Cuadernillo Semestral de Actividades

Actualizado: 26 de agosto de 2024

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 cuales 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. Para importar el proyecto, siga los pasos explicados en el documento “*Trabajando con proyectos Maven*, *importar un proyecto”*. Allí verá que existe la interface Wallpost y la clase WallpostImpl que implementa la interfaz anterior. Una vez importado, dentro del mismo, debe completar la clase WallPostImpl para que entienda:

```
/*
* Permite construir una instancia del WallpostImpl.
* 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 WallPostImpl()
```

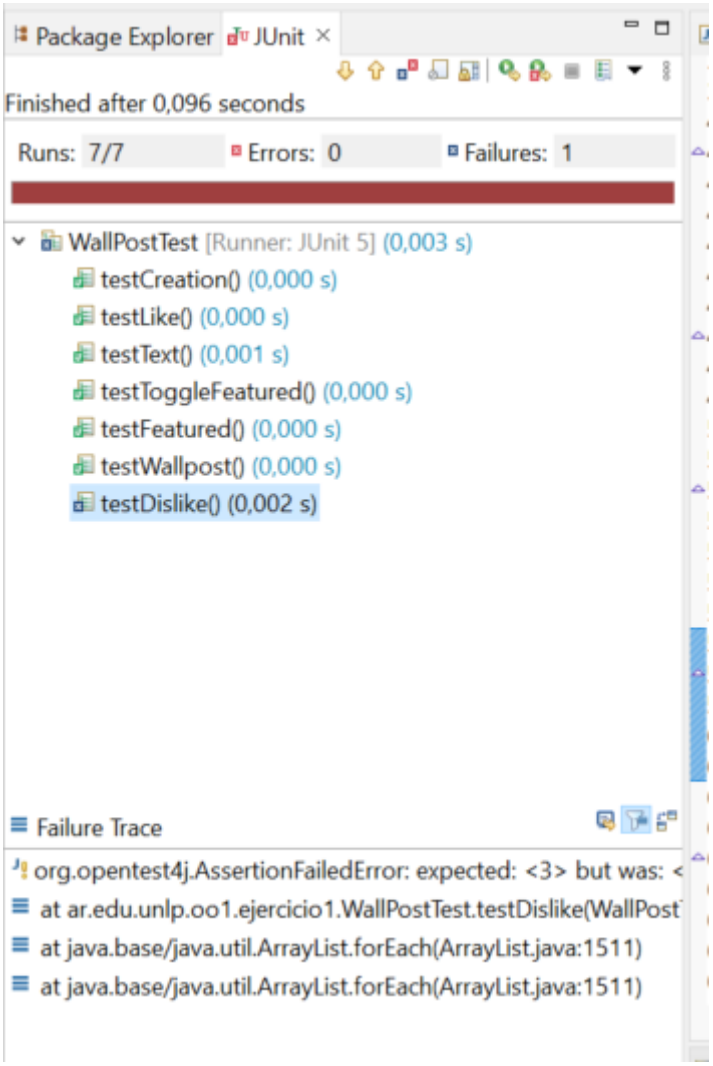
E implemente el protocolo definido en la interfaz Wallpost como se detalla a continuación

```
/*
* 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.

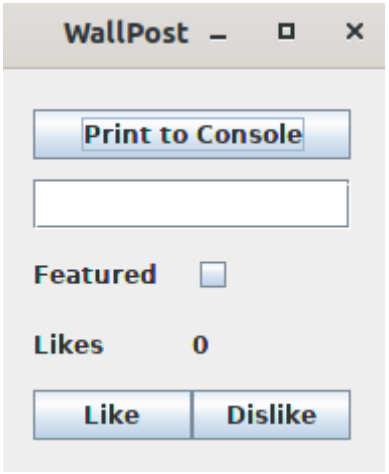
Para ejecutar los tests simplemente haga click derecho sobre el proyecto y utilice la opción Run As >> JUnit Test. Al ejecutarlo, se abrirá una ventana con el resultado de la evaluación de los tests. Siéntase libre de investigar la implementación de la clase de test. Ya veremos en detalle cómo implementarlas.



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

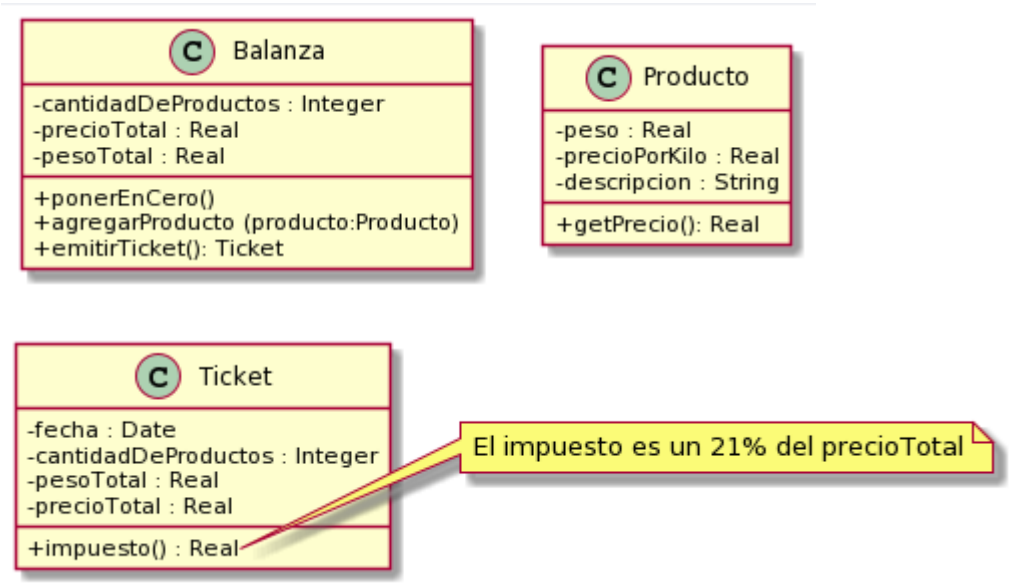
En el taller de programación ud programó una balanza electrónica. Volveremos a programarla, con algún requerimiento adicional.

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.

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/javase/8/docs/api/java/time/LocalDate.html

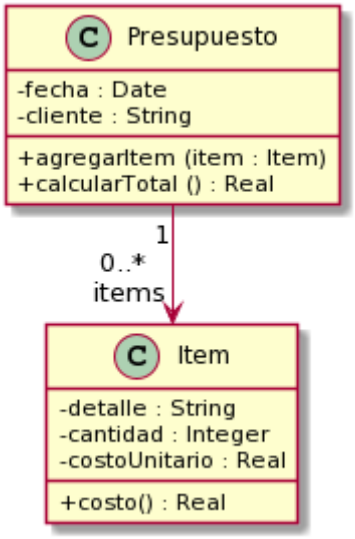
Probando su implementación:

Para realizar este ejercicio, utilice el recurso que se encuentra en el sitio de la cátedra. 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

Defina el proyecto Ejercicio 3 - Presupuesto y dentro de él implemente las clases que se observan en el siguiente diagrama. Ambas son subclases de Object. Preste atención a los siguientes aspectos:



- ¿Cuáles son las variables de instancia de cada clase?
- ¿Qué variables inicializa y cómo?

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 3 - Bis: 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 anteriormente.

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 el 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?

Tarea 3

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

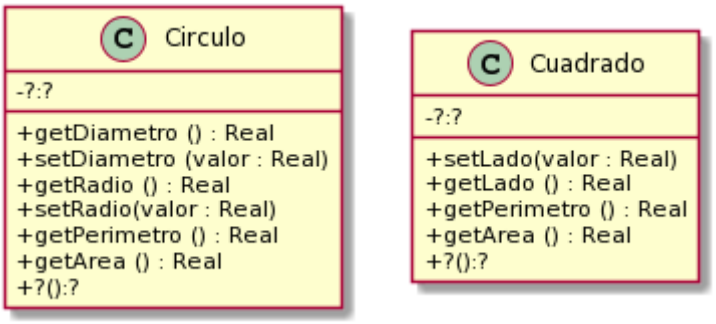
Ejercicio 4: Figuras y cuerpos

Figuras en 2D

Defina un nuevo proyecto figurasYCuerpos

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. Decida usted qué variables de instancia son necesarias. Ambas clases son subclases de Object. Puede agregar mensajes adicionales si lo cree necesario.

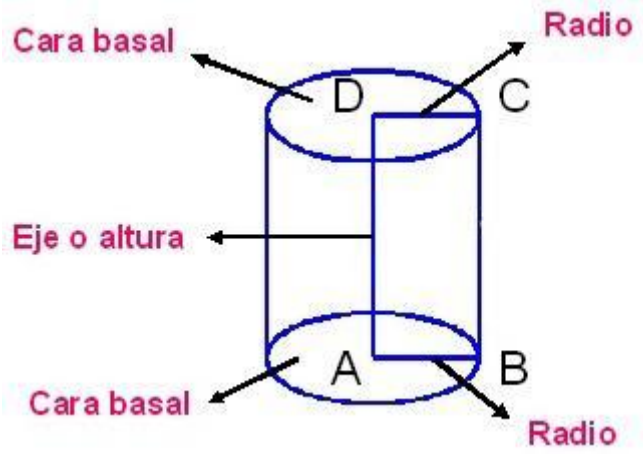


Fórmulas y mensajes útiles:

- 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)

Cuerpos en 3D

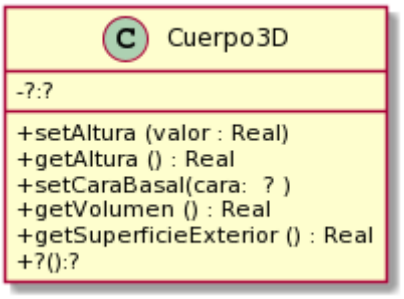
Ahora que tenemos Círculos y Cuadrados, podemos usarlos para construir cuerpos (en 3D) y calcular su volumen y superficie o área exterior. Vamos a pensar a un cilindro como "un cuerpo que tiene una figura 2D como cara basal y que tiene una altura (vea la siguiente imagen)" . Si en el lugar de la figura2D 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. Decida usted qué variables de instancia son necesarias. Cuerpo3D es subclase de Object.

Decida usted si es necesario hacer cambios en las figuras 2D.



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.

Pruebas automatizadas

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

Discuta y reflexione

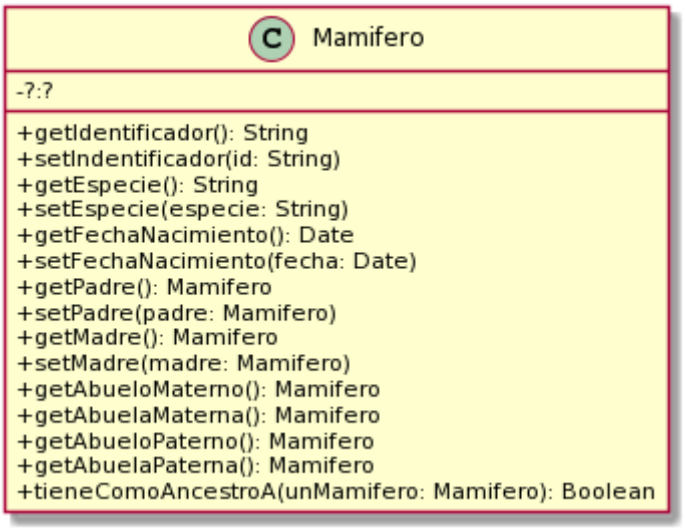
Discuta con el ayudante sus elecciones de variables de instancia y métodos adicionales. ¿Es necesario todo lo que definió?

Ejercicio 5: 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:

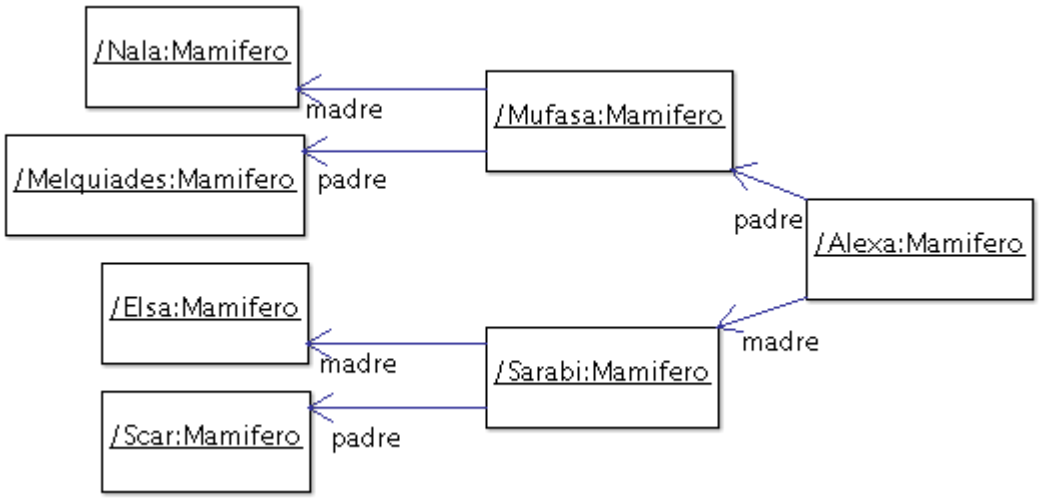
a) Modelar en objetos y programar la clase Mamífero (como subclase de Object). El siguiente diagrama de clases (incompleto) nos da una idea de los mensajes que un mamífero entiende.
Deje tieneComoAncestroA para el final y discuta su solución con el ayudante.

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



b) Complete el diagrama de clases para reflejar los atributos y relaciones requeridos.

c) 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.