración:
    - Uml: - numAsientos : int

- fechaReserva: Contiene la fecha en la que se realiza la reserva del viaje. Declaración:
    - Uml: - fechaReserva : LocalDate

- idReserva: Contiene el código identificador de la reserva en el sistema. Declaración:
    - Uml: - idReserva : string

- viaje: Contiene la referencia al objeto Viaje del cual se ha realizado la reserva. Declaración:
    - Uml: - viaje : Viaje

Métodos:

- GetNumAsientos: Método consultor de asientos. Devuelve el número de asientos reservados. Declaración:
    - Uml: + GetNumAsientos () : int

- GetIdReserva: Método consultor del id de la reserva. Devuelve un string con el identificador de la reserva. Declaración:
    - Uml: + GetIdReserva () : String

- GetViaje: Método consultor de viaje. Devuelve el viaje del cual se realiza la reserva. Declaración:
    - Uml: + GetViaje () : Viaje

- GetFechaReserva: Método consultor de fecha. Devuelve la fecha de la reserva del viaje. Declaración:
    - Uml: + GetFechaReserva () : LocalDate

2.7. Clase *ListadoViajes*:

Genera y representa un listado de viajes.

Descripción:

Clase encargada de generar y representar un listado de los viajes disponibles para una fecha determinada entre una ciudad origen y una ciudad destino.

Atributos:

- fecha: Contiene la fecha para la que se desea generar un listado de viajes. Declaración:
  - Uml: - fecha : LocalDate

- origen: Contiene la ciudad que se desea que sea el origen de los viajes a listar. Declaración:
  - Uml: - origen : Ciudad

- destino: Contiene la ciudad que se desea que sea el destino de los viajes a listar. Declaración:
  - Uml: - destino : Ciudad

- Viajes: Contiene un conjunto de referencias a objetos Viajes, que serán los viajes a listar. Declaración:
  - Uml: - Viajes : ArrayList<Viaje>

Métodos:

- GetOrigen: Método consultor de ciudad origen. Devuelve la ciudad origen de los viajes para los cuales se ha generado el listado. Declaración:
  - Uml: + GetOrigen() : Ciudad

- GetDestino: Método consultor de ciudad destino. Devuelve la ciudad destino de los viajes para los cuales se ha generado el listado. Declaración:
  - Uml: + GetDestino() : Ciudad

- GetFecha: Método consultor de fecha. Devuelve la fecha para la cual se ha pedido el listado de viajes. Declaración:
  - Uml: + GetFecha() : LocalDate

- ListarViajesConPrecio: Método consultor de listado. Genera y devuelve un listado de los viajes con sus correspondientes precios. Declaración:
  - Uml: + ListarViajesConPrecio() : Map<Viaje, Float>

- ListarViajesPorAsientoDisponible: Método consultor de listado. Genera y devuelve un listado de los viajes que disponen de asientos libres. Declaración:
  - Uml: + ListarViajesPorAsientoDisponible() : ArrayList<Viaje>

- GetViajes: Genera y devuelve un conjunto de objetos Viaje, que serán los viajes que forman parte del listado. Declaración:
  - Uml: + GetViajes() : ArrayList<Viaje>

2.8. Clase *GestionReserva*:

Genera y gestiona una Reserva.

Descripción:

Clase encargada de generar y gestionar una reserva determinada, para un viaje de nuestro sistema. Será la encargada de interaccionar con el usuario a la hora de realizar la reserva.

Atributos:

- fecha: Contiene la fecha para la que se desea reservar un viaje. Declaración:
  - Uml: - fecha : LocalDate

- origen: Contiene el nombre de la ciudad que se desea que sea el origen del viaje a reservar. Declaración:
  - Uml: - origen : Ciudad

- destino: Contiene el nombre de la ciudad que se desea que sea el destino del viaje a reservar. Declaración:
  - Uml: - destino : Ciudad

- hora: Contiene la hora de salida del tren para el cual se desea reservar un viaje. Declaración:
  - Uml: - hora : LocalTime

- tren: Contiene el nombre del tren en el que se va a realizar el viaje a reservar. Declaración:
  - Uml: - tren : Tren

- numAsientos: Contiene el número de asientos que queremos reservar para el viaje. Declaración:
  - Uml: - numAsientos : int

- Reserva: Contiene una referencia a un objeto Reserva, que será la reserva que está gestionando. Declaración:
  - Uml: - reserva : Reserva

Métodos:

