Cuadernillo Semestral de Actividades

Actualizado: 30 de octubre de 2025

El presente cuadernillo posee un compilado con todos los ejercicios que se usarán durante el semestre en la asignatura. Los ejercicios están organizados en forma secuencial, siguiendo los contenidos que se van viendo en la materia.

Cada semana les indicaremos cuáles son los ejercicios en los que deberían enfocarse para estar al día y algunos de ellos serán discutidos en la explicación de práctica.

Recomendación importante:
Los contenidos de la materia se incorporan y fijan mejor cuando uno intenta aplicarlos - no alcanza con ver un ejercicio resuelto por alguien más. Para sacar el máximo provecho de los ejercicios, es importante que asistan a las consultas de práctica habiendo intentado resolverlos (tanto como les sea posible). De esa manera podrán hacer consultas más enfocadas y el docente podrá darles mejor feedback.

Ejercicio 1: WallPost

Primera parte

Se está construyendo una red social como Facebook o Twitter. Debemos definir una clase `WallPost` con los siguientes atributos: un texto que se desea publicar, cantidad de likes (“me gusta”) y una marca que indica si es destacado o no. La clase es subclase de Object.

Para realizar este ejercicio, utilice el recurso que se encuentra en el sitio de la cátedra (o que puede descargar desde acá). Para importar el proyecto, siga los pasos explicados en el documento “*Trabajando con proyectos Maven*, *importar un proyecto”*.

Una vez importado, dentro del mismo, debe completar la clase WallPost de acuerdo a la siguiente especificación:

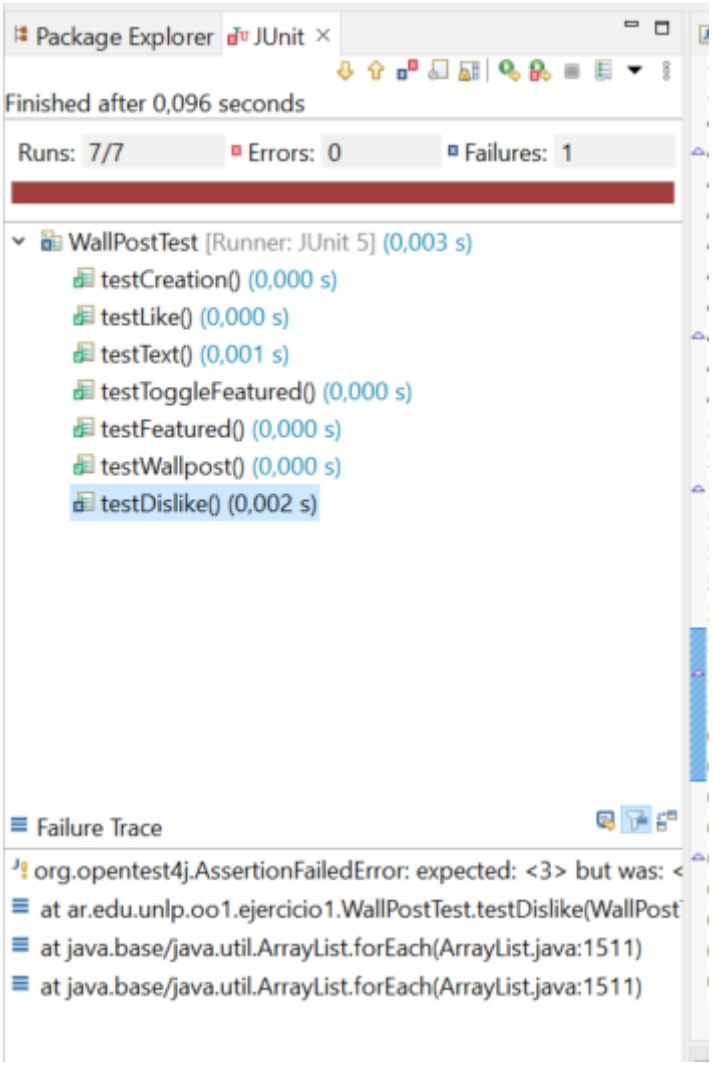
```
/*
* Permite construir una instancia del WallPost.
* Luego de la invocación, debe tener como texto: “Undefined post”,
* no debe estar marcado como destacado y la cantidad de “Me gusta” debe ser 0.
*/
public WallPost()
```

```
/*
* Retorna el texto descriptivo de la publicación
*/
public String getText()

/*
* Asigna el texto descriptivo de la publicación
*/
public void setText (String descriptionText)

/*
* Retorna la cantidad de “me gusta”
*/
public int getLikes()

/*
 * Incrementa la cantidad de likes en uno.
*/
public void like()

/*
 * Decrementa la cantidad de likes en uno. Si ya es 0, no hace nada.
*/
public void dislike()

/*
 * Retorna true si el post está marcado como destacado, false en caso contrario
*/
public boolean isFeatured()

/*
 * Cambia el post del estado destacado a no destacado y viceversa.
*/
public void toggleFeatured()
```

Segunda parte

Utilice los tests provistos por la cátedra para comprobar que su implementación de Wallpost es correcta. Estos se encuentran en el mismo proyecto, en la carpeta test, clase WallPostTest.

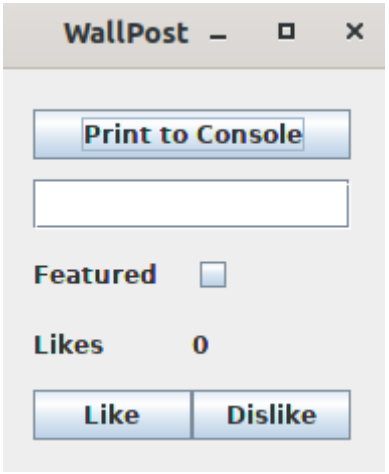
Para ejecutar los tests simplemente haga click derecho sobre el proyecto y utilice la opción Run As >> JUnit Test. Al ejecutarlo, se abrirá una ventana con el resultado de la evaluación de los tests. Siéntase libre de investigar la implementación de la clase de test. Ya veremos en detalle cómo implementarlas.



En el informe, Runs indica la cantidad de test que se ejecutaron. En Errors se indica la cantidad que dieron error y en Failures se indica la cantidad que tuvieron alguna falla, es decir, los resultados no son los esperados. Abajo, se muestra el Failure Trace del test que falló. Si lo selecciona, mostrará el mensaje de error correspondiente a ese test, que le ayudará a encontrar la falla. Si hace click sobre alguno de los test, se abrirá su implementación en el editor.

Tercera parte

Una vez que su implementación pasa los tests de la primera parte puede utilizar la ventana que se muestra a continuación, la cual permite inspeccionar y manipular el post (definir su texto, hacer like / dislike y marcarlo como destacado).



Para visualizar la ventana, sobre el proyecto, usar la opción del menú contextual Run As >> Java Application. La ventana permite cambiar el texto del post, incrementar la cantidad de likes, etc. El botón Print to Console imprimirá los datos del post en la consola.

Ejercicio 2: Balanza Electrónica

En el taller de programación ud programó una balanza electrónica. Volveremos a programarla, con algún requerimiento adicional.

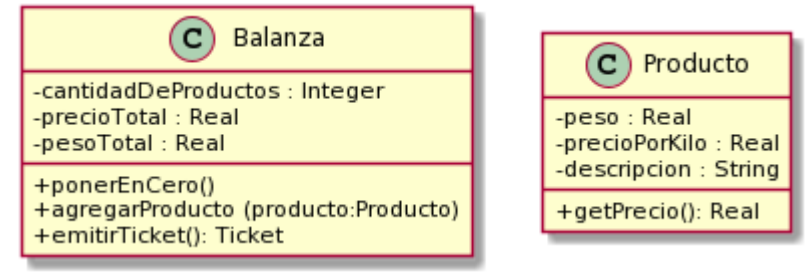
En términos generales, la Balanza electrónica recibe productos (uno a uno), y calcula dos totales: peso total y precio total. Además, la balanza puede poner en cero todos sus valores.

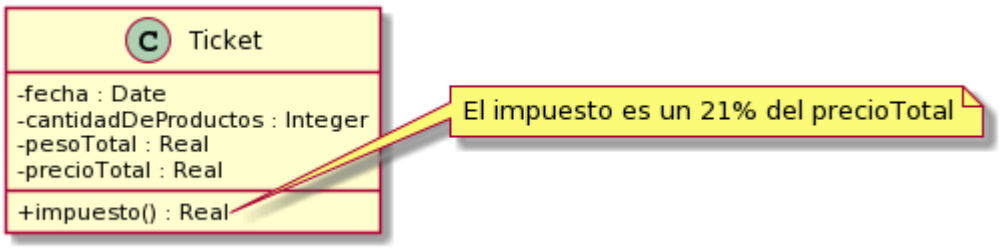
La balanza no guarda los productos. Luego emite un ticket que indica el número de productos considerados, peso total, precio total.

Tareas:

a) **Implemente:**

Cree un nuevo proyecto Maven llamado balanzaElectronica, siguiendo los pasos del documento "*Trabajando con proyectos Maven, crear un proyecto Maven nuevo*". En el paquete correspondiente, programe las clases que se muestran a continuación.





Observe que no se documentan en el diagrama los mensajes que nos permiten obtener y establecer los atributos de los objetos (accessors). Aunque no los incluimos, verá que los tests fallan si no los implementa. Consulte con el ayudante para identificar, a partir de los tests que fallan, cuales son los accessors necesarios (pista: todos menos los setters de balanza).

Todas las clases son subclases de Object.

Nota: Para las fechas, utilizaremos la clase java.time.LocalDate. Para crear la fecha actual, puede utilizar LocalDate.now(). También es posible crear fechas distintas a la actual. Puede investigar más sobre esta clase en https://docs.oracle.com/en/java/javase/11/docs/api/index.html

b) Probando su implementación:

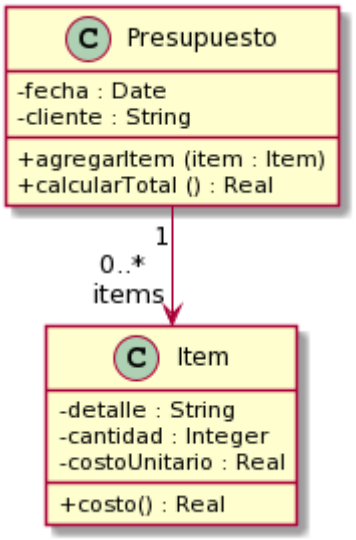
Para realizar este ejercicio, utilice el recurso que se encuentra en el sitio de la cátedra o que puede descargar desde este link. En este caso, se trata de dos clases, BalanzaTest y ProductoTest, las cuales debe agregar dentro del paquete tests. Haga las modificaciones necesarias para que el proyecto no tenga errores.

Si todo salió bien, su implementación debería pasar las pruebas que definen las clases agregadas en el paso anterior. El propósito de estas clases es ejercitar una instancia de la clase Balanza y verificar que se comporta correctamente.

Ejercicio 3: Presupuestos

Un presupuesto se utiliza para detallar los precios de un conjunto de productos que se desean adquirir. Se realiza para una fecha específica y es solicitado por un cliente, proporcionando una visión de los costos asociados.

