UD06.POO Avanzada. Tipos Predefinidos y Colecciones

Ejercicios

DAM1-Programación 2024-25

Colecciones

Paquete: **coleccionesejercicios**

Contenedores

EP1211_ContenedorEnteros. Utilizando la clase `ContenedorLista` definida en la actividad **E1202**, implementa una aplicación donde se guardan 30 enteros aleatorios entre 1 y 10 y luego ordenan de menor a mayor. La aplicación debe mostrar el contenedor antes y después de ordenar.

A continuación ordena y muestra los elementos de mayor a menor.

EP1212. Añade a la clase `ContenedorLista` los siguientes métodos:

- `T get (int indice)` que devuelve el elemento que ocupa el lugar `indice` dentro del contenedor.
- `T extract (int indice)` que **extrae** y devuelve el elemento que ocupa el lugar `indice` dentro del contenedor.
- `int[] buscar Todos (Object e)`: que devuelve una tabla con los indices de todas las ocurrencias de `e`.
- `void ordenar (Comparator<T> c)` que ordena los elementos del contenedor según el criterio de `c`.
- `boolean eliminar Todos (Object e)`: que elimina todas las ocurrencias de `e`. Devuelve `true` si la lista queda alterada.

Escribe una aplicación que pruebe todos estos métodos.

Métodos genéricos

EP1215. Implementa un **método genérico** al que se le pasa una lista de valores de la clase genérica `T` y devuelve otra donde se han eliminado las repeticiones.

EP1230. Implementa una función

```
<T> List<T> eliminaRepetidos(List<T> lista)
```

a la que se pase una lista de objetos y devuelva una copia sin elementos repetidos.

EP1232. Implementa la función

```
static <E> List<E> clonaLista (List<E>)
```

que realice una copia exacta de una lista.

Estructuras de datos: Pilas, Colas, etc.

EP1217. Implementa la clase `Cola` genérica utilizando un objeto `ArrayList` para guardar los elementos.

EP1218. Implementa la clase `Pila` genérica utilizando un objeto `ArrayList` para guardar los elementos.

EP1231. Implementa las clases `Cola` y `Pila` genéricas **heredando** de `ArrayList`.

EP1220. Utilizando colecciones, implementa la clase `Supercola`, que tiene como atributos dos colas para enteros, en las que se encola y desencola por separado. Sin embargo, si una de las colas queda vacía, al llamar a su método `desencolar()`, se desencola de la otra mientras tenga elementos. Solo cuando las dos colas estén vacías, cualquier llamada a desencolar devolverá `null`. Escribe un programa con el menú:

1. Encolar en `cola1`.
2. Encolar en `cola2`.
3. Desencolar de `cola1`.
4. Desencolar de `cola2`.
5. Salir

Después de cada operación se mostrará el estado de las dos colas para seguir su evolución.

EP1233. Define la clase `ListaOrdenada`, que almacena una serie de objetos de tipo genérico `E`, de forma ordenada, pudiéndose repetir. Los elementos se ordenarán según el orden natural de `E` o bien con el criterio de orden definido en un comparador que se le pasa al constructor.

Socios

EP1216. Implementa una aplicación que gestione los socios de un club usando la clase `Socio` implementada en la Actividad **E1211**. En particular, se deberán ofrecer las opciones de alta, baja y modificación de los datos de un socio. Además, se listarán los socios por nombre o por antigüedad en el club.

EP1225. Implementa una aplicación que gestione un club donde se identifica a los socios por un apodo personal y único. De cada socio, además del apodo, se guarda el nombre y su fecha de ingreso en el club. Utiliza un mapa donde las claves serán los apodos y los valores, objetos de la clase `Socio`. Los datos se guardarán en un fichero llamado “club.dat”, de donde se leerá el mapa al arrancar y donde se volverá a guardar actualizado al salir. Las operaciones se mostrarán en un menú que tendrá las siguientes opciones:

1. Alta socio.
2. Baja socio.
3. Modificación socio.
4. Listar socios por apodo.
5. Listar socios por antigüedad.
6. Listar los socios con alta anterior a un año nado.
7. Salir.

Variados

EP1219. Escribe un programa donde se introduzca por consola una frase que conste exclusivamente de palabras separadas por espacios. Las palabras de la frase se almacenarán en una lista. Finalmente, se mostrarán por pantalla las palabras que estén repetidas y, a continuación, las que no lo estén.

EP1221. Implementa una aplicación donde se insertan 20 números enteros aleatorios distintos, menores que 100, en una colección. Deberán guardarse por orden decreciente a medida que se vayan generando, y se mostrará la colección resultante por pantalla.

EP1224. Implementa una función a la que se le pasen dos listas de enteros ordenadas en sentido creciente y nos devuelva una única lista, también ordenada, fusión de las dos anteriores. Desarrolla el algoritmo de forma no destructiva, es decir, que las listas utilizadas como parámetros de entrada se mantengan intactas.

EP1222. Introduce por teclado, hasta que se introduzca “fin”, una serie de nombres, que se insertarán en una colección, de forma que se conserve el orden de inserción y que no puedan repetirse. Al final, la colección se mostrará por pantalla.

Añade una segunda colección en la que se inserten los nombres manteniendo el orden alfabético.

EP1227. Implementa la función `LeeCadena()`, con el siguiente prototipo:

```
List<Character> leeCadena()
```

que lee una cadena por teclado y nos la devuelve en una lista con un carácter en cada elemento.

EP1228. Implementa la función `uneCadenas`, con el siguiente prototipo:

```
List<Character> uneCadenas(List<Character> cad1, List<Character> cad2)
```

que devuelve una lista con la concatenación de `cad`1 y `cad2`.

EP1226. Un **centro educativo** necesita distribuir de forma aleatoria a los alumnos de un curso entre los grupos disponibles para ese curso. Diseña la función

```
List<List<String>> repartoAlumnos(List<String> lista, int numGrupos)
```

que devuelve una lista de listas, cada una de las cuales corresponde a un grupo. Cada nombre de la lista de alumnos se asigna a uno de los grupos.

EP1234. Amplía la Actividad **E1214**, de forma que se gestionen los registros de temperatura de diferentes días en la misma aplicación. Para ello, implementa un mapa cuyas entradas tendrán como clave la fecha y como valor el conjunto con los registros de un dia. Implementa también un programa que gestione los registros del dia actual y permita visualizar los de un día cualquiera, junto con sus estadísticas. Al arrancar el programa se cargará en memoria el mapa a partir del fichero correspondiente y, al terminar, se volverá a guardar actualizado.

EP1235. Con la clase `Jornada` definida en la Actividad de ampliación **EP0928**, Implementa una aplicación que gestione las jornadas de los trabajadores de una empresa por medio de colecciones, incluyendo altas y bajas de trabajadores y altas de jornadas, así como el listado de las jornadas de un trabajador. Los datos se guardarán en un fichero binario.

EP1236. Repite la Actividad de ampliación **EP0932** utilizando una colección para guardar y manipular las Llamadas.

EP1237. Queremos gestionar la plantilla de un equipo de fútbol, en la que a cada jugador se le asigna un dorsal que no puede estar repetido. Para ello vamos a crear una estructura de tipo `Map` cuyas entradas corresponden a los jugadores, con el dorsal como clave y un objeto de la clase `Jugador` como valor. De cada jugador se guarda el DNI, el nombre, la posición en el campo - para simplificar, los jugadores pueden ser porteros, defensas, centrocampistas y delanteros - y su estatura. Define la clase `Jugador` y un enumerado para la posición en el campo, e implementa los siguientes métodos estáticos:

1. `static void altaJugador(Map<Integer, Jugador> plantilla, Integer dorsal, Jugador jugador)`, que añade una entrada al mapa con el dorsal pasado como parámetro y el jugador creado dentro del método.
2. `static Jugador eliminarJugador(Map<Integer, Jugador> plantilla, Integer dorsal)`, que elimina la entrada correspondiente al jugador cuyo dorsal se pasa como parámetro. Dicho dorsal desaparece del mapa hasta que se asigne a otro jugador por medio de un alta. El método devuelve el jugador eliminado.

3. `static void mostrar (Map<Integer, Jugador> plantilla)`, que muestra una lista de los dorsales con los nombres de los jugadores correspondientes.

```
dorsal - jugador
dorsal2 - jugador2
…
```

4. `static void mostrar (Map<Integer, Jugador> plantilla, String posicion)`, que muestra una lista de los jugadores que comparten una misma posición. Por ejemplo, todos los defensas o todos los delanteros.
5. `static boolean editarJugador(Map<Integer, Jugador> plantilla, Integer dorsal, Jugador jugador)`, que permite modificar los datos de un jugador, excepto su dorsal y su DNI. Devuelve `true` si el dorsal existe y `false` en caso contrario.

Festival das Meigas

Paquete: festivalmeigas

A vila imaxinaria de **San Comadro**, en pleno corazón da Galiza máxica, celebra cada ano o seu tradicional **Festival das Meigas**, onde se organizan concursos de **feitizos**, se intercambian **ingredientes máxicos** e se premia ás **meigas máis creativas**.

Funcionalidades obrigatorias:

1. **Rexistro de Meigas**:
   Cada meiga ten un **nome**, un **alcume máxico**, e unha lista de **feitizos favoritos**.
   Os feitizos non se poderán repetir. As meigas poderanse amosar ordenadas por diferentes criterios.

2. **Base de datos de Feitizos**:
   Cada feitizo ten un **nome**, unha lista de **ingredientes**, e un nivel de **dificultade**.
   a. A partir de todos os feitizos rexistrados queremos obter unha colección de **todos os ingredientes únicos** empregados no festival.
   b. Xera un listado dos feitizos que se poden facer cunha lista de ingredientes dados.
   c. Permite listar os 3 ingredientes máis usados en total (pista: podes usar un `Map<String, Integer>` para contar).

3. **Funcionalidades extra**:

   a. Xerar unha lista de meigas que comparten un mesmo ingrediente nun dos seus feitizos favoritos.
   b. Implementa un sistema de recomendación: dado un ingrediente, suxerir ás meigas un novo feitizo que o use e que **non teñan nos seus favoritos**.