# Trabajo Práctico III: HERENCIA

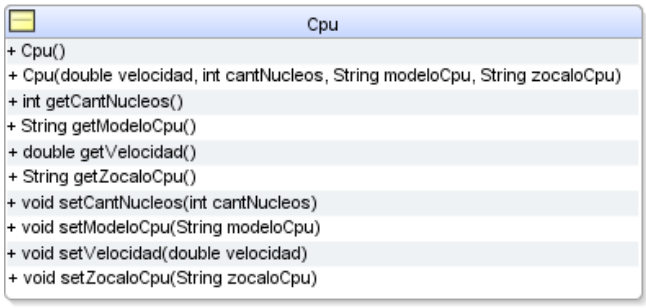
| |
|---|
| Concepto de Herencia. Herencia de Contrato. Herencia de Implementación. |
| Sobreescritura de métodos. Ocultamiento de atributos. |
| La palabra reservada “super”. |
| Constructores en clases extendidas. |
| Especificadores de acceso. |
| El operador FINAL en clases, métodos, atributos. |
| Concepto de Delegación. Herencia vs. Delegación. |
| Clases Abstractas. |
| Generalización. Especialización. Diseño de una clase extensible. |
| SOLID |
| La clase Object, el método toString(). |
| Patrón Factory |

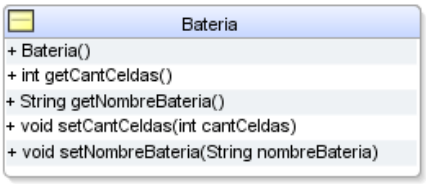
Objetivo de la práctica

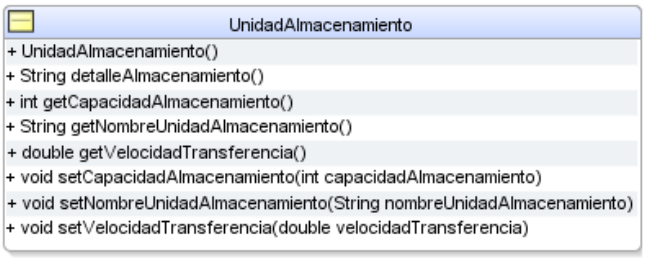
1. Diseñar un diagrama de clases utilizando herencia.
2. Diferenciar los casos en dónde aplicar herencia o composición.
3. Sobreescribir métodos.
4. Ocultar atributos
5. Utilizar especificadores de acceso con herencia.
6. Comprender la necesidad y uso de clases abstractas.
7. Utilizar la palabra reservada super en métodos y constructores.
8. Buen uso del constructor en clases extendidas.
9. Uso del especificador final.
10. Usar métodos concretos que invocan métodos abstractos.
11. Redefinir métodos abstractos que invocan métodos concretos (de la clase padre)

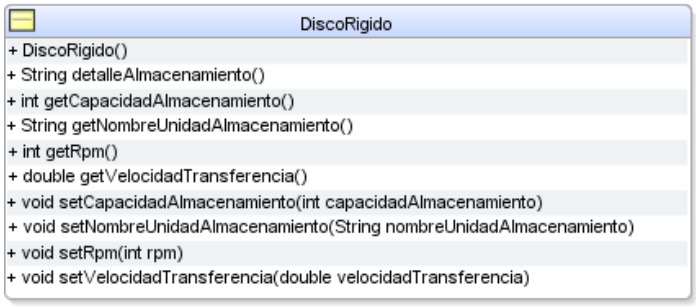
## *Ejercicio 1*

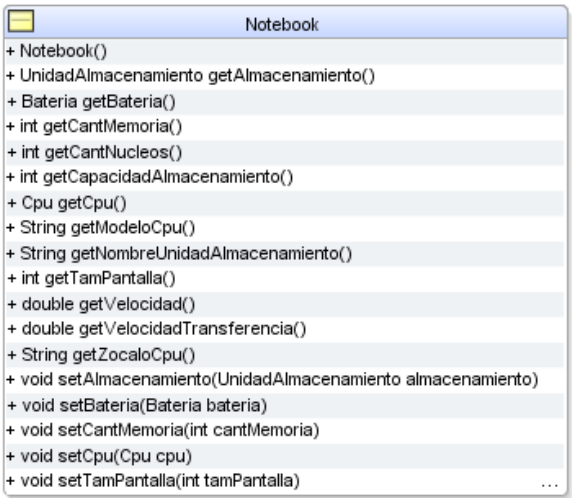
Una casa de informática desea llevar un registro de sus productos los cuales se modelarán a partir de las siguientes clases:

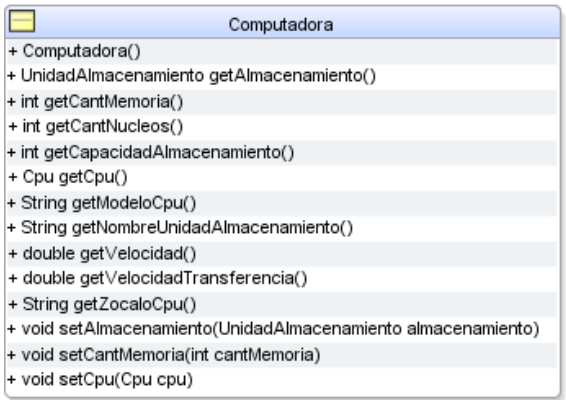


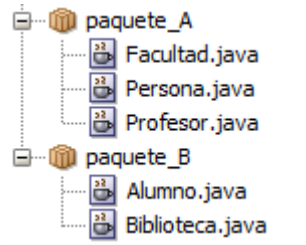




(solo se muestran los constructores y los métodos)

No hace falta escribir el código Java. Realiza un diagrama de clases prestando especial atención en las relaciones que existen entre las clases, y en qué casos conviene utilizar herencia o delegación.

## *Ejercicio 2 (modificadores de acceso)*

Considera las siguientes clases distribuidas en los paquetes **paquete_A** y **paquete_B**




La clase **Persona** tiene los siguientes modificadores de acceso en sus atributos:

```
public class Persona
{
    public String nombre;
    protected Date fechaNac;
    private String dni;
    String email;
….
 }
```

y sabiendo que las clases **Profesor** y **Alumno** se extienden de **Persona** y que las clases **Facultad** y **Biblioteca** no tienen relación de herencia con la clase **Persona**, indica que atributos de Persona son visibles desde las diferentes clases:

| | nombre | fechaNac | dni | email |
|---|---|---|---|---|
| Persona | | | | |
| Profesor | | | | |
| Facultad | | | | |
| Alumno | | | | |
| Biblioteca | | | | |

## *Ejercicio 3*

Modele las clases necesarias que representen cuentas bancarias que tienen un saldo y un titular. y que pueden realizar operaciones de extracciones y depósitos.
Para poder crear una cuenta bancaria debe conocerse el nombre del titular, el cual no podra modificarse, el saldo inicial de toda cuenta bancaria es igual a cero.
Existen tres tipos de cuentas bancarias: Cuentas corrientes, Cajas de ahorro y Cuentas Universitarias.
Si revisamos el comportamiento nos encontraremos con las siguientes características en común:

- Todas llevan cuenta de su saldo.
- Todas permiten realizar depósitos.
- Todas permiten realizar extracciones.

Pero cada una tiene un tipo de restricción distinto en cuanto a las extracciones:

- Cuentas corrientes: permiten que el cliente gire en descubierto (con un tope pactado con cada cliente). Dicho tope podrá modificarse, pero debe indicarse al momento de crear la cuenta corriente. (Ver constructores encadenados)
- Cajas de ahorro: poseen una cantidad máxima de extracciones mensuales (para todos los clientes). No se permite girar en descubierto.
- Cuenta universitarias: permite extracciones de no más de $1.000.- diarios.

Modele y codifique estas clases y los métodos correspondientes.

## *Ejercicio 4*

Una empresa solicita un sistema para administrar los datos del personal. La empresa puede tener varios departamentos, que a su vez tienen varios empleados.

Los empleados pueden ser permanentes o temporarios, dentro de los empleados permanentes, estos pueden ser principiantes, intermedios o expertos. Esta última clasificación decidirá la forma de calcular su sueldo.

De todos los empleados se desea conocer:

- nombre
- número de legajo
- domicilio

De los empleados temporarios interesa conocer, el sueldo por hora y la cantidad de horas trabajadas.

De los empleados permanentes interesa su antigüedad y su sueldo base.

Los departamentos tendrán un nombre, y una lista de empleados.

Un departamento deberá permitirnos agregar y quitar empleados, y entregar un planilla informativa de los sueldos de sus empleados.

La planilla informativa será una lista que solo contendrá el nombre del empleado y el sueldo neto.

La forma de calcular el sueldo será la siguiente:

- En el caso de los trabajadores temporarios, el sueldo neto a cobrar será el sueldo por hora multiplicado por la cantidad de horas trabajadas. Por tratarse de un trabajador temporario, los aportes jubilatorios y de obra social corren por su cuenta.
- En el caso de los trabajadores permanentes, el sueldo neto siempre será el suelo bruto menos el 11% de aporte jubilatorio y el 6% por obra social. Estos descuentos están fijados por ley, por lo tanto este cálculo no debe poder modificarse.

El bruto de los trabajadores permanentes se calculará de acuerdo al siguiente criterio:

- Si el trabajador es principiante, se pagará un plus por antigüedad de acuerdo a la siguiente tabla:

| | |
|---|---|
| De 2 a 5 años | 5% |
| De 5 a 10 años | 7% |
| De 10 a 15 años | 10% |
| De 15 a 20 años | 15% |
| Mas de 20 años | 20% |

- Si el trabajador es intermedio se incrementa su sueldo base un 25%, y se suma un 1% por cada año de antigüedad.
- Si es experto, el sueldo base se incrementa un 50%, y se suma un 1,5% por cada año de antigüedad.

**Inciso a)**
Define las clases que consideres necesarias y realiza un diagrama de las mismas.

**Inciso b)**
Implementa una clase Prueba en cuyo método main se instanciarán las clases necesarias para comprobar el funcionamiento de una empresa ficticia con los siguientes datos:

Departamento de mantenimiento:

- Juan Perez, empleado intermedio con 7 años de antigüedad, Legajo 1234, domicilio Matheu 2343, Sueldo base $12.000
- Julio García, empleado temporal, Legajo 3209, domicilio Colon 5561, Sueldo por hora, $80, horas trabajadas 160.
- Martin Rodriguez, empleado temporal, Legajo 3210, domicilio Mitre 3451, Sueldo por hora, $70, horas trabajadas 100.

Departamento de Contabilidad:

- Mara Anchorena, empleado experto con 18 años de antigüedad, legajo 9876, domicilio Luro 3489, Sueldo base $13.000
- Sandra Fernandez, empleado principiante con 6 años de antigüedad, legajo 1276, domicilio Córdoba 3843, Sueldo base $11.500
- Luis Gomez, empleado principiante con 2 años de antigüedad, legajo 1544, domicilio Formosa 2354, Sueldo base $10.500
- Lucas Benitez, empleado intermedio con 12 años de antigüedad, legajo 1634, domicilio Castelli 4563.

Realiza una consulta de planillas de sueldos de ambos departamentos.

## *Ejercicio 5*

Escribe la clase abstracta Automovil, y las clases extendidas AutomovilManual y AutomovilAutomatico.

Los atributos de los Automoviles son:
- String patente.
- double velocidad
- double velocidadmaxima
- int marcha (0 representa punto muerto, y -1 marcha atrás)

Tendrá además los siguientes métodos:

- Automovil (String patente, double velmax)
- Automovil (String patente) (utiliza el constructor anterior setea la velocidad maxima en 160 km/h)

- getPatente()
- getVelocidad()
- getMarcha()
- acelerar(double vel) (Método abstracto que incrementa la velocidad según el parámetro vel)
- frenar (double vel) (Método abstracto que decrementa la velocidad según el parámetro vel)
- toString()
- setMarcha(int nuevamarcha)

Los métodos acelerar y frenar aceptarán valores positivos, en caso de recibir valores negativos, no modificarán la velocidad.

Al acelerar no podrá superarse la velocidad máxima, ni tampoco al frenar se podrá obtener un valor negativo.

El método setMarcha será de tipo protected, y validará el parámetro de entrada ignorando cualquier valor menor a -1 o superior a 5 (en nuestro ejemplo los automoviles tendrán todos caja de quinta).

La clase AutomovilManual deberá implementar los métodos acelerar y frenar.

La clase AutomovilAutomático sobreescribirá los métodos acelerar y frenar de forma que las marchas se cambien de forma automática. De acuerdo a la siguiente tabla:

- velocidad = 0 punto muerto
- 0 < velocidad <= 10 Primera
- 10 < velocidad <= 35 Segunda
- 35 < velocidad <= 50 Tercera
- 50 < velocidad <= 90 Cuarta
- 90 < velocidad Quinta

Bajo este planteo ¿los automóviles automáticos tienen forma de poner la marcha atrás? ¿cómo lo solucionarías?

Por qué el método setMarcha() de la clase Automovil() no lo hemos hecho privado?

Como lograríamos que una instancia de AtumovilManual pueda cambiar de marcha?

Implementa un clase Prueba en cuyo método main se instanciará un automóvil manual y un automóvil automático. Verifica el funcionamiento de las clases escritas.

### *Ejercicio 6:*

Defina las clases (nombre, superclase, atributos y métodos) para implementar una solución orientada a objetos para el siguiente problema:

Un Sistema de Archivos debe organizar y manipular: archivos, directorios, links y archivos comprimidos. Los archivos se definen por un nombre, una fecha de creación, una fecha de última modificación y un tamaño. Los directorios tienen un nombre, una fecha de creación y además contienen un conjunto de archivos y un conjunto de subdirectorios. El tamaño de un directorio está dado por el tamaño de sus sub-directorios, sus archivos, sus links y sus archivos comprimidos. Los links son vínculos a otro archivo o directorio y tienen un nombre, una fecha de creación y su tamaño en disco es siempre igual a 1Kb. Los archivos comprimidos son un tipo particular de archivo el cual contiene otros archivos y/o directorios en formato comprimido según una taza de compresión dada y tiene un nombre y una fecha de creación.

El sistema debe ser capaz de informar el tamaño de cualquiera de los elementos antes mencionados. Además, en el caso de los directorios y de los archivos comprimidos, se debe poder pedir un listado total de los subdirectorios y archivos.

Implementa las clases necesarias para comprobar la funcionalidad del sistema considerando la siguiente estructura:

```
C:.
│  Acceso directo a DSC08910.JPG.lnk
│  Acceso directo a Save me.mp3.lnk
│  Queen.zip (Ocupa el 80% del total)
│  Mis Documentos.zip (Ocupa el 30% del total)
│  Recordatorio.txt (5 kb)
├──fotos
│  │  CAM00053.jpg (150 kb)
│  │  CAM00054.jpg (200 kb)
│  │  CAM00055.jpg (170 kb)
│  │  CAM00056.jpg (150 kb)
│  │  CAM00057.jpg (250 kb)
│  │
│  └──viaje
│        Acceso directo a raiz.lnk
│        DSC08904.JPG (1500 kb)
│        DSC08909.JPG (1000 kb)
│        DSC08910.JPG (2000 kb)
│        DSC08911.JPG (2500 kb)
│
├──mis documentos
│      carta.doc (30 kb)
│      curriculum.doc (60 kb)
│      receta de cocina.doc (80 kb)
│
└──mp3
   │  El choclo.mp3 (3500 kb)
   │  El dia que me quieras.Mp3 (4500 kb)
   │  Naranjo en flor.MP3 (5000 kb)
   │
   ├──Queen
   │      Bohemian Rhapsody.mp3 (5300 kb)
```

```
│      Made in heave.mp3 (6500 kb)
│      Save me.mp3 (2500 kb)
│
└──────The beatles
       Let it be.mp3 (3530 kb)
       Yesterday.mp3 (3000 kb)
```

Nota: Los archivos Queen.zip y Mis Documentos.zip contienen los mismos archivos que los directorios correspondientes.

## *Ejercicio 7*

**Inciso a)**
Se desea implementar un sistema que maneje los datos de diferentes equipos de futbol. El sistema pretende mediante un cálculo probabilístico (de dudoso rigor científico) informar los índices de efectividad (ataque y defensa) que tiene cada equipo.
Un equipo debe constar de un nombre identificatorio, y un conjunto de jugadores. Se debe poder agregar, eliminar jugadores y obtener el conjunto de los mismos.

De los jugadores se sabe su nombre de tipo String. Para realizar los cálculos probabilísticos, cada jugador tiene un valor de velocidad, y otro de potencia (ambos valores están entre 0 y 1 y deberán conocerse a la hora de crear el jugador.

Además tendrá los métodos

double indiceDefensa() y double indiceAtaque().

Existen tres clases de jugadores: Delanteros, Defensores, y Arqueros.

La forma de calcular los índices de defensa y ataque varían de acuerdo al tipo de jugador de acuerdo a la siguiente tabla:

| | indiceDefensa | indiceAtaque |
|---|---|---|
| Delantero | velocidad * 0.5 | velocidad * potencia |
| Defensor | $velocidad^2$ | $potencia^2$ |
| Arquero | efectividad (atributo propio del arquero. Siempre entre 0 y 1, valor por defecto 0.5) | 0.1 * velocidad * potencia |

Sobreescribe el método toString de los diferentes tipos de jugadores.

Cada equipo tendrá los métodos double indiceDefensa() y double indiceAtaque()que retornará la suma de los correspondientes índices de sus jugadores.

La clase equipo contara con los métodos:

String agregaDelantero(String nombre,  double velocidad, double potencia);

String agregaDefensor(String nombre,double velocidad, double potencia);

String agregaArquero(String nombre,  double velocidad, double potencia);

void eliminaJugador(Jugador jugador);

Iterator<Jugador> getJugadores();

double indiceDefensa();

double indiceAtaque();

Los  métodos que agregan jugadores retornarán un String con el mensaje “Jugador agregado” en caso de éxito, o un String informando la imposibilidad de crear un jugador si alguno de los parámetros no estaba en el rango permitido de 0 a 1 (por ejemplo “Imposible crear un delantero con velocidad = 1.5”).

**Inciso b)**

Considera los métodos  agregaDelantero,  agregaDefensor,  agregaArquero. Dichos métodos agregan un jugador al equipo realizando las comprobaciones pertinentes. ¿Sería posible generalizar estas acciones en un único método agregarJugador?. ¿Podría delegarse la responsabilidad de creación de jugadores en otro lugar?