El siguiente diagrama muestra un diseño para este dominio.



Tareas:

a) **Implemente:**

Defina el proyecto Ejercicio 3 - Presupuesto y dentro de él implemente las clases que se observan en el diagrama. Ambas son subclases de Object.

b) **Discuta y reflexione**

Preste atención a los siguientes aspectos:

- ¿Cuáles son las variables de instancia de cada clase?
- ¿Qué variables inicializa? ¿De qué formas se puede realizar esta inicialización?
- ¿Qué ventajas y desventajas encuentra en cada una de ellas?

c) **Probando su código:**

Utilice los tests provistos para confirmar que su implementación ofrece la funcionalidad esperada. En este caso, se trata de dos clases: ItemTest y PresupuestoTest, que debe agregar dentro del paquete tests. Haga las modificaciones necesarias para que el proyecto no tenga errores. Siéntase libre de explorar las clases de test para intentar entender qué es lo que hacen.

Ejercicio 4: Balanza mejorada

Realizando el ejercicio de los presupuestos, aprendimos que un objeto puede tener una colección de otros objetos. Con esto en mente, ahora queremos mejorar la balanza implementada en el ejercicio 2.

Tarea 1

Mejorar la balanza para que recuerde los productos ingresados (los mantenga en una colección). Analice de qué forma puede realizarse este nuevo requerimiento e implemente el mensaje

```
public List<Producto> getProductos()
```

que retorna todos los productos ingresados a la balanza (en la compra actual, es decir, desde la última vez que se la puso a cero).

¿Qué cambio produce este nuevo requerimiento en la implementación del mensaje *ponerEnCero()* ?

¿Es necesario, ahora, almacenar los totales en la balanza? ¿Se pueden obtener estos valores de otra forma?

Tarea 2
Con esta nueva funcionalidad, podemos enriquecer al Ticket, haciendo que él también conozca a los productos (a futuro podríamos imprimir el detalle). Ticket también debería entender el mensaje `public List<Producto> getProductos()`.

- ¿Qué cambios cree necesarios en Ticket para que pueda conocer a los productos?
- ¿Estos cambios modifican las responsabilidades ya asignadas de realizar cálculo del precio total?. ¿El ticket adquiere nuevas responsabilidades que antes no tenía?

Tarea 3
Después de hacer estos cambios, ¿siguen pasando los tests? ¿Está bien que sea así?

Ejercicio 5: Inversores

Estamos desarrollando una aplicación móvil para que un inversor pueda conocer el estado de sus inversiones. El sistema permite manejar dos tipos de inversiones: Inversión en acciones e inversión en plazo fijo. En todo momento, se desea poder conocer el valor actual de cada inversión y de las inversiones realizadas por el inversor.

Para las inversiones en acciones el valor actual se calcula multiplicando el valor unitario de una acción por la cantidad de acciones que se posee. De las acciones se conoce el nombre que las identifica en el mercado de valores, un inversor puede invertir en diferentes acciones con diferentes valores unitarios. Por su parte, para los plazo fijos, el valor actual consiste en el cálculo del valor inicial de constitución del plazo fijos sumando los intereses diarios desde la fecha de constitución hasta hoy.

De las inversiones en acciones es importante poder conocer su nombre, la cantidad de acciones en las que se invertirá y el valor unitario de cada acción. Por su parte, los plazos fijos se constituyen en una fecha, es importante conocer el monto depositado y cuál es el porcentaje de interés que genera.

Por último, el valor de inversión actual de un inversor es la suma de los valores actuales de todas las inversiones que posee. Un inversor puede agregar y sacar inversiones de su cartera de inversiones cuando lo desee. Las inversiones pueden ser tanto en acciones como en plazo fijos y pueden estar mezcladas.

Tareas:

1. Realice la lista de conceptos candidatos. Clasifique cada concepto dentro de las categorías vistas en la teoría.
2. Grafique el modelo de dominio usando UML.
3. Actualice el modelo de dominio incorporando los atributos a los conceptos
4. Agregue asociaciones entre conceptos, indicando para cada una de ellas la categoría a la que pertenece, de acuerdo a lo explicado en la teoría, y demás atributos, según sea necesario.

Ejercicio 6: Distribuidora Eléctrica

Una distribuidora eléctrica desea gestionar los consumos de sus usuarios para la emisión de facturas de cobro.
De cada usuario se conoce su nombre y domicilio. Se considera que cada usuario sólo puede tener un único domicilio en donde se registran los consumos.

Los consumos de los usuarios se dividen en dos componentes:

- **Consumo de energía activa**: tiene un costo asociado para el usuario. Se mide en kWh (kilowatt/hora).
- **Consumo de energía reactiva**: no genera ningún costo para el usuario, es decir, se utiliza solamente para determinar si hay alguna bonificación. Se mide en kVArh (kilo voltio-amperio reactivo hora).

Se cuenta con un **cuadro tarifario** que establece el precio del kWh para calcular el costo del consumo de energía activa. Este cuadro tarifario puede ser ajustado periódicamente según sea necesario (por ejemplo, para reflejar cambios en los costos).

Para emitir la factura de un cliente se tiene en cuenta **solo su último consumo registrado**. Los datos que debe contener la factura son los siguientes:

- El usuario a quien se está cobrando.
- La fecha de emisión.
- La bonificación, sí aplica.
- El monto final de la factura: se calcula restando la bonificación al costo del consumo:
    - El costo del consumo se calcula multiplicando el consumo de energía activa por el **precio del kWh** proporcionado por el cuadro tarifario.
    - Se calcula su **factor de potencia** para determinar si hay alguna bonificación aplicable. Si el factor de potencia estimado (fpe) del último consumo del usuario es mayor a 0.8, el usuario recibe una bonificación del 10%.

Tareas:

1. Realice la lista de conceptos candidatos.
2. Grafique el modelo de dominio usando UML.

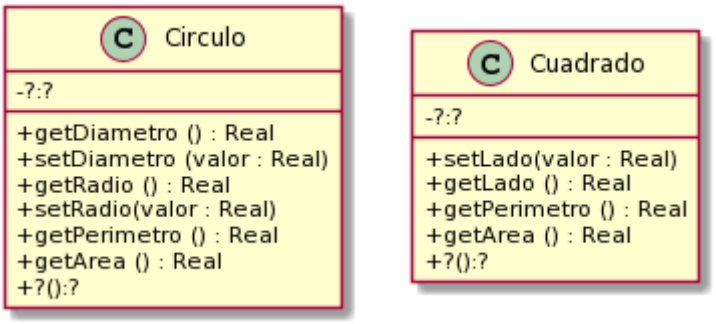
3. Actualice el modelo de dominio incorporando los atributos a los conceptos
4. Agregue asociaciones entre conceptos, indicando para cada una de ellas la categoría a la que pertenece, de acuerdo a lo explicado en la teoría, y demás atributos, según sea necesario.

Ejercicio 7: Figuras y cuerpos

Figuras en 2D

En Taller de Programación definió clases para representar figuras geométricas. Retomaremos ese ejercicio para trabajar con Cuadrados y Círculos.

El siguiente diagrama de clases documenta los mensajes que estos objetos deben entender.



Fórmulas y mensajes útiles:

- Diámetro del círculo: radio * 2
- Perímetro del círculo: π * diámetro
- Área del círculo: π * radio ^{2}
- π se obtiene enviando el mensaje #pi a la clase Float (Float pi) (ahora Math.PI)

Tareas:

a) Implementación:

Defina un nuevo proyecto figurasYCuerpos. Implemente las clases Círculo y Cuadrado, siendo ambas subclases de Object. Decida usted qué variables de instancia son necesarias. Puede agregar mensajes adicionales si lo cree necesario.

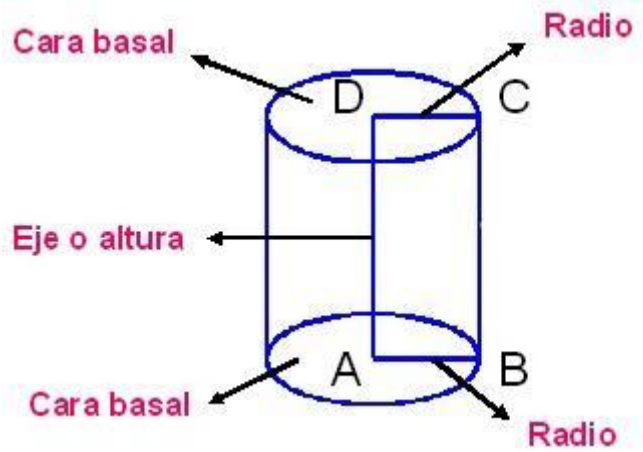
b) Discuta y reflexione

¿Qué variables de instancia definió? ¿Pudo hacerlo de otra manera? ¿Qué ventajas encuentra en la forma en que lo realizó?

Cuerpos en 3D

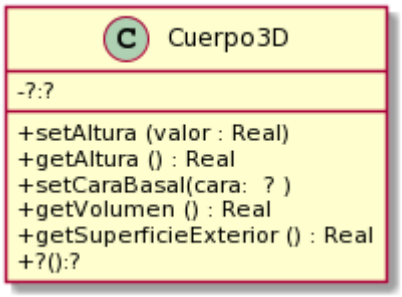
Ahora que tenemos Círculos y Cuadrados, podemos usarlos para construir cuerpos (en 3D) y calcular su volumen y superficie o área exterior.

- Vamos a pensar a un cilindro como "un cuerpo que tiene una figura 2D como cara basal y que tiene una altura (vea la siguiente imagen)". Es decir, si en el lugar de la figura2D tuviera un círculo, se formaría el siguiente cuerpo 3D.



- Si reemplazamos la cara basal por un rectángulo, tendremos un prisma (una caja de zapatos).

El siguiente diagrama de clases documenta los mensajes que entiende un cuerpo3D.



Fórmulas útiles:

- El área o superficie exterior de un cuerpo es:
  2* área-cara-basal + perímetro-cara-basal * altura-del-cuerpo
- El volumen de un cuerpo es: área-cara-basal * altura

Más info interesante: A la figura que da forma al cuerpo (el círculo o el cuadrado en nuestro caso) se le llama directriz. Y a la recta en la que se mueve se llama generatriz. En wikipedia (Cilindro)[1] se puede aprender un poco más al respecto.

Tareas:

a) Implementación

Implemente la clase Cuerpo 3D, la cuál es subclase de Object. Decida usted qué variables de instancia son necesarias. También decida si es necesario hacer cambios en las figuras 2D.

b) Pruebas automatizadas

Siguiendo los ejemplos de ejercicios anteriores, ejecute las pruebas automatizadas provistas. En este caso, se trata de tres clases (CuerpoTest, TestCirculo y

[1] https://es.wikipedia.org/wiki/Cilindro

TestCuadrado) que debe agregar dentro del paquete tests. Haga las modificaciones necesarias para que el proyecto no tenga errores. Si algún test no pasa, consulte al ayudante.

c) Discuta y reflexione

Discuta con el ayudante sus elecciones de variables de instancia y métodos adicionales. ¿Es necesario todo lo que definió?