- getPrecio: Devuelve el precio de la reserva. Declaración:
  - Uml: + getPrecio(in origen : String, in destino : String, in fecha : LocalDate, in hora : LocalTime) : double

- GetNumAsientos: Devuelve el número de asientos reservados. Declaración:
  - Uml: + GetNumAsientos() : double

- GetOrigen: Método consultor de origen. Devuelve la ciudad origen del viaje que se pretende reservar. Declaración:
  - Uml: + GetOrigen() : Ciudad

- GetHora: Método consultor de hora. Devuelve la hora para la cual se va a realizar la reserva. Declaración:
    - Uml: + GetHora() : LocalTime

- GetDestino: Método consultor de destino. Devuelve la ciudad destino del viaje que se pretende reservar. Declaración:
    - Uml: + GetDestino() : Ciudad

- GetFecha: Método consultor de fecha. Devuelve la fecha para la cual se va a realizar la reserva. Declaración:
    - Uml: + GetFecha() : LocalDate

- GetTren: Método consultor de Tren. Devuelve el tren en el cual se va a viajar. Declaración:
    - Uml: + GetTren() : Tren

- reservaAsiento: Permite realizar una reserva del viaje. En el caso de que alguno de los parámetros no sea correcto, debe de devolver una excepción IllegalArgumentException. Se encarga de crear la reserva, utilizando para ello los atributos de la clase y devuelve un string con el identificador de la reserva. Declaración:
    - Uml: + reservaAsiento(in origen : String, in destino : String, in fecha : LocalDate, in hora : LocalTime) : String

- cancelaReserva: Se encarga de cancelar, por tanto eliminar del sistema, la reserva indicada en el atributo 'Reserva' de la clase, dejando el asiento indicado libre. Declaración:
    - Uml: - cancelaReserva(in codigoReserva : String) : void

2.9. Clase *CargaDatos*:

Clase encargada de cargar datos.

Descripción:

Realiza una carga de los trenes y trayectos que obtiene de los ficheros csv destinados para ello.

Atributos:

- TrenesCargados: Contiene un conjunto de trenes que han sido cargados del fichero trenes.csv. Declaración:
    - Uml: - TrenesCargados : ArrayList<Tren>

- TrayectosCargados: Contiene un conjunto de trayectos que han sido cargados del fichero trayectos.csv. Declaración:
    - Uml: - TrayectosCargados : ArrayList<Trayecto>

Métodos:

- StringToLocalTime: Método que convierte un string que contiene una hora determinada, en un objeto LocalTime con dicha hora. Declaración:
    - Uml: + StringToLocalTime (in valor : String) : LocalTime

- GetTrenesCargados: Método consultor de trenes. Devuelve el conjunto de trenes que han sido cargados del fichero trenes.csv. Declaración:
    - Uml: + GetTrenesCargados () : ArrayList<Tren>

- GetTrayectosCargados: Método consultor de trayectos. Devuelve el conjunto de trayectos que han sido cargados del fichero trayectos.csv. Declaración:
    - Uml: + GetTrayectosCargados () : ArrayList<Trayecto>

- GetTren: Devuelve el tren de entre los trenes cargados cuyo nombre coincide con el string que recibe por parámetro. Declaración:
    - Uml: + GetTren(in nombre : string) : Tren

2.10. Clase *AdaptadorListado*:

Clase adaptadora de la interfaz.

Descripción:

Se encarga de implementar la funcionalidad de la interfaz, en otras palabras, adapta la interfaz a la clase de gestión de listados

Atributos:

- datos: Contiene un objeto de la clase CargaDatos que contiene los trenes y trayectos cargados de los ficheros csv. Declaración:
    - Uml: - datos : CargaDatos

- listado: Contiene un objeto de la clase ListadoViajes que contiene los listados de viajes necesarios. Declaración:
    - Uml: - listado : ListadoViajes

Métodos:

- GetHorarios: Permite obtener un listado con la hora de inicio del viaje. No se mostrarán las horas para las cuales ya no hay asientos disponibles. Declaración:
    - Uml: + GetHorarios(in origen : String, in destino : String, in fecha : LocalDate) : List<LocalTime>

- GetDatos: Método consultor de datos. Devuelve el objeto de la clase CargaDatos que contiene los trenes y trayectos precargados. Declaración:
    - Uml: + GetDatos() : CargaDatos

3. Documentación de los casos de uso:

3.1. Caso de uso: Listado de viajes (filtrados por asientos libres)

- Descripción del caso de uso:

