PRÁCTICA 3 de JAVA:

MVC, herencia, this y this(), super(), protected, métodos abstractos (abstract), cast, instanceof, final y paquetes.

Fecha tope de entrega	Miércoles, 6 de octubre	**Fecha tope de defensa**	Viernes, 8 de octubre.
Tipo	Individual	**Asunto e-mail**	JAVA03
Formato fichero: ApellidoNombreJAVA03.zip (compartir por Google Drive)			

Consistirá en una aplicación gráfica para gestionar una lista de objetos que heredan de la clase Empleado (práctica 1). Requisitos:

1.- Modifica la clase Empleado. Tendrá:

1.1.- Atributos no estáticos: nombre (String), salario (float) y fecha de alta (GregorianCalendar).

1.2.- El atributo sueldo máximo (no puede sobrepasarse) será el mismo paro todos los empleados (static)

1.3.- Dos constructores (sobrecarga).

El nombre siempre debe tener valor.

Debe usarse this(?) en el segundo constructor.

1.4.- Métodos get y set para todos los atributos.

1.5.- El método setFechaAlta(GregorianCalendar fecha), modificará la fecha de forma manual, para que pueda ser diferente a la dada por el sistema.

1.6.- Las fechas se pueden pedir y mostrar como int o String. Se debe crar una clase estática (como Terminal de la prácticas 01) para su tratamiento.

1.7.- - Contendrán el método abstracto toString().

2.- Crea dos clases de forma libre, que hereden de la Empleado base con la siguientes peculiaridades:

2.1.- Usarán dos constructores que debe hacer uso de super(???).

2.2.- Cada una debe tener un atributo (libre) distinto al de la otra pero del mismo tipo, al que se el aplicará el método getDato() definido en la clase madre como abstracto y implementado en las hijas.

2.3.- Sobre anterior atributo actuará el método getDato().

Su uso es obligatorio para acceder a los valores de las clases hijas sin usar casting.

Ser definirá en Empleado como abstracto.

Cuidado en las clases hijas.

2.3.- Habrá otro atributo libre con diferente nombre y tipo para cada hija, con los métodos get y set correspondientes.

2.4.- Sobrescribirán toString().

4.- Desarrolla una clase Lista independiente de los datos.

4.1.- Se basará en una lista de Object enlazados y con un apuntador inicial.

4.2.- Desarrolla lo métodos necesarios para su manejo. Deben ser genéricos, es decir, independientes de los objeto que pueda albergar. Por ejemplo: crear (new), insertar, avanzar, retroceder, leer, inicio, fin, esVacia, tamaño, etc.

4.3.- La clase no puede ser estática. Se necesita una clase Nodo para almacenar los elementos de la lista.

4.4.- Habrá dos clases: Lista (como una “collection”) y Nodo (almacenar cualquier elemento (Object), en nuestros caso “hijos” de la clase Empleado).

5.- Gráficos. Se desarrollarán a partir de las herramientas que ofrece el IDE Netbeans utilizado en clase:

5.1.- Se basarán en un JFrame que incorpora un menú básico para las opciones: bienvenida, altas y mostrar, para los objetos creados (hijos de la clase base).

6.- JPanel Alta Empleado.

6.1.- Permite crear un objeto de cualquiera de las clases hijos e insertarlo en la lista.

6.2.- Se podrán crear objetos con una fecha introducida manualmente o que coja la del sistema (Internamente se usará el constructor o el método setFecha).

7.- JPanel Mostrar:

7.1.- Visualizará una a una todos los atributos de los empleados en la lista, indicando a que clase hija pertenece (que tipo de empleado es).

7.2.- Tendrá botones para avanzar y retroceder.

7.3.- Los objetos visualizados podrán ser todos y con los siguientes filtros.

7.4.- Filtrado por fecha (por un día, un mes o año)

7.5.- Filtrado por el atributo del método getDato().

7.6.- Filtrado por uno de lor atributos de una de las clases hija.

8.- JPanel de Bienvenida: con el nombre del autor, versión, fecha, etc.

9. De una de las clases hijas no podrá heredarse y de la otra sí. De esta última, un método no podrá sobrescribirse por sus posibles hijas (“nietas de Empleado”).

10.- Usaremos el modelo vista controlador y se debe utilizar un paquete llamado “Apellido1Apellido2Nombre” y dentro los paquetes modelo, vita y controlador.

Ayuda sobre paneles:

// Si en el JFrame principal se declara los objetos, como JpanelAlta y JPanelMostrar, y luego se instancia, no se perderá el contenido de estos Jpanel al pasar de un panel a otro.

```
private JPanelAlta panelAlta = new JPanelAlta();

    // Acción asociada al menu

private void verMenuItemActionPerformed(java.awt.event.ActionEvent evt){
    cambiarContenedor(panelAlta)
}

// Para cargar el JPanel deseado en la Application principal

private void cambiarContenedor(javax.swing.JPanel aux){
      this.setContentPane(aux);
      pack();
}
```