Ingeniería del Software

Grado de Ingeniería Informática
Curso 2020-2021
Antonio Moruno Gracia, David Perez Dueñas, Marcos Rivera Gavilán

Práctica 3

Proyecto 26

1. Introducción

La práctica 3 se ha realizado a lo largo de 2 sesiones, concretamente la del 12 y la del 22 de noviembre de 2020. En esta práctica hemos tenido que realizar el diseño de la estructura y del comportamiento del sistema. Para ello hemos hecho uso de UML y de la herramienta Visual Paradigm Community Edition.

Los objetivos a cumplir con la realización de la práctica eran los siguientes:

- Aplicar la técnica de diagramas de clase de UML para el diseño estructural del sistema.
- Avanzar en el conocimiento y desarrollo del sistema validando y refinando la especificación de requisitos.
- Aplicar la técnica de diagramas de secuencia de UML para el diseño del comportamiento del sistema.
- Avanzar en el conocimiento y desarrollo del sistema para poder abordar la fase de implementación y pruebas

2. Planificación

Práctica 2

En esta práctica le ha tocado ser Product Owner a Marcos Rivera Gavilán y lo largo de estas tres semanas, hemos distribuido el trabajo de la siguiente forma:

2.1 Semana 1

El trabajo que debíamos realizar para esta semana era el siguiente:

- Extraer requisitos a partir de la entrevista con el cliente.
- Identificar, refinar y documentar los distintos tipos de requisitos del sistema.

Y acordamos distribuirlo de la siguiente forma:

- Antonio: Requisitos funcionales
- Marcos: Requisitos de información
- David: Requisitos no funcionales

Siendo cada uno responsable de:

- Añadir y documentar sus requisitos en el documento de Requisitos del Sistema

2.2 Semana 2

El trabajo que debíamos realizar para esta semana era el siguiente:

- Utilizar historias de usuario para crear la lista de producto según la metodología Scrum.
- Comenzar a planificar y priorizar funcionalidades de cara a su futura implementación.

Y acordamos distribuirlo de la siguiente forma:

- Antonio: Análisis de los requisitos funcionales para obtener historias de usuario
- Marcos: Análisis de los requisitos de información para obtener historias de usuario
- David: Análisis de los requisitos no funcionales para obtener historias de usuario

Siendo cada uno responsable de:

- Añadirlas al tablón haciendo uso de las funcionalidad de historias de usuario
- Añadirlas al documento Historias de Usuario

2.3 Semana 3

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aplicar la técnica de casos de uso de UML para detallar el análisis de requisitos funcionales. Uso de Visual Paradigm.
- Validar los requisitos con el cliente para resolver dudas surgidas durante la toma de requisitos

Y acordamos distribuirlo de la siguiente forma:

- Antonio: Creación de los casos de uso
- Marcos: Validación de los requisitos y redacción de este documento.
- David: Diseño del diagrama UML

Siendo cada uno responsable de aportar su parte en tiempo y forma antes de la fecha límite para la entrega de la práctica.

Todos los documentos citados en los párrafos anteriores así como las tarjetas con las historias de usuario, se encuentran respectivamente en la sección Knowledge Base y en el Panel de Desarrollo de la sección Agile Boards de nuestra instancia de YouTrack cuyo enlace es el siguiente:

https://uco-is2021-eq26.myjetbrains.com/

Práctica 3

En esta práctica le ha tocado ser Product Owner a Antonio Moruno Gracia y lo largo de estas dos semanas, hemos distribuido el trabajo de la siguiente forma:

3.1 Semana 1

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aplicar la técnica de diagramas de clase de UML para el diseño estructural del sistema.
- Avanzar en el conocimiento y desarrollo del sistema validando y refinando la especificación de requisitos.

Y acordamos distribuirlo de la siguiente forma:

- Antonio: Validación de los requisitos funcionales frente a los casos de uso y validación de los casos de uso frente a las clase
- Marcos: Diseño de clases
- David: Diagrama de clases

Siendo cada uno responsable de:

- Aportar su parte en tiempo y forma antes de la fecha de límite para la entrega de la práctica

3.2 Semana 2

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aplicar la técnica de diagramas de secuencia de UML para el diseño del comportamiento del sistema.
- Avanzar en el conocimiento y desarrollo del sistema para poder abordar la fase de implementación y pruebas

Y acordamos distribuirlo de la siguiente forma:

- Antonio: Diagrama de secuencia 1 y redacción de este documento
- Marcos: Diagrama de secuencia 2
- David: Diagrama de secuencia 3

Siendo cada uno responsable de:

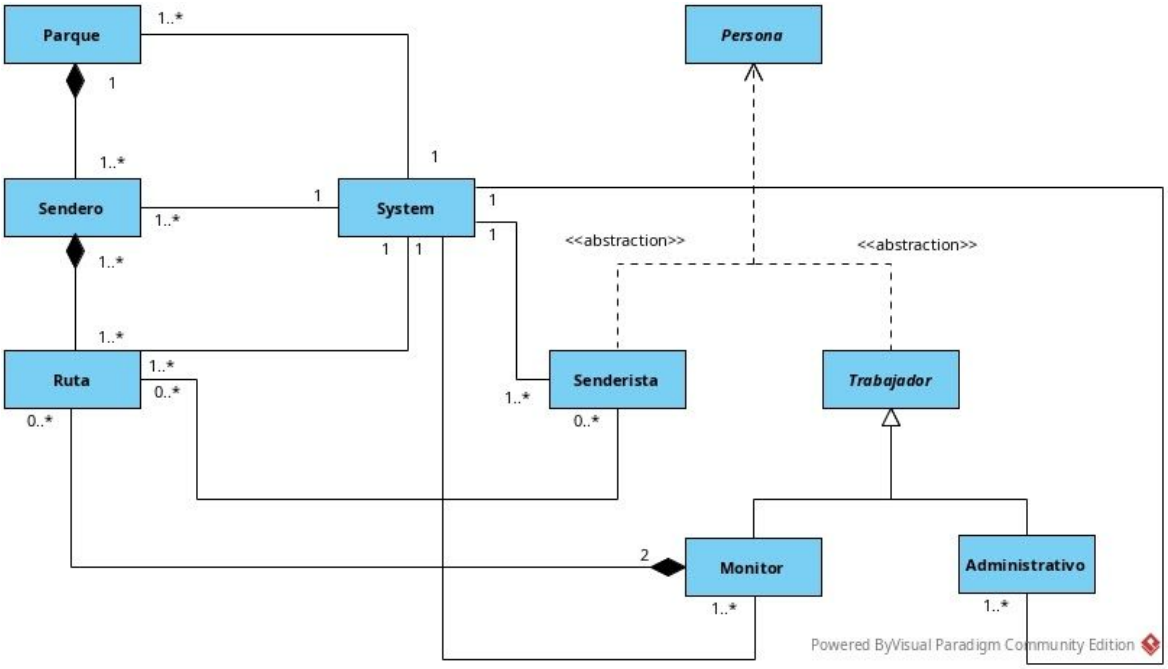
- Aportar su parte en tiempo y forma antes de la fecha límite para la entrega de la práctica.
- La organización del trabajo así como las actas de las reuniones se encuentran en el siguiente enlace:

https://uco-is2021-eq26.myjetbrains.com/

3. Diseño del sistema (P3)

3.1 Diseño estructural

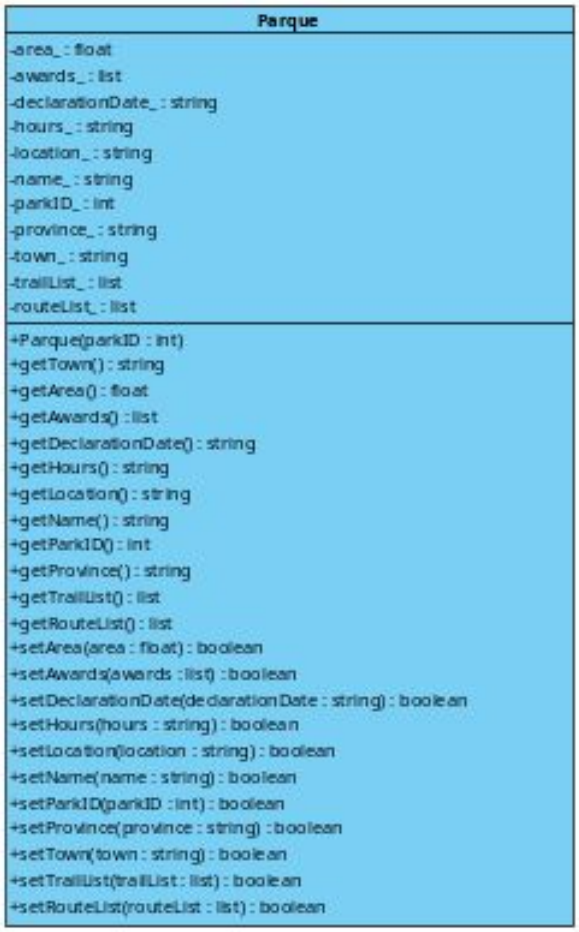
3.1.1 Diagrama de clases



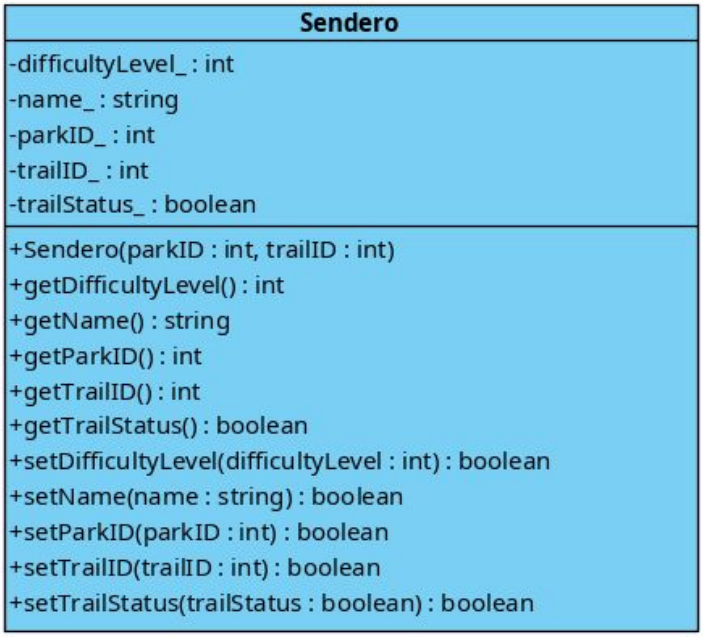
"La imagen 1 muestra el diagrama de clases con las correspondientes relaciones"

La imagen anterior muestra el diagrama de clases simplemente con las relaciones establecidas entre clases, ya que si incluimos en la misma imagen todos los atributos y operaciones de las clases, quedaría una imagen difícil de

interpretar. A continuación se muestran las distintas clases con sus atributos y operaciones.

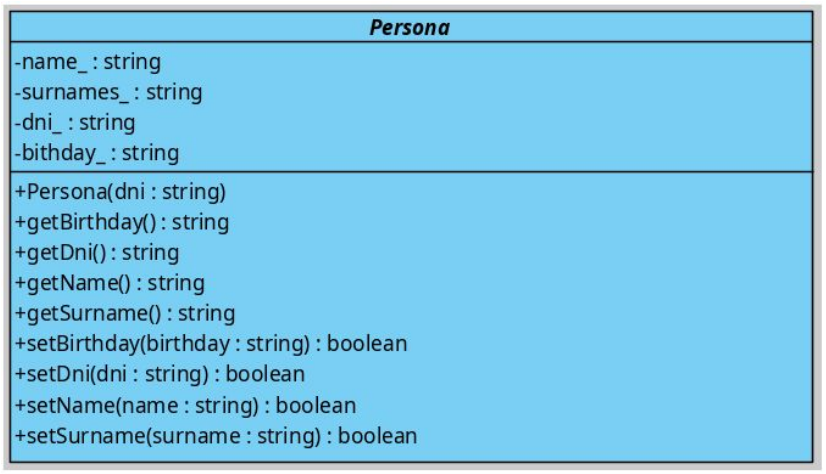


"La imagen 2 muestra la clase parque con sus atributos y operaciones"