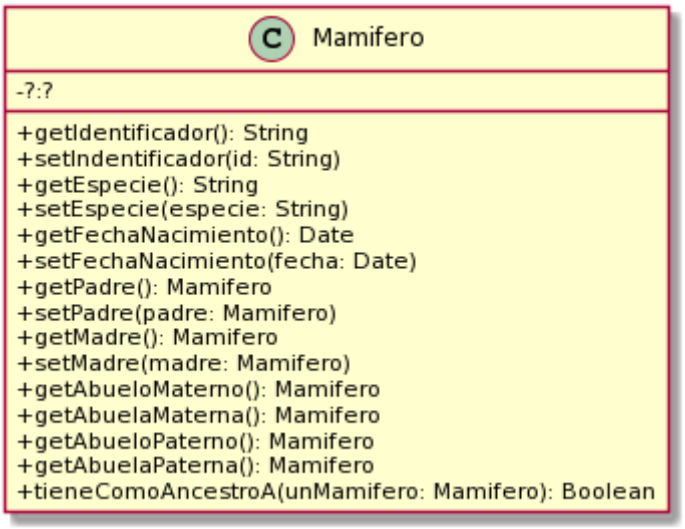
Ejercicio 8: Genealogía salvaje

En una reserva de vida salvaje (como la estación de cría ECAS, en el camino Centenario), los cuidadores quieren llevar registro detallado de los animales que cuidan y sus familias. Para ello nos han pedido ayuda. Debemos:

Tareas:

a) Complete el diseño e implemente

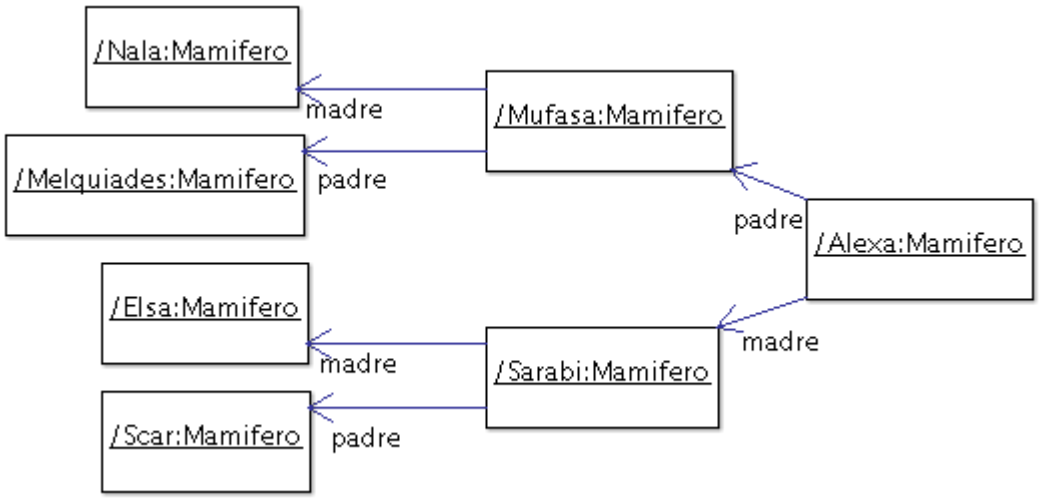
Modelar una solución en objetos e implementar la clase Mamífero (como subclase de Object). El siguiente diagrama de clases (incompleto) nos da una idea de los mensajes que un mamífero entiende.
Proponga una solución para el método *tieneComoAncestroA(...) y deje la implementación para el final y discuta su solución con el ayudante.*



Complete el diagrama de clases para reflejar los atributos y relaciones requeridas en su solución.

b) Pruebas automatizadas

Siguiendo los ejemplos de ejercicios anteriores, ejecute las pruebas automatizadas provistas. En este caso, se trata de una clase, MamiferoTest, que debe agregar dentro del paquete tests. En esta clase se trabaja con la familia mostrada en la siguiente figura.



En el diagrama se puede apreciar el nombre/identificador de cada uno de ellos (por ejemplo Nala, Mufasa, Alexa, etc).

Haga las modificaciones necesarias para que el proyecto no tenga errores. Si algún test no pasa, consulte al ayudante.

Ejercicio 9: Red de Alumbrado

Imagine una red de alumbrado donde cada farola está conectada a una o varias vecinas formando un grafo conexo[2]. 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 (inicialmente están apagadas)
2. conectar farolas a tantas vecinas como uno quiera (las conexiones son bi-direccionales)
3. encender una farola (y obtener el efecto antes descrito)
4. apagar una farola (y obtener el efecto antes descrito)

Tareas:

a) Modele e implemente

1. Realice el diagrama UML de clases de la solución al problema.

2. Implemente en Java, la clase Farola, como subclase de Object, con los siguientes métodos:

```
/*
* Crear una farola. Debe inicializarla como apagada
*/
public Farola ()
```

[2] https://es.wikipedia.org/wiki/Grafo_conexo

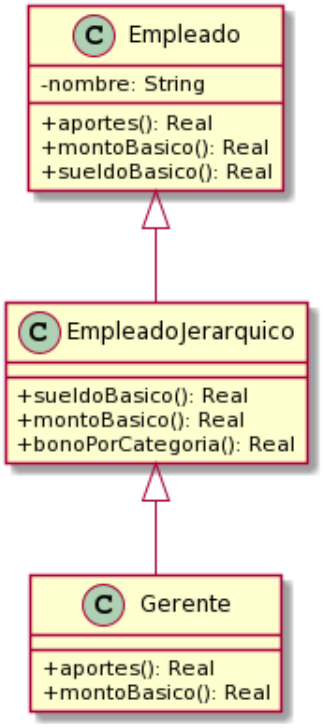
```
/*
* Crea la relación de vecinos entre las farolas. La relación de vecinos
entre las farolas es recíproca, es decir el receptor del mensaje será vecino
de otraFarola, al igual que otraFarola también se convertirá en vecina del
receptor del mensaje
*/
public void pairWithNeighbor( Farola otraFarola )

/*
* Retorna sus farolas vecinas
*/
public List<Farola> getNeighbors ()

/*
* Si la farola no está encendida, la enciende y propaga la acción.
*/
public void turnOn()

/*
* Si la farola no está apagada, la apaga y propaga la acción.
*/
public void turnOff()

/*
* Retorna true si la farola está encendida.
*/
public boolean isOn()

/*
* Retorna true si la farola está apagada.
*/
public boolean isOff()
```

b) Verifique su solución con las pruebas automatizadas

Utilice los tests provistos por la cátedra para probar las implementaciones del punto 2.

Ejercicio 10: Method lookup con Empleados

Sea la jerarquía de `Empleado` como muestra la figura de la izquierda, cuya implementación de referencia se incluye en la tabla de la derecha.



Empleado	EmpleadoJerarquico	Gerente
`public double montoBasico() { return 35000; }`	`public double sueldoBasico() { return super.sueldoBasico()+ this.bonoPorCategoria(); }`	`public double aportes() { return this.montoBasico() * 0.05d; }`
`public double aportes(){ return 13500; }`	`public double montoBasico() { return 45000; }`	`public double montoBasico() { return 57000; }`
`public double sueldoBasico() { return this.montoBasico() + this.aportes();}`	`public double bonoPorCategoria() { return 8000; }`	

Analice cada uno de los siguientes fragmentos de código y resuelva las tareas indicadas abajo:

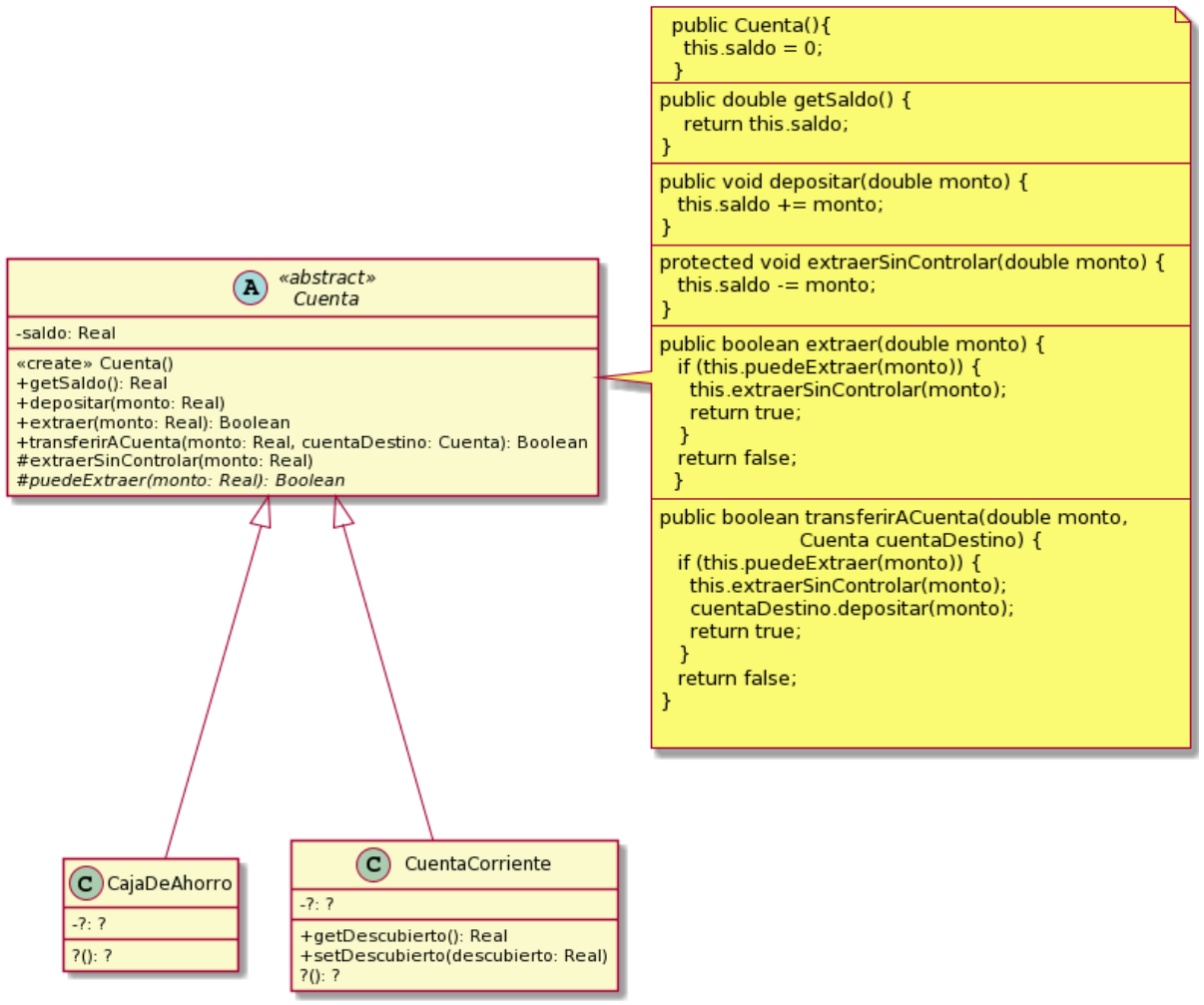
Gerente alan = **new** Gerente("Alan Turing"); **double** aportesDeAlan = alan.aportes();	Gerente alan = **new** Gerente("Alan Turing"); **double** sueldoBasicoDeAlan = alan.sueldoBasico();

Tareas:

1. Liste todos los métodos, indicando nombre y clase, que son ejecutados como resultado del envío del último mensaje de cada fragmento de código (por ejemplo, (1) método +aportes de la clase Empleado, (2) ...)
2. ¿Qué valores tendrán las variables aportesDeAlan y sueldoBasicoDeAlan luego de ejecutar cada fragmento de código?

