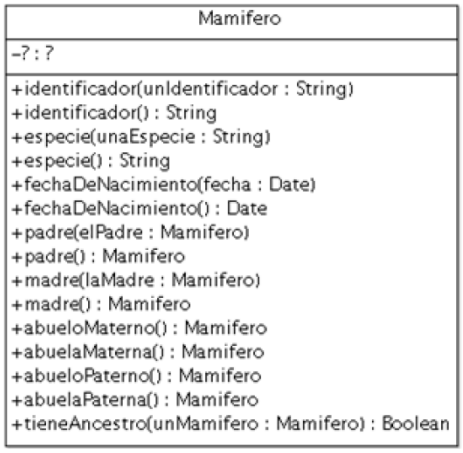
Practica 1

Esta práctica tiene como objetivo evaluar al estudiante en competencias básicas de modelado e implementación de soluciones basadas en la programación orientada a objetos, aplicando conocimientos del paradigma (herencia, polimorfismo, encapsulamiento, reuso, etc.) y de modelado UML (específicamente diagramas de clase, diagramas de objetos y diagramas de secuencia).

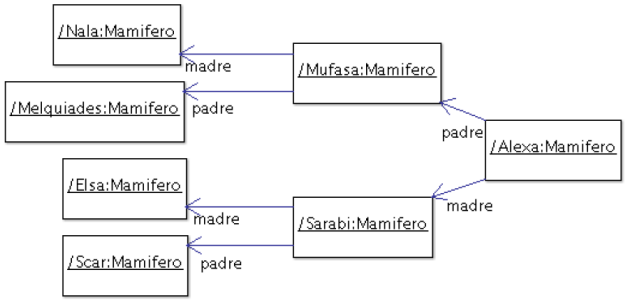
EJERCICIOS INTRODUCTORIOS

EJERCICIO A

En la reserva Castillo de las Guardas, los cuidadores quieren llevar registro detallado de los animales que cuidan y sus familias. Para ello nos han pedido ayuda para crear una aplicación cuyo principal objetivo es mantener el árbol genealógico de los animales, en principio solo mamíferos:



Considere el siguiente diagrama de objetos como base para analizar la implementación de la clase Mamífero:



Tareas

1. Específique una implementación mediante pseudocódigo.

EJERCICIO B

Imagine una red de alumbrado inteligente donde cada farola está conectada a una o varias vecinas formando un grafo conexo. Cada una de las farolas tiene un interruptor. Es suficiente con encender o apagar una farola cualquiera para que se enciendan o apaguen todas las demás. Sin embargo, si se intenta apagar una farola apagada (o si se intenta encender una farola encendida) no habrá ningún efecto, ya que no se propagará esta acción hacia las vecinas.

La funcionalidad a proveer permite:
1. crear farolas, siendo que inicialmente están apagadas
2. conectar farolas, de manera tal que al conectar una farola a otra, se asegure que la conexión es bi-direccional, y además
3. encender una farola (y obtener el efecto antes descrito)
4. apagar una farola (y obtener el efecto antes descrito)
5. saber si una farola esta prendida

Tareas
1. Realice el diagrama UML de clases de la solución al problema.
2. Específique una implementación mediante pseudocódigo.

EJERCICIOS CON EVALUACIÓN

EJERCICIO 1

Un sistema de alquiler de propiedades permite a los usuarios registrar sus propiedades y también reservar o alquilar propiedades de otros usuarios.

Cuando un usuario registra una propiedad para alquilar indica cuánto cuesta el alquiler por cada noche. Cuando un usuario reserva una propiedad en alquiler, fija la fecha de inicio y la fecha de finalización. Además se calcula el precio en el momento que se hace la reserva. Una propiedad puede ser reservada para un rango de fechas si no existe una reserva previa que incluya alguna de esas fechas.