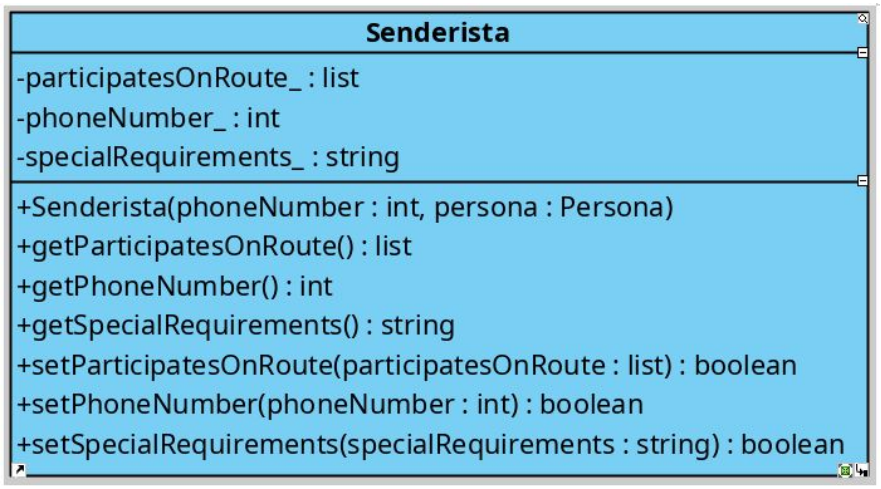


"La imagen 3 muestra la clase sendero con sus atributos y operaciones"

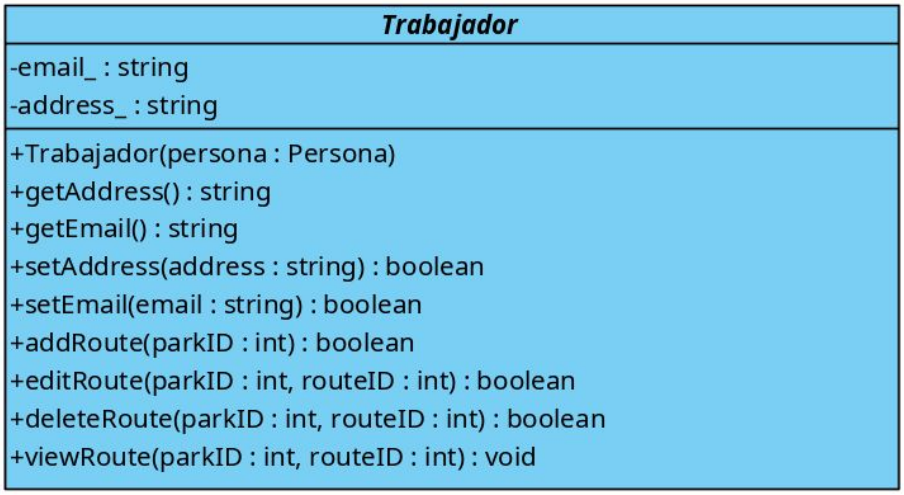
"La imagen 4 muestra la clase ruta con sus atributos y operaciones"



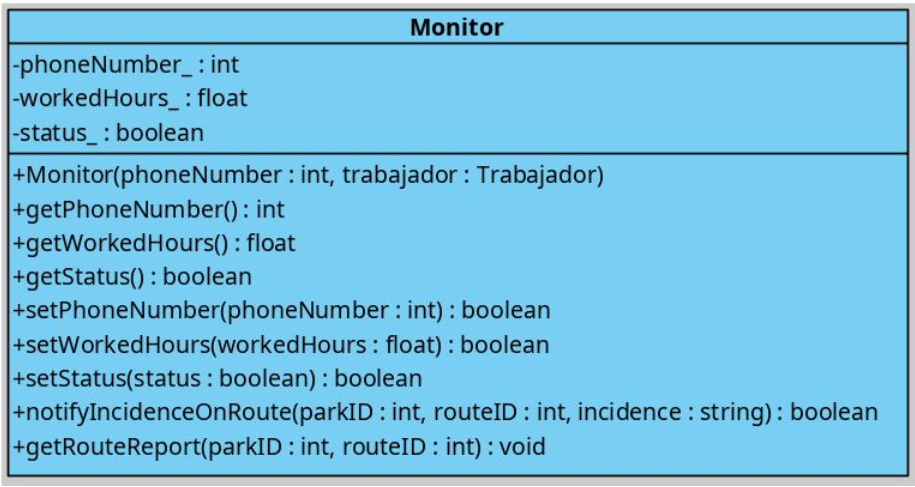
"La imagen 5 muestra la clase persona con sus atributos y operaciones"



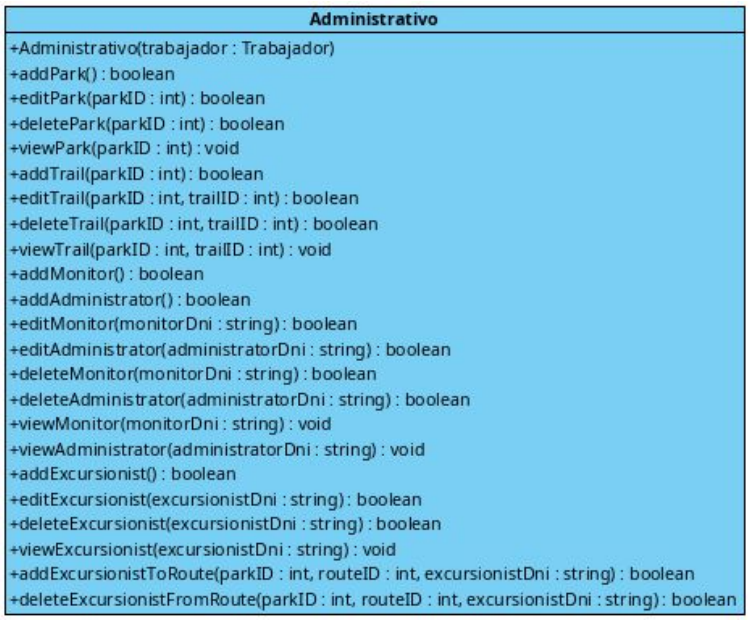
"La imagen 6 muestra la clase senderista con sus atributos y operaciones"