Ejercicio 11: Cuenta con ganchos

Observe con detenimiento el diseño que se muestra en el siguiente diagrama. La clase cuenta es *abstracta*. El método `puedeExtraer()` es abstracto. Las clases `CajaDeAhorro` y `CuentaCorriente` son concretas y están incompletas.



Tarea A: Complete la implementación de las clases `CajaDeAhorro` y `CuentaCorriente` para que se puedan efectuar depósitos, extracciones y transferencias teniendo en cuenta los siguientes criterios.

1) Las **cajas de ahorro** *solo pueden extraer y transferir* cuando cuentan con fondos suficientes.
2) Las extracciones, los depósitos y las transferencias desde **cajas de ahorro** tienen un costo adicional de 2% del monto en cuestión (téngalo en cuenta antes de permitir una extracción o transferencia desde caja de ahorro).
3) Las **cuentas corrientes** pueden extraer aún cuando el saldo de la cuenta sea insuficiente. Sin embargo, no deben superar cierto límite por debajo del saldo. Dicho límite se conoce como límite de descubierto (algo así como el máximo saldo negativo permitido). Ese límite es diferente para cada cuenta (lo negocia el cliente con la gente del banco).
4) Cuando se abre una **cuenta corriente**, su límite descubierto es 0 (no olvide definir el constructor por default).

Tarea B: Reflexione, charle con el ayudante y responda a las siguientes preguntas.

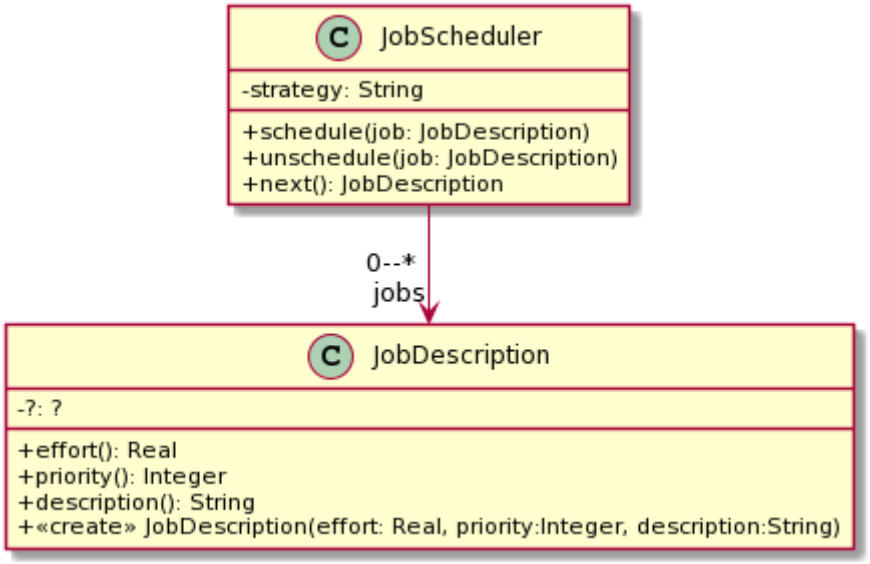
a) ¿Por qué cree que este ejercicio se llama "Cuenta con ganchos"?

b) En las implementaciones de los métodos `extraer()` y `transferirACuenta()` que se ven en el diagrama, ¿quién es `this`? ¿Puede decir de qué clase es `this`?
c) ¿Por qué decidimos que los métodos `puedeExtraer()` y `extraerSinControlar` tengan visibilidad "protegido"?
d) ¿Se puede transferir de una caja de ahorro a una cuenta corriente y viceversa? ¿por qué? ¡Pruébelo!
e) ¿Cómo se declara en Java un método abstracto? ¿Es obligatorio implementarlo? ¿Qué dice el compilador de Java si una subclase no implementa un método abstracto que hereda?

Tarea C: Escriba los tests de unidad que crea necesarios para validar que su implementación funciona adecuadamente.

Ejercicio 12: Job Scheduler

El JobScheduler es un objeto cuya responsabilidad es determinar qué trabajo debe resolverse a continuación. El siguiente diseño ayuda a entender cómo funciona la implementación actual del JobScheduler.



- El mensaje `schedule(job: JobDescription)` recibe un job (trabajo) y lo agrega al final de la colección de trabajos pendientes.
- El mensaje `next()` determina cuál es el siguiente trabajo de la colección que debe ser atendido, lo retorna, y lo quita de la colección.

En la implementación actual del método `next()`, el JobScheduler utiliza el valor de la variable `strategy` para determinar cómo elegir el siguiente trabajo.

Dicha implementación presenta dos serios problemas de diseño:
- Secuencia de ifs (o sentencia switch/case) para implementar alternativas de un mismo comportamiento.
- Código duplicado.

Tareas:

1) **Analice el código existente**
   Utilice el código y los tests provistos por la cátedra y aplique lo aprendido (en particular en relación a herencia y polimorfismo) para eliminar los problemas mencionados. Siéntase libre de agregar nuevas clases como considere necesario. También puede cambiar la forma en la que los objetos se crean e inicializan. Asuma que una vez elegida una estrategia para un scheduler no puede cambiarse.
2) **Verifique su solución con las pruebas automatizadas**
   Sus cambios probablemente hagan que los tests dejen de funcionar. Corríjalos y mejórelos como sea necesario.

Ejercicio 13: ¡A implementar Inversores!

Retomando el Ejercicio 5, trabajamos en el diseño y modelado UML de un sistema de inversiones, donde definimos las clases, atributos y asociaciones necesarias para representar inversores y sus diferentes tipos de inversiones. Ahora es el momento de llevar el diseño a la práctica: vamos a implementar en Java lo diseñado y asegurar su calidad mediante pruebas automatizadas.

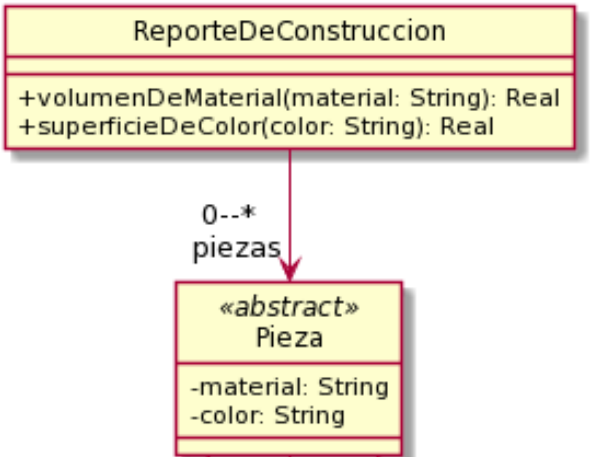
Tareas:

1. **Implemente**
   a. Realice el mapeo del modelo conceptual, a un diagrama de clases de UML.
   b. Implemente en Java lo necesario para que se pueda conocer el valor actual de cada inversión. Y también el monto total de las inversiones realizadas por un inversor.
2. **Pruebas automatizadas**
   a. Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.
   b. Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.
3. **Discuta con el ayudante**
   a. Consulte con un ayudante los casos de prueba diseñados

Ejercicio 14: Volumen y superficie de sólidos

Una empresa siderúrgica quiere introducir en su sistema de gestión nuevos cálculos de volumen y superficie exterior para las piezas que produce. El volumen le sirve para determinar cuánto material ha utilizado. La superficie exterior le sirve para determinar la cantidad de pintura que utilizó para pintar las piezas.

El siguiente diagrama UML muestra el diseño actual del sistema. En el mismo puede observarse que un ReporteDeConstruccion tiene la lista de las piezas que fueron construidas. Pieza es una clase abstracta.



Tareas:

a) **Complete el diseño e implemente**

Su tarea es completar el diseño considerando que las piezas pueden ser **cilindros, esferas** o **prismas rectangulares**.

Luego complete la implementación de la clase **ReporteDeConstruccion** siguiendo la especificación que se muestra a continuación:

*volumenDeMaterial(nombreDeMaterial: String***)**

> "Recibe como parámetro un nombre de material (un string, por ejemplo 'Hierro'). Retorna la suma de los volúmenes de todas las piezas hechas en ese material"

superficieDeColor(unNombreDeColor: String)

> "Recibe como parámetro un color (un string, por ejemplo 'Rojo'). Retorna la suma de las superficies externas de todas las piezas pintadas con ese color".

Fórmulas a utilizar

- **Cilindro**
  Volumen de un cilindro: π * radio ^{2} * h.
  Superficie de un cilindro: 2 * π * radio * h + 2 * π * radio ^{2}
- **Esfera**
  Volumen de una esfera: ⅓ * π * radio ^{3}.
  Superficie de una esfera: 4 * π * radio ^{2}
- ***Prisma Rectangular***
  Volumen del prisma: ladoMayor * ladoMenor * altura
  Superficie del prisma: 2 * (ladoMayor * ladoMenor + ladoMayor * altura + ladoMenor * altura)

- Para obtener π, utilizamos Math.PI
- Para elevar un número a cualquier potencia, utilizamos Math.pow(numero: double, potencia: double). Ej: 8^2 = Math.pow(8, 2)

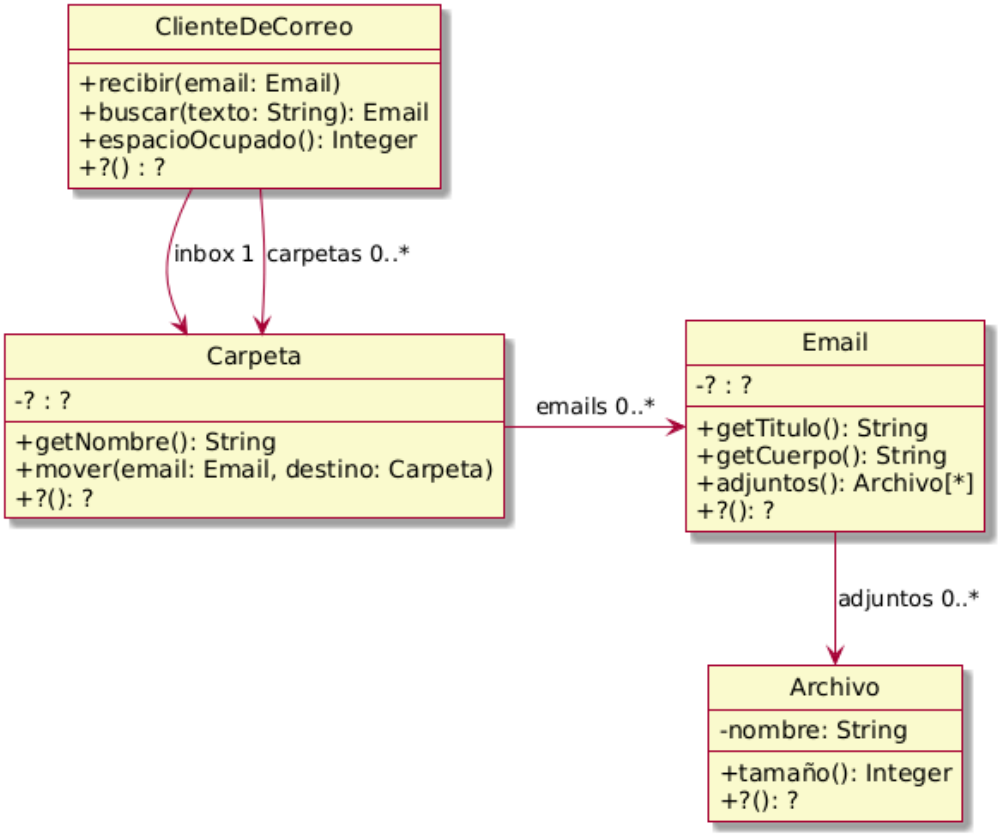
b) **Pruebas automatizadas**
Implemente los tests (JUnit) que considere necesarios.

