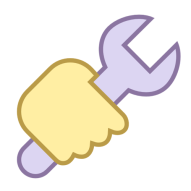
**CFP 8**

**Módulo 2:**

**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Unidad 4:**

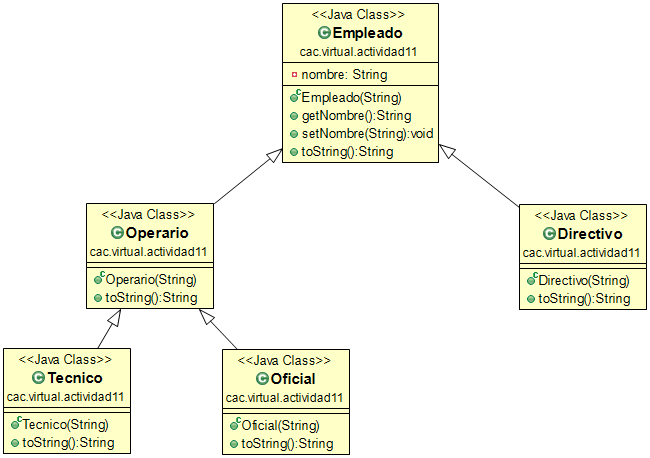
**HERENCIA**

****

## ACTIVIDADES PRÁCTICAS

### Herencia, Subclases y superclases, Sobrecarga, protected vs private, final, Herencia vs Composición, constructores en herencia

Considere la siguiente jerarquía de clases java representada por este diagrama UML para los **ejercicios 1 a 5**:



**1)**

Crear el código para la clase base Empleado. Esta clase contiene:

* Un atributo privado nombre de tipo String que **heredan** el resto de clases.
* Un constructor por defecto.
* Un constructor con parámetros que inicializa el nombre con el String que recibe.
* Método set y get para el atributo nombre.
* Un método toString() que devuelve el String: "Empleado " + nombre.

**Nota:** antes de escribir el resto de clases, que se mencionan en los ejercicios siguientes, considere que debe sobrescribir el método toString() en cada una de ellas y declarar el constructor adecuado de forma tal que al ejecutar las siguientes instrucciones:

* Empleado E1 = new Empleado("Rafael");
* Directivo D1 = new Directivo("Mario");
* Operario OP1 = new Operario("Alfonso");
* Oficial OF1 = new Oficial("Luis");
* Tecnico T1 = new Tecnico("Pablo");
* System.out.println(E1);
* System.out.println(D1);
* System.out.println(OP1);
* System.out.println(OF1);
* System.out.println(T1);

De como resultado lo siguiente:

* Empleado Rafael
* Empleado Mario -> Directivo
* Empleado Alfonso -> Operario
* Empleado Luis -> Operario -> Oficial
* Empleado Pablo -> Operario -> Tecnico

**2)**

Vamos a crear es la clase Operario que **hereda** de forma directa de la clase Empleado. Recuerde que en Java la herencia se expresa mediante la palabra **extends**. *Ejemplo: Operario extends Empleado indica que la clase Operario deriva o hereda de la clase Empleado.* La clase Operario hereda de Empleado el atributo nombre, los métodos get y set y el método toString(). El constructor de la clase recibe el nombre del empleado y lo pasa al constructor de la clase base mediante la instrucción **super**(nombre). En esta clase se modifica (**override**) el método toString() heredado de la clase base para que muestre un mensaje como pide el ejercicio.

**3)**

Escriba el código de la clase Oficial que hereda de la clase Operario. La clase Oficial hereda de Operario el atributo nombre, los métodos get y set y el método toString(). El constructor de la clase recibe el nombre del empleado y lo pasa al constructor de su clase base, en este caso la clase Operario, mediante la instrucción **super**. En esta clase también se modifica el método toString() heredado de su clase base para que muestre un mensaje como pide el ejercicio. Declare a la clase Oficial de tipo **final** y pruebe generar una nueva clase que hereda de ella. ¿Qué es lo que ocurre?

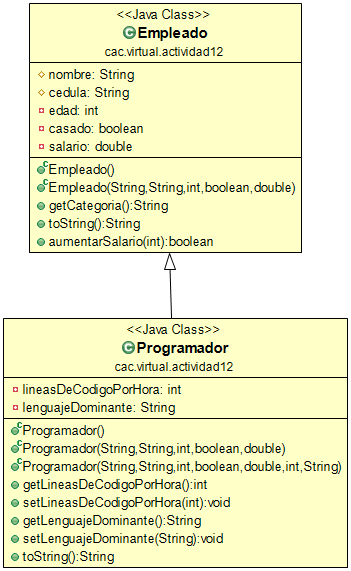
**4)**

Escribir la clase Directivo, similar a la clase Operario. Declare a la clase directivo de tipo **final**.

**5)**

Escribir la clase Tecnico, similar a la clase Oficial. Declare a la clase Tecnico de tipo **final**.

**6)**



La empresa informática “Codo a Codo Tech” necesita llevar un registro de todos sus empleados que se encuentran en la oficina central, para eso ha creado un diagrama de clases que debe incluir la clase Empleado con las siguientes características:

Atributos:

* nombre: tipo cadena (nombre y apellido)
* cedula: tipo cadena
* edad : entero (rango entre 18 y 45 años)
* casado: boolean
* salario: tipo numérico doble

Métodos:

* Cree un constructor con y sin parámetros de entrada
* Cree un método getCategoria que permita mostrar la clasificación según la edad de acuerdo al siguiente algoritmo:
  + Si edad es menor o igual a 21, Principiante
  + Si edad es >= 22 y <= 35, Intermedio
  + Si edad es >35, Senior
* Sobrescriba el método toString() para poder imprimir los datos del empleado por pantalla (puede utilizar salto de línea \n para separar los atributos).
* Cree un método que permita aumentar el salario en un porcentaje que sería pasado como parámetro al método.

**7)**

Teniendo en cuenta la clase creada en el ejercicio anterior, cree la clase Programador (especialización de Empleado). La clase Programador hereda de Empleado todos sus atributos y métodos. Los atributos que serán propios de la clase Programador serán:

* lineasDeCodigoPorHora : tipo entero
* lenguajeDominante: tipo cadena

Para la clase Programador deberá crear también el método constructor con y sin parámetros de entrada. Redefina algunos de los atributos de la clase Empleado con el tipo de modificador de acceso **protected**. ¿Qué es lo puede hacer ahora desde la clase Programador al haber redefinido así la clase Empleado?

**public** **class** Empleado {

**protected** String nombre;

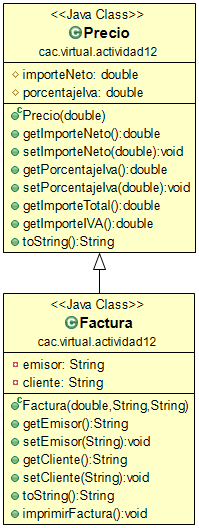
**protected** String cedula;

**private** **int** edad;

**private** **boolean** casado;

**private** **double** salario;

**8)**



Construir una clase denominada Factura, hija de la clase Precio y que posea 2 atributos específicos llamados emisor y cliente (que deberán ser los números de CUIT de ellos) y, al menos, 1 método llamado imprimirFactura que deberá mostrar los datos de esa factura.

La clase Precio tendrá los atributos que se listan a continuación, al menos un constructor, sus getters y setters y el método toString:

**public** **class** Precio {

**protected** **double** importeNeto;

**protected** **double** porcentajeIva;

La clase Factura tendrá los atributos anteriormente mencionados, al menos un constructor que invoque al constructor de la clase Precio, sus getters y setters y el método toString.

**public** **class** Factura **extends** Precio{

**private** String emisor;

**private** String cliente;

En ejemplo de ejecución del método imprimirFactura podría ser:

*Factura:*

*Emisor: 20123456789*

*Cliente: 33123456789*

*Precio:*

*Importe Neto: $100.0*

*IVA 21.0%: $21.0*

*Importe Total: $121.0*

**9)**

Crear una clase llamada Cuentaque tendrá los siguientes atributos: dniTitular y saldo(puede tener decimales). El dniTitular será obligatorio y el saldo es opcional. Crea dos constructores que cumplan con lo anterior. Crea sus métodos get y set (excepto para saldo que no deberá tener set) y toString.

La clase tendrá dos métodos más:

* depositar(double importe): se ingresa un importe a la cuenta, si el importe es negativo o cero, no se hará nada y el método devolverá false (error), caso contrario deberá retornar true (ok) y actualizar el saldo de la cuenta.
* extraer(double importe): se retira una cantidad determinada de la cuenta. Si la resta entre el saldo actual y el importe a extraer da negativo, retornar false; caso contrario, retornar true y actualizar el saldo de la cuenta.

**10)**

Reutilizando la clase Cuenta del ejercicio anterior, modifique los atributos de la clase para que tenga un objeto de tipo Persona como atributo en lugar de dniTitular (**composición**). Modifique el resto del código de la clase Cuenta para que su implementación no se vea alterada.

La clase Persona deberá respetar las siguientes condiciones:

* Sus atributos son: dni, nombre, edad, sexo (M y F, Masculino y Femenino). No queremos que se accedan directamente a ellos. Piensa que modificador de acceso es el más adecuado, también su tipo.
* Por defecto, todos los atributos menos el dni serán valores por defecto según su tipo (0 números, cadena vacía para String).
* Se implantarán varios constructores: un constructor que sólo solicitará el dni, un constructor con el dni y sexo, el resto por defecto, y un constructor con todos los atributos como parámetro.
* Métodos set de cada parámetro, excepto para DNI. getters para todos los parámetros.
* toString(): devuelve toda la información del objeto.
* esMayorDeEdad(): indica si es mayor de edad, devuelve un tipo de dato booleano.

Crea una clase instanciada que pida por teclado los datos necesarios para instanciar 3 objetos de tipo Persona utilizando los 3 constructores anteriormente realizados. En caso de ser mayores de edad, crearles una cuenta con la nueva clase Cuenta creada en este ejercicio y que contenga el correspondiente objeto de tipo Persona. Por último, mostrar la información de cada objeto Cuenta y Persona creado.

**11)**

Escriba un método para realizar la búsqueda del nombre de un cliente (Clase Cliente) guardado en un **ArrayList** de clientes. El cliente a buscar será ingresado por consola por el usuario. El método deberá devolver true en caso de que ese nombre exista, sino false. El método buscarCliente podría estar dentro de una clase llamada Local. La clase Cliente deberá tener al menos los atributos: id (int) y nombre (String), sus correspondientes getters y setters y su constructor Cliente(int id, String nombre).

**12)**

Un supermercado nos pide que hagamos una pequeña aplicación que almacene los productos pasados por el escáner. La aplicación debe almacenar Productos (clase) y la cantidad comprada. Para ello deberá tener un **ArrayList** de compras y cada elemento de la lista deberá tener un objeto de tipo Producto y una cantidad de tipo int. La compra al crearse contendrá un precio (puede generarlo aleatoriamente) y una cantidad asociada que también puede ser generado de esa forma si lo desea. El nombre del producto será básico (producto1, producto2, producto3, etc.). Cree la clase Producto, la clase Compra y la clase Supermercado que contendrá el ArrayList de compras. Calcule el precio total de una lista de entre 1 y 8 productos (aleatorio). Mostrar un ticket con todo lo vendido y el precio final como se hacen en los supermercados. Más o menos con este formato, lo importante son los datos, no el estilo:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cantidad\*\*\*Precio\*\*\*\*\*Total

Producto1 5 3.5 17.5

Producto2 7 2.5 17.5

Precio final 35

**Nota:** este ejercicio es una simplificación de un caso real. En un sistema real, la compra debería estar asociada a un Cliente y cada compra a su vez tendría un ArrayList de ítems de la compra que a su vez contendría la información del producto y su correspondiente cantidad comprada.