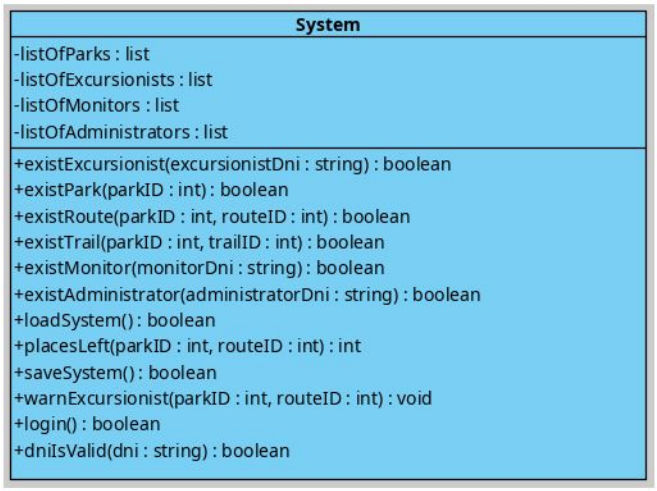
"La imagen 7 muestra la clase trabajador con sus atributos y operaciones"



"La imagen 8 muestra la clase monitor con sus atributos y operaciones"



"La imagen 9 muestra la clase administrativo con sus atributos y operaciones"



"La imagen 10 muestra la clase system con sus los atributos y operaciones"

3.1.2 Especificación de clases

Se muestran a continuación la información de diseño en forma de tabla para que quede aún más clara la visualización del diagrama de clases.

Clase	**Parque**		
La clase parque representa a un parque con sus correspondientes atributos y operaciones			
Atributos			
-	area_	float	Área del parque
-	awards_	list	Lista de los premios del parque
-	declarationDate_	string	Fecha de declaración del parque
-	hours_	string	Horario del parque
-	location_	string	Localización (latitud y longitud)
-	name_	string	Nombre del parque
-	parkID_	int	ID del parque
-	province_	string	Provincia del parque
-	town_	string	Ciudad del parque
-	trailList_	list	Lista de los senderos del parque

-	routeList_	list	Lista de las rutas del parque
Operaciones			
+	Parque(int parkID)	Constructor	Constructor de la clase parque. Recibe el ID del parque creado
+	getArea()	float	Devuelve el área del parque
+	getAwards()	list	Devuelve el listado de premios
+	getDeclarationDate()	string	Devuelve la fecha de declaración
+	getHours()	string	Devuelve el horario
+	getLocation()	string	Devuelve la localización
+	getName()	string	Devuelve el nombre
+	getParkID()	int	Devuelve el ID del parque
+	getProvince()	string	Devuelve la provincia
+	getTown()	string	Devuelve la ciudad
+	getTrailList()	list	Devuelve la lista de senderos del parque
+	getRoutesList()	list	Devuelve la lista de rutas del parque
+	setArea(float area)	bool	Asigna un área
+	setAwards(list awards)	bool	Asigna la lista de premios
+	setDeclarationDate(string date)	bool	Asigna la fecha de declaración

+	setHours(string hours)	bool	Asigna el horario
+	setLocation(string location)	bool	Asigna la localización
+	setName(string name)	bool	Asigna el nombre
+	setParkID(int parkID)	bool	Asigna un ID
+	setProvince(string province)	bool	Asigna la provincia
+	setTown(string town)	bool	Asigna la ciudad
+	setTrailList(list trailList)	bool	Asigna los senderos del parque
+	setRouteList(list routeList)	bool	Asigna las rutas del parque

"La tabla 1 muestra la clase parque con sus atributos y operaciones"

Clase	Sendero		
La clase sendero representa a un sendero con sus correspondientes atributos y operaciones			
Atributos			
-	difficultyLevel_	int	Nivel de dificultad del sendero (del 1 al 10)
-	name_	string	Nombre del sendero
-	parkID_	int	ID del parque del sendero
-	trailID_	int	ID del sendero
-	trailStatus_	bool	Estado del sendero (disponible o deshabilitado)
Operaciones			
+	Sendero(int parkID, int trailID)	Constructor	Constructor de la clase sendero. Recibe el ID de su parque y el ID del sendero
+	getDifficultyLevel()	int	Devuelve el nivel de dificultad del sendero
+	getName()	string	Devuelve el nombre del sendero
+	getParkID	int	Devuelve el ID del parque del sendero
+	getTrailID()	int	Devuelve el ID del sendero
+	getTrailStatus()	bool	Devuelve el estado del sendero

+	setDifficultyLevel(int difficultyLevel)	bool	Asigna un nivel de dificultad al sendero
+	setName(string name)	bool	Asigna un nombre al sendero
+	setParkID(int parkID)	bool	Asigna el ID del parque al que pertenece el sendero
+	setTrailID(int trailID)	bool	Asigna un ID al sendero
+	setTrailStatus(bool trailStatus)	bool	Asigna un estado al sendero

"La tabla 2 muestra la clase sendero con sus atributos y operaciones"

Clase	Ruta		
La clase ruta representa a una ruta con sus correspondientes atributos y operaciones			
Atributos			
-	adaptations_	string	Adaptaciones de la ruta (por si algún senderista tiene requisitos especiales)
-	dniMonitor_	string	Dni del monitor de la ruta
-	dniMonitorAlternate_	string	Dni del monitor alternativo de la ruta
-	duration_	int	Duración de las rutas en minutos
-	exclusiveness_	bool	Almacena si una ruta es exclusiva o no (solo para colegios por ejemplo)
-	length_	float	Longitud de la ruta en kilómetros
-	modality_	string	Modalidad de la ruta (a pie, en bici...)
-	name_	string	Nombre de la ruta
-	numberOfPlaces_	int	Número de plazas de la ruta
-	excursionistRegistered_	list	Lista de senderistas apuntados a la ruta
-	parkID_	int	ID del parque de la ruta
-	routeID_	int	ID de la ruta