c) Discuta con el ayudante

Es probable que note una similitud entre este ejercicio y el de "Figuras y cuerpos" que realizó anteriormente, ya que en ambos se pueden construir cilindros y prismas rectangulares. Sin embargo las implementaciones varían. Enumere las diferencias y similitudes entre ambos ejercicios y luego consulte con el ayudante.

Ejercicio 15: Cliente de Correo

El diagrama de clases de UML que se muestra a continuación documenta parte del diseño simplificado de un cliente de correo electrónico.



Su funcionamiento es el siguiente:

- En respuesta al mensaje **recibir**, almacena en el inbox (una de las carpetas) el email que recibe como parámetro.
- En respuesta al mensaje **mover**, mueve el email desde una carpeta de origen a una carpeta destino (asuma que el email está en la carpeta origen cuando se recibe este mensaje).
- En respuesta al mensaje **buscar** retorna el primer email en el Cliente de Correo cuyo título o cuerpo contienen el texto indicado como parámetro. Busca en todas las carpetas.

- En respuesta al mensaje **espacioOcupado**, retorna la suma del espacio ocupado por todos los emails de todas las carpetas.
- El tamaño de un email es la suma del largo del título, el largo del cuerpo, y del tamaño de sus adjuntos.
- Para simplificar, asuma que el tamaño de un archivo es el largo de su nombre.

Tareas:

a) **Modele e implemente**
   i) Complete el diseño y el diagrama de clases UML.
   ii) Implemente en Java la funcionalidad requerida.

b) **Pruebas automatizadas**
   i) Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.
   ii) Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior

Ejercicio 16: Intervalo de tiempo

En Java, las fechas se representan normalmente con instancias de la clase java.time.LocalDate. Se pueden crear con varios métodos "`static`" como por ejemplo `LocalDate.now()`.

- Investigue cómo hacer para crear una fecha determinada, por ejemplo 15/09/1972.
- Investigue cómo hacer para determinar si la fecha de hoy se encuentra entre las fechas 15/12/1972 y 15/12/2032. Sugerencia: vea los métodos permiten comparar LocalDates y que retornan `booleans`.
- Investigue cómo hacer para calcular el número de días entre dos fechas. Lo mismo para el número de meses y de años Sugerencia: vea el método `until`.
  Tenga en cuenta que los métodos de LocalDate colaboran con otros objetos que están definidos a partir de enums, clases e interfaces de java.time; por ejemplo `java.time.temporal.ChronoUnit.DAYS`

Tareas:

a) **Implemente**
   Implemente la clase **DateLapse** (Lapso de tiempo). Un objeto DateLapse representa el lapso de tiempo entre dos fechas determinadas. La primera fecha se conoce como "from" y la segunda como "to". Una instancia de esta clase entiende los mensajes:

```
public LocalDate getFrom()
"Retorna la fecha de inicio del rango"

public LocalDate getTo()
"Retorna la fecha de fin del rango"

public int sizeInDays()
"retorna la cantidad de di´as entre la fecha 'from' y la fecha 'to'"
```

```
public boolean includesDate(LocalDate other)
"recibe un objeto LocalDate y retorna true si la fecha esta´ entre el from y
el to del receptor y false en caso contrario".
```

b) Pruebas automatizadas

i) Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.

ii) Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior

Ejercicio 17: Intervalo de tiempo (¡otra vez!)

Asumiendo que implementó la clase **DateLapse** con dos variables de instancia "from" y "to", realice otra implementación de la clase para que su representación sea a través de los atributos "from" y "sizeInDays" y coloquela en otro paquete. Es decir, debe basar su nueva implementación en estas variables de instancia solamente.

Los cambios en la estructura interna de un objeto sólo deben afectar a la implementación de sus métodos. Estos cambios deben ser transparentes para quien le envía mensajes, no debe notar ningún cambio y seguir usándolo de la misma forma. Tenga en cuenta que los tests que implementó en el ejercicio anterior deberían pasar **sin que se requiera realizar modificaciones**. Solamente deberá actualizar el setup, es decir, la instanciación del intervalo de tiempo.

Sugerencia: Para facilitar el uso de estas clases en los tests, considere definir una interfaz Java que ambas soluciones implementen.

Ejercicio 18: Filtered Set

En el apunte de uso de colecciones se detalla el protocolo de la interface Collection y se ejemplifica el uso mediante la clase concreta ArrayList. Existen otras implementaciones de Collection que tienen ciertas diferencias, en particular, el **Set** (*java.util.Set*) es una colección que no admite duplicados y no tiene un índice para acceder a sus elementos.
Implemente una clase **EvenNumberSet** (conjunto de números pares). Esta especialización se diferencia en que únicamente permite agregar números enteros que sean pares. Por simplicidad, considere únicamente el tipo de datos **Integer** para su solución (ignore el resto de tipos de datos numéricos).
Tenga en cuenta que la clase **EvenNumberSet** debe implementar la interface **Set<Integer>** de Java. Esto significa que a las variables de tipo *Set<Integer>* se les puede asignar un objeto concreto de tipo **EvenNumberSet** y luego utilizarlo enviando los mensajes que están definidos en el protocolo de **Set<Integer>**.

El siguiente fragmento de código ejemplifica cómo se podría usar la clase **EvenNumberSet**:

```
Set<Integer> numbers = new EvenNumberSet();
```

```
// inicialmente el Set está vacío => []
numbers.add(1); // No es par, entonces no se agrega => []
numbers.add(2); // Es par, se agrega al set => [2]
numbers.add(4); // Es par, se agrega al set => [2, 4]
numbers.add(2); // Es par, pero ya está en el set, no se agrega => [2, 4]
```

Evalúe las distintas opciones para implementar la clase **EvenNumberSet**. Para evitar reinventar la rueda, considere reutilizar alguna de las clases existentes en Java que ofrezcan funcionalidades similares.

Tareas:

a. Investigue qué clases se pueden utilizar para implementar la clase **EvenNumberSet**. Consulte la documentación de Set.

b. Explique brevemente cómo propone utilizar las clases investigadas anteriormente para implementar su solución. Por ejemplo:
- "Se debe subclasificar una determinada clase y redefinir un método para que haga lo siguiente"
- "Se debe crear una nueva clase que contenga un objeto de un determinado tipo al cual se le delegará está responsabilidad"

c. Implemente en Java las alternativas que haya propuesto.

d. Implemente tests automatizados utilizando JUnit para verificar sus implementaciones.

e. Compare las soluciones y liste las ventajas y desventajas de cada una.

Ejercicio 19: Alquiler de propiedades

Se desea diseñar e implementar una **plataforma para gestión de reservas** de **propiedades** que llamaremos OOBnB. En la misma, los **usuarios** pueden gestionar sus **inmuebles** para su **alquiler** así como también realizar **reservas** sobre estos.
De los usuarios se conoce el **nombre**, la **dirección** y el **DNI**. Cada usuario posee propiedades que desea alquilar, de las cuales se guarda la **dirección**, un **nombre descriptivo** y el **precio** que se desea cobrar por noche. Además, los usuarios pueden realizar reservas sobre cualquiera de las propiedades disponibles.

Nos piden implementar la siguiente funcionalidad:

- **Consultar la disponibilidad de una propiedad en un período específico**: dada una propiedad, una fecha inicial y una fecha final, se debe determinar si la propiedad está disponible el período indicado.
- **Crear una reserva**: Un usuario puede realizar una reserva para un período de tiempo determinado. Si la propiedad está disponible, se crea la reserva y la propiedad pasa a estar ocupada durante ese período. Si no lo está, la reserva no será creada.
- **Calcular el precio de una reserva**: Dada una reserva, se debe poder calcular su precio. El mismo se obtiene multiplicando la cantidad de noches por el precio por noche.

- **Cancelar una reserva:** Se debe permitir cancelar una reserva. En este caso, la propiedad pasa a estar disponible durante el período de tiempo indicado en la reserva. Esta operación sólo es permitida si el período de la reserva no está en curso.
- **Calcular los ingresos de un propietario:** Se debe calcular la retribución a un propietario, la cual es el 75% de la suma de precio totales de las reservas incluidas en un período específico de tiempo.

Sugerencia:
Para el manejo de los períodos de reserva se sugiere añadir un nuevo método a la interfaz **DateLapse** definida en el ejercicio anterior (**ejercicio de Intervalos de tiempo**).

A modo de sugerencia, la especificación del mismo puede ser la siguiente:

```
/**
      Retorna true si el período de tiempo del receptor se superpone con el
      recibido por parámetro
**/
public boolean overlaps (DateLapse anotherDateLapse)
```

Tareas:

a) **Modelado de dominio**
   i) Realice la lista de conceptos candidatos, teniendo en cuenta los métodos vistos en la teoría.
   ii) Grafique el modelo de dominio usando UML.
   iii) Actualice el modelo de dominio incorporando los atributos a los conceptos
   iv) Agregue asociaciones entre conceptos, indicando para cada una de ellas la categoría a la que pertenece, de acuerdo a lo explicado en la teoría, y demás atributos, según sea necesario.

b) **Modelado e Implementación**
   i) Diagrama de clases UML a partir del modelo de dominio realizado en el punto anterior.
   ii) Implemente en Java la funcionalidad requerida.

c) **Pruebas automatizadas**
   i) Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría para verificar la disponibilidad de una propiedad en una fecha, reservar y cancelar una reserva.
   ii) Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.