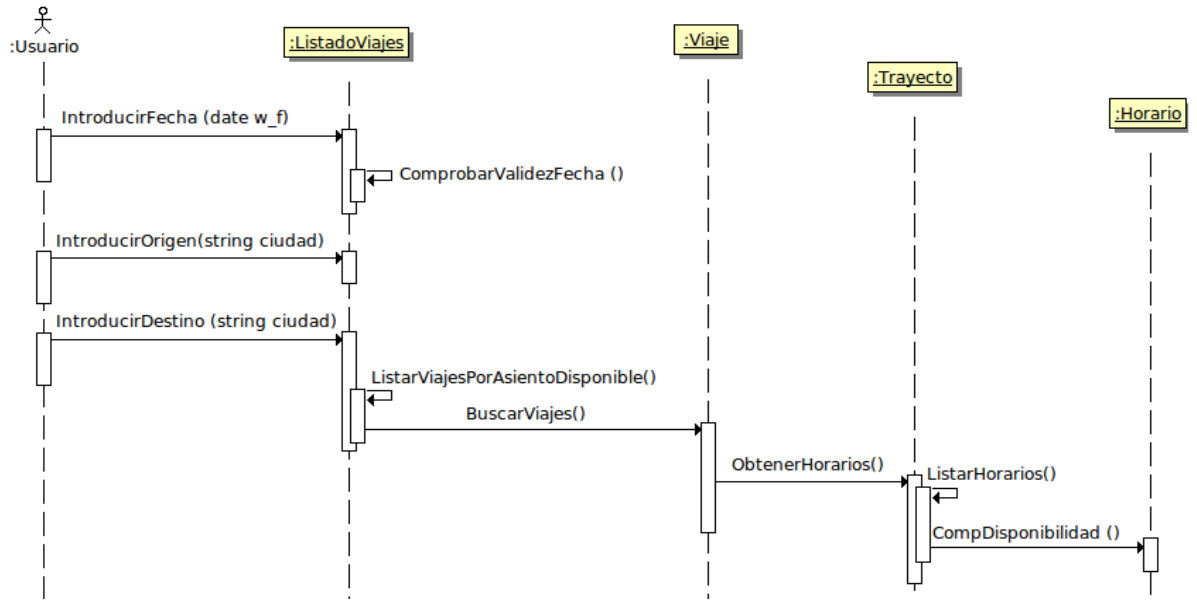
  - Descripción: El sistema ofrece al usuario un listado de viajes con asientos disponibles para la fecha, origen y destino indicados por el usuario.

  - Actores: El usuario del sistema.

  - Flujo básico: (sin errores inesperados)
    1. El usuario desea informarse sobre los viajes, de los que se disponen asientos libres, entre dos ciudades determinadas en una determinada fecha.
    2. El usuario introduce una fecha en el sistema.
    3. El sistema comprueba que la fecha es válida.
    4. El usuario elige la ciudad de origen del viaje.
    5. El usuario elige la ciudad de destino del viaje.
    6. El sistema busca los viajes entre las dos ciudades elegidas para la fecha indicada y obtiene sus horarios.
    7. El sistema comprueba si existen asientos libres en cada uno de los trenes que cubren los horarios.
    8. El sistema visualiza un listado con los horarios cuyos trenes disponen de asientos libres.

  - Flujo alternativo:
    2-6a. El usuario cancela el proceso.
        1. El sistema cancela el proceso y vuelve al menú principal.
    3a. El sistema detecta que la fecha introducida por el usuario es incorrecta.
        1. El sistema muestra el mensaje de aviso.
        2. El usuario vuelve a introducir una fecha.
    6a. El sistema detecta que no existen viajes entre las dos ciudades elegidas para la fecha indicada por el usuario.
        1. El sistema informa al usuario.
        2. El usuario modifica los datos del viaje.
    7a. El sistema detecta que no existen asientos libres para los trenes disponibles.
        1. El sistema informa al usuario.
        2. El usuario modifica los datos del viaje.