-	traversedTrail_	list	Lista de los senderos atravesados por la ruta
-	incidences_	list	Lista de las incidencias
Operaciones			
+	Ruta(string dniMonitor, int parkID, int routeID, list traversedTrail, int numberOfPlaces)	Constructor	Constructor de la clase ruta. Recibe el dni del monitor, el ID de su parque, el ID de la ruta, la lista de senderos atravesados por la ruta y el número de plazas
+	getAdaptations()	string	Devuelve el número de adaptaciones de la ruta
+	getDniMonitor()	string	Devuelve el dni del monitor de la ruta
+	getDniMonitorAlternate()	string	Devuelve el dni del monitor alternativo de la ruta
+	getDuration()	int	Devuelve la duración de la ruta
+	getExclusiveness()	bool	Devuelve si la ruta es exclusiva
+	getIncidences()	list	Devuelve la lista de incidencias de la ruta
+	getLength()	float	Devuelve la longitud de la ruta
+	getModality()	string	Devuelve la modalidad de la ruta
+	getName()	string	Devuelve el nombre de la ruta
+	getNumberOfPlaces()	int	Devuelve el número de plazas de la ruta

+	getExcursionistRegistered()	list	Devuelve la lista de senderistas apuntados a la ruta
+	getParkID()	int	Devuelve el ID del parque de la ruta
+	getRouteID()	int	Devuelve el ID de la ruta
+	getTraversedTrail()	int	Devuelve el número de senderos atravesados por la ruta
+	setAdaptations(string adapts)	bool	Asigna la adaptación de una ruta
+	setDniMonitor(string dni)	bool	Asigna el dni del monitor
+	setDniMonitorAlternate(string dni)	bool	Asigna el dni del monitor alternativo
+	setDuration(int duration)	bool	Asigna la duración de la ruta
+	setExclusiveness(bool exclnss)	bool	Asigna la exclusividad a la ruta
+	setIncidences(list incidences)	bool	Asigna incidencias a la ruta
+	setLength(float length)	bool	Asigna la longitud de la ruta
+	setModality(string modality)	bool	Asigna la modalidad de la ruta
+	setName(string name)	bool	Asigna un nombre a la ruta

+	setNumberOfPlaces(int n)	bool	Asigna el número de plazas de la ruta
+	setExcursionistRegistered(list excursionist)	bool	Asigna la lista de senderistas apuntados a una ruta
+	setParkID(int parkID)	bool	Asigna el ID del parque al que pertenece la ruta
+	setRouteID(int routeID)	bool	Asigna el ID de la ruta
+	setTraversedTrails(list traversedTrails)	bool	Asigna la lista de senderos que atraviesa la ruta

"La tabla 3 muestra la clase ruta con sus atributos y operaciones"

Clase	Persona		
La clase persona representa a una persona con sus correspondientes atributos y operaciones			
Atributos			
-	name_	string	Nombre de la persona
-	surnames_	string	Apellidos de la persona
-	dni_	string	Dni de la persona
-	birthday_	string	Fecha de nacimiento de la persona
Operaciones			
+	Persona(string dni)	Constructor	Constructor de la clase Persona. Recibe un dni
+	getBirthday()	string	Devuelve la fecha de nacimiento
+	getDni()	string	Devuelve el dni de la persona
+	getName()	string	Devuelve el nombre de la persona
+	getSurname()	string	Devuelve los apellidos de la persona
+	setBirthday(string birthday)	bool	Asigna la fecha de nacimiento
+	setDni(string dni)	bool	Asigna el dni de la persona

+	setName(string name)	bool	Asigna el nombre de la persona
+	setSurname(string name)	bool	Asigna los apellidos de la persona

“La tabla 4 muestra la clase persona con sus atributos y operaciones”

Clase	Senderista		
La clase senderista representa a un senderista con sus correspondientes atributos y operaciones			
Atributos			
-	participatesOnRoute_	list	Lista de rutas a las que está apuntado el senderista
-	phoneNumber_	int	Número de teléfono del senderista
-	specialRequirements_	string	Requisitos especiales del senderista
Operaciones			
+	Senderista(int number, Persona persona)	Constructor	Constructor de la clase Senderista. Recibe un teléfono y un objeto de la clase Persona
+	getParticipatesOnRoute()	list	Devuelve la lista de rutas a las que está apuntado el senderista
+	getPhoneNumber()	int	Devuelve el número de teléfono del senderista
+	getSpecialRequirements()	string	Devuelve los requisitos especiales del senderista
+	setParticipatesOnRoute(list participatesOnRoute)	bool	Asigna la lista de rutas a las que está apuntado el senderista

+	setPhoneNumber(int number)	bool	Asigna el número de teléfono del senderista
+	setSpecialRequirements(string specialRequirements)	bool	Asigna los requisitos especiales del senderista

"La tabla 5 muestra la clase senderista con sus atributos y operaciones"

Clase	Trabajador		
La clase trabajador representa a un trabajador con sus correspondientes atributos y operaciones			
Atributos			
-	email_	string	Email del trabajador
-	address_	string	Dirección
Operaciones			
+	Trabajador(Persona persona)	Constructor	Constructor de la clase trabajador. Recibe un objeto del tipo persona
+	getAddress()	string	Devuelve la dirección del trabajador
+	getEmail()	string	Devuelve el email del trabajador
+	setAddress(string address)	bool	Asigna la dirección
+	setEmail(string email)	bool	Asigna el email
+	addRoute(int parkID)	bool	Añade una ruta. Recibe el ID del parque donde se encuentre
+	editRoute(int parkID, int routeID)	bool	Modifica una ruta. Recibe el ID del parque donde se encuentre y el ID de la ruta a modificar

+	deleteRoute(int parkID, int routeID)	bool	Elimina una ruta. Recibe el ID del parque donde se encuentre y el ID de la ruta a eliminar
+	viewRoute(int parkID, int routeID)	void	Visualiza una ruta. Recibe el ID del parque donde se encuentre y el ID de la ruta a visualizar

"La tabla 6 muestra la clase trabajador con sus atributos y operaciones"

Clase	Monitor		
La clase monitor representa a un monitor con sus correspondientes atributos y operaciones			
Atributos			
-	phoneNumber_	int	Número de teléfono del monitor
-	workedHours_	float	Horas trabajadas del monitor
-	status_	bool	Almacena si el monitor está disponible o está de baja
Operaciones			
+	Monitor(int phoneNumber, Trabajador trabajador)	Constructor	Constructor de la clase monitor. Recibe un número de teléfono y un objeto de la clase Trabajador
+	getPhoneNumber()	int	Devuelve el número de teléfono del monitor
+	getWorkedHours()	float	Devuelve el número de horas trabajadas del monitor
+	getStatus()	bool	Devuelve el status del monitor
+	setPhoneNumber(int number)	bool	Asigna el número de teléfono del monitor
+	setWorkedHours(float hours)	bool	Asigna el número de horas trabajadas del monitor

+	setStatus(bool status)	bool	Asigna el status del monitor
+	notifyIncidentOnRoute(int parkID, int routeID, string incidence)	bool	Guarda en incidencias una incidencia ocurrida en la ruta
+	getRouteReport(int parkID, int routeID)	void	Muestra por pantalla la información de la ruta

"La tabla 7 muestra la clase monitor con sus atributos y operaciones"

Clase	Administrativo		
La clase administrativo representa a un administrativo con sus correspondientes atributos y operaciones			
Operaciones			
+	Administrativo(Trabajador t)	Constructor	Constructor de la clase administrativo. Recibe un objeto t de la clase Trabajador
+	addPark()	bool	Añade un parque
+	editPark(int parkID)	bool	Modifica un parque
+	deletePark(int parkID)	bool	Elimina un parque
+	viewPark(int parkID)	void	Visualiza un parque
+	addTrail(int parkID)	bool	Añade un sendero
+	editTrail(int parkID, int trailID)	bool	Modifica un sendero
+	deleteTrail(int parkID, int trailID)	bool	Elimina un sendero
+	viewTrail(int parkID, int trailID)	void	Visualiza un sendero
+	addMonitor()	bool	Añade un monitor
+	addAdministrator()	bool	Añade un administrativo
+	editMonitor(string dni)	bool	Modifica un monitor
+	editAdministrator(string dni)	bool	Modifica un administrativo
+	deleteMonitor(string dni)	bool	Elimina un monitor

+	deleteAdministrator(string dni)	bool	Elimina un administrativo
+	viewMonitor(string dni)	void	Visualiza un monitor
+	viewAdministrator(string dni)	void	Visualiza un administrativo
+	addExcursionist()	bool	Añade un senderista
+	editExcursionist(string dni)	bool	Modifica un senderista
+	deleteExcursionist(string dni)	bool	Elimina un senderista
+	viewExcursionist(string dni)	void	Visualiza un senderista
+	addExcursionistToRoute(int parkID, int routeID, string dni)	bool	Añade un senderista a una ruta
+	deleteExcursionistFromRoute(int parkID, int routeID, string dni)	bool	Elimina a un senderista de una ruta

"La tabla 8 muestra la clase administrativo con sus atributos y operaciones"

Clase	System		
La clase system representa al sistema con sus correspondientes atributos y variables			
-	listOfParks	list	Lista de parques del sistema
-	listOfExcursionists	list	Lista de senderistas del sistema
-	listOfMonitors	list	Lista de monitores del sistema
-	listOfAdministrators	list	Lista de administrativos del sistema
Operaciones			
+	existExcursionist(string dni)	bool	Comprueba si existe el senderista en el sistema
+	existPark(int parkID)	bool	Comprueba si existe el parque en el sistema
+	existRoute(int parkID, int routeID)	bool	Comprueba si existe la ruta en el sistema
+	existTrail(int parkID, int trailID)	bool	Comprueba si existe el sendero en el sistema
+	existMonitor(string dni)	bool	Comprueba si existe el monitor en el sistema
+	existAdministrator(string dni)	bool	Comprueba si existe el administrativo en el sistema
+	loadSystem()	bool	Carga en memoria los datos de los ficheros parque, administrativo, senderista y monitor

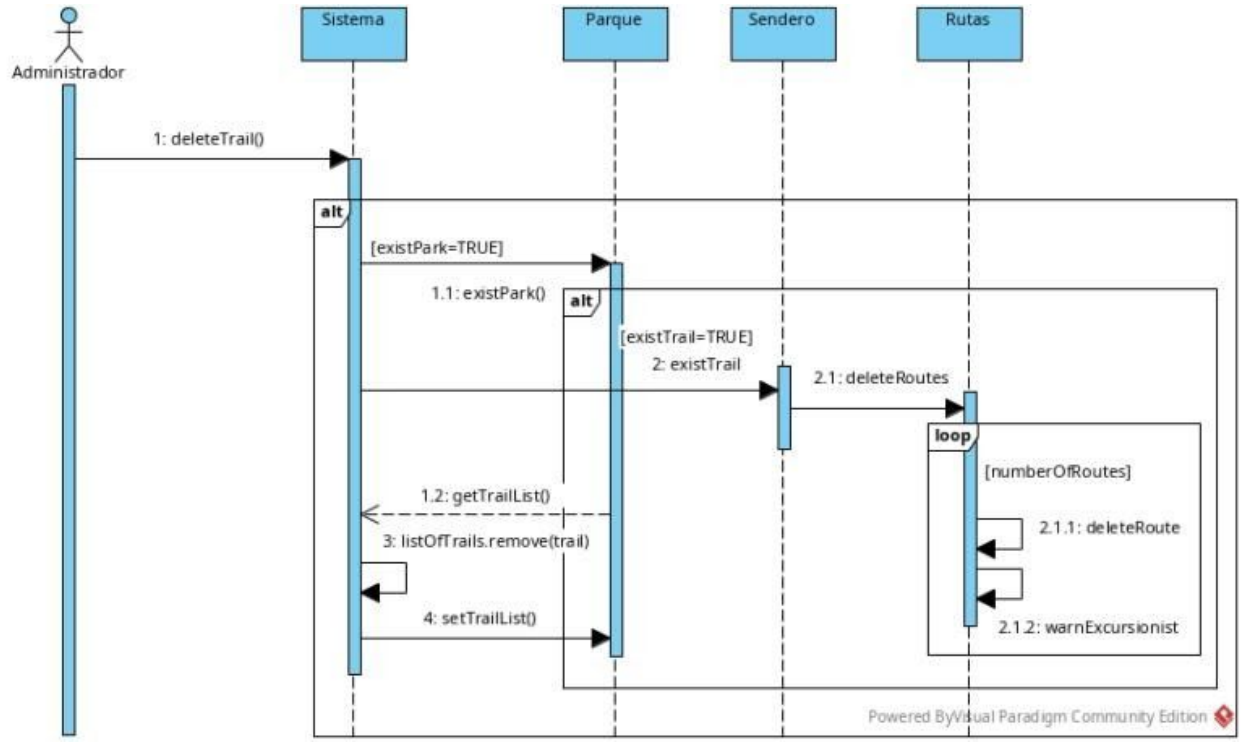
+	placesLeft(int parkID, int routeID)	int	Devuelve el número de plazas libres que quedan en una ruta
+	saveSystem()	bool	Guarda en los ficheros parque, administrativo, senderista y monitor la información cargada en memoria
+	warnExcursionist(int parkID, int routeID)	void	Muestra los datos de contacto de los senderistas apuntados a la ruta que se acaba de eliminar
+	login()	boolean	Permite a los trabajadores iniciar sesión en el sistema como monitores o administrativos
+	dniIsValid(string dni)	boolean	Comprueba si el dni introducido es correcto o no

"La tabla 9 muestra la clase system con sus atributos y operaciones"

Cabe destacar que la clase system no es propiamente una clase. En ella almacenaremos los atributos y operaciones que se tengan que hacer sobre las otras clases. Es decir, no podemos almacenar los atributos y operaciones en las otras clases, ya que estos operan e interactúan con todas ellas.

3.2 Diseño de comportamiento

3.2.1 Diagrama de secuencia 1



"La imagen 11 muestra el diagrama de secuencia del CU9"

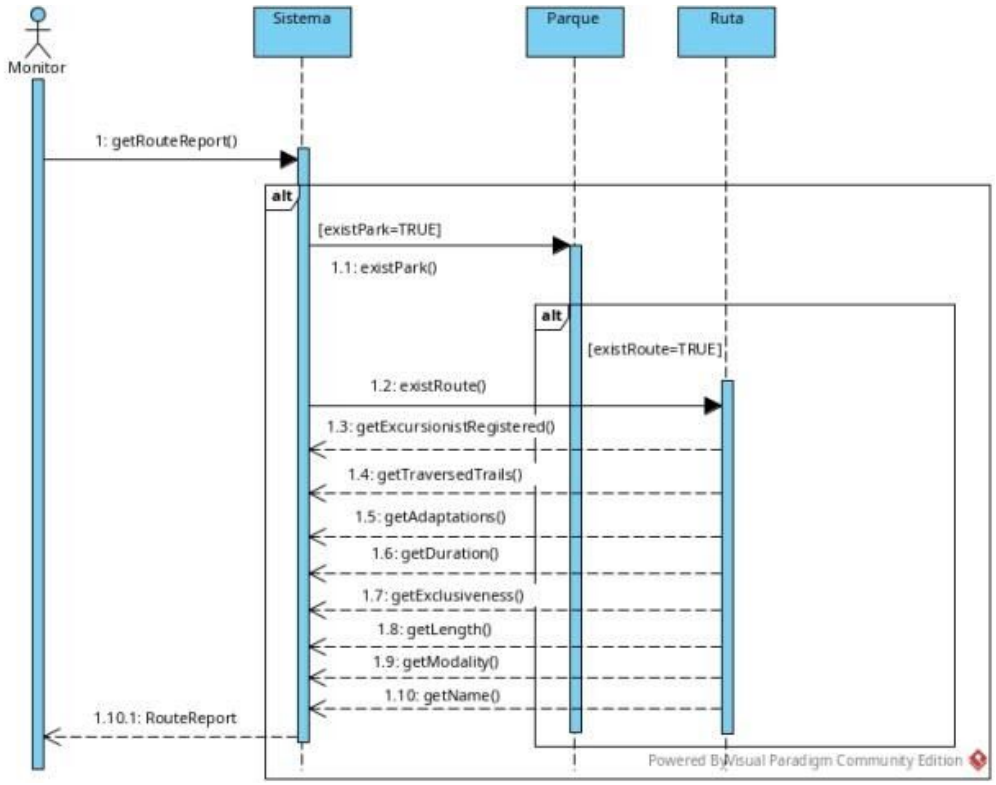
En este caso de uso el administrador desea eliminar un sendero de un parque.

Para ello lo primero que hace es introducir el ID del parque al que pertenece el sendero y el ID del sendero que queremos eliminar.

El sistema comprueba si existe el parque y en caso afirmativo comprueba que exista el sendero. Si se cumplen ambas condiciones elimina el sendero con ese ID de la lista de

senderos del parque así como las rutas que pasan por ese sendero y avisa a los senderistas.

3.2.2 Diagrama de secuencia 2



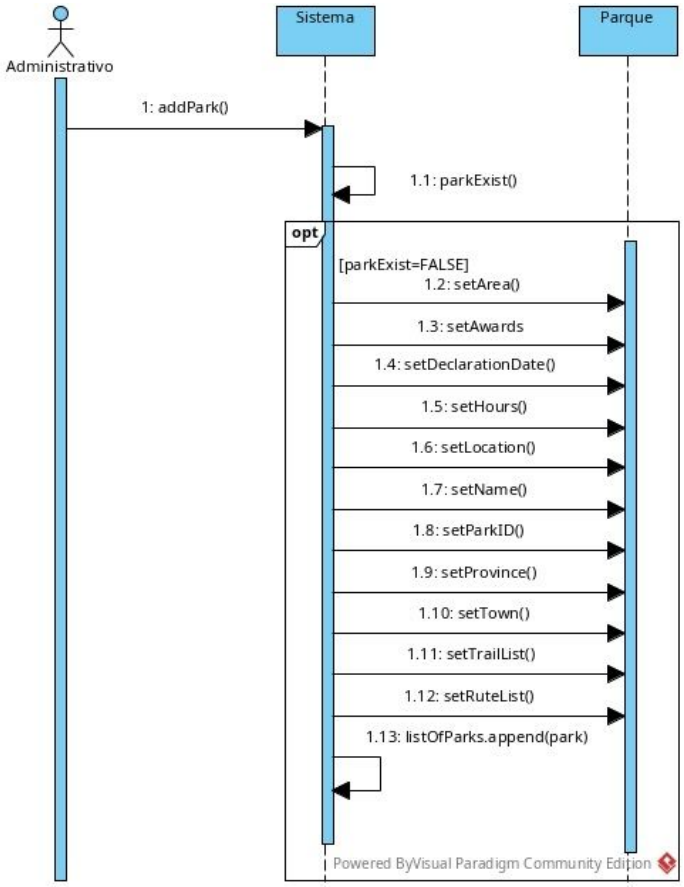
"La imagen 12 muestra el diagrama de secuencia del CU30"

En este caso de uso el monitor que está apunto de comenzar la ruta desea obtener un reporte con la información de la ruta.

Este lo solicita al sistema pasándole el ID del parque y de la ruta.

El sistema se encarga de comprobar que la información introducida es correcta antes de empezar a generar el reporte.
Una vez comprobado comienza a extraer la información correspondiente de la ruta y la vuelca en un fichero (route ID)Report.txt donde route ID representa en ID de la ruta de la que se extrae el reporte.

3.2.3 Diagrama de secuencia 3



"La imagen 13 muestra el diagrama de secuencia del CU1"

En este caso de uso el administrativo desea añadir un parque al sistema para ello introduce el ID del parque que desea añadir y el sistema comprueba si ya existe un parque con ese ID

Si no existe se procede a crear y rellenar un nuevo objeto de tipo parque haciendo uso de los métodos set de dicha clase y al final el sistema lo añade al listado de parques.

4. Validación del sistema (P3)

4.1 Requisitos funcionales frente a casos de uso

RF vs CU	CU1	CU2	CU3	CU4	CU5
RF1	X	X	X	X	X
RF2					
RF3					
RF4					
RF5					

"La tabla 10 muestra los requisitos funcionales frente a los casos de uso 1-2-3-4-5"

RF vs CU	CU6	CU7	CU8	CU9	CU10
RF1	X	X	X	X	X
RF2					
RF3					
RF4					
RF5					

"La tabla 11 muestra los requisitos funcionales frente a los casos de uso 6-7-8-9-10"

RF vs CU	CU11	CU12	CU13	CU14	CU15
RF1	X	X	X	X	X
RF2		X	X		
RF3		X			
RF4					
RF5					

"La tabla 12 muestra los requisitos funcionales frente a los casos de uso 11-12-13-14-15"

RF vs CU	CU16	CU17	CU18	CU19	CU20
RF1	X	X	X	X	X
RF2	X	X	X	X	X
RF3	X				
RF4					
RF5					

"La tabla 13 muestra los requisitos funcionales frente a los casos de uso 16-17-18-19-20"

RF vs CU	CU21	CU22	CU23	CU24	CU25
RF1	X	X	X	X	X
RF2	X	X	X		
RF3	X				X
RF4					
RF5					X

"La tabla 14 muestra los requisitos funcionales frente a los casos de uso 21-22-23-24-25"

RF vs CU	CU26	CU27	CU28	CU29	CU30
RF1	X	X	X	X	X
RF2					
RF3			X		X
RF4					
RF5			X	X	

"La tabla 15 muestra los requisitos funcionales frente a los casos de uso 26-27-28-29-30"

RF vs CU	CU31	CU32	CU33
RF1	X	X	X
RF2		X	
RF3			X
RF4	X		
RF5			

"La tabla 16 muestra los requisitos funcionales frente a los casos de uso 31-32"

4.2 Casos de uso frente a clases

Matrices de validación de clases frente a los casos de uso. Las clases son, según se indican:

CL1: Parque **CL2: Sendero** **CL3: Ruta** **CL4: Persona**

CL5: Senderista **CL6: Trabajador** **CL7: Monitor** **CL8: Administrativo**

CL9: System

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU1	X			X		X		X	X
CU2	X								X
CU3	X			X		X		X	X
CU4	X	X	X	X		X		X	X
CU5	X			X		X		X	X

"La tabla 17 muestra los casos de uso 1-2-3-4-5 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU6	X	X		X		X		X	X
CU7	X	X							X
CU8	X	X		X		X		X	X
CU9	X	X	X	X		X		X	X
CU10	X	X		X		X		X	X

"La tabla 18 muestra los casos de uso 6-7-8-9-10 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU11	X	X	X	X		X	X	X	X
CU12	X	X	X						X
CU13	X	X	X	X	X	X	X	X	X
CU14	X	X	X	X	X	X	X	X	X
CU15	X	X	X	X	X	X	X	X	X

"La tabla 19 muestra los casos de uso 11-12-13-14-15 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU16	X	X	X	X		X	X	X	X
CU17				X	X	X		X	X
CU18				X	X	X		X	X
CU19	X	X	X	X	X	X		X	X
CU20	X	X	X	X	X	X		X	X

"La tabla 20 muestra los casos de uso 16-17-18-19-20 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU21	X	X	X	X	X	X		X	X
CU22	X	X	X	X	X	X		X	X
CU23	X	X	X	X	X	X		X	X
CU24				X		X	X	X	X
CU25				X		X			X

"La tabla 21 muestra los casos de uso 21-22-23-24-25 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU26				X		X	X	X	X
CU27				X		X	X	X	X
CU28				X		X	X	X	X
CU29				X		X	X	X	X
CU30	X	X	X	X	X	X	X		X

"La tabla 22 muestra los casos de uso 26-27-28-29-30 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU31			X	X		X	X	X	X
CU32	X	X	X						X
CU33				X		X	X	X	X

"La tabla 23 muestra los casos de uso 31-32 frente a las clases"

5. Referencias

La documentación utilizada para elaborar esta práctica ha sido la aportada por la profesora en el moodle de la asignatura. Esto incluye los contenidos teóricos aportados en los diferentes documentos de cada una de las semanas de prácticas y la información adjuntada en los siguientes enlaces:

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/