El cálculo del precio final de una reserva tiene dos pasos. En el primero se calcula el precio base que surge de multiplicar el valor por noche de la propiedad por la cantidad de días de la reserva. Luego se pueden aplicar reglas específicas que cada propietario puede configurar cuando pone su propiedad en alquiler. El sistema contempla dos tipos de reglas por ahora (puede haber más en el futuro):

- Regla por rango de fechas. El propietario define si en un rango de fechas el precio por noche aumenta o disminuye. Por ejemplo, un propietario A puede establecer que entre el 1ro de enero y el 28 de febrero el precio base aumenta un 10%. O que entre el 1ro de marzo y el 30 de junio disminuye un 10%. Esta regla se aplica sólo a los días de la reserva que se encuentren en el rango que fija la regla. Por ejemplo, una reserva del 23 de diciembre al 15 de enero, sólo verá afectado el precio de los días de enero. Los días de diciembre se calculan sólo con precio base. Puede

haber varias reglas de este tipo por cada propiedad. Podemos asumir que el sistema se encarga automáticamente de controlar que para una misma propiedad no haya reglas de este tipo que solapen rangos de fechas. Este tipo de regla tiene prioridad 1.

- Reglas por estancias prolongadas. El propietario define en qué porcentaje disminuye el precio base por noche cuando la cantidad de días reservados supera una cantidad mínima de días que establece en la regla. Por ejemplo, el mismo propietario A indica que por una reserva de más de 10 días el precio disminuye 5%. Por cada propiedad sólo puede haber una regla de este tipo. Este tipo de regla tiene prioridad 2.

Las reglas son acumulables y se aplican por nivel de prioridad. Es decir, dada una reserva, primero se calcula el precio base y luego se aplican las reglas que existan en el orden de prioridad indicado. En la reserva del 23/12 al 15/1 del ejemplo de Regla por rangos, primero se calcula cuánto sale según precio base, después se aplica la regla de rangos (1/1 al 28/2) para los días de enero y después se aplica un 5% menos para toda la reserva porque supera 10 días.

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 2

Se desea desarrollar un Sistema de Administración de Usuarios en Cursos Online que debe permitir:

- Registro de usuarios:
  - Una persona puede registrarse como usuario (se guardan nombre de usuario, email y clave).
  - Un usuario registrado puede inscribirse en cualquiera de los cursos disponibles.
  - Un usuario registrado puede cargar su valoración de un curso en el que está inscripto.
- Gestión de inscripciones en cursos
  - El Sistema tiene cursos disponibles de los que se conoce un nombre, una descripción textual y una lista de las valoraciones que cargaron los usuarios inscriptos
  - Los cursos pueden ser de dos tipos: grabados o en vivo.
  - Los cursos grabados tienen cupo de inscripción ilimitado. El precio es un valor fijo (que puede cambiar si el productor del curso lo establece). No tienen fecha de inicio.
  - Los cursos en vivo tienen un cupo máximo de inscripciones y una fecha de inicio. La inscripción sólo está permitida hasta el día anterior a la fecha de inicio del curso.

    - Los cursos en vivo ofrecen un bono por cantidad de estudiantes. El productor del curso indica la cantidad mínima de inscriptos y el porcentaje del bono. En el momento de comienzo del curso, si se superó la cantidad mínima de inscriptos, se genera para cada inscripto un bono correspondiente al porcentaje indicado. Este bono no modifica lo pagado por el curso, sino que queda "a favor" de pagos para otros cursos. Por ejemplo, el curso A tiene una cantidad mínima de 15 inscriptos, un valor de €100 y bono 10%. Si al comenzar el curso existen 16 inscriptos, a cada uno se le anotan €10 "a favor". Asuma que el proceso de computar montos "a favor" se dispara automáticamente cuando inicia un curso en vivo.
- Inscripción y pago:
    - Cuando un usuario se inscribe en un curso se registra: la fecha y el monto pagado realmente.
    - El sistema debe controlar que la inscripción sea posible según el curso.
    - En caso que un usuario tenga monto "a favor" de bonos por cantidad, el monto se descuenta del valor a pagar por el curso (hasta un 50%). Por ejemplo, un usuario se registra en un curso B que vale €100 y tiene "a favor" €60 de inscripciones anteriores. En el curso B paga €50 y mantiene €10 "a favor".
- Valoraciones:
    - Cada valoración que hace un usuario sobre un curso consiste en un puntaje (un número entre 1 y 5), una fecha y un comentario.
- Requerimientos extras:
    - Dado un usuario, la lista de cursos en los que se inscribió y el monto pagado por cada uno.
    - La cantidad promedio de inscriptos por cada tipo de curso (grabados y en vivo)
    - Los 10 cursos que tienen mejor valoración (la valoración de un curso es el promedio de todas las valoraciones recibidas)

