Facultad de Ciencias de la Administración
UNER

TRABAJO PRÁCTICO N° 3 – ASOCIACIÓN Y DEPENDENCIA

PROGRAMACIÓN II - 2025 – 2do cuatrimestre

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

Objetivos

- Aplicar y reforzar los conceptos referidos a asociación y dependencia de clases y objetos, como así también los mecanismos de encapsulamiento y abstracción.

Condiciones de Entrega

- Deberá realizarse una entrega grupal, con grupos de no más de 3 (tres) alumnos.
- Los ejercicios a entregar son los correspondientes a la sección "b". La entrega deberá consistir de una carpeta comprimida que contenga un archivo con extensión .py por cada clase creada (producto.py, empleado.py, y empresa.py) y un cuarto archivo main.py donde se realice el resto de los ejercicios que no implican crear o alterar la definición de una clase. Se deberá indicar con claridad la sección del código que se corresponde con cada uno de los ejercicios del documento. La carpeta debe ser subida en la sección correspondiente del Campus.
- Se deberá respetar la fecha de entrega, la misma será informada en el campus.
- El Trabajo Práctico será calificado como **Aprobado** o **Desaprobado**.
- Las soluciones del grupo deberán ser de autoría propia. De encontrarse soluciones idénticas entre diferentes grupos, dichos trabajos prácticos

serán calificados como **Desaprobado**, lo cual será comunicado en la devolución.

- Las entregas individuales serán calificadas como **Desaprobado**.

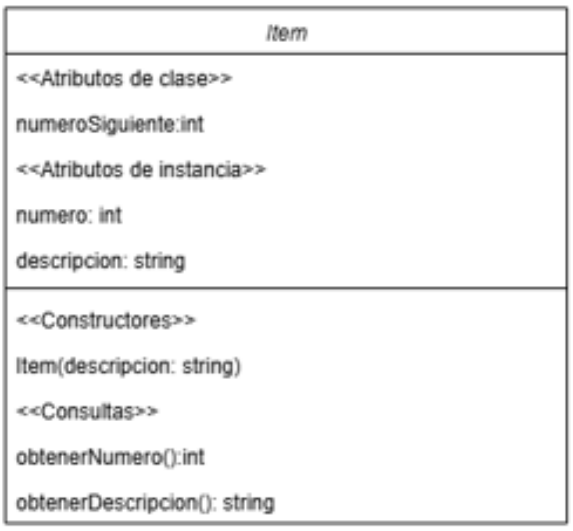
EJERCICIOS:

***Nota**: asumir que todos los parámetros recibidos en los métodos correspondientes a los ejercicios del trabajo son válidos, con excepción de los casos donde se especifique la necesidad de agregar validaciones específicas.*

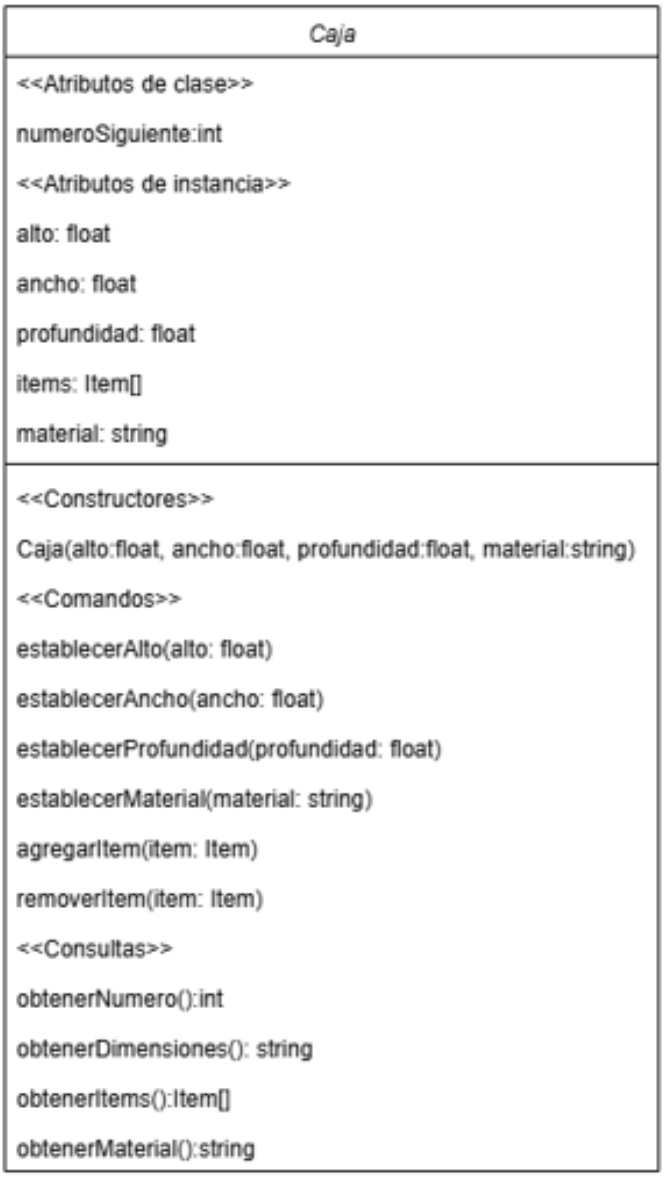
Sección A (para ver en clase):

Dado el objetivo de modelar los depósitos de una organización, se van a generar una clase "Deposito", una clase "Estanteria", una clase "Caja", y una clase "Item".

1) En el archivo item.py, generar la clase correspondiente al siguiente diagrama:

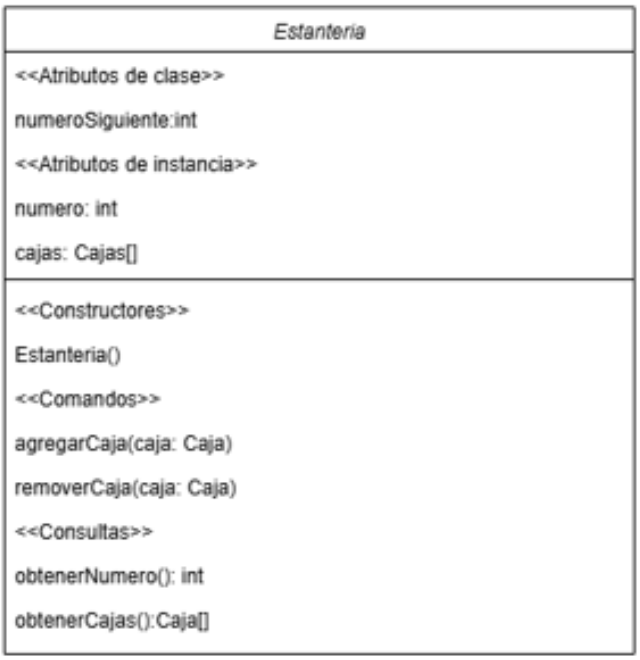


2) En el archivo caja.py, generar la clase correspondiente al siguiente diagrama:

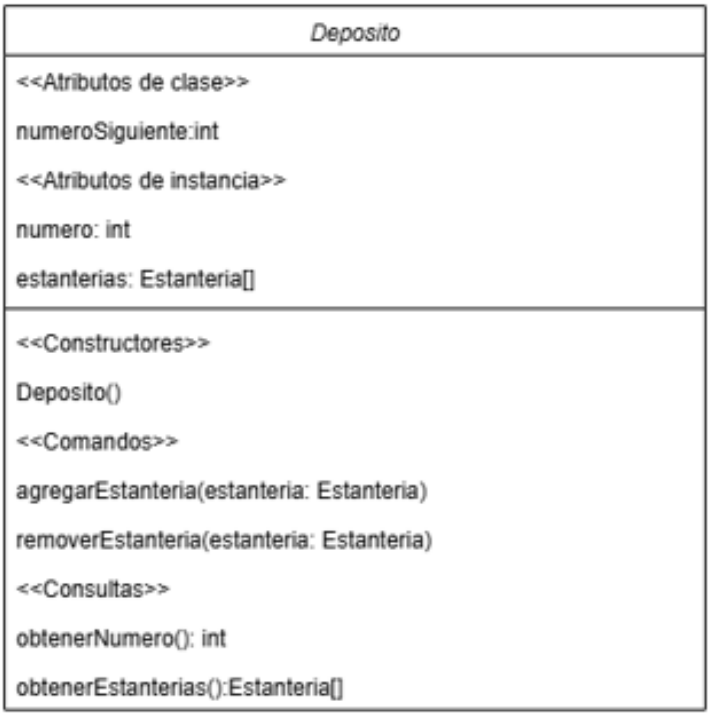


Nota: las dimensiones de la caja están medidas en centímetros

3) En el archivo estanteria.py, generar la clase correspondiente al siguiente diagrama:



4) En el archivo deposito.py, generar la clase correspondiente al siguiente diagrama:



5) Asegurar que todos los métodos constructores de las clases creadas asignen como valor al atributo de instancia "numero" el valor actual del

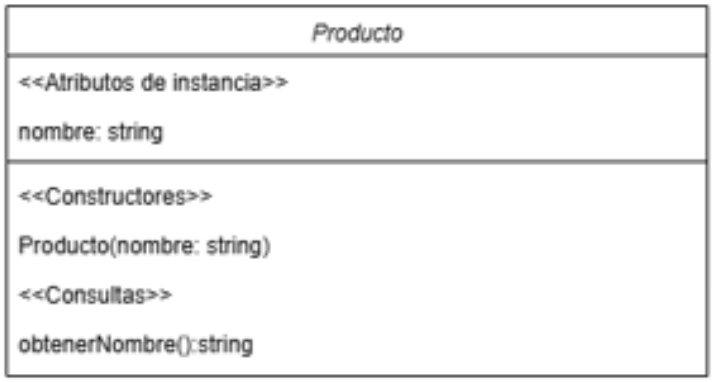
atributo de clase "numeroSiguiente", e incremente el valor de este último en una unidad.

6) Implementar en cada una de las clases creadas los métodos "__str__" y "__eq__ " (comparando por "numero").
7) Asegurar que todos los atributos de instancia de las clases creadas son privados (con Name Mangling).
8) En el archivo main.py, crear un depósito. El depósito debe tener 2 estanterías, cada estantería debe tener 3 cajas, y cada caja debe tener 2 ítems.
9) En el archivo main.py, imprimir el contenido del depósito, incluyendo el detalle de sus estanterías, cajas, e items.
10) En el archivo main.py, utilizar los métodos disponibles para pasar una caja de una estantería a la otra, e imprimir nuevamente el contenido del depósito, siguiendo las condiciones establecidas en el punto anterior.

Sección B (para entregar):

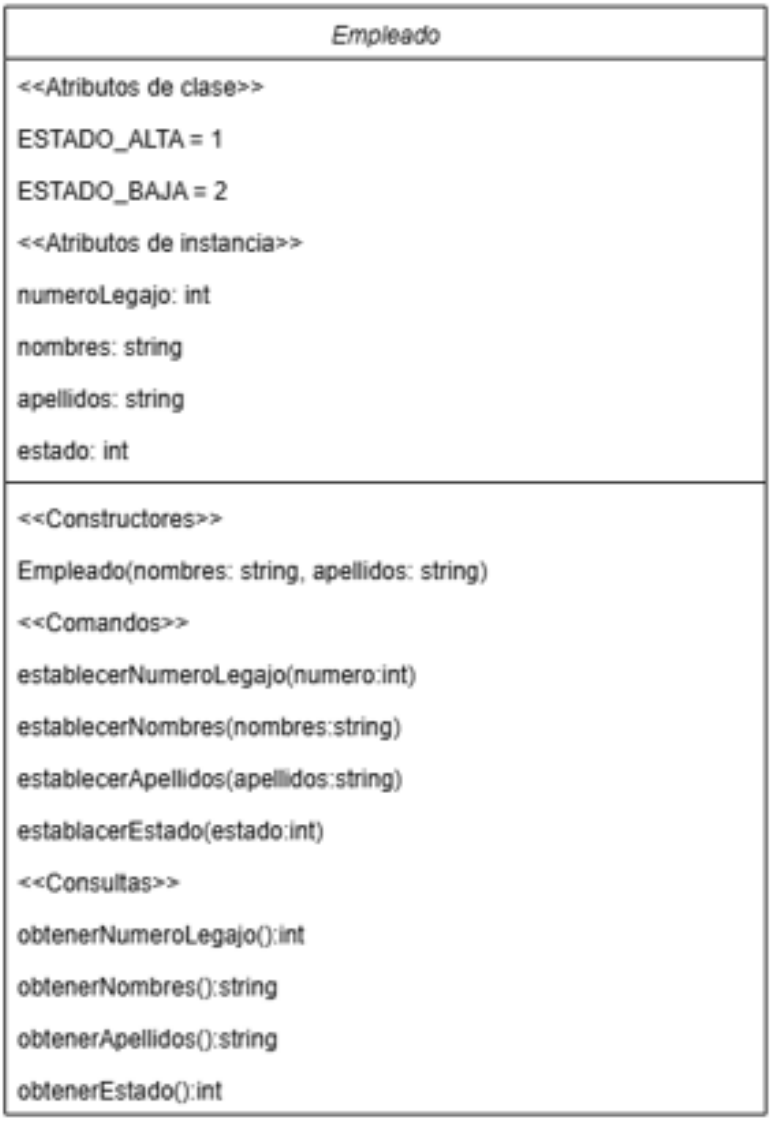
Con el objetivo de modelar empresas, se van a generar una clase "Empresa", una clase "Producto", y una clase "Empleado".

1) En el archivo producto.py, generar la clase correspondiente al siguiente diagrama:



Nota: Cada producto dentro de una instancia de "Empresa" se puede identificar de forma única por su nombre

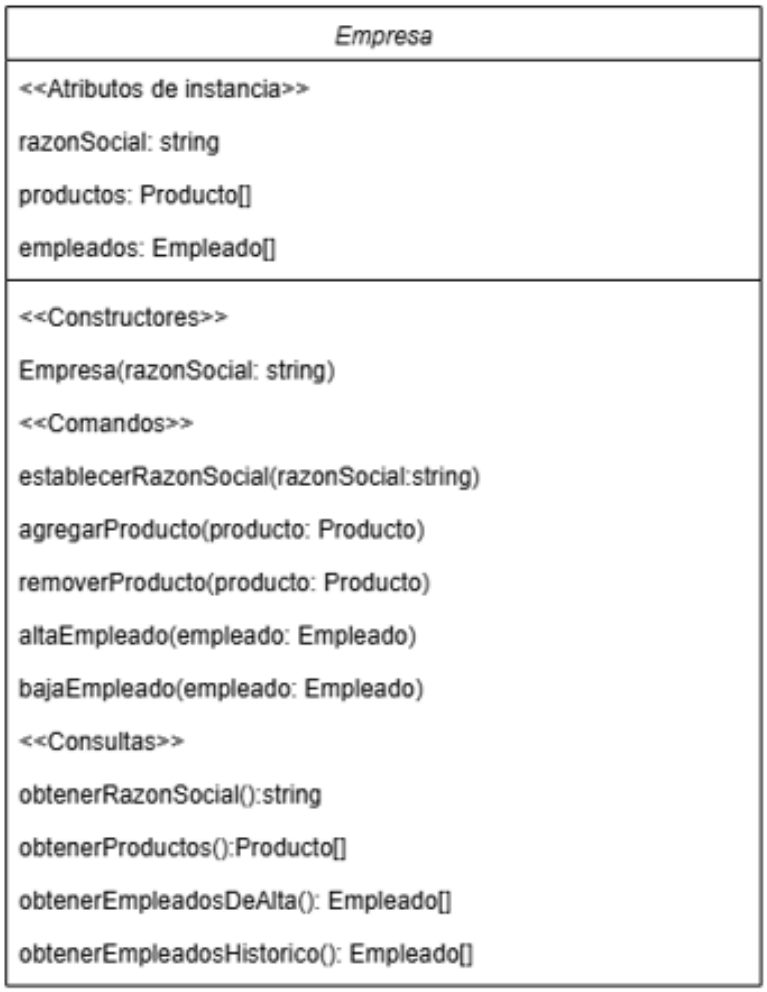
2) En el archivo empleado.py, generar la clase correspondiente al siguiente diagrama:



Nota: Cada empleado dentro de una instancia de "Empresa" se puede identificar de forma única por su número de legajo

3) Asegurar que el constructor asigne el valor del atributo de clase "ESTADO_ALTA" al atributo de instancia "estado" al crear un empleado.

4) En el archivo empresa.py, generar la clase correspondiente al siguiente diagrama:



Nota: Cada instancia de la clase"Empresa" se identifica de forma única por su razón social.

5) Asegurar que el método "altaEmpleado" asigne como número de legajo el número siguiente al número de legajo máximo de todos los empleados registrados de la empresa (sin importar su estado actual) antes de agregar al empleado a la lista de empleados. Asumir que este método no va a ser llamado varias veces para el mismo empleado.
6) Asegurar que el método "bajaEmpleado" actualice el estado del empleado al valor del atributo de clase "ESTADO_BAJA". El empleado no debe ser removido de la lista de empleados. Asumir que un empleado en estado de baja no va a cambiar de estado en el futuro.

7) Asegurar que el método "obtenerEmpleadosDeAlta" retorne únicamente los empleados que estén en estado de alta, y que el método "obtenerEmpleadoHistorico" retorne todos los empleados sin importar su estado actual.
8) Implementar en cada una de las 3 clases (Producto, Empleado, y Empresa) los métodos __str__ y __eq__ (comparando por nombre, número de legajo, y razón social, respectivamente). El método __str__ de la clase empresa debe imprimir información detallada de los productos y de los empleados en estado de alta (recuperados utilizando el método creado para ese fin), haciendo uso del método str() sobre las instancias de las clases "Empleado" y "Producto". Asumir que el método __eq__ de la clase “Empleado” nunca se va a utilizar sobre 2 instancias de empleado de empresas distintas, y que el método __eq__ de la clase “Producto” nunca se va a utilizar sobre 2 instancias de producto de empresas distintas.
9) Asegurar que todos los atributos de instancia en cada una de las 3 clases (Producto, Empleado, y Empresa) son privados (con Name Mangling).
10) En el archivo main.py, crear 2 empresas, cada una con 3 empleados y 2 productos. A continuación, remover 2 empleados de una de ellas. Por último, imprimir la información disponible de cada empresa, haciendo uso del método str().