Ejercicio 20. Políticas de cancelación

A la implementación del ejercicio anterior se quiere extender la funcionalidad de cancelar una reserva de manera tal que calcule el monto que será reembolsado (devuelto) al inquilino. Para hacer esto posible, las propiedades deben conocer una política de

cancelación que se define al momento de crearla. Esta política puede ser una de las siguientes: flexible, moderada, o estricta y **puede cambiarse en cualquier momento.**

Al momento de reembolsar, las políticas se comportan de la siguiente manera:

a) **Política de cancelación flexible:** reembolsa el monto total sin importar la fecha de cancelación (que de todas maneras debe ser anterior a la fecha de inicio de la reserva).
b) **Política de cancelación moderada:** reembolsa el monto total si la cancelación se hace hasta una semana antes y 50% si se hace hasta 2 días antes.
c) **Política de cancelación estricta**: no reembolsará nada (0, cero) sin importar la fecha tentativa de cancelación.

Actualice su diseño, implementación y tests de acuerdo a los nuevos requerimientos.

Ejercicio 21. Servicio de envíos de paquetes

Una **empresa de envíos de paquetes** ofrece a sus clientes distintos servicios, como envíos locales, interurbanos e internacionales. Los envíos locales son envíos dentro de la misma ciudad y cuentan con una opción de entrega rápida. Los envíos interurbanos son envíos entre ciudades. Los envíos internacionales son envíos a destinos fuera del país y también cuentan con una opción de envío rápido.

De cada envío se registra la fecha en la cual se realiza el despacho, la dirección de origen, la dirección de destino y el peso expresado en gramos. Para los interurbanos además, la distancia expresada en km.

La empresa trabaja con dos tipos de clientes: **personas físicas**, que son individuos, y **clientes corporativos,** empresas que tienen un volumen alto de envíos. De las personas físicas, se conoce el nombre, dirección y DNI. De los clientes corporativos se conoce nombre de la empresa, dirección y CUIT.

Nos piden implementar la siguiente funcionalidad:
Agregar un envío para un cliente: dado un cliente y un envío, se agrega ese envío al cliente indicado.

Monto a pagar por los envíos realizados dentro de un período. Se indica el cliente para el cual se quiere calcular el monto y las fechas de inicio y fin del período a considerar. Para calcular el monto total a pagar, se suma el costo de todos los envíos despachados durante el período especificado.

- Los envíos locales tienen un costo fijo de $1000 para las entregas estándar y $500 adicional por entrega rápida .
- Los envíos interurbanos tienen un costo que depende de la distancia entre el origen y el destino (utilice $20 para menos de 100 km por cada gramo de peso, $25 para distancias entre 100 km y 500 km por gramo de peso, y $30 para distancias de más de 500 km por gramo de peso).
- Los envíos internacionales tienen un costo que depende del país destino y del peso del paquete. Por ahora, utilice $5000 para cualquier destino y $10 por gramo de

peso para envíos de hasta 1 kg y $12 para envíos de más de 1 kg. Además si tiene entrega rápida debe cobrarse un adicional de 800$

Los envíos efectuados por personas físicas tienen un 10% de descuento.

Tareas:

a) **Modelado**
   i) Realice la lista de conceptos candidatos, teniendo en cuenta los métodos vistos en la teoría.
   ii) Grafique el modelo de dominio usando UML.
   iii) Actualice el modelo de dominio incorporando los atributos a los conceptos
   iv) Agregue asociaciones entre conceptos, indicando para cada una de ellas la categoría a la que pertenece, de acuerdo a lo explicado en la teoría, y demás atributos, según sea necesario.

b) **Implementación**
   i) Diagrama de clases UML a partir del modelo de dominio realizado en el punto anterior.
   ii) Implemente en Java la funcionalidad requerida.

c) **Pruebas automatizadas**
   i) Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.
   ii) Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.

d) Es probable que los montos utilizados para los cálculos le hayan quedado fijos dentro del código (hardcoded). Piense qué pasaría si al calcular el monto a pagar se proveyera (como un parámetro más) el "cuadro tarifario". ¿Cómo sería ese objeto? ¿Qué responsabilidad le podría delegar? ¿Cómo haríamos para tener montos diferentes para los distintos países en los envíos internacionales según los pesos de los envíos?

Ejercicio 22. Sistema de pedidos

Se desea diseñar un sistema para una empresa que ofrece viandas preparadas y artículos de supermercado. De las viandas se conoce su nombre, precio, si es apta para celíacos y disponibilidad. De los artículos de supermercado se conoce su nombre, categoría, precio, peso en kg y stock actual.

Para realizar un pedido, no es necesario registrarse en la aplicación, alcanza con indicar el email al momento de solicitar los productos. Por cuestiones logísticas, cada pedido deberá ser de uno de los dos rubros: supermercado o viandas. Para coordinar la entrega, se puede optar por envío por delivery, en ese caso se debe indicar la dirección y fecha/hora en la que se recibirá el pedido; o bien retiro en sucursal, para lo cual deberá indicar el DNI de la persona autorizada a retirar y la fecha/hora estimadas en la que retira.

La empresa dispone de cocina sin TACC, por lo que es muy importante al momento de realizar un pedido que se indique sí es apto celíaco o no.
Los pedidos de supermercado pueden obtener un 10% de descuento en caso de que se utilice un cupón de descuento al solicitarlos.

Queremos que el sistema permita las siguientes funcionalidades:

- **Crear pedido:** Se indica el tipo de pedido, los productos solicitados (por simplicidad, se puede pedir sólo una unidad de cada uno) y la forma de entrega. En caso de solicitar viandas, debe indicar sí es apto celíaco. En caso de solicitar artículos de supermercado, se puede incluir un cupón de descuento.
- **Calcular costo pedido**: Dado un pedido, se calcula el costo sumando todos los precios de los productos que contenga el pedido.
- **Calcular peso pedido:** Dado un pedido de supermercado, se calcula el peso sumando todos los pesos de los productos que contenga el pedido.
- **Calcular tiempo estimado de entrega:** Dado un pedido y de acuerdo a la forma de entrega, se calculan los minutos estimados hasta la entrega de la siguiente manera:
    - Retiro: Es la diferencia entre la fecha/hora de creación del pedido y la fecha/hora estimada de retiro
    - Envío por delivery: La empresa lo calcula en tiempo real utilizando GPS, el estado del tráfico y fórmulas de distancia.
- **Agregar artículo a pedido**: Agrega el artículo al pedido siempre y cuando su tipo sea supermercado y no supere el peso máximo de 35 kg.
- **Agregar vianda a pedido**: Agrega la vianda al pedido siempre y cuando su tipo sea de viandas y falten menos de 40 min estimados para su entrega; sí el pedido es sólo apto celiaco, sólo admitirá viandas apto celíaco.

Tarea: Realice un diagrama de clases UML donde identifique todas las clases del dominio, sus atributos, roles, cardinalidades y asociaciones. No debe incluir comportamiento.

Ejercicio 23. Poolcar

Poolcar es una aplicación para organizar viajes de larga distancia, que permite compartir gastos y abaratar el costo del viaje.

De cada usuario se conoce su nombre y dirección. Además se mantiene un estado de cuenta, que se usa para el pago de sus viajes. Al momento de su alta, los usuarios se registran como conductor o pasajero, lo cual es una característica que no se puede modificar.

Los conductores deben registrar al menos un vehículo. De cada uno se conoce una descripción, su capacidad (incluyendo al conductor), el año de fabricación y el valor de mercado. Un vehículo pertenece siempre a un único conductor, y cada conductor sólo podrá viajar en calidad de chofer de su propio vehículo.

Los viajes son creados por los conductores y quedan asociados a uno de sus vehículos. De cada viaje se registra el costo total, la localidad de origen, la localidad de destino y la fecha de salida.

Los pasajeros pueden inscribirse en los viajes creados por conductores, siempre y cuando haya lugares disponibles en el vehículo y su saldo de cuenta sea positivo. La inscripción sólo está permitida hasta dos días antes de la fecha de salida del viaje.
Cuando un viaje se lleva a cabo, el costo total del mismo se divide en partes iguales entre todos los participantes, tanto pasajeros como conductor, descontándose de sus respectivos saldos de cuenta. Además, se aplican descuentos:

- Para el conductor, se descuenta un monto equivalente al 0,1% del valor de su vehículo.
- Para cada pasajero que ya haya realizado al menos un viaje previamente, se descuenta un monto fijo de $5000.

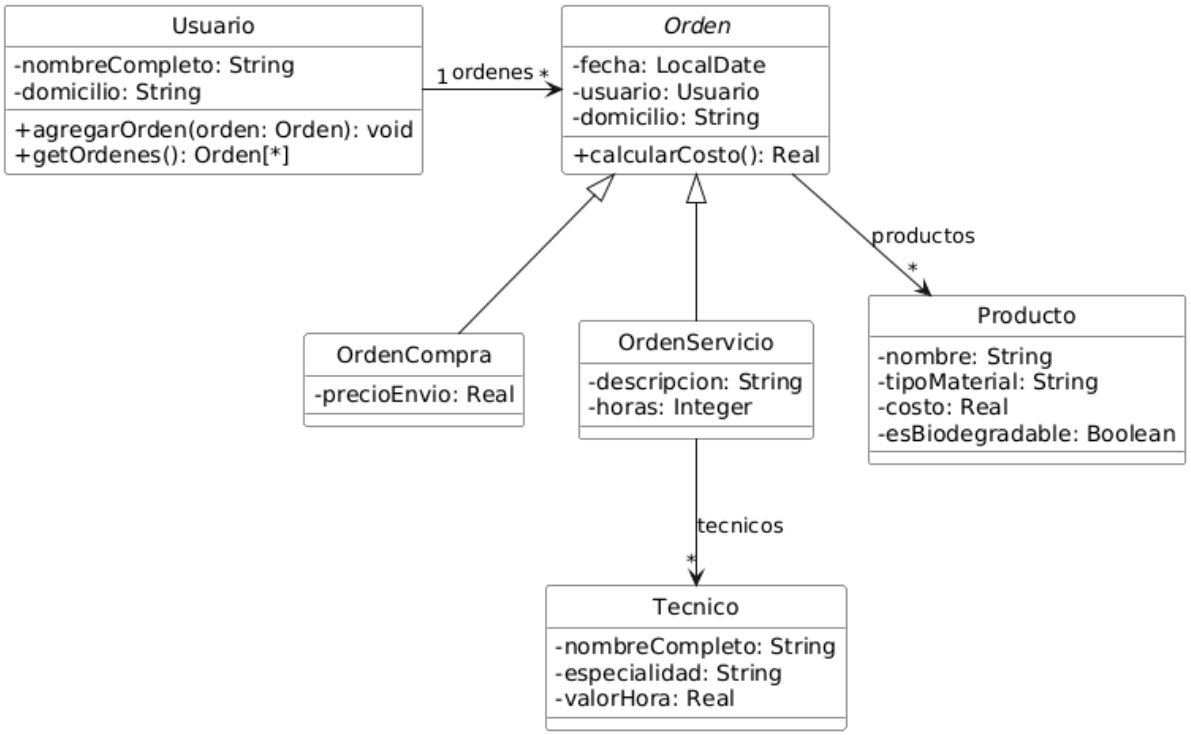
El sistema debe permitir:

- **Cargar saldo para la cuenta del usuario:** dado un usuario y un monto, se carga ese monto en el saldo de cuenta del usuario.
- **Crear un viaje desde un origen a un destino:** dado un chofer y un vehículo de su propiedad, crear un viaje a un destino, indicando la fecha de salida.
- **Registrar un pasajero para un viaje:** dado un viaje, y un pasajero, se registra el pasajero en el viaje indicado
- **Cobrar a cada participante el costo de un viaje:** Dado un viaje, descontar del saldo de cada participante el costo que le corresponde por el mismo. El saldo de la cuenta puede quedar en negativo.
- **Listar los destinos visitados por un usuario:** Dado un usuario, se quiere obtener la lista de destinos visitados.