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 3

Se desea desarrollar una aplicación para compartir archivos entre usuarios de una aplicación. De los usuarios se sabe su nombre, email y password. Los usuarios tienen los archivos organizados en directorios que ellos mismos crean. Cada directorio puede contener archivos y también otros directorios. De los directorios se sabe su nombre. Todo archivo de un usuario debe estar dentro de un directorio. De un archivo se sabe su nombre, fecha de creación, fecha de modificación y tamaño. Los archivos pueden ser compartidos de dos maneras, pero solo de una manera a la vez:

- El usuario propietario de un archivo puede definir una lista de usuarios de la aplicación para que puedan acceder.
- El usuario propietario puede definir el archivo como público, para que cualquier usuario de la aplicación pueda acceder. Esta manera de compartir permite definir una fecha límite a partir de la cual el documento no podrá ser accedido. Si un archivo no tiene definida una manera de ser compartido, entonces no puede ser accedido mas que por el propietario del mismo.

La aplicación debe satisfacer los siguientes requerimientos:

- Controlar que dado un usuario y un archivo, el usuario pueda acceder al archivo.
- Dado un usuario, retornar la cantidad de archivos que tiene.
- Dado un usuario, retornar el tamaño total ocupado por sus archivos.

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 4

El municipio de Dos Hermanas desea instalar un servicio de bicicletas públicas que los ciudadanos y turistas podrán utilizar siempre y cuando estén registrados en una aplicación. El municipio planea construir varias estaciones en distintos puntos de la ciudad para recoger y devolver las bicicletas, y cada estación tendrá un conjunto de estacionamientos de bicicletas con cerraduras inteligentes. En este sentido, la aplicación a construir será utilizada por los usuarios registrados para desbloquear dichas cerraduras.

El sistema debe permitir:

- Registrar estaciones: indicando geoposición y cantidad de estacionamientos.
- Registrar bicicletas: se indica un identificador.
- Registrar un usuario: indicando número de identificación (DNI o pasaporte), nombre y apellido. Además, se requieren los datos de una tarjeta de crédito la cual será de garantía para cobrar infracciones en el uso del sistema.
- Registrar abono de usuario, siendo que existen distintos tipos de abonos y que un usuario puede cambiar de abono cuando lo desee:
    - Abono Anual: Pensado para usuarios cuyo uso es muy frecuente. El usuario realiza un pago de €150 por el uso del sistema para todo un año. Se registra la fecha de inicio del abono. Este tipo de abono permite realizar, durante el tiempo de validez del abono, una cantidad ilimitada de usos de bicicletas sin costo adicional excepto que el uso sea mayor a 30 minutos. Superando este tiempo de uso,

se cobrará €2 por cada 5 minutos extras. Si el abono ha vencido, el usuario no puede hacer uso de las bicicletas.
    - Abono Prepago: Pensado para usuarios cuyo uso es ocasional. El usuario con este abono compra crédito y se le irá descontando del mismo cada uso de bicicleta. El saldo no tiene vencimiento y puede recargarse mas saldo cuando el usuario lo desee. A un usuario con este abono el costo de uso de la bicicleta se calcula por periodos de 15 minutos, siendo €5 el costo de este periodo de tiempo. Para que el usuario pueda hacer uso de las bicicletas, debe tener el menos 5€ de crédito, y en ese caso si el uso de la bicicleta implico un costo de mas de 5€, debe quedar con crédito negativo.
    - Abono turístico: Pensado para visitantes de la ciudad. Se registra la fecha de inicio y tiene una duración de 7 días. El abono cuesta €50 y permite realizar usos de bicicletas de periodos de 2 horas. Superado este tiempo, se le cargara un costo de €10 por cada 15 minutos extra de viaje.
    - Para cualquiera de los abonos, si han pasado mas de 24hs entre la recogida de la bicicleta y su devolución, se cargará en la tarjeta de crédito del usuario un monto de €30.
    - NOTA: para los cobros a tarjetas de crédito puede utilizar la clase **Payment**, en cuyo protocolo público dispone del método *#processPayment*(credit_card_number: String, amount: Float) y que retorna True si el pago fue realizado con éxito.
- Registrar usos de bicicleta: se requiere saber el usuario, la bicicleta, la estación donde recogió la bicicleta, y se registra fecha y hora de recogida. Cuando el usuario devuelve la bicicleta se registra la estación donde la devolvió y la fecha y hora de la devolución. Además al momento de devolver la bicicleta se realiza el cobro correspondiente según sea el abono del usuario.
- Por otro lado, la aplicación debe permitir que un usuario pueda ver:
    - Cuáles estaciones tienen bicicletas disponibles.
    - Cuáles estaciones tienen estacionamientos disponibles.