- Diagrama de secuencia del flujo básico:



3.2. Caso de uso: Listado de viajes (calculando el precio de cada uno)

- Descripción del caso de uso:

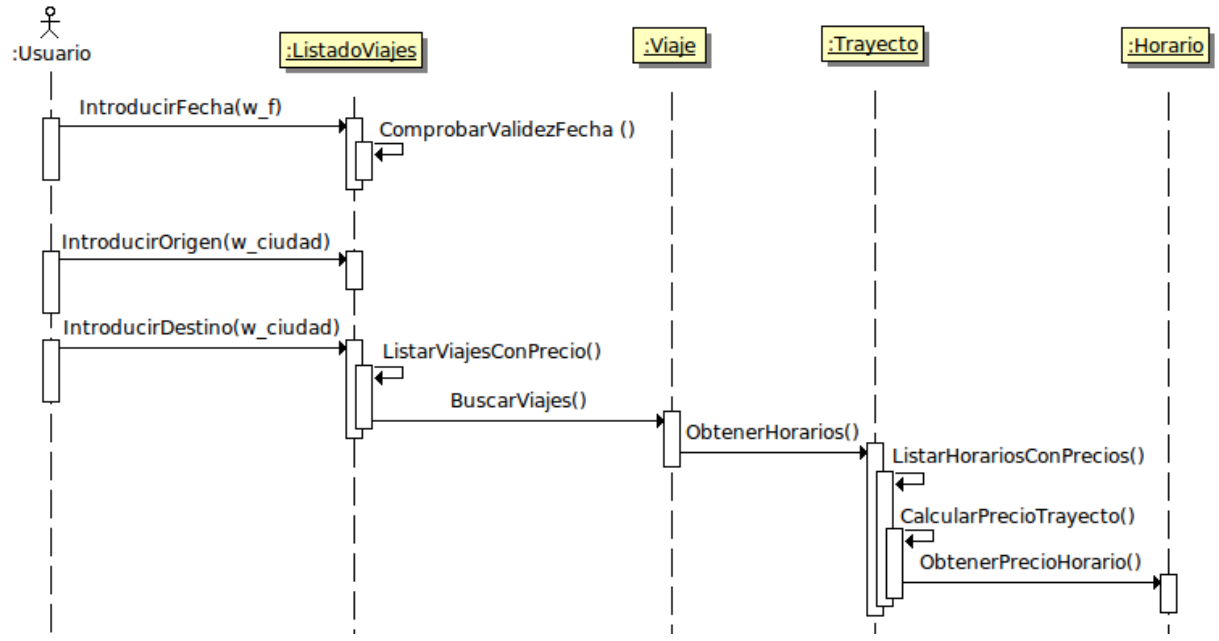
  - Descripción: El sistema ofrece al usuario un listado de viajes con sus precios correspondientes para la fecha, origen y destino indicados por el usuario.

  - Actores: El usuario del sistema.

  - Flujo básico: (sin errores inesperados)
    1. El usuario desea informarse sobre los viajes, y los precios de los mismos, entre dos ciudades determinadas en una determinada fecha.
    2. El usuario introduce una fecha en el sistema.
    3. El sistema comprueba que la fecha es válida.
    4. El usuario elige la ciudad de origen del viaje.
    5. El usuario elige la ciudad de destino del viaje.
    6. El sistema busca los viajes entre las dos ciudades elegidas para la fecha indicada y obtiene sus horarios.
    7. El sistema obtiene el precio por tramo para cada horario.
    8. El sistema calcula el precio para cada horario en función del número de tramos del trayecto.
    9. El sistema visualiza un listado con los horarios y precios correspondientes.

  - Flujo alternativo:
    2-6a. El usuario cancela el proceso.
        1. El sistema cancela el proceso y vuelve al menú principal.
    3a. El sistema detecta que la fecha introducida por el usuario es incorrecta.
        1. El sistema muestra el mensaje de aviso.
        2. El usuario vuelve a introducir una fecha.
    6a. El sistema detecta que no existen viajes entre las dos ciudades elegidas para la fecha indicada por el usuario.
        1. El sistema informa al usuario.
        2. El usuario modifica los datos del viaje.

- Diagrama de secuencia del flujo básico:



3.3. Caso de uso: Reserva de viaje

- Descripción del caso de uso:

