

EJERCICIOS COLECCIONES

PROBLEMA 1 – Arrays estáticos de String

Veremos las principales propiedades de un array estático de strings:

a) Crea el array de strings llamado fruits, formado por las cadenas: Pineapple, Apple, Orange y Banana:

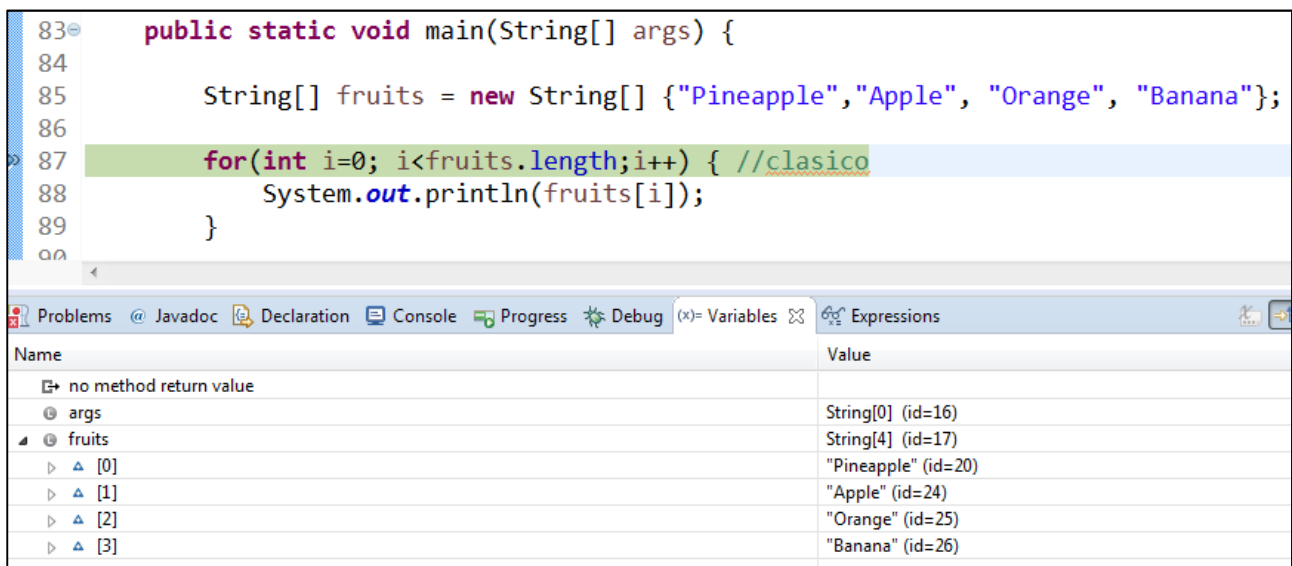
```
public static void main(String[] args) {  
    String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};
```

b) Indica dos formas de mostrar los elementos de un array de strings:

```
public static void main(String[] args) {  
    String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};  
  
    for(int i=0; i<fruits.length; i++) { //clasico  
        System.out.println(fruits[i]);  
    }  
    for(String temp: fruits){           //foreach desde java 8  
        System.out.println(temp);  
    }
```

c) ¿Cómo quedan almacenados los elementos dentro de un array estático?

Los arrays estáticos siempre se han indexado por números, donde en la posición 0 encontraremos el primer elemento, y en la posición n-1 encontraremos el último elemento, siendo n la longitud del array.



The screenshot shows a Java IDE with the following code in the editor:

```
83 public static void main(String[] args) {  
84  
85     String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};  
86  
87     for(int i=0; i<fruits.length; i++) { //clasico  
88         System.out.println(fruits[i]);  
89     }  
90 }
```

Below the code, the IDE's variable viewer is open, showing the state of the program. The 'Variables' tab is selected, displaying the following information:

Name	Value
no method return value	
args	String[0] (id=16)
fruits	String[4] (id=17)
[0]	"Pineapple" (id=20)
[1]	"Apple" (id=24)
[2]	"Orange" (id=25)
[3]	"Banana" (id=26)

En el array fruits anterior tenemos que la 1ª componente es fruits[0] = "Pineapple", la 2ª componente fruits[1] = "Apple", la 3ª componente es fruits[2] = "Orange" y la última componente es fruits[3] = "Banana".

d) ¿Existe alguna forma de eliminar o agregar un elemento de un array estático de strings o de un array en general?

No hay un método específico para ello. En los últimos años han salido librerías que resuelven esta problemática, como por ejemplo:

- ArrayUtils.remove. de las librerías Apache Commons library
- System.arraycopy(src, i + 1, newArray, i, newArray.length - i);

No obstante la idea clásica para esto siempre ha pasado por crear otra estructura de menor tamaño e ir copiando los elementos que interesen.

```
String[] fruits2 = new String[fruits.length - 1];
int indiceaBorrar = 2;
for (int i = 0, j = 0; i < fruits.length; i++) {
    if (i != indiceaBorrar) {
        fruits2[j++] = fruits[i];
    }
}
```

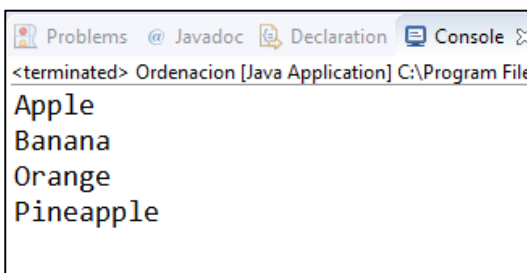
Por supuesto la mejor idea en caso de que se quieran eliminar elementos, es la de utilizar estructuras de almacenamiento dinámico, cuyos objetos se puedan borrar y agregar sin problemas, y evitar los array estáticos. Cuando se crea un array estático se hace por una determinada longitud que no se puede modificar en un futuro.

e) Ordena el array de strings alfabéticamente mediante el método estático sort de Arrays.

```
public static void main(String[] args) {

    String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};

    Arrays.sort(fruits);
    for(String temp: fruits){
        System.out.println(temp);
    }
}
```



The screenshot shows a Java IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the program: Apple, Banana, Orange, and Pineapple, which are the fruits from the array sorted alphabetically.

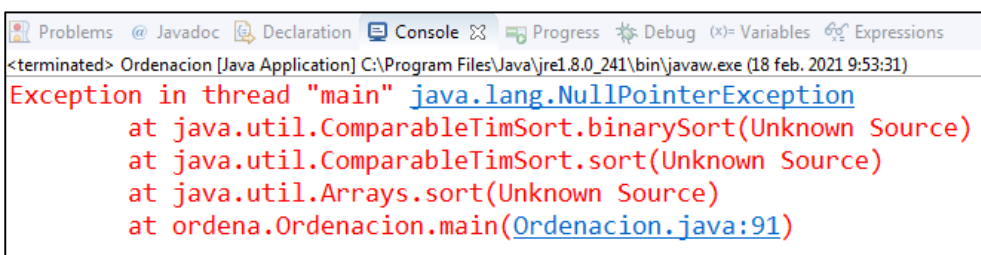
f) Define un array de 5 strings e introduce los 4 elementos anteriores:

```
public static void main(String[] args) {
    //String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};
    String[] fruits = new String[5];
    fruits[0] = "Pineapple";
    fruits[1] = "Apple";
    fruits[2] = "Orange";
    fruits[3] = "Banana";
}
```

g) Realiza la ordenación del array anterior ¿Qué ocurre?

```
public static void main(String[] args) {
    //String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};
    String[] fruits = new String[5];
    fruits[0]= "Pineapple";
    fruits[1]= "Apple";
    fruits[2]= "Orange";
    fruits[3]= "Banana";

    Arrays.sort(fruits);
    for(String temp: fruits){
        System.out.println(temp);
    }
}
```