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML.
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 5

Una empresa requiere una plataforma para que diseñadores y creadores puedan comercializar por allí sus diseños y recursos (imágenes, íconos, tipografías, templates web, etc.). Cada vez que un creador de recursos registra una nueva producción, puede configurar una estrategia de comercialización, que son ideadas por los propietarios de la plataforma. Los creadores de recursos juntan puntos cada vez que un usuario adquiere uno de sus productos. Y estos puntos sirven a la plataforma no solo para posicionar a los creadores dentro de

la misma sino también para computar su costo de suscripción a la plataforma en un futuro.

El sistema debe permitir:

- Registrar nuevo creador de recursos: se indica su nombre, email, contraseña y se inicia con 0 su cantidad de puntos.
- Registrar un nuevo recurso: se indica el creador, una descripción del recurso, una imagen de previsualización, una URL de descarga, fecha de carga, su precio base y una estrategia de comercialización (que no podrán cambiar en el futuro), que inicialmente puede ser:
    - Normal: implica que los usuarios podrán descargar el recurso pagando el costo base del recurso.
    - Oferta: se define una fecha límite y un porcentaje de descuento. Si una compra se realiza antes de la fecha límite se aplica el porcentaje de descuento, mientras que luego de dicha fecha se cobra el costo base completo.
    - Crowd-based: se define una cantidad de usuarios mínimos que deben comprar el recurso. Solo cuando la cantidad de usuarios que lo compran alcanza al menos ese límite, dichos compradores podrán descargarlo, y se realizará el cobro considerando el precio base del recurso.
- Registrar nuevo usuario: se indica su nombre, email y contraseña.
- Registrar una nueva compra de recurso: se indica el usuario y el recurso a adquirir. Cuando se crea una nueva compra, deben pasar varias cosas:
    - Calcular el costo de la compra, la cual depende de la estrategia de comercialización del producto.
    - Calcular y registrar la cantidad de puntos que el creador sumará, que se calcula también según la estrategia de comercialización:
        - i. Normal: el puntaje resulta del costo base * 10.
        - ii. Oferta: el puntaje resulta de costo de la compra * 5 cuando la oferta aun no venció, y * 10 cuando ya ha vencido.
        - iii. Crowd-based: el puntaje resulta de multiplicar el costo base * 50 / la cantidad de usuarios mínimos.

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML.
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 6

Se requiere realizar una aplicación para el seguimiento de mantenimiento de trenes. Cada tren está registrado con su modelo (varios trenes pueden ser del mismo modelo), y cada uno de los viajes realiza (fecha de realización y km recorridos). Cada modelo de tren tiene un plan de mantenimiento, donde se registran tareas que deben realizarse a los mismos. El plan de mantenimiento de un tren puede incluir dos tipos de tareas:

- Por tiempo: se hacen con una frecuencia predeterminada (mensual, semestral, etc);
- Por rodadura: deben realizarse cuando el tren alcanzó la cantidad de km rodados que determina la tarea. Cualquiera de los tipos de tarea puede requerir la colocación de algún repuesto (de cada repuesto se conoce el nombre y el costo). Todas las tareas tienen un costo que se calcula como la suma de un monto fijo (el costo base de la tarea) más la suma de los costos de los repuestos usados. Para una tarea por tiempo el costo base es un monto fijo. Para una tarea por rodadura, el costo base es la suma de un monto fijo más el 5% de la cantidad de km rodados por el tren.

El primer día de cada mes, se requiere tener un listado de tareas a realizar en cada tren. Una vez realizado el mantenimiento mensual, debe registrarse en el sistema las tareas realizadas para cada tren.