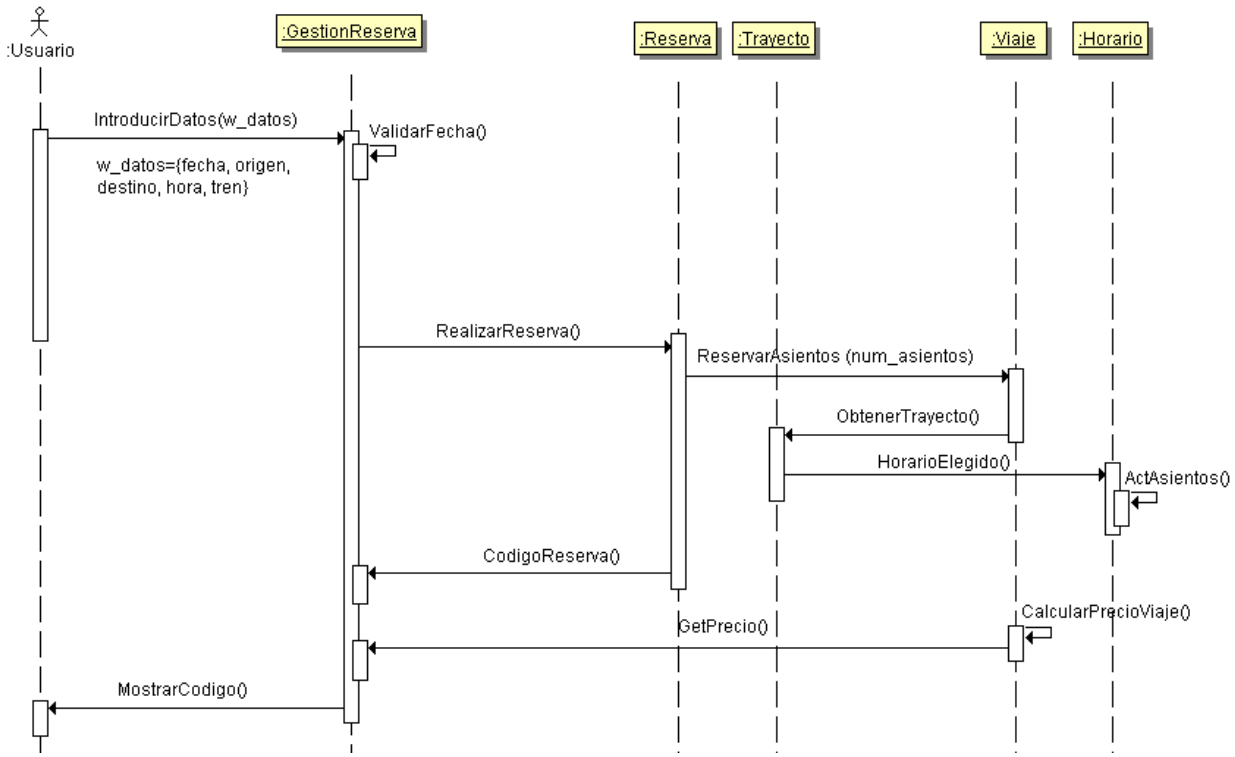
  - Descripción: El sistema Reserva un viaje, indicándole todos los datos (fecha, origen, destino, hora,tren), devolviendo el código de reserva.

  - Actores: El usuario del sistema.

  - Flujo básico: (sin errores inesperados)
    1. El usuario desea reservar un viaje entre dos ciudades.
    2. El usuario introduce una fecha en el sistema.
    3. El sistema comprueba que la fecha es válida.
    4. El usuario elige la ciudad de origen y destino del viaje.
    6. El usuario elige la hora y el tren del viaje.
    7. El sistema comprueba si existen asientos libres para dicho viaje.
    8. El usuario confirma la reserva.
    9. El sistema devuelve el código de la reserva.

  - Flujo alternativo:
    2-8a. El usuario cancela el proceso.
        1. El sistema cancela el proceso y vuelve al menú principal.
    3a. El sistema detecta que la fecha introducida por el usuario es incorrecta.
        1. El sistema muestra el mensaje de aviso.
        2. El usuario vuelve a introducir una fecha.
    6a. El sistema detecta que no existen viajes entre las dos ciudades elegidas para la fecha indicada por el usuario.
        1. El sistema informa al usuario.
        2. El usuario modifica los datos del viaje.
    7a. El sistema detecta que no existen asientos libres para los trenes disponibles.
        1. El sistema informa al usuario.
        2. El usuario modifica los datos del viaje.

- Diagrama de secuencia del flujo básico:



4. Reparto de funcionalidad

Por último describimos el reparto de la implementación de la funcionalidad del sistema entre los miembros de desarrollo del mismo. Las clases del sistema, es decir, las pertenecientes al diagrama de clases de diseño, se repartirán del siguiente modo:

- Miguel Ángel Pérez Montero implementará las clases 'Ciudad', 'Tren', 'Horario' y 'GestionReservas'.

- Manuel Jesús de la Calle Brihuega implementará las clases 'Reserva', 'Viaje', 'Trayecto' y 'ListadoViajes'.