Tarea: Realice un diagrama de clases UML donde identifique todas las clases del dominio, sus atributos, roles, cardinalidades y asociaciones. No debe incluir comportamiento.

Ejercicio 24. GreenHOOme

GreenHome es una empresa que promueve prácticas sustentables mediante la venta de productos ecológicos y la realización de servicios de mantenimiento en los hogares.
El sistema gestiona a los usuarios del servicio, registrando para cada uno las órdenes de compra de productos y las órdenes de servicio que ha solicitado para el mantenimiento de su hogar. Las órdenes de servicio pueden requerir la intervención de técnicos especializados y el uso de determinados productos, mientras que las órdenes de compra solo registran los productos adquiridos.
El costo de una orden de compra se calcula sumando los valores de los productos adquiridos, mientras que el costo de una orden de servicio se obtiene sumando el costo de los productos utilizados y el valor correspondiente a las horas de trabajo de los técnicos involucrados.
El siguiente diagrama muestra el diseño general de la aplicación.



Uno de los usuarios es **Juan Martínez**, que vive en **Larrea 5800, Mar del Plata**, tiene registradas dos órdenes en el sistema de GreenHome: una orden de compra y una orden de servicio.

La orden de compra corresponde a la adquisición de productos ecológicos. En este caso, Juan compró un panel solar y una compostera. El panel solar está hecho con materiales reciclables y tiene un costo de $35.000 pesos, mientras que la compostera es un producto biodegradable que cuesta $8.000 pesos. Esta orden se registró con la fecha actual, tiene asociado el domicilio de Juan y no interviene ningún técnico, ya que se trata de una compra directa. El costo total de la orden se obtiene sumando el valor de ambos productos, lo que da un total de $43.000 pesos.

La orden de servicio, en cambio, corresponde a la instalación de un calefón solar en el domicilio del usuario. En esta orden se detalla la descripción del trabajo (“instalación de calefón solar”) y la cantidad de horas estimadas para realizarlo, que son cinco horas. Para llevar a cabo esta tarea participa una técnica llamada Lucía Iraola, cuya especialidad son las instalaciones solares y cuyo valor por hora de trabajo es de $4500 pesos.
Además, para completar la instalación se utilizó un producto: el calefón solar, fabricado con materiales reciclables y con un costo de $50.000 pesos.
El costo total de la orden de servicio se calcula sumando el valor de los productos utilizados más el valor del trabajo de la técnica (cinco horas a $4.500 cada una), lo que da un total de $72.500 pesos.

Tareas:

1. Instancie los elementos correspondientes al usuario **Juan Martínez**, incluyendo sus datos personales, sus órdenes (una de compra y una de servicio), los productos y

los técnicos asociados a dichas órdenes, de manera consistente con el diseño presentado en el diagrama. Puede hacerlo en un script de código o un test simple donde se verifique los valores totales de las órdenes. Debe definir los constructores necesarios para resolver la tarea.

2. Para incentivar compras y servicios más extensos, si una orden de compra incluye cinco o más productos, o si una orden de servicio requiere más de diez horas de trabajo, se aplica automáticamente un **10% de descuento** sobre el costo total. Diseñe los casos de prueba para el método calcularCosto de una orden, teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.

Ejercicio 25. Bag

Primera parte

Un Map en Java es una colección que asocia objetos que actúan cómo claves (*keys*), a otros objetos que actúan cómo valores (*values*). En otros lenguajes también se llaman Dictionary (Diccionario). Cada clave está vinculada a un único valor; no pueden existir claves duplicadas en un mapa, aunque sí podrían haber valores repetidos. A los pares clave-valor se los denomina entradas (*entries*).
Para definir un Map, es necesario indicar el tipo que tendrán sus claves y valores: por ejemplo, una variable de tipo `Map<String, Integer>` define un mapa en donde sus claves son de tipo *String*, y sus valores de tipo *Integer.* Observe que `Map<K, V>` es una interfaz.

Tareas:

a. Lea la documentación de Map en Java y responda:
   a. ¿Qué implementaciones provee Java para utilizar un Map? ¿Cuáles de ellas son destinadas a uso general?
   b. Investigue cómo consultar sí un mapa contiene una determinada clave (key). Explique qué métodos deben implementar las claves para que esto funcione correctamente
   c. ¿Con qué método se puede recuperar el objeto asociado a una clave? ¿Qué pasa sí la clave no existe en el mapa?
   d. Investigue cómo agregar claves y valores a un mapa. ¿Qué pasa sí la clave ya se encontraba en el mapa? ¿Permite agregar claves y/o valores nulos?
   e. Determine cómo se pueden eliminar claves y valores de un mapa. ¿Es necesario controlar la presencia de alguno de ellos?
   f. Investigue cómo reemplazar un valor en un mapa
   g. Teniendo en cuenta los métodos `keySet(), values() y entrySet(),` explique de qué formas se puede iterar un mapa ¿Es posible utilizar streams?

b. Para practicar los mensajes investigados anteriormente, escriba un test de unidad que contenga lo siguiente:
   a. Cree un map un `Map<String, Integer>`, y agregue las tuplas (“Lionel Messi”, 111), (“Gabriel Batistuta”, 56), (“Kun Agüero”, 42)
   b. Elimine la entrada con clave “Kun Agüero”
   c. Messi hizo 112 goles a día de la fecha; actualice la cantidad de goles

    d. Intente repetir la clave “Gabriel Batistuta” y verifique que no es posible.
    e. Obtenga la cantidad total de goles
c. Como se mencionó, cualquier objeto puede actuar como clave. Es decir, pueden ser instancias de clases definidas por el programador. Modele e implemente la clase Jugador con apellido y nombre. Escriba otro test de unidad similar al de la tarea 2, pero utilizando `Map<Jugador, Integer>`

Segunda parte

Un **Bag** (bolsa) es una colección que permite almacenar elementos sin ningún orden específico y admite elementos repetidos. Este objeto requiere un buen tiempo de respuesta para conocer la cardinalidad de sus elementos, y por esa razón almacena la **cardinalidad** de cada elemento (cantidad de veces que fue agregado en la bolsa). Por ejemplo, sí agregamos 3 veces un objeto en la bolsa, y luego eliminamos 1 referencia, la cardinalidad de ese objeto en la bolsa es 2.

El protocolo de la interface Bag<T> es:

```
public interface Bag<T> extends Collection<T> {
    /**
     * Agrega un elemento al Bag, incrementando en 1 su cardinalidad.
     */
    @Override
    boolean add(T element);

    /**
     * Devuelve la cardinalidad del elemento. Sí el elemento no está en el Bag,
     * devuelve 0.
     */
    int occurrencesOf(T element);

    /**
     * Elimina una referencia del elemento del Bag. Sí el elemento no está en
     * el Bag, no hace nada.
     */
    void removeOccurrence(T element);

    /**
     * Elimina el elemento del Bag. Sí el elemento no está en el Bag, no hace
     * nada
     */
    void removeAll(T element);

    /**
     * Devuelve el número total de elementos en el Bag, es decir, la suma de
     * todas las cardinalidades de todos sus elementos.
     */
    @Override
```

```
    int size();
}
```

Observe que la interfaz Bag<T> extiende Collection<T>.

Tareas:

1. Liste los métodos que debe contener una clase que implementa la interface Bag<T>.
2. Explique cómo implementaría un **Bag<T>** usando composición con un **Map<K, V>**. ¿De qué tipo tendrían que ser las claves y valores del Map?
3. Implemente la interfaz Bag<T>, utilizando **AbstractCollection<T>** como superclase, y componga con un **Map<T, V>**. Para simplificar la implementación utilice la clase BagImpl que se encuentra en este link.

4. Discuta con un ayudante:
    a. ¿Cuáles son los beneficios de utilizar **AbstractCollection** como superclase para implementar el Bag?
    b. ¿Qué ventajas tiene componer con un Map para implementar el Bag?
    c. En lugar de componer con un Map, ¿es posible extenderlo para poder implementar el Bag? ¿Qué diferencias tendría esa solución con respecto a la planteada en este ejercicio?
    d. ¿Para qué cree que podría ser útil un objeto Bag?

Ejercicio 26. Estadísticas del Cliente de Correo

Extienda el Ejercicio 15: Cliente de Correo:

Nos piden implementar la siguiente funcionalidad:

- cantidad de emails que tiene una carpeta
- cantidad total de emails en el cliente de correo: considerando todas las carpetas existentes.
- cantidad de mails por categoría: para cada carpeta se debe calcular y retornar en un solo objeto, la cantidad de emails categorizados por tamaño siguiendo el siguiente criterio
    - Pequeño: el email tiene un tamaño entre 0 y 300
    - Mediano: el email tiene un tamaño entre 301 y 500
    - Grande: el email tiene un tamaño mayor a 501

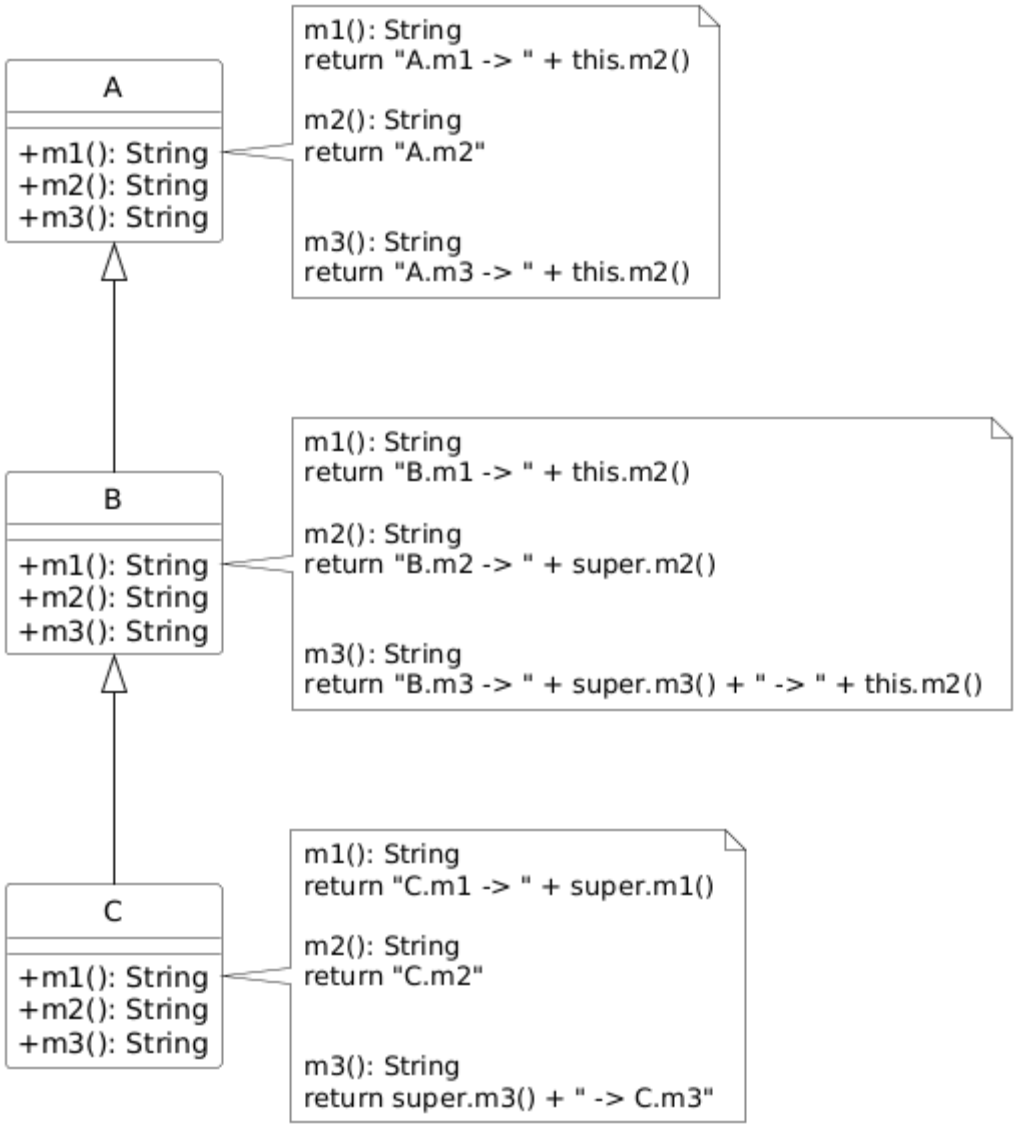
Tareas:

a) Modele e implemente
   i) Modifique el diagrama de clases UML según lo necesario
   ii) Implemente en Java los cambios solicitados

b) Pruebas automatizadas
   i) Diseñe los casos de prueba teniendo en cuenta los conceptos de valores de borde y particiones equivalentes vistos en la teoría.
   ii) Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.

Ejercicio 27. Method Lookup

Dado el siguiente modelo, y los fragmentos de códigos mostrados, marque la respuesta para cada una de las preguntas enunciadas. Sólo una opción es la correcta.



Observando el diagrama, indique qué texto retorna el siguiente fragmento de código:

```
C c = new C();
c.m1();
```

1. C.m1 -> B.m1 -> C.m2
2. C.m1 -> A.m1 -> A.m2
3. C.m1 -> B.m1 -> B.m2 -> A.m2
4. C.m1 -> A.m1 -> C.m2

Observando el diagrama, indique qué texto retorna el siguiente fragmento de código:

```
C c = new C();
c.m2();
```

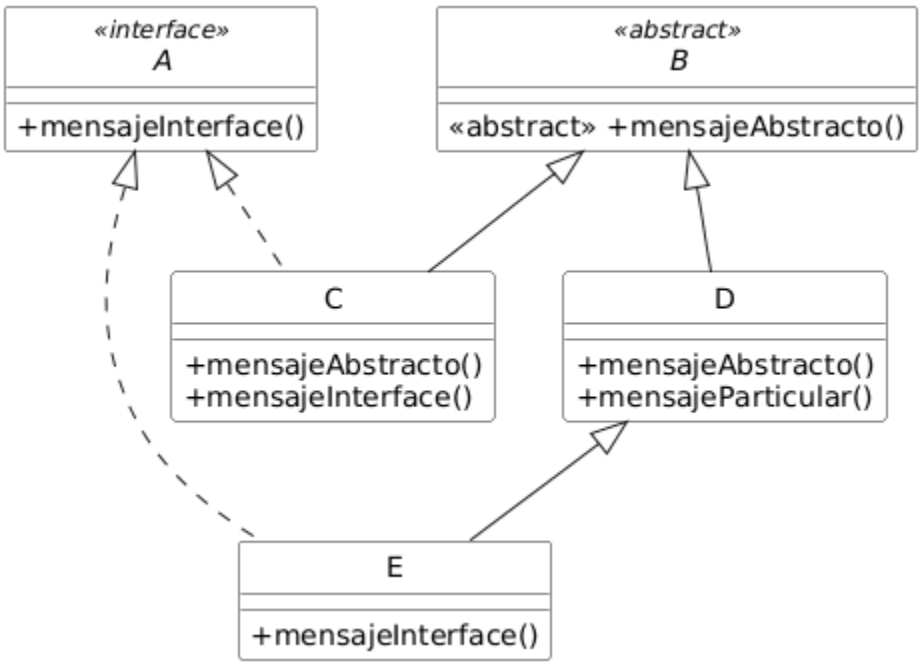
1. A.m2
2. B.m2 -> A.m2
3. C.m2
4. B.m2 -> C.m2

Observando el diagrama, indique qué texto retorna el siguiente fragmento de código:

```
C c = new C();
c.m3();
```

1. B.m3 -> A.m3 -> C.m2 -> C.m2
2. B.m3 -> A.m3 -> C.m2 -> C.m2 -> C.m3
3. B.m3 -> B.m3 -> A.m2 -> A.m2 -> C.m3
4. B.m3 -> A.m3 -> C.m2 -> C.m3

Ejercicio 28. TipadOOs



Tarea: Dado el diagrama de clases UML proporcionado, complete todos los bloques de código reemplazando los signos de interrogación (???) con los tipos y métodos correctos. Escriba todas las combinaciones válidas posibles.

1)
`A objeto = new ???();`
`objeto.???();`
	4)
	`C objeto = new C();`
	`objeto.???();`

2)
`B objeto = new ???();`
`objeto.???();`
	5)
	`??? objeto = new C();`
	`objeto.mensajeAbstracto();`

3)
`D objeto =  new ???();`
`objeto.???();`
	6)
	`??? objeto = new C();`
	`objeto.mensajeInterface();`

Ejercicio 29. Plataforma de Streaming

Una plataforma de streaming registra las actividades realizadas por los clientes para calcular los montos a cobrar a cada uno. De cada cliente se conoce el nombre y su fecha de alta. Los clientes pueden contratar un plan que puede ser individual o grupal. Al momento de contratar el plan, se establece una cantidad límite en las direcciones IP desde las cuales se permite acceder a las actividades, sin generar un cargo adicional. En el caso de los planes individuales se permite acceder desde una única dirección IP; mientras que en los planes grupales se permite acceder desde varias direcciones IP, con una cantidad máxima establecida al contratar el plan. En los planes individuales se registra además la cantidad de minutos contratados. Los clientes pueden cambiar el plan contratado (y no es necesario mantener el registro de planes anteriores). Un cliente tendrá siempre un solo plan.

Hay dos tipos de actividades: reproducciones de video y sesiones de juegos.

- En las reproducciones de video se conoce la fecha de inicio de la actividad y la dirección IP desde la cual se accedió (almacenado como un String). Además, una reproducción de vídeo conoce la duración total y la duración en publicidad (ambas expresadas en minutos). Se puede obtener la duración real utilizada calculada como la duración total menos la duración en publicidad.
- En el caso de las sesiones de juegos se conoce la fecha de inicio de la actividad, la duración (expresada en minutos) y la dirección IP desde la cual se accedió. Además, el usuario tiene la opción de utilizar diversos ítems que mejoran la experiencia o brindan ventajas dentro del juego. Cada uno de estos ítems tiene nombre, cantidad y precio unitario. El usuario deberá pagar, si corresponde, el precio final de cada ítem como cargo adicional (calculado como la cantidad multiplicada por precio unitario).

La empresa cobra un precio base según el plan contratado, al que le suma el monto correspondiente a las actividades registradas y un monto extra por penalización (si el cliente excede la cantidad de IPs según su plan).

Nos piden implementar la siguiente funcionalidad:

Calcular el precio base de un plan: Para los planes individuales, el precio base es $20 por la cantidad de minutos contratados. Para los planes grupales, el precio base es de $800 por la cantidad límite de IPs contratadas.

Monto total a cobrar: dado un cliente y un intervalo de tiempo (fecha inicial - fecha final), retorna el monto total a cobrar para ese intervalo. El monto total a cobrar es la suma del precio base según el plan, el monto por las actividades registradas y el monto por penalización.

Monto a cobrar por las actividades registradas: dado un cliente y un intervalo de tiempo (fecha inicial - fecha final), retorna el monto a cobrar en ese intervalo, según cada caso:

- **Video:** Se cobra $10 por cada minuto de duración real de la reproducción del video.
- **Juegos:** Sí la duración de la actividad supera 360 minutos, se cobra el precio final de todos los ítems utilizados durante la sesión. En caso contrario, no se cobra nada.

Monto por penalización por superar el límite contratado: dado un cliente y un intervalo de tiempo (fecha inicial - fecha final), retorna el monto por penalización en ese intervalo. El monto de penalización varía según el plan:

- Si el cliente tiene un plan individual será penalizado si utilizó más de una dirección IP en las actividades registradas durante ese intervalo. El costo por cada IP adicional que haya utilizado es de $300.
- Si el cliente tiene un plan grupal será penalizado si el número de direcciones IP utilizadas en el intervalo supera la cantidad máxima establecida al contratar el plan. El costo por cada IP adicional es de $500. En caso de que la antigüedad del cliente sea mayor a 10 años, no se cobra esta penalización.

Ayuda: Para conocer los años entre dos fechas: `Period.between(fechaInicio, fechaFin).getYears()`

Tareas:

1) **Modelado de dominio**
   i) Realice un diagrama de clases UML donde identifique las clases del dominio, atributos, y las relaciones con sus roles y cardinalidades cuando corresponda. No debe incluir comportamiento.
2) **Modelado e Implementación**
   i) Continúe completando el diagrama anterior con el comportamiento de cada clase.
   ii) Realice el Diagrama de clases UML a partir del modelo de dominio realizado en el punto anterior.
   iii) Implemente en Java la funcionalidad requerida. Puede usar las clases implementadas en los ejercicios de la práctica: DateLapse, FilteredSet, Bag, según se requiera.
3) **Pruebas automatizadas**
   i) Diseño de los casos de prueba: Enfocándose en la funcionalidad que permite calcular el monto por penalización por superar el límite contratado (todos los métodos de todas las clases involucradas en conseguir esa funcionalidad), determine y enumere **qué métodos** testear, **indicando clases** y casos de prueba (teniendo en cuenta los conceptos de valores de borde y particiones

equivalentes)**.** Identifique claramente las particiones que encuentra y los valores de borde para cada caso.

ii) Implemente utilizando JUnit los tests automatizados diseñados en el punto anterior.

4) Instanciación

i) En un script de código o un test simple, escriba **el código Java** necesario para crear un cliente con un plan grupal con una cantidad máxima de 6 direcciones IP, que accedió a dos actividades: una reproducción de video, y una sesión de juegos, donde se utilizó un ítem de nombre “daño aumentado”, con 2 unidades y precio unitario de 1000. Luego el cliente cambia a un plan individual. Complete con los valores que necesite según corresponda.