Curso de  
**Java y java.util: Colecciones, Wrappers y Lambda expressions**

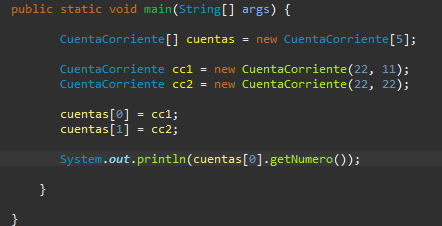
**Conociendo Arrays:**

Conociendo Arrays:



Operaciones:

Objetos:



Lo que Aprendimos:

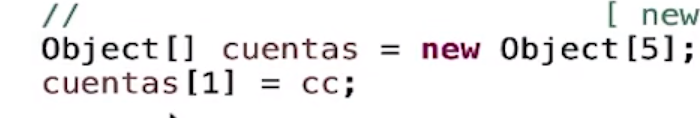
En esta clase sobre *arrays* aprendimos:

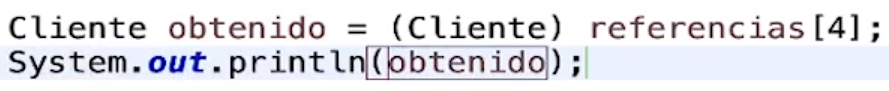
* Un array es una estructura de datos y se usa para almacenar elementos (valores primitivos o referencias)
* Los arrays usan corchetes ([]) sintácticamente
* ¡Los arrays tienen un tamaño fijo!
* ¡Un array también es un objeto!
* Los arrays son *zero-based*(el primer elemento se encuentra en la posición 0)
* Un array siempre se inicializa con los valores padron.
* Al acceder a una posición no válida recibimos la excepción ArrayIndexOutOfBoundException
* Las matrices tienen un atributo *length* para conocer el tamaño
* La forma literal de crear un array, utilizando llaves {}.

**Guardando Referencias:**

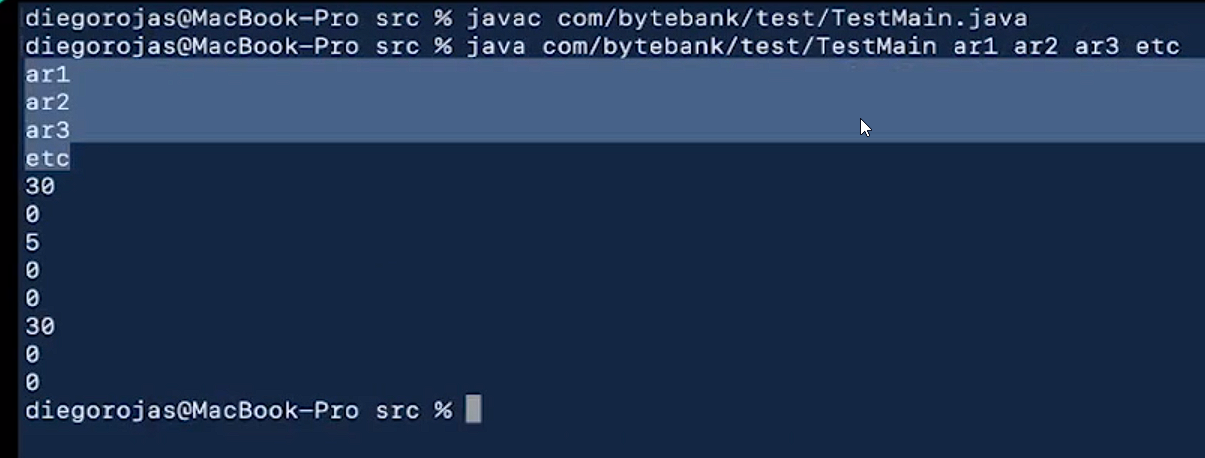
Cast Objeto #1:

Cast Objeto #2:





Método Main:



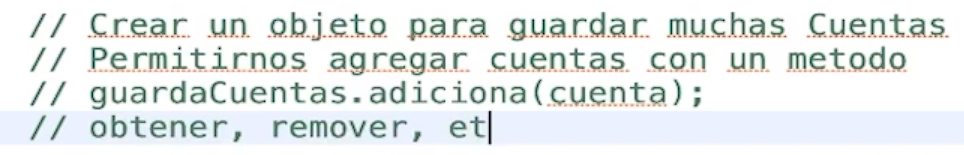
Lo que Aprendimos:

En esta clase aprendimos:

* Un array de tipo *Object* puede contener cualquier tipo de referencia.
* Cuando convertimos una referencia genérica a una referencia más específica, necesitamos usar un *type cast*.
* El *cast* solo compila cuando es posible, aún así puede fallar al ejecutarse.
* Cuando falla el *type cast*, podemos recibir un *ClassCastException*.
* Para recibir valores al llamar al programa Java en la línea de comando, podemos usar la matriz *String[]* en el método *main*.

**ArrayList y Generics:**

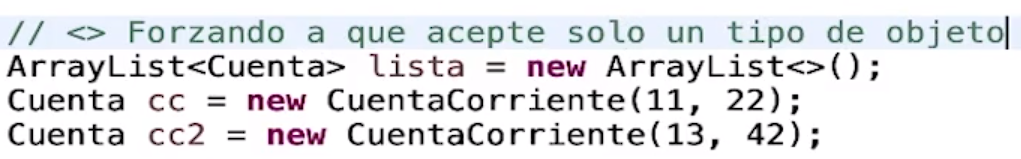
Operaciones Array:



Guarda Cuentas:

Guarda Referencias:

ArrayList:



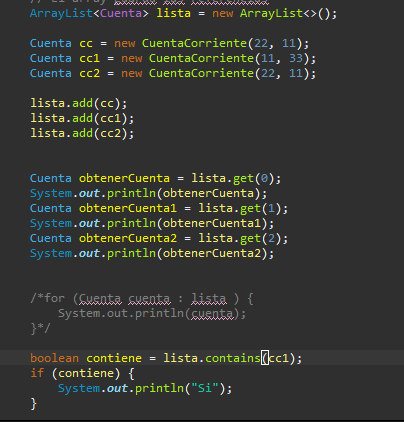
Lo que Aprendimos:

En esta clase comenzamos a hablar sobre la lista y llegamos a conocer la clase *java.util.ArrayList*. Aprendimos:

* Que la clase *java.util.ArrayList* encapsula el uso de array y ofrece varios métodos de más alto nivel.
* Que una lista guarda referencias.
* Cómo usar los métodos *size*, *get*, *remove*.
* Cómo usar *foreach* para iterar a través de *ArrayList*.
* Que los generics parametrizan clases
* Que en el caso de *ArrayList* podemos definir el tipo de los elementos mediante generics.

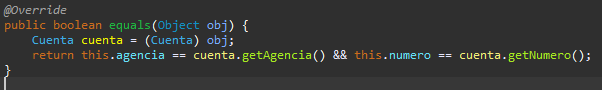
**Equals y más listas:**

Comparando Elementos:

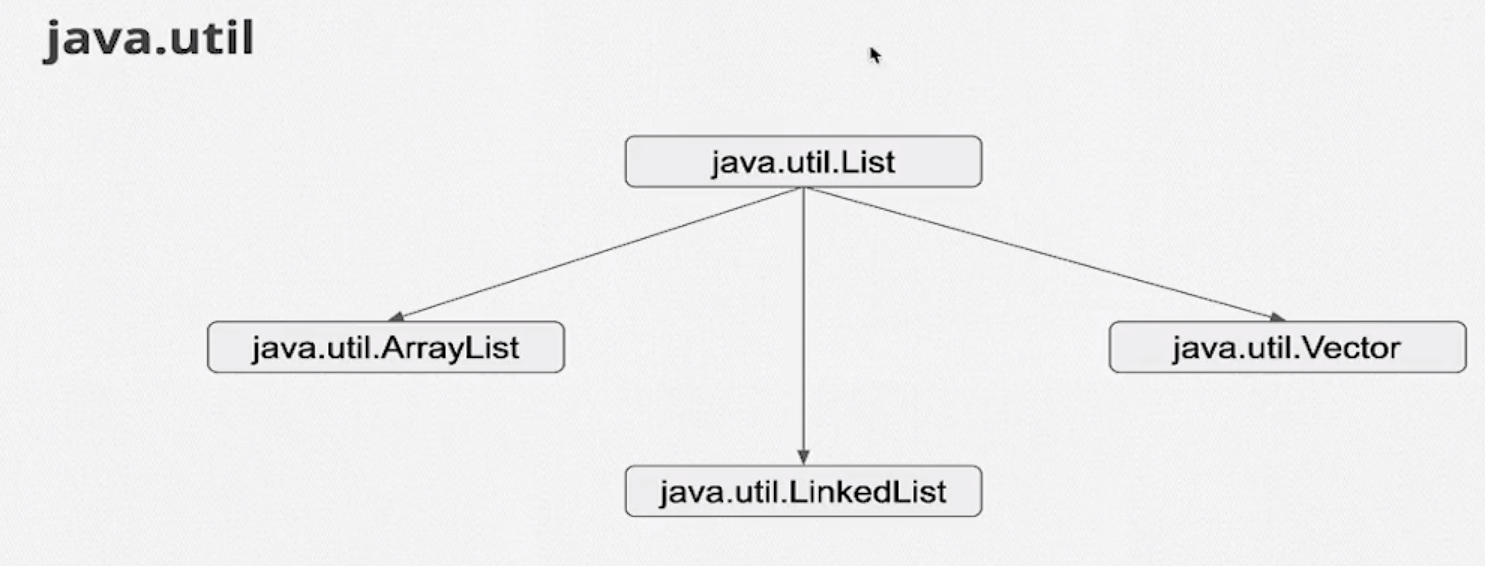


Contains compara con la referencia.

Sobrescribiendo Equals:



Listas Anexadas:



Lo que Aprendimos:

