9.H. Actividades propuestas.

Sitio:	Formación Profesional a Distancia
Curso:	Programación
Libro:	9.H. Actividades propuestas.
Imprimido por:	Iván Jiménez Utiel
Día:	viernes, 17 de abril de 2020, 18:08

Tabla de contenidos

1. ActividadUT09-1: Clase genérica y métodos genéricos.

Crear una clase genérica **ClaseGenerica** que tenga como único atributo un objeto genérico, como constructor, el constructor general, que inicializa el atributo, y que además disponga de un método llamado **tipoDeClase()**, que muestre el tipo de dato que se ha cargado para la clase genérica (usaremos **getClass().getName()** sobre el objeto).

Además hacer una clase de prueba donde instanciemos una **ClaseGenerica** para **Integer** y veamos el valor del entero correspondiente. También instanciaremos una **ClaseGenerica** para **String** y mostraremos su valor. Por último, intentaremos instanciar una **ClaseGenerica** para un **char**, y veremos el resultado.

2. ActividadUT09-2: Recorriendo una LinkedList.

Crear una **LinkedList** sobre la cual incorporaremos objetos de tipo **String** (por ejemplo 5 nombres leídos por teclado). Una vez incorporados mostraremos:

- Tamaño de la lista.
- Nodo en primera posición.
- Nodo en última posición.
- Nodo en una posición cualquiera dentro del rango.
- Nodo en una posición cualquiera fuera del rango. Incorporar control de excepciones.
- Recorrer la lista con bucle for.
- Recorrer la lista con un iterador.
- Recorrer la lista con bucle for desde el final hasta el principio.
- Recorrer la lista con un iterador desde el final hasta el principio.
- ¿Se puede recorrer la lista con un bucle for-each?
- ¿Se puede recorrer la lista con un bucle for-each desde el final hasta el principio?

3. ActividadUT09-3: Estante sin clase genérica.

Supongamos que tenemos una clase **Estante** destinada a almacenar o bien libros, o bien discos. Dichos elementos los cargaremos en un **ArrayList** de elementos, siendo necesario definir un tamaño máximo de elementos. Además sobre la clase **Estante** incorporaremos un método para añadir elementos que controlará no sobrepasar el número máximo que soporta. Además implementará la interfaz **Iterable** devolviendo un iterador de la lista de elementos.

Crearemos una clase **Libro** con atributos **titulo**, **autor** y **número de páginas**, su constructor general y sus métodos getters y setters.

Crearemos una clase **Disco** con atributos **titulo**, **autor** y **número de canciones**, su constructor general y sus métodos getters y setters.

Crearemos una clase de prueba que cargará en un objeto Estante de 10 elementos una serie de objetos Libro y objetos Disco. Recorreremos a continuación los elementos de dicho estante con un for-each, mostrando el número de páginas para los libros, y el número de canciones para las canciones.

4. ActividadUT09-4: Estante con clase genérica.

Rehacer el ejercicio anterior haciendo que la clase **Estante** utilice la clase genérica. Las clases **Libro** y **Disco**, no cambiarán respecto al anterior ejercicio. Hacer una clase de prueba, en la cual crearemos un estante específico de libros, y otro específico de discos. Comprobar cómo solo permitirán almacenar datos del tipo definido y no del otro. Mostrar la información de los libros o los discos según sea el estante mediante un for-each que recorra cada estante.

5. ActividadUT09-5: Colección de objetos Alumno.

Crear una colección de objetos **Alumno**, que a su vez estarán descritos por un nombre y una nota entera entre 0 y 10,y un constructor general y sus métodos getters y setters.

La colección se llamará **listaAlumnos** y será de tipo **Alumno**, utilizándose como constructor la clase **ArrayList** derivada de la clase **Collection**. Incorporemos una serie de alumnos. Mostraremos el contenido de la colección en forma de pares (nombre, nota). A continuación eliminaremos algún elemento, y tras esto, volveremos a mostrar el contenido para comprobar que efectivamente se elimina el objeto u objetos **Alumno**.

6. ActividadUT09-6: Ordenación de ArrayList mediante clase Collections

Dado un **ArrayList** de números enteros, ordenar los datos de menor a mayor mediante método estático proporcionado por la clase **Collections**. A continuación hacer la operación de ordenación de mayor a menor a través de **Collections** con ayuda de la clase **Comparator**.

Tendremos una salida similar a:

Datos en el mismo orden en el que se han insertado

Posición: 0 --->Valor: 3

Posición: 1 --->Valor: 2

Posición: 2 --->Valor: 4

Posición: 3 --->Valor: 1

Posición: 4 --->Valor: 7

Posición: 5 --->Valor: 5

Posición: 6 --->Valor: 6

Posición: 7 --->Valor: 0

Datos del ArrayList ordenados de menor a mayor:

Posición: 0 --->Valor: 0

Posición: 1 --->Valor: 1

Posición: 2 --->Valor: 2

Posición: 3 --->Valor: 3

Posición: 4 --->Valor: 4

Posición: 5 --->Valor: 5

Posición: 6 --->Valor: 6

Posición: 7 --->Valor: 7

Datos del ArrayList ordenados de mayor a menor:

Posición: 0 --->Valor: 7

Posición: 1 --->Valor: 6

Posición: 2 --->Valor: 5

Posición: 3 --->Valor: 4

Posición: 4 --->Valor: 3

Posición: 5 --->Valor: 2

Posición: 6 --->Valor: 1

Posición: 7 --->Valor: 0

7. ActividadUT09-7: ArrayList de Personas con métodos estáticos.

Crear una aplicación que utilice objetos de la clase **Persona**, definida por atributos **nombre**, **apellidos** y **año de nacimiento**, y que además dispondrá de sus métodos getters y setters correspondientes , así como un método **toString()** sobrescrito que mostrará todos sus datos.

Por otro lado, crear una clase **ListadoPersonas**, que incorpore un **ArrayList** de objetos **Persona**. Además del método **main()**, dispondrá de una serie de métodos estáticos o de clase que permitirán:

- Buscar por nombre.
- Buscar por apellido.
- Buscar por apellido que contenga una cadena (si buscamos "Pérez", encontrará "López Pérez" y "Pérez Romero").
- Buscar por año de nacimiento.

Nota: En todos los casos debe mostrar todas las apariciones, no solo la primera.

8. ActividadUT09-8: ArrayList de Personas con métodos de instancia.

Rehacer el ejercicio anterior considerando el uso de métodos de instancia. Además separemos la clase **ListadoPersonas** de la clase **PruebaListadoPersonas**, la cual tendrá, ésta última el método **main()** para cargar el **ArrayList** de objetos de clase **Persona** y hacer todas las pruebas de métodos anteriormente indicadas.

9. ActividadUT09-9: Conjuntos HashSet y TreeSet.

Implementar la interfaz **Set** con la clase **HashSet** y la interfaz **SortedSet** con la clase **TreeSet**, para crear un conjunto de objetos de la clase **Persona**, clase que vendrá definida por los atributos **id**, de tipo int, **nombre**, de tipo String, y **altura** (en cm) de tipo int. Esta clase **Persona** deberá implementar la interfaz **Comparable**, lo cual obligará a sobrescribir el método **compareTo()**. Dicha implementación tendrá 4 posibles formas que habrá que probar una a una comentado las demás. La primera, permitirá que el **SortedSet** se ordene por altura de manera creciente, la segunda por altura de manera decreciente, la tercera por nombre, alfabéticamente de manera creciente, y la cuarta por nombre de manera decreciente.

Además sobreescribir el método **toString()** para mostrar toda la información de cada elemento.

10. ActividadUT09-10: HashMap para selección de fútbol.

Iniciamos con este ejercicio un grupo de tres, que utilizarán los conjuntos clave/valor para representar mapas de datos o arrays asociativos. Estos 3 ejercicios, están desarrollados a partir de ejemplos de la página www.jarroba.com.

Concretamente, en este primer ejercicio modelaremos los jugadores de fútbol que fueron campeones del mundo con la Selección española en el mundial del 2010. Dichos jugadores son identificados a través de una clave/valor, donde la clave puede ser el número de dorsal (será un Integer), y el valor el nombre (será un String). De esta forma tendremos:

CLAVE	1	15	3	5	11	14	16	8	18	6	7
VALOR	Casillas	Ramos	Piqué	Puyol	Capdevila	Xabi Alonso	Busquets	Xavi Hernández	Pedrito	Iniesta	Villa

En este ejercicio instanciaremos nuestro mapa(**Map**) con la subclase **HashMap**, de la forma:

```
Map<Integer, String> hashMap=new HashMap<Integer, String>();
```

Recorremos el **HashMap** para ver si ordena los elementos clave/valor.

Además haremos las siguientes operaciones características de la interfaz **Map**:

```
//Operaciones a realizar:

        System.out.println("********** Trabajando con los métodos de Map **********");

        System.out.println("Mostramos el numero de elementos que tiene el hashMap: hashMap.size() = "+hashMap.size());

        System.out.println("Vemos si el hashMap esta vacio : hashMap.isEmpty() = "+ hashMap.isEmpty());

        System.out.println("Obtenemos un elemento del Map pasándole la clave 6: hashMap.get(6) = "+hashMap.get(6));

        System.out.println("Borramos un elemento del Map el 18 (porque fue sustituido): hashMap.remove(18)"); hashMap.remove(18);

        System.out.println("Vemos que pasa si queremos obtener la clave 18 que ya no existe: hashMap.get(18) = "+hashMap.get(18));

        System.out.println("Vemos si existe un elemento con la clave 18: hashMap.containsKey(18) = "+hashMap.containsKey(18));

        System.out.println("Vemos si existe un elemento con la clave 1: hashMap.containsKey(1) = "+hashMap.containsKey(1));
```

```
System.out.println("Vemos si existe el valor 'Villa' en el Map: hashMap.containsValue(\"Villa\") = "+hashMap.containsValue("Villa"));

System.out.println("Vemos si existe el valor 'Pelé' en el Map: hashMap.containsValue(\"Pelé\") = "+hashMap.containsValue("Pelé"));

System.out.println("Borramos todos los elementos del Map: hashMap.clear()");
hashMap.clear();

System.out.println("Comprobamos si lo hemos eliminado viendo su tamaño: hashMap.size() = "+hashMap.size());

System.out.println("Lo comprobamos tambien viendo si esta vacio hashMap.isEmpty() = "+hashMap.isEmpty());
```

Comprobaremos que en un **HashMap**, los elementos que se insertan en el **Map** no tendrán un orden específico.

11. ActividadUT09-11: TreeMap para selección fútbol.

Rehacer el ejercicio anterior utilizando un **TreeMap** en lugar de un **HashMap**, mediante:

Map<Integer, String> treeMap=**new** TreeMap<Integer, String>();

Haremos las mismas operaciones que antes.

Comprobaremos que con un **TreeMap**, el mapa se ordena de manera "natural" teniendo en cuenta la clave. Por ejemplo, si la clave son valores enteros, como es el caso, los ordena de menos a mayor.

12. ActividadUT09-12: LinkedHashMap para selección fútbol.

Rehacer el ejercicio anterior utilizando un **LinkedhashMap** en lugar de un **HashMap** o un **TreeMap**, mediante:

Map<Integer, String> linkedHashMap=new LinkedHashMap<Integer, String>();

Haremos las mismas operaciones que antes.

Comprobaremos que con un **LinkedHashMap**, el mapa se mantiene en el mismo orden en que fueron insertados los datos, es decir, no modifica la posición de inserción.

13. ActividadUT09-13: Interfaz Comparable y Comparator.

Crear una aplicación que tenga una clase **Alumno** que tenga atributos **idAlumno** (int), **nombre** (String) y **nota** (int entre 0 y 10). Dicha clase debe implementar la interfaz **Comparable**, debiendo sobrescribirse el método **compareTo()** para que ordene alfabéticamente por nombre. Además sobrescribiremos el método **toString()** para que muestre adecuadamente toda los atributos de un alumno.

Por otro lado, crear las siguiente clases que implementen la interfaz **Comparator**, sobrescribiéndose el método **compare()** para que ordene con respecto a un criterio específico, indicado por el propio nombre según verás a continuación:

- Clase **OrdenarAlumnoPorNombreCreciente**.
- Clase **OrdenarAlumnoPorNombreDecreciente**.
- Clase **OrdenarAlumnoPorNotaCreciente**.
- Clase **OrdenarAlumnoPorNotaDecreciente**.

Crear una clase de prueba en la cual carguemos un **ArrayList** de objetos **Alumno**, y a través de la clase **Collections**, mediante su método de ordenación **sort()**, se pueda apreciar la ordenación basada en el uso de la interfaz **Comparable** y las diferentes ordenaciones basadas en el uso de la interfaz **Comparator**.

14. ActividadUT09-14: Lista enlazada simple.

Escribe las clases necesarias para implementar una lista enlazada simple (será programada íntegramente por el alumno, sin usar librerías específicas de listas de Java). Los métodos serán: **insertarPrincipio()**, **insertarFinal()**, **buscar()**, **mostrarLista()**, **eliminarNodo()**.

15. ActividadUT09-15: Cola dinámica.

Implementa una cola utilizando una lista enlazada simple (será programada íntegramente por el alumno, sin usar librerías específicas de listas de Java). La clase debe llamarse **ColaDinamica**. Para probarlo, incorporaremos objetos de la clase **Character**, cuyos valores serán las letras del abecedario.

16. ActividadUT09-16: Pila dinámica.

Implementa una pila utilizando una lista enlazada simple (será programada íntegramente por el alumno, sin usar librerías específicas de listas de Java). La clase debe llamarse **PilaDinamica**. Para probarlo, incorporaremos objetos de la clase **Character**, cuyos valores serán las letras del abecedario.