TXURDINAGA CIFP LHII	Curso / *Kurtsoa*	Fecha / *Data*	Nivel / *Maila*	Eval. / *Ebal.*
	2018/2019	19/06/2019	1	Final 2

Módulo / *Modulua*	*Kodea / Código*	*U.Didak /Unid didác*	*Tipo / Mota*	Calificación/*Kalifikazioa*
Programación	PROG	Todas	Todos	

Nombre / Izena	

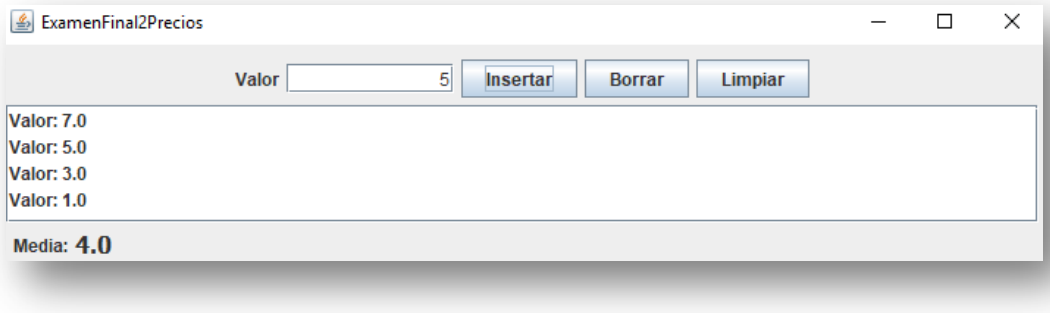
Crea un nuevo proyecto de Java de nombre Java si todavía no existe. Crea dentro un paquete con el nombre **examenFinal2** y guarda dentro de él las clases de este examen.

Parte 1. (1,25 Ptos.)
Crea una nueva clase Java de nombre **Precio** que constará de un atributo de tipo **double** llamado **valor** y de los siguientes métodos:

- Un constructor por defecto que tenga como valor por defecto 1.0 para el valor.
- Un constructor que permita dar valores al valor.
- Un constructor copia.
- Setters y Getters necesarios para el correcto funcionamiento de la clase.
- Método toString que devuelve un String con el formato **"Valor: " + this.valor**
- Métodos equals, hashCode.
- Método **compareTo** para comparar objetos de la clase. Los objetos se comparan **en función de su valor**, haciendo que los de mayor valor aparezcan antes (**orden descendente**).

Si falla sólo uno de los apartados y el resto está bien la nota será de 0,50 Ptos.

Parte 2. (3,75 Ptos.)
Crea una nueva clase Java de nombre **ExamenFinal2Precios**. Para la realización de la clase se utiliza la clase Precio y la siguiente interfaz gráfica



La aplicación cuenta con los siguientes componentes:
lblValor. JLabel con el texto "Valor ".

txtValor. JTextField con el texto "1.0". Al pulsar enter hará lo mismo que btnInsertar. Al coger el foco se seleccionará todo el texto y al perder el foco el texto dejará de estar seleccionado. (0,25 Ptos.)

btnInsertar. JButton con el texto "Insertar". Al hacer clic sobre él se comprobará que el valor de txtPrecio sea un valor válido. **Si no es un valor válido**, mostrará un mensaje de error con el texto “Valor no válido” y no añadirá el valor a la lista. **Si el valor es válido**, creará un nuevo objeto de la clase **Precio** que tendrá como valor el valor que hay en el campo de texto txtValor y, **si** ese nuevo objeto de la clase Precio **todavía no está en la lista**, lo añadirá a los datos de la lista **ordenado según** el criterio de ordenación por defecto de la clase Precio (**compareTo**). (1,25 Ptos.)

btnBorrar. JButton con el texto "Borrar". Al hacer clic sobre él se borrarán **todos los elementos seleccionados** en la lista, si es que hay alguno. (0,5 Ptos.)

btnLimpiar. JButton con el texto "Limpiar". Al hacer clic sobre él se borrarán todos los elementos de la lista. (0,25 Ptos.)

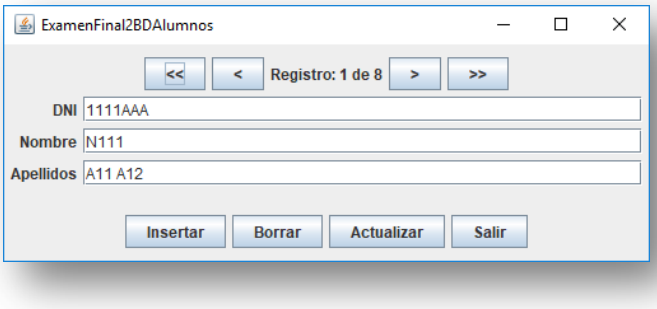
lstPrecios. JList que permite la **selección múltiple** y que contiene elementos de la clase **Precio**. Los datos de la lista se guardan en un **DefaultListModel** de nombre **dlmPrecios** que guarda elementos de la clase **Precio**. Además cuenta con un JScrollPane de nombre **scrollPane** que permite que se vean los elementos de la lista.

panelEstado. JPanel que tiene una JLabel de nombre **lblMedia** con el texto "**Media:** " y otra JLabel de nombre **lblMediaValor** cuyo valor será la media de todos los elementos de la lista. Para facilitar el proceso del cálculo de la media se creará en la clase un nuevo método de nombre **calcularMedia** que será llamado cada vez que cambien los datos de la lista y que **calculará el nuevo valor de la media** y lo asignará a la etiqueta **lblMediaValor**. (1,25 Ptos.)