La aplicación a desarrollar debe permitir:

1. Registrar trenes: incluyendo número de serie, modelo, marca, fecha de incorporación a la flota y kilometraje inicial del tren. Se asume que cada tren que se incorpora ya tiene realizado el mantenimiento actualizado según fecha y kilometraje.
2. Registrar plan de mantenimiento: incluyendo versión y a qué modelos de tren aplica.
3. Agregar tareas a un plan de mantenimiento:
    a. Agregar tareas periódicas: se registra un identificador de tarea, el tiempo estimado de realización, el costo base, el conjunto de repuestos que se requieren y la periodicidad con la que debe realizarse (en cantidad de días)
    b. Agregar tareas basadas en uso: se registra un identificador de tarea, el tiempo estimado de realización, el costo base, el conjunto de repuestos que se requieren y cada cuanta distancia recorrida (en cantidad de kilómetros) debe ser realizada.
4. Registrar viaje de tren: incluyendo la fecha del viaje y el kilometraje rodado.
5. Registrar tarea realizada: se registra que tarea del plan de mantenimiento fue aplicada a un tren en particular, indicando su fecha y una descripción.
6. Obtener el costo de las tareas realizadas en un tren en un mes específico.
7. Obtener los 5 planes de mantenimiento más costosos

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML.
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 7

Se desea una plataforma para encontrar trabajos de manera freelance (es decir, por cuenta propia). En dicha plataforma, quienes requieren que se efectúe

un trabajo (por ejemplo, desarrollar una aplicación) lo publican como un proyecto y quienes buscan trabajos (los freelancers) se ofrecen a desarrollar alguno de esos proyectos publicados. La aplicación debe resolver la asignación entre proyectos y freelancers.

Cuando un proyecto se publica, acepta ofertas de freelancers hasta una fecha específica. Las ofertas son cotización que pueden hacerse de dos maneras, por hora de trabajo o por posición. Luego, la aplicación asigna un freelancer en función de la mejor oferta, que se determina según el costo final de desarrollo. Una vez hecha la asignación, la plataforma se desliga de su desarrollo, aunque permite al proyectista establecer que el proyecto fue terminado para que el freelancer sume los puntos correspondientes.

El sistema ofrece la siguiente funcionalidad (que usted deberá implementar):

- Registrar un proyectista: Se le indica el nombre del proyectista y su dirección de email.
- Registrar un freelancer: Se le indica el nombre del freelancer, su dirección de email, el precio de hora de trabajo, y las categorías (ej.: Desarrollo Web, Diseño Gráfico, etc.).
- Registrar un proyecto: Se indica el nombre, la descripción, la fecha hasta la que se aceptan ofertas.
- Registrar oferta para proyecto: para brindarle flexibilidad a cada freelancer a la hora de cotizar los proyectos según aspectos profesionales no contemplados en la aplicación, las ofertas de trabajo pueden ser:
    - por hora de trabajo: Una oferta por hora de trabajo implica que el freelancer cotiza el proyecto en función de las horas especificadas en su oferta y su propio precio por hora de trabajo. Se requiere saber la fecha de la oferta, la cantidad de horas estimadas y la fecha de entrega estimada.
    - por posición de trabajo: Una oferta por posición implica que el freelancer cotiza el proyecto en función de cobrar un sueldo mensual por una cantidad de horas de trabajo por mes por una cantidad de meses determinada.
- Buscar un proyecto por categoría: Se recibe el nombre de una categoría y retorna los proyectos que la incluyen.
- Buscar un freelancer por categoría: Se recibe el nombre de una categoría y retorna los freelancer que la incluyen.
- Recomendar oferta de proyectos: Para ayudar al proyectista a escoger una oferta, debe ser posible obtener un listado de ofertas ordenadas por puntaje. El puntaje de una oferta se define por el precio total dividido la cantidad de días que deben pasar para la entrega del proyecto:
    - El precio final de una oferta por hora de trabajo es la cantidad de horas establecidas por el freelancer multiplicadas por el precio de hora de trabajo del mismo.
    - El precio final de una oferta por posición es el salario estipulado multiplicado por la cantidad de meses de posición definida.
    - La cantidad de días de entrega para una oferta por hora de trabajo es definida por la cantidad de días entre la fecha de inicio (considerando como fecha de inicio la fecha en la que el proyecto ya no acepta más ofertas) y la fecha de entrega estimada.

    - La cantidad de días de entrega para una oferta por posición es definida por la cantidad días entre la fecha de inicio (también considerando como fecha de inicio la fecha en la que el proyecto ya no acepta más ofertas) y la de fecha finalización, calculada en función de los meses de posición definidos en la oferta.
- Asignar oferta a un proyecto: dado un proyecto y una oferta del mismo, se registra asigna como ganadora la oferta al proyecto.
- Retornar el freelancer asignado a un proyecto: dado un proyecto, retorna el freelancer asignado. Si no tiene freelancer asignado, retorna nil.
- Registrar finalización de proyecto: Para un proyecto con freelancer asignado, el proyectista puede registrar la finalización del proyecto, sumando un puntaje del 1 al 50 para el freelancer. Además del puntaje establecido, también debe registrarse la fecha de finalización del proyecto.

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML.
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.

EJERCICIO 8

Se desea implementar un sistema para administrar los proyectos de desarrollo de software. El sistema contemplar a todos los, quienes pueden tomar dos roles: líder técnico y desarrollador. Cada integrante se identifica en el sistema con su nombre y un correo electrónico. Además de cada integrante se conoce un puntaje que está relacionado con su participación en los proyectos, lo cual se explica posteriormente.

Cada proyecto se presenta en el sistema como un Tablero que incluye diferentes listas de tareas. Al comienzo del proyecto todas las tareas están en una lista creada por defecto llamada Backlog y durante el proyecto, las tareas se van moviendo a otras listas. Las listas se utilizan para agrupar tareas que deban ser resueltas en siguientes versiones del sistema en desarrollo.

Cuando se crea un Tablero se indica el nombre del mismo y el líder, que requiere un nombre. Automáticamente el Tablero debe disponer de la lista de tareas llama Backlog, pero el líder podrá generar nuevas listas de tareas a donde mover tareas del Backlog, para lo cual se debe registrar el nombre de la nueva lista de tareas.

Cada tarea puede ser asignada a un integrante del equipo. Puede ocurrir que como parte del trabajo una tarea tenga que cambiar de desarrollador. Es necesario que se puedan conocer todos integrantes que estuvieron asignados una tarea y entre cuáles fechas fue dicha asignación. También puede ocurrir que una tarea que originalmente estaba en una lista sea asignada a otra lista.

Al crearse una tarea se requiere saber su título, una descripción, y su complejidad (un número de 1 a 5) y una fecha tope de realización. Cuando se

crea una tarea también se registra el día actual como fecha de creación, y cuando se termina, debe registrar su fecha de finalización. Además de estos datos, debe tenerse en cuenta que las tareas podrán ser del Programación o de Diseño de Interfaz de Usuario (DIU). Para calcular el puntaje que se asigne a cada integrante se tendrá encuenta que:

- Una Tarea de Diseño de Interfaz de Usuario que se cierra a tiempo (fecha de cierre <= fecha tope) asigna un puntaje igual a 2 x complejidad. Si se cierra fuera de fecha asigna 1 punto.
- Una Tarea de Programación que se cierra a tiempo (fecha de cierre <= fecha tope) asigna un puntaje igual a complejidad al cuadrado. Si se cierra fuera de fecha no asigna puntaje.
- El líder de un tablero obtiene como puntaje 1 punto por cada tarea completada a tiempo.

Se desea que se puedan llevar a cabo las siguientes funcionalidades con el sistema:

- Crear un tablero. Se indica el nombre y el líder. Se obtiene el tablero creado.
- Crear una tarea. Se indica el título, descripción y fecha límite para la tarea. Se crea la tarea con la fecha actual y se devuelve completamente inicializada, la tarea debe quedar asignada a la lista Backlog del proyecto.
- Crear una lista. Se indica el título de la lista que se crea y se agrega al tablero.
- Asignar una tarea. Se indica el integrante que queda a cargo de realizar la tarea a partir del día de la asginación. Si estaba asignada a otro desarrollador, se cierra esa asignación (con fecha de hoy) y se abre otra. Se devuelve la tarea con la asignación completada.
- Mover una tarea de una lista a otra. Se indica a qué lista debe pasar la tarea.
- Finalizar una tarea. Se le asigna el día actual como fecha de finalización.
- Cerrar un sprint para un tablero. Se toman las tareas finalizadas durante el último sprint (últimos 15 días) y se asigna el puntaje al actual responsable de la misma.
- Listar tareas pendientes. Se obtiene un listado de todas las tareas del Tablero que aún no han sido finalizadas ordenadas por antigüedad (es decir primero las que tienen una fecha anterior de creación)
- Listar tareas de usuario. Se obtiene el listado de todas las tareas en las que participó un usuario.

Tareas

1. Diseñe su solución y documente la misma con un diagrama de clases UML.
2. Implementar el modelo de clases en Python.
3. Especificar en un workspace la instanciación de las clases y la vinculación entre las instancias de acuerdo a un diagrama de objetos que contemple instancias de todas las clases.