The screenshot shows the IDE's console window with the following error message:

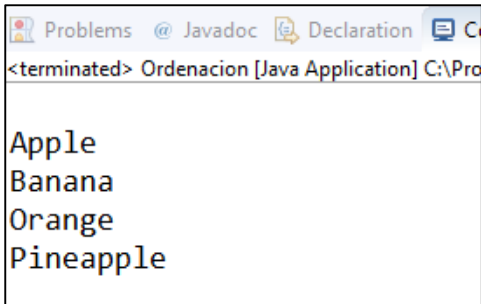
```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (18 feb. 2021 9:53:31)
Exception in thread "main" java.lang.NullPointerException
    at java.util.ComparableTimSort.binarySort(Unknown Source)
    at java.util.ComparableTimSort.sort(Unknown Source)
    at java.util.Arrays.sort(Unknown Source)
    at ordena.Ordenacion.main(Ordenacion.java:91)
```

Por defecto Java inicializa los strings de un array a null. Si se intenta ordenar un array de strings con un elemento a null, la ordenación falla.

h) ¿Cómo se puede corregir el problema?

```
public static void main(String[] args) {
    //String[] fruits = new String[] {"Pineapple", "Apple", "Orange", "Banana"};
    String[] fruits = new String[5];
    fruits[0]= "Pineapple";
    fruits[1]= "Apple";
    fruits[2]= "Orange";
    fruits[3]= "Banana";
    fruits[4]= "";

    Arrays.sort(fruits);
    for(String temp: fruits){
        System.out.println(temp);
    }
}
```



The screenshot shows the IDE's console window with the following output:

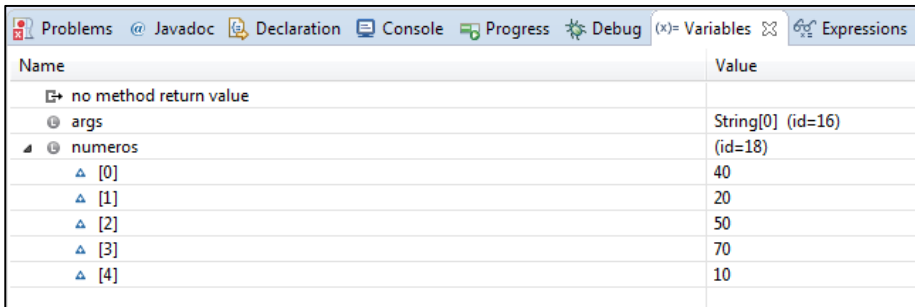
```
<terminated> Ordenacion [Java Application] C:\Pro
Apple
Banana
Orange
Pineapple
```

PROBLEMA 2 – Array estático de enteros

Veremos cómo funciona un array estático de enteros:

a) Crea un array de enteros llamado numeros, formado por 40, 20, 50, 70, 10:

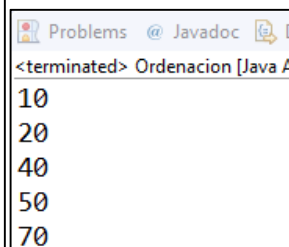
```
public static void main(String[] args) {  
    int[] numeros = new int[] {40,20,50,70,10};  
}
```



Name	Value
no method return value	
args	String[0] (id=16)
numeros	(id=18)
[0]	40
[1]	20
[2]	50
[3]	70
[4]	10

b) Ordena el array de enteros mediante el método estático sort de la clase Arrays.

```
public static void main(String[] args) {  
    int[] numeros = new int[] {40,20,50,70,10};  
  
    Arrays.sort(numeros);  
    for(int temp: numeros){  
        System.out.println(temp);  
    }  
}
```

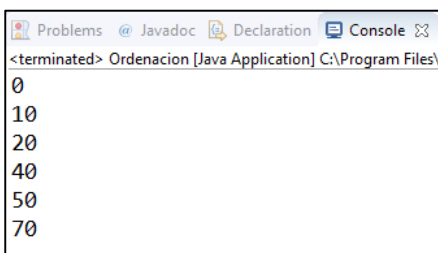


```
<terminated> Ordenacion [Java Ap  
10  
20  
40  
50  
70
```

c) Define un array estático de 6 enteros e introduce los 5 elementos anteriores:

```
public static void main(String[] args) {  
    //int[] numeros = new int[] {40,20,50,70,10};  
    int[] numeros = new int[6];  
    numeros[0]=40;  
    numeros[1]=20;  
    numeros[2]=50;  
    numeros[3]=70;  
    numeros[4]=10;  
  
    Arrays.sort(numeros);  
    for(int temp: numeros){  
        System.out.println(temp);  
    }  
}
```

d) Realiza la ordenación numérica de este array de enteros. ¿Ocurre algún error?

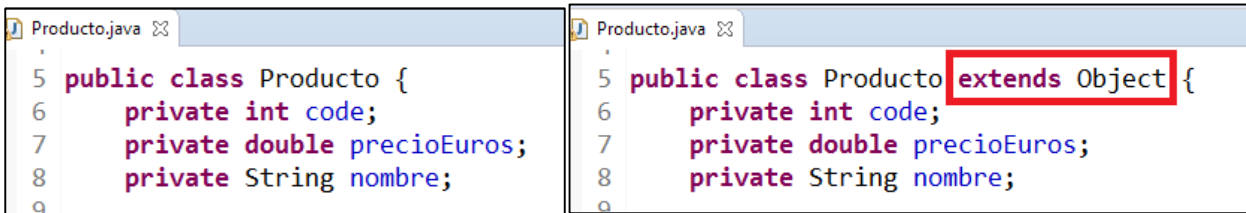


```
<terminated> Ordenacion [Java Application] C:\Program Files\  
0  
10  
20  
40  
50  
70
```

Por defecto Java inicializa los enteros de un array a 0. La ordenación de un array de enteros no fallara porque los enteros no pueden ser nulos.

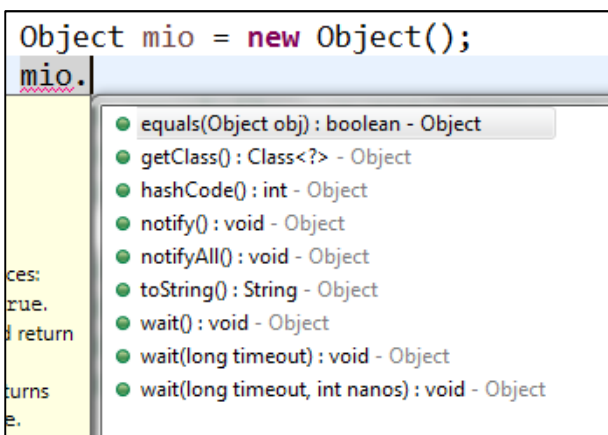
PROBLEMA 3 – Métodos del objeto Object de Java

a) ¿Que es la clase Object de Java?



La clase Object es la clase base de la cual heredan todos los objetos de java. Aunque cuando creamos una clase no se suele poner.

b) ¿Cuáles son los métodos de la clase Object?



Toda clase de Java incorpora una serie de métodos heredados de la clase Object:

- `public void notify();` → Se usan en threads
- `public void notifyAll();` → Se usan en threads
- `public void wait();` → Se usan en threads
- `public Class<?> getClass();` → Devuelve la clase a la que pertenece el objeto
- `public String toString();` → Devuelve un string explicando los elementos de la clase.
- `public boolean equals(Object o);` → La implementación por defecto de este método devuelve true si y solo si `o==this` (siendo `this` la referencia al objeto que contiene el método `equals` en ejecución). En otras palabras, devuelve true solo si `o` y `this` referencian a un único objeto (referencian la misma zona de memoria). Lo crea eclipse
- `public int hashCode();` → Este método devuelve un valor `int` que se calcula a partir de los contenidos del objeto. En el caso de los objetos `String` este entero está relacionado con la cadena de caracteres que forma dicho `String`. Si dos objetos son iguales según el método `equals()` antes mencionado, sus `hashCode()` DEBEN ser iguales. Lo contrario no tiene por qué pasar. Tener una definición consistente de estos dos métodos es fundamental a la hora de introducir objetos en conjuntos `HashSet`.

PROBLEMA 4 – Array de Objetos

a) Crea la clase Habitación, con los siguientes atributos y constructores:

```
Habitacion.java
8
9 public class Habitacion {
10     //Atributos
11     private String name;
12     private int id;
13
14     public Habitacion() {
15     }
16
17     public Habitacion(String name, int id) {
18         this.name = name;
19         this.id = id;
20     }
21 }
```

b) Crea un array estático de 5 objetos Habitación:

```
Habitacion []habitacionesArray =
    {new Habitacion("James",20),
      new Habitacion("Mary",10),
      new Habitacion("John",80),
      new Habitacion("Amanda",40),
      new Habitacion("Charles",30)};
```

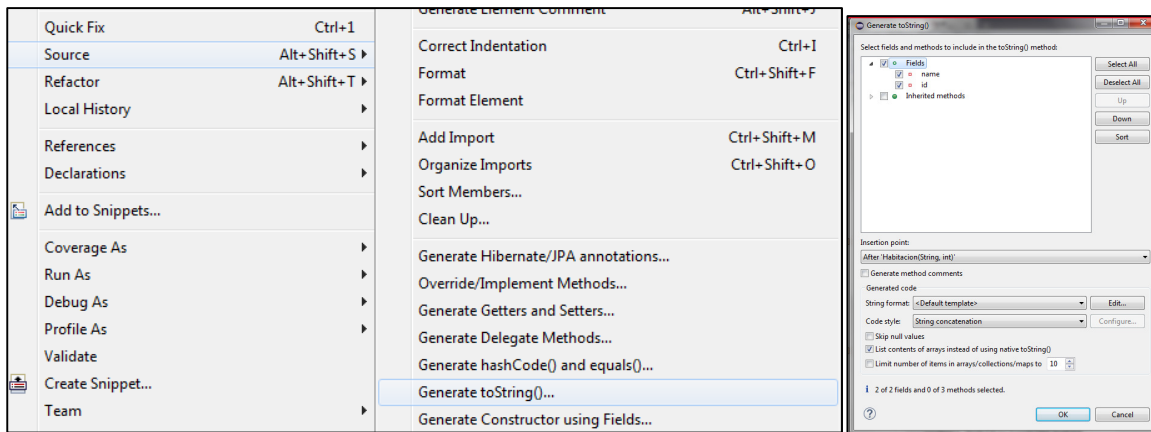
c) Imprime el array de habitaciones anterior ¿Qué ocurre? ¿Qué se imprime?

```
for(int i=0; i<habitacionesArray.length; i++){
    System.out.println(habitacionesArray[i]);
}
for(Habitacion temp: habitacionesArray){
    System.out.println(temp);
}
```

```
Problems @ Javadoc Declaration Console
<terminated> Ordenacion [Java Application] C:\Program Files\Java\
ordena.Habitacion@7852e922
ordena.Habitacion@4e25154f
ordena.Habitacion@70dea4e
ordena.Habitacion@5c647e05
ordena.Habitacion@33909752
```

Se imprime la dirección de memoria de cada objeto, puesto que no está definido el método toString en la clase Habitación.

d) Define el método toString en la clase Habitación. Utiliza el wizard de eclipse:



```

8
9 public class Habitación {
10     //Atributos
11     private String name;
12     private int id;
13
14     public Habitación() {
15     }
16
17     public Habitación(String name, int id) {
18         this.name = name;
19         this.id = id;
20     }
21
22     @Override
23     public String toString() {
24         return "Habitacion [name=" + name + ", id=" + id + "]";
25     }
26

```

e) Vuelve a mostrar el contenido del array estático de habitaciones.

```

<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8
Habitacion [name=James, id=20]
Habitacion [name=Mary, id=10]
Habitacion [name=John, id=80]
Habitacion [name=Amanda, id=40]
Habitacion [name=Charles, id=30]

```

f) Realiza la ordenación del array anterior mediante Arrays.sort. ¿Qué ocurre?

```

Habitacion []habitacionesArray =
    {new Habitación("James",20),
      new Habitación("Mary",10),
      new Habitación("John",80),
      new Habitación("Amanda",40),
      new Habitación("Charles",30)};

Arrays.sort(habitacionesArray);

for(Habitacion temp: habitacionesArray){
    System.out.println(temp);
}

```

```
Problems @ Javadoc Declaration Console Progress Debug (x) Variables Expressions
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (22 feb. 2021 12:00:26)
Exception in thread "main" java.lang.ClassCastException: ordena.Habitacion cannot be cast to java.lang.Comparable
    at java.util.ComparableTimSort.countRunAndMakeAscending(Unknown Source)
    at java.util.ComparableTimSort.sort(Unknown Source)
    at java.util.Arrays.sort(Unknown Source)
    at ordena.Ordenacion.main(Ordenacion.java:168)
```

Para ordenar un array de objetos es necesario que el objeto contenido implemente la interfaz Comparable.

g) Implementa la interfaz comparable en la clase Habitacion. Implementa el método compareTo de esta interfaz, de manera que permita ordenar por el nombre de la Habitacion:

```
Habitacion.java
9 public class Habitacion implements Comparable<Habitacion>{
10     //Atributos
11     private String name;
12     private int id;
13
14     public Habitacion() {
15     }
16
17     public Habitacion(String name, int id) {
18         this.name = name;
19         this.id = id;
20     }
21     @Override
22     public int compareTo(Habitacion o) {
23         return this.getName().compareTo(o.getName());
24     }
25
26     @Override
27     public String toString() {
28         return "Habitacion [name=" + name + ", id=" + id + "]";
29     }
30 }
```

La interfaz Comparable sirve para ordenar un array estático o dinámico de objetos. Se implementa sólo dentro de la clase y obliga a definir el método compareTo. Este método sirve tanto para definir una ordenación por atributos enteros como de tipo strings. Esta operativa obliga a definir los típicos getters and setters de la clase.

h) Prueba de ejecutar la ordenación y comprueba el resultado:

```
Habitacion []habitacionesArray =
    {new Habitacion("James",20),
    new Habitacion("Mary",10),
    new Habitacion("John",80),
    new Habitacion("Amanda",40),
    new Habitacion("Charles",30)};

Arrays.sort(habitacionesArray);

for(Habitacion temp: habitacionesArray){
    System.out.println(temp);
}
```

```
Problems @ Javadoc Declaration Console
<terminated> Ordenacion [Java Application] C:\Program Files\
Habitacion [name=Amanda, id=40]
Habitacion [name=Charles, id=30]
Habitacion [name=James, id=20]
Habitacion [name=John, id=80]
Habitacion [name=Mary, id=10]
```


i) Modifica el método `compareTo` para realizar la ordenación del array estático de habitaciones según su atributo entero `id`.

```
Habitacion.java
9 public class Habitacion implements Comparable<Habitacion>{
10     //Atributos
11     private String name;
12     private int id;
13
14     public Habitacion() {
15     }
16
17     public Habitacion(String name, int id) {
18         this.name = name;
19         this.id = id;
20     }
21
22     @Override
23     public int compareTo(Habitacion o) {
24         //return this.getName().compareTo(o.getName());
25         return this.getId()-o.getId();
26     }
27 }
```

```
Problems @ Javadoc Declaration Console
<terminated> Ordenacion [Java Application] C:\Program Files\J...
Habitacion [name=Mary, id=10]
Habitacion [name=James, id=20]
Habitacion [name=Charles, id=30]
Habitacion [name=Amanda, id=40]
Habitacion [name=John, id=80]
```

j) ¿Existe algún mecanismo que permita una ordenación puntual de un array por un criterio diferente al indicado en el método `compareTo` de la interfaz `Comparable`, sin tocar naturalmente la clase contenedora?

```
174 Arrays.sort(habitacionesArray, new Comparator<Habitacion>() {
175     @Override
176     public int compare(Habitacion first, Habitacion second) {
177         //return first.getId() - second.getId();
178         return first.getName().compareTo(second.getName());
179     }
180 });
181
182 for(Habitacion temp: habitacionesArray){
183     System.out.println(temp);
184 }
185
```

```
Problems @ Javadoc Declaration Console Progress Debug (x)= Variables Expressions
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (22 feb. 2021 12:26:16)
Habitacion [name=Amanda, id=40]
Habitacion [name=Charles, id=30]
Habitacion [name=James, id=20]
Habitacion [name=John, id=80]
Habitacion [name=Mary, id=10]
```

La interfaz `Comparator` permite definir un nuevo criterio de ordenación puntual dentro de `Arrays.sort`, sin modificar internamente la clase contenedora.

Ambas interfaces permiten ordenar por atributos enteros y/o strings. La diferencia radica en que `Comparable` es interna a la clase y `Comparator` se define externo a la clase y de forma puntual.

PROBLEMA 5 – ArrayList

La característica principal de un ArrayList respecto un array estático, es que es dinámico y por tanto permite eliminar y agregar componentes fácilmente en tiempo de ejecución. Cada vez que se agrega un objeto, crece dinámicamente en tamaño. Igual que en un array estático, los elementos de un ArrayList se indexan con números dentro de un rango [0, n-1]: el primer elemento agregado ocupa la posición 0 y el último la posición longitud-1, como en un array estático.

a) Crea un ArrayList de objetos Habitación:

```
ArrayList <Habitacion> lista = new ArrayList<Habitacion>();
```

Problems @ Javadoc Declaration Console Progress Debug (x)= Variables Expressions	
Name	Value
<init>() returned	(No explicit return value)
args	String[0] (id=22)
lista	ArrayList<E> (id=21)

b) Agrega 5 habitaciones:

```
ArrayList <Habitacion> lista = new ArrayList<Habitacion>();  
  
lista.add(new Habitacion("James",20));  
lista.add(new Habitacion("Mary",10));  
lista.add(new Habitacion("John",80));  
lista.add(new Habitacion("Charles",30));  
lista.add(new Habitacion("Amanda",40));
```

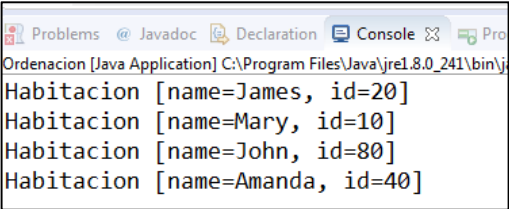
c) Indica 3 formas diferentes de mostrar un ArrayList:

```
ArrayList <Habitacion> lista = new ArrayList<Habitacion>();  
lista.add(new Habitacion("James",20));  
lista.add(new Habitacion("Mary",10));  
lista.add(new Habitacion("John",80));  
lista.add(new Habitacion("Charles",30));  
lista.add(new Habitacion("Amanda",40));  
  
for(int i=0; i<lista.size(); i++)  
    System.out.println(lista.get(i));  
  
for(Habitacion temp: lista)  
    System.out.println(temp);  
  
System.out.println(lista);
```

Problems @ Javadoc Declaration Console Progress Debug (x)= Variables Expressions	
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (25 feb. 2021 11:13:47)	
Habitacion [name=James, id=20]	
Habitacion [name=Mary, id=10]	
Habitacion [name=John, id=80]	
Habitacion [name=Charles, id=30]	
Habitacion [name=Amanda, id=40]	
Habitacion [name=James, id=20]	
Habitacion [name=Mary, id=10]	
Habitacion [name=John, id=80]	
Habitacion [name=Charles, id=30]	
Habitacion [name=Amanda, id=40]	
[Habitacion [name=James, id=20], Habitacion [name=Mary, id=10], Ha	

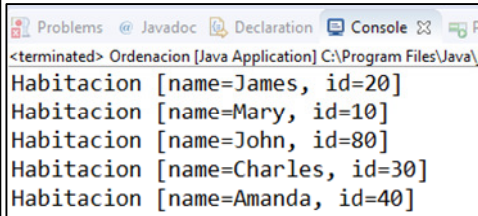
d) Borra el elemento habitación que ocupa la posición 3.

```
198 lista.add(new Habitacion("James",20));
199 lista.add(new Habitacion("Mary",10));
200 lista.add(new Habitacion("John",80));
201 lista.add(new Habitacion("Charles",30));
202 lista.add(new Habitacion("Amanda",40));
203
204 lista.remove(3);
205
206 for(Habitacion temp: lista){
207     System.out.println(temp);
208 }
209
```



e) Borra la habitación de "Charles,30" usando el método remove(object). ¿Funciona?

```
198 lista.add(new Habitacion("James",20));
199 lista.add(new Habitacion("Mary",10));
200 lista.add(new Habitacion("John",80));
201 lista.add(new Habitacion("Charles",30));
202 lista.add(new Habitacion("Amanda",40));
203
204 //lista.remove(3);
205 lista.remove(new Habitacion("Charles",30));
206
207 for(Habitacion temp: lista){
208     System.out.println(temp);
209 }
210
```



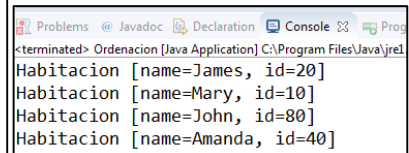
Los métodos del ArrayList que tienen como parámetro Object (remove, contains, indexOf), no funcionan si el objeto a buscar no pertenece previamente al array. Es decir, cuando intentamos borrar new Habitacion("Charles", 30), este es un nuevo objeto diferente al insertado anteriormente. Son punteros objetos diferentes, cada uno con su espacio de memoria, teniendo solo en común el estado (valor de los atributos), que es el mismo

f) Define el método equals en la clase Habitacion, usando el wizard de eclipse:

```
Habitacion.java
56 @Override
57 public boolean equals(Object obj) {
58     if (this == obj)
59         return true;
60     if (obj == null)
61         return false;
62     if (getClass() != obj.getClass())
63         return false;
64     Habitacion other = (Habitacion) obj;
65     if (id != other.id)
66         return false;
67     if (name == null) {
68         if (other.name != null)
69             return false;
70     } else if (!name.equals(other.name))
71         return false;
72     return true;
73 }
74 }
```

g) Prueba de hacer de nuevo el borrado con las siguientes condiciones:

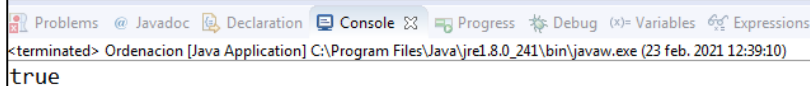
```
199 ArrayList <Habitacion> lista = new ArrayList<Habitacion>();
200 lista.add(new Habitacion("James",20));
201 lista.add(new Habitacion("Mary",10));
202 lista.add(new Habitacion("John",80));
203 lista.add(new Habitacion("Charles",30));
204 lista.add(new Habitacion("Amanda",40));
205
206 lista.remove(new Habitacion("Charles",30));
207
208 for(int i=0; i<lista.size(); i++)
209     System.out.println(lista.get(i));
210
```



Con equals definido, el método remove(object) funciona. Esto pasa también en los siguientes métodos de un ArrayList: contains(Object), indexOf (Object) y lastIndexOf(Object). La definición del método equals permite borrar y buscar objetos en un arrayList, mirando el estado del objeto, aunque su puntero no esté incluido en el array.

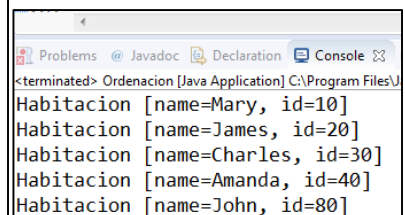
h) Realiza una prueba con el método contains, comentando y descomentando el método equals:

```
196 ArrayList <Habitacion> lista = new ArrayList<Habitacion>();
197
198 lista.add(new Habitacion("James",20));
199 lista.add(new Habitacion("Mary",10));
200 lista.add(new Habitacion("John",80));
201 lista.add(new Habitacion("Charles",30));
202 lista.add(new Habitacion("Amanda",40));
203
204 System.out.println(lista.contains(new Habitacion("Amanda",40)));
205
```



i) Ordena el arrayList inicial de habitaciones, usando en este caso el método Collections.sort:

```
196 ArrayList <Habitacion> lista = new ArrayList<Habitacion>();
197 lista.add(new Habitacion("James",20));
198 lista.add(new Habitacion("Mary",10));
199 lista.add(new Habitacion("John",80));
200 lista.add(new Habitacion("Charles",30));
201 lista.add(new Habitacion("Amanda",40));
202
203 Collections.sort(lista);
204 for(Habitacion temp: lista){
205     System.out.println(temp);
206 }
207
```



La ordenación se ha realizado correctamente debido a que la clase Habitacion tiene implementada la interfaz Comparable. En un ArrayList de objetos siguen funcionando las mismas interfaces de ordenación existentes en un array estático Comparable y Comparator. La diferencia respecto los arrays estáticos radica en que ahora se utiliza Collections.sort en lugar de Arrays.sort.

j) Realiza una ordenación puntual de lista de habitaciones, según su nombre:

```
196 ArrayList <Habitacion> lista = new ArrayList<Habitacion>();
197 lista.add(new Habitacion("James",20));
198 lista.add(new Habitacion("Mary",10));
199 lista.add(new Habitacion("John",80));
200 lista.add(new Habitacion("Charles",30));
201 lista.add(new Habitacion("Amanda",40));
202
203 Collections.sort(lista, new Comparator<Habitacion>() {
204     @Override
205     public int compare(Habitacion first, Habitacion second) {
206         //return first.getId() - second.getId();
207         return first.getName().compareTo(second.getName());
208     }
209 });
210 for(Habitacion temp: lista)
211     System.out.println(temp);
```

<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (23 feb. 2021 12:47:06)

Habitacion [name=Amanda, id=40]
Habitacion [name=Charles, id=30]
Habitacion [name=James, id=20]
Habitacion [name=John, id=80]
Habitacion [name=Mary, id=10]

La interfaz Comparator sirve para realizar una ordenación puntual en un array dinámico, sin tocar la clase contenedora

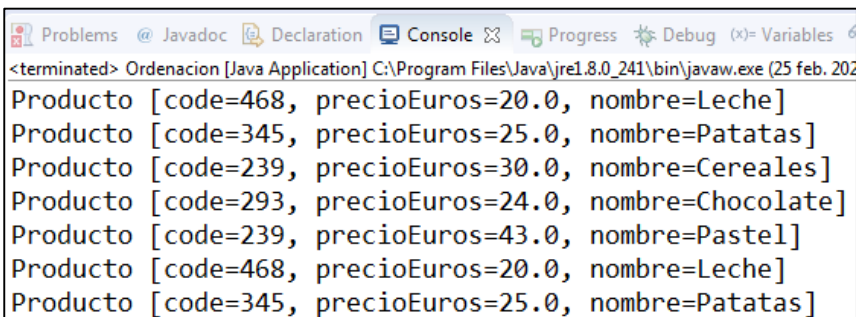
PROBLEMA 6 – Colección HashSet

a) Crea la clase Producto con los siguientes 3 atributos, 2 constructores y la función toString:

```
Producto.java
5 public class Producto {
6     private int code;
7     private double precioEuros;
8     private String nombre;
9
10    //Constructor por defecto
11    public Producto () {
12    }
13
14    //Constructor full 3 parametros
15    public Producto (int code, double precioEuros, String nombre) {
16        this.code = code;
17        this.precioEuros = precioEuros;
18        this.nombre = nombre;
19    }
20
21    @Override
22    public String toString() {
23        return "Producto [code=" + code + ", precioEuros=" +
24            precioEuros + ", nombre=" + nombre + "]";
25    }
26 }
```


b) Crea un ArrayList de Productos e inserta 7 productos, repitiendo 2. Por ejemplo introduce los productos Leche, Patatas, Cereales, Chocolate y Pastel, repitiendo por ejemplo los productos Leche y Patatas. Imprime el ArrayList e indica cuantos elementos se ven.

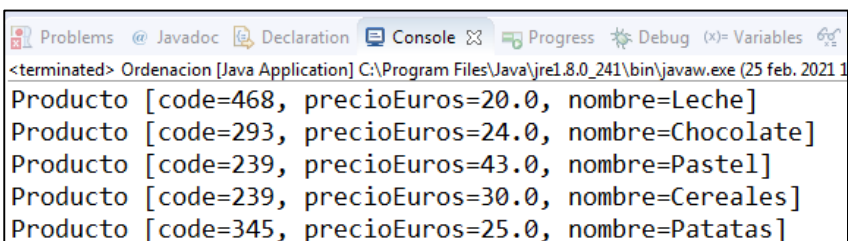
```
List<Producto>productos = new ArrayList <Producto>();
Producto p1 = new Producto(468, 20, "Leche");
productos.add(p1);
Producto p2 = new Producto(345, 25, "Patatas");
productos.add(p2);
productos.add(new Producto(239, 30, "Cereales"));
productos.add(new Producto(293, 24, "Chocolate"));
productos.add(new Producto(239, 43, "Pastel"));
productos.add(p1);
productos.add(p2);
for(Producto temp: productos){
    System.out.println(temp);
}
```



```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (25 feb. 2021)
Producto [code=468, precioEuros=20.0, nombre=Leche]
Producto [code=345, precioEuros=25.0, nombre=Patatas]
Producto [code=239, precioEuros=30.0, nombre=Cereales]
Producto [code=293, precioEuros=24.0, nombre=Chocolate]
Producto [code=239, precioEuros=43.0, nombre=Pastel]
Producto [code=468, precioEuros=20.0, nombre=Leche]
Producto [code=345, precioEuros=25.0, nombre=Patatas]
```

c) Crea una HashSet de Productos e inserta los mismos productos del ArrayList. Imprime la HashSet e indica cuantos elementos se ven.

```
Set <Producto> productosh = new HashSet <Producto> ();
Producto p1 = new Producto(468, 20, "Leche");
productosh.add(p1);
Producto p2 = new Producto(345, 25, "Patatas");
productosh.add(p2);
productosh.add(new Producto(239, 30, "Cereales"));
productosh.add(new Producto(293, 24, "Chocolate"));
productosh.add(new Producto(239, 43, "Pastel"));
productosh.add(p1);
productosh.add(p2);
for(Producto temp: productosh){
    System.out.println(temp);
}
```

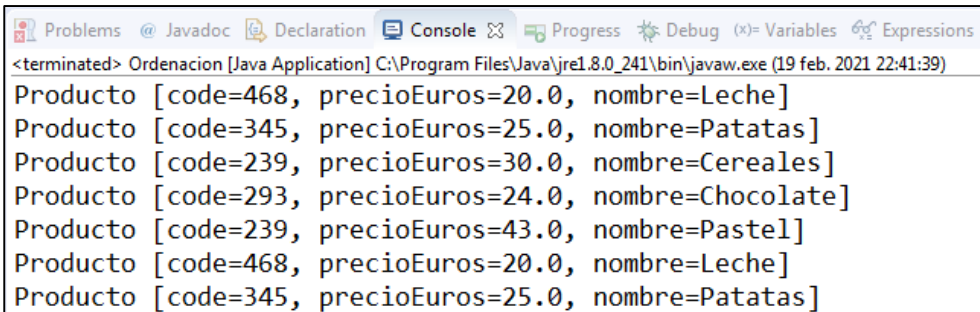


```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (25 feb. 2021)
Producto [code=468, precioEuros=20.0, nombre=Leche]
Producto [code=293, precioEuros=24.0, nombre=Chocolate]
Producto [code=239, precioEuros=43.0, nombre=Pastel]
Producto [code=239, precioEuros=30.0, nombre=Cereales]
Producto [code=345, precioEuros=25.0, nombre=Patatas]
```

La hashSet no permite insertar objetos repetidos.

d) Inserta en una HashSet de Productos los mismo productos . Imprime la HashSet e indica cuantos elementos se ven.

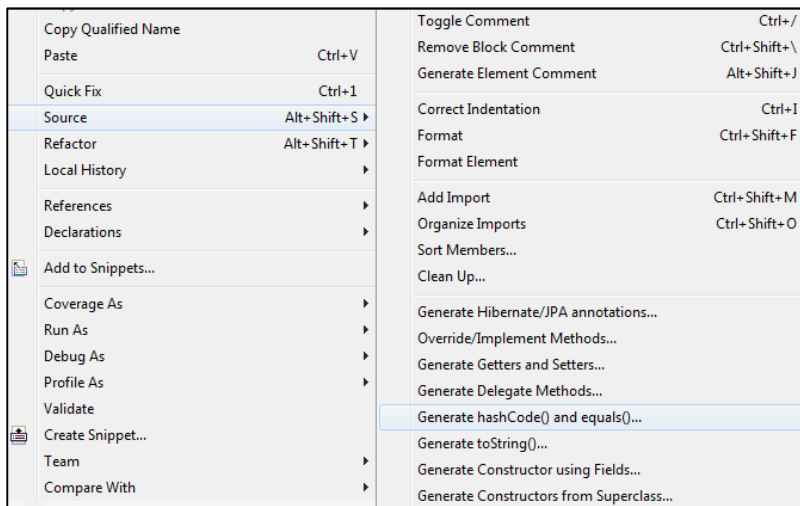
```
Set <Producto> productosh = new HashSet <Producto> ();
productosh.add(new Producto(468, 20, "Leche"));
productosh.add(new Producto(345, 25, "Patatas"));
productosh.add(new Producto(239, 30, "Cereales"));
productosh.add(new Producto(293, 24, "Chocolate"));
productosh.add(new Producto(239, 43, "Pastel"));
productosh.add(new Producto(468, 20, "Leche"));
productosh.add(new Producto(345, 25, "Patatas"));
for(Producto temp: productosh){
    System.out.println(temp);
}
```



```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (19 feb. 2021 22:41:39)
Producto [code=468, precioEuros=20.0, nombre=Leche]
Producto [code=345, precioEuros=25.0, nombre=Patatas]
Producto [code=239, precioEuros=30.0, nombre=Cereales]
Producto [code=293, precioEuros=24.0, nombre=Chocolate]
Producto [code=239, precioEuros=43.0, nombre=Pastel]
Producto [code=468, precioEuros=20.0, nombre=Leche]
Producto [code=345, precioEuros=25.0, nombre=Patatas]
```

La HashSet no permite diferenciar objetos diferentes con el mismo estado. Todos los objetos creados con new son independientes.

e) Define los métodos hashCode y equals en la clase Producto según el wizard de eclipse:



```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + code;
    result = prime * result + ((nombre == null) ? 0 : nombre.hashCode());
    long temp;
    temp = Double.doubleToLongBits(precioEuros);
    result = prime * result + (int) (temp ^ (temp >>> 32));
    return result;
}
```

```

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Producto other = (Producto) obj;
    if (code != other.code)
        return false;
    if (nombre == null) {
        if (other.nombre != null)
            return false;
    } else if (!nombre.equals(other.nombre))
        return false;
    if (Double.doubleToLongBits(precioEuros) !=
        Double.doubleToLongBits(other.precioEuros))
        return false;
    return true;
}

```

e) Vuelve a visualizar la lista de la hashSet. ¿Qué ocurre?

```

240 Set <Producto> productosh = new HashSet <Producto> ();
241 productosh.add(new Producto(468, 20, "Leche"));
242 productosh.add(new Producto(345, 25, "Patatas"));
243 productosh.add(new Producto(239, 30, "Cereales"));
244 productosh.add(new Producto(293, 24, "Chocolate"));
245 productosh.add(new Producto(239, 43, "Pastel"));
246 productosh.add(new Producto(468, 20, "Leche"));
247 productosh.add(new Producto(345, 25, "Patatas"));
248 for(Producto temp: productosh){
249     System.out.println(temp);
250 }

```

Problems @ Javadoc Declaration Console Progress Debug (x)= Variables Expressions

<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (23 feb. 2021 12:56:07)

```

Producto [code=468, precioEuros=20.0, nombre=Leche]
Producto [code=345, precioEuros=25.0, nombre=Patatas]
Producto [code=239, precioEuros=43.0, nombre=Pastel]
Producto [code=239, precioEuros=30.0, nombre=Cereales]
Producto [code=293, precioEuros=24.0, nombre=Chocolate]

```

Una HashSet de objetos no admite repetición de objetos, que según sus métodos equals(Object o) y hashCode() son iguales. El método add() de HashSet() hace uso de los métodos hashCode() y equals(Object o) de la clase Producto.

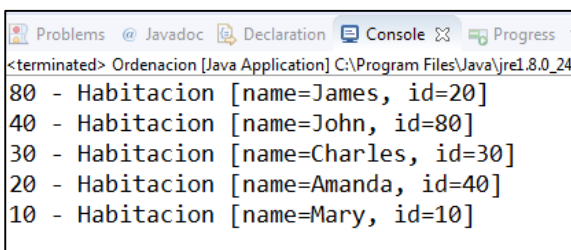
PROBLEMA 7 – Colección HashMap

La colección HashMap se utiliza para almacenar pares de datos de la forma {clave, valor}. Se suele denominar como HashMap <clave, valor> o HashMap <K, V>. La novedad respecto un array o un arrayList es que la información ya no está obligatoriamente indexada por números enteros. Está indexada por claves de datos únicas, que puede ser de distinto tipo (números, cadenas de texto, etc.). Pero al igual que el ArrayList es una estructura dinámica que va creciendo a medida que se van insertando elementos mediante el método put.

a) Crea una `HashMap<String, Habitacion>` indexada por claves de tipo `String` y valores de tipo `Habitacion`. Agrega 5 habitaciones y muéstralas junto con sus claves. ¿Qué se observa?

```
HashMap<String, Habitacion> habitacionesHash = new HashMap<String, Habitacion>();
habitacionesHash.put("80",new Habitacion("James",20));
habitacionesHash.put("10",new Habitacion("Mary",10));
habitacionesHash.put("40",new Habitacion("John",80));
habitacionesHash.put("20",new Habitacion("Amanda",40));
habitacionesHash.put("30",new Habitacion("Charles",30));

for(Map.Entry<String, Habitacion> entry : habitacionesHash.entrySet()) {
    String key = entry.getKey();
    Habitacion value = entry.getValue();
    System.out.println(key + " - "+value);
}
```



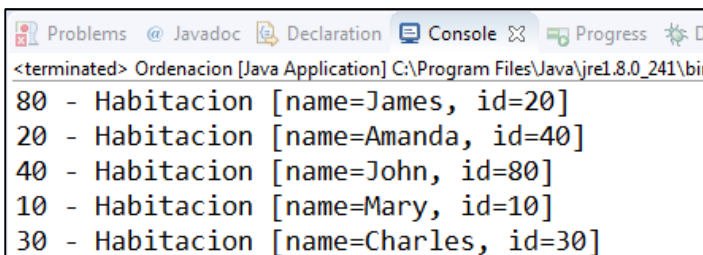
```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_24
80 - Habitacion [name=James, id=20]
40 - Habitacion [name=John, id=80]
30 - Habitacion [name=Charles, id=30]
20 - Habitacion [name=Amanda, id=40]
10 - Habitacion [name=Mary, id=10]
```

HashMap por defecto realiza una ordenación de sus elementos por su clave alfanumérica en orden descendiente

b) Crea una `HashMap<Integer, Habitacion>` indexada por claves de tipo `Integer` y valores de tipo `Habitacion`. Agrega 5 habitaciones y muéstralas junto con sus claves. ¿Qué se observa?

```
HashMap<Integer, Habitacion> habHash = new HashMap<Integer, Habitacion>();
habHash.put(80,new Habitacion("James",20));
habHash.put(10,new Habitacion("Mary",10));
habHash.put(40,new Habitacion("John",80));
habHash.put(20,new Habitacion("Amanda",40));
habHash.put(30,new Habitacion("Charles",30));

for(Map.Entry<Integer, Habitacion> entry : habHash.entrySet()) {
    Integer key = entry.getKey();
    Habitacion value = entry.getValue();
    System.out.println(key + " - "+value);
}
```



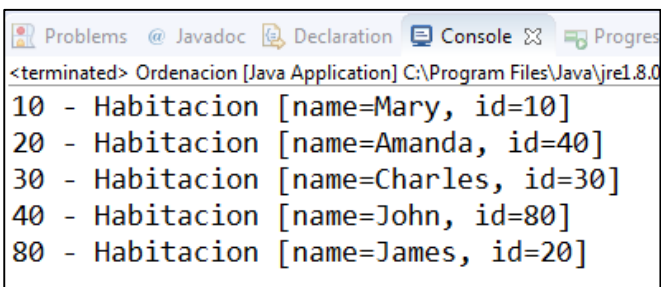
```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_241\bin
80 - Habitacion [name=James, id=20]
40 - Habitacion [name=John, id=80]
30 - Habitacion [name=Charles, id=30]
20 - Habitacion [name=Amanda, id=40]
10 - Habitacion [name=Mary, id=10]
```

HashMap por defecto realiza una ordenación de sus elementos por su clave numérica en orden descendiente

c) Realiza una ordenación de una hashMap por su clave alfanumérica en orden creciente. Usa el método que ordena las claves de forma separada y después muestra la HashMap siguiendo ese orden:

```
HashMap<String, Habitacion> habitacionesHash = new HashMap<String, Habitacion>();
habitacionesHash.put("80", new Habitacion("James", 20));
habitacionesHash.put("10", new Habitacion("Mary", 10));
habitacionesHash.put("40", new Habitacion("John", 80));
habitacionesHash.put("20", new Habitacion("Amanda", 40));
habitacionesHash.put("30", new Habitacion("Charles", 30));

SortedSet<String> keys = new TreeSet<>(habitacionesHash.keySet());
for (String key : keys) {
    Habitacion value = habitacionesHash.get(key);
    System.out.println(key + " - " + value);
}
```

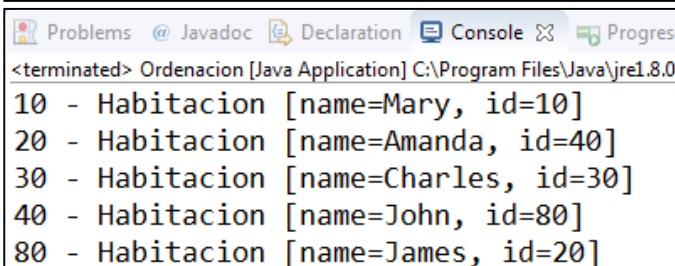


```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0
10 - Habitacion [name=Mary, id=10]
20 - Habitacion [name=Amanda, id=40]
30 - Habitacion [name=Charles, id=30]
40 - Habitacion [name=John, id=80]
80 - Habitacion [name=James, id=20]
```

d) Realiza una ordenación de una hashMap por su clave alfanumérica en orden creciente. En este caso usa la clase TreeMap:

```
HashMap<String, Habitacion> habitacionesHash = new HashMap<String, Habitacion>();
habitacionesHash.put("80", new Habitacion("James", 20));
habitacionesHash.put("10", new Habitacion("Mary", 10));
habitacionesHash.put("40", new Habitacion("John", 80));
habitacionesHash.put("20", new Habitacion("Amanda", 40));
habitacionesHash.put("30", new Habitacion("Charles", 30));

TreeMap treeMap = new TreeMap<String, Habitacion>(habitacionesHash);
Iterator it = treeMap.entrySet().iterator();
while(it.hasNext()){
    Map.Entry entry = (Map.Entry) it.next();
    String key = (String) entry.getKey();
    Habitacion value = (Habitacion) entry.getValue();
    System.out.println(key + " - " + value);
}
```



```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0
10 - Habitacion [name=Mary, id=10]
20 - Habitacion [name=Amanda, id=40]
30 - Habitacion [name=Charles, id=30]
40 - Habitacion [name=John, id=80]
80 - Habitacion [name=James, id=20]
```

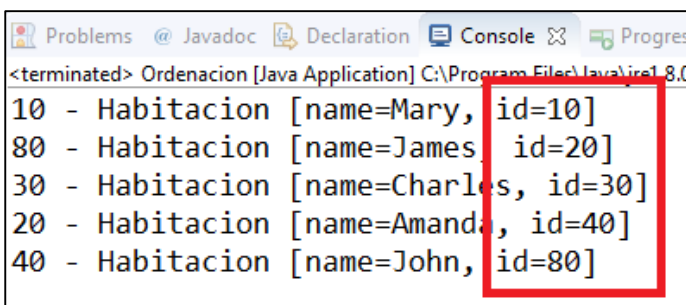
e) Realiza una ordenación de una HashMap por el atributo entero de Habitación en orden creciente.

```
HashMap<String, Habitacion> habitacionesHash = new HashMap<String, Habitacion>();
habitacionesHash.put("80", new Habitacion("James", 20));
habitacionesHash.put("10", new Habitacion("Mary", 10));
habitacionesHash.put("40", new Habitacion("John", 80));
habitacionesHash.put("20", new Habitacion("Amanda", 40));
habitacionesHash.put("30", new Habitacion("Charles", 30));

habitacionesHash = sortedMapByInteger(habitacionesHash);

for (Map.Entry<String, Habitacion> entry : habitacionesHash.entrySet()) {
    String key = entry.getKey();
    Habitacion value = entry.getValue();
    System.out.println(key + " - " + value);
}
```

```
private static HashMap<String, Habitacion> sortedMapByInteger(HashMap<String, Habitacion> unsorted) {
    List<Entry<String, Habitacion>> list = new LinkedList<Entry<String, Habitacion>>(unsorted.entrySet());
    Collections.sort(list, new Comparator<Entry<String, Habitacion>>() {
        @Override
        public int compare(Entry<String, Habitacion> t, Entry<String, Habitacion> t1) {
            return t.getValue().getId() - t1.getValue().getId();
        }
    });
    HashMap<String, Habitacion> sorted = new LinkedHashMap<String, Habitacion>();
    for (Entry<String, Habitacion> item : list) {
        sorted.put(item.getKey(), item.getValue());
    }
    return sorted;
}
```



```
<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\java.exe
10 - Habitacion [name=Mary, id=10]
80 - Habitacion [name=James, id=20]
30 - Habitacion [name=Charles, id=30]
20 - Habitacion [name=Amanda, id=40]
40 - Habitacion [name=John, id=80]
```

f) Realiza una ordenación de una HashMap por el atributo String de Habitación en orden creciente.

```

HashMap<String, Habitacion> habitacionesHash = new HashMap<String, Habitacion>();
habitacionesHash.put("80",new Habitacion("James",20));
habitacionesHash.put("10",new Habitacion("Mary",10));
habitacionesHash.put("40",new Habitacion("John",80));
habitacionesHash.put("20",new Habitacion("Amanda",40));
habitacionesHash.put("30",new Habitacion("Charles",30));

habitacionesHash=sortedMapByString(habitacionesHash);

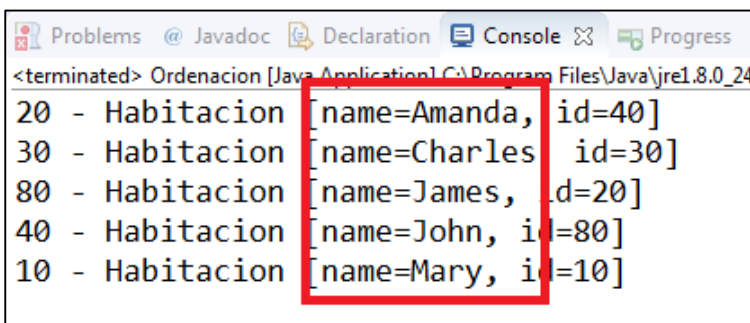
for(Map.Entry<String, Habitacion> entry : habitacionesHash.entrySet()) {
    String key = entry.getKey();
    Habitacion value = entry.getValue();
    System.out.println(key + " - " +value);
}

```

```

private static HashMap<String, Habitacion> sortedMapByString( HashMap<String, Habitacion> unsorted){
    List<Entry<String, Habitacion>> list = new LinkedList<Entry<String, Habitacion>>(unsorted.entrySet());
    Collections.sort(list, new Comparator<Entry<String, Habitacion>>(){
        @Override
        public int compare(Entry<String, Habitacion> t, Entry<String, Habitacion> t1) {
            return t.getValue().getName().compareTo(t1.getValue().getName());
        }
    });
    HashMap<String, Habitacion> sorted = new LinkedHashMap<String, Habitacion>();
    for (Entry<String, Habitacion> item:list){
        sorted.put(item.getKey(), item.getValue());
    }
    return sorted;
}

```



```

<terminated> Ordenacion [Java Application] C:\Program Files\Java\jre1.8.0_24
20 - Habitacion [name=Amanda, id=40]
30 - Habitacion [name=Charles, id=30]
80 - Habitacion [name=James, id=20]
40 - Habitacion [name=John, id=80]
10 - Habitacion [name=Mary, id=10]

```

PROBLEMA 8 – Trabajando con HashMap

Diseñaremos una aplicación que permita introducción de menús del día y la posterior selección por parte de un cliente usando HashMap y ArrayList.

a) Crea la clase Menu:

```

public class Menu {
    private String descripcion;
    private double precio;

    @Override
    public String toString() {
        return "Menu [descripcion=" + descripcion + ", precio=" + precio + "]";
    }

    public Menu(String descripcion, double precio) {
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
}

```

b) Construye un sistema de opciones que permita realizar 3 funcionalidades:

- Introducción de menús
- Exposición de menús
- Elección de menus

```

public static void main(String[] args) {
    boolean fin = false;
    while(!fin) {
        System.out.println("1. Introduce los menus del dia");
        System.out.println("2. Muestra menus del dia");
        System.out.println("3. Elige menu");
        System.out.println("4. Salir");
        System.out.print("Introduce tu opcion: ");
        int opcion = reader.nextInt();
        switch (opcion) {
            case 1:
                introduce_menu();
                break;
            case 2:
                muestra_menu();
                break;
            case 3:
                cliente_elige_menu();
                break;
            case 4:
                fin=true;
                break;
        }
    }
}

```

c) Crea la función `introduce_menu`, de manera que se puedan introducir 5 menus, en una `HashMap`, preguntando el plato y su precio.

```
public static void introduce_menu() {
    for(int i=1; i<=5;i++) {
        System.out.print("Introduce plato menu " + i + ": ");
        String descripcion = reader.next();
        System.out.print("Introduce precio menu " + i + ": ");
        double precio = reader.nextDouble();
        menuG.put(i,new Menu(descripcion,precio));
    }
}
```

d) Crea la función `muestra_menu`, de manera que recorra y muestre los menus de la `HashMap` junto con sus claves.

```
public static void muestra_menu() {
    for(Map.Entry<Integer, Menu> entry : menuG.entrySet()) {
        Integer key = entry.getKey();
        Menu value = entry.getValue();
        System.out.println(key + " - " + value);
    }
}
```

e) Crea la función `cliente_elige_menu`. La aplicación debe de mostrar los 5 menus indexados por su clave, y mediante un bucle debe de dar la oportunidad de seleccionarlos e introducirlos en un `ArrayList`. Una vez finalizada la selección debe de mostrar el precio de dicha selección:

```
public static void cliente_elige_menu() {
    ArrayList<Menu>menuC =new ArrayList<Menu>();
    int opcion=-1;
    muestra_menu();
    System.out.print("Introduce menu (fin 0):");
    opcion = reader.nextInt();
    while (opcion!=0){
        Menu m = menuG.get(opcion);
        menuC.add(m);
        System.out.print("Introduce menu (fin 0):");
        opcion = reader.nextInt();
    }
    double precio=0.0;
    for(Menu m: menuC) {
        precio+=m.getPrecio();
    }
    System.out.println("Ha elegido un menu de " + precio);
}
```