Parte 3. (0,75 Ptos.)
Al hacer clic sobre la X para **cerrar la ventana** de la aplicación **sólo si los datos han sido modificados** se guardarán los datos del modelo de datos de la lista en el fichero **precios.ser** usando para ello el método **guardarPrecios** que es el que se encarga de grabar los datos de los **objetos** de la clase **Precio** que hay en la lista en el fichero precios.ser usando **serialización**.

Parte 4. (0,50 Ptos.)
Al **iniciar la aplicación** se cargan los datos de los **objetos** de la clase **Precio** que hay en el fichero **precios.ser** usando **serialización** en el modelo de datos de la lista. Para ello usa el método **cargarPrecios** que carga los datos desde el fichero precios.ser en el modelo de datos de la lista al comienzo de la ejecución de la aplicación.

Parte 5. (3,75 Ptos.)
Crea en MySQL una base de datos de nombre **bdfinal2** con cotejamiento utf8_spanish_ci. Crea una tabla de nombre **alumnos** con los campos de texto **dni, nombre, y apellidos**. El valor del campo dni es la clave primaria y no puede estar repetido.
Configura Eclipse para que permita conexiones a bases de datos MySQL.
Crea clase Java **ExamenFinal2BDAlumnos** que tiene un campo de texto por cada campo de la tabla alumnos de la base de datos bdfinal2 que son dni, nombre, y apellidos.
Para facilitar el trabajo con los datos, al iniciar la aplicación se cargan los datos de la tabla alumnos de la base de datos bdfinal2 en objetos de la clase **ArrayList** y al cerrar la aplicación se guardan los datos de los ArrayList en la base de datos, pero sólo si han sido modificados.
La interfaz gráfica contiene los siguientes componentes



btnPrimero. JButton que al hacer clic sobre él permite ir al primer registro de datos (si es que hay).
btnAnterior. JButton que al hacer clic sobre él permite ir al registro anterior (si es que hay).
btnSiguiente. JButton que al hacer clic sobre él permite ir al registro siguiente (si es que hay).
btnUltimo. JButton que al hacer clic sobre él permite ir al último registro (si es que hay).
Si funcionan correctamente los 4 botones la nota será de 1 Pto, si funcionan 3 botones la nota será de 0,5 Ptos. En cualquier otro caso la nota será de 0 Ptos.
btnInsertar. JButton que al hacer clic sobre él añade un nuevo alumno al ArrayList, si todavía no ha sido añadido (si todavía no existe ese dni). (0,5 Ptos.)
btnBorrar. JButton que al hacer clic sobre él borra el registro actual del ArrayList, si hay registros. (0,25 Ptos.)
btnActualizar. JButton que al hacer clic sobre él actualiza los datos del registro actual del ArrayList salvo el campo dni que no se modifica, si hay registros. (0,25 Ptos.)
btnSalir. JButton que al hacer clic sobre él guarda los datos en la base de datos, sólo si han sido modificados. Para ello, borra todos los registros de la tabla alumnos y añade uno a uno todos los registros que haya en los ArrayList. (1,25 Ptos.)
lblRegistros. JLabel que si en la lista no hay registros muestra el texto "No hay registros" y, si hay registros, muestra el texto "Registro: " + registroActual + " de "+numeroRegistros. La etiqueta lblRegistros se tiene que actualizar cada vez que se produzcan cambios ya sea porque se va a otro registro o porque se inserta o se borra un registro. Para facilitar el proceso se usará un método de nombre **actualizarCampos**. Para tener la puntuación de 0,5 Ptos la etiqueta se tiene que actualizar correctamente en todas las situaciones en las que se produzca un cambio. Si falla en alguna situación la nota final será de 0 Ptos.

Criterios de Corrección
Para tener la puntuación correspondiente a cada parte es necesario que todo lo que se pide en esa parte sea correcto. Si no es así la puntuación de esa parte será de 0 salvo que existan subpartes.
Si existen subpartes se valorará cada subparte, siendo necesario que todo lo que se pide en esa subparte sea correcto para tener la puntuación asignada. Si no es así la puntuación de esa subparte será de 0.
Al finalizar el tiempo se recoge el examen y se guarda el paquete con las clases dentro de una carpeta de nombre **Apellido, Nombre** donde Apellido hace referencia tu primer apellido y nombre a tu nombre. Por ejemplo, De Miguel, Txema.
No se permitirá ningún tipo de comunicación durante el examen. Los cables de red permanecerán desconectados, las conexiones inalámbricas (si las hubiera) se deshabilitarán y los móviles deberán estar apagados. Si un alumno incumple alguna de estas normas entregará su examen y tendrá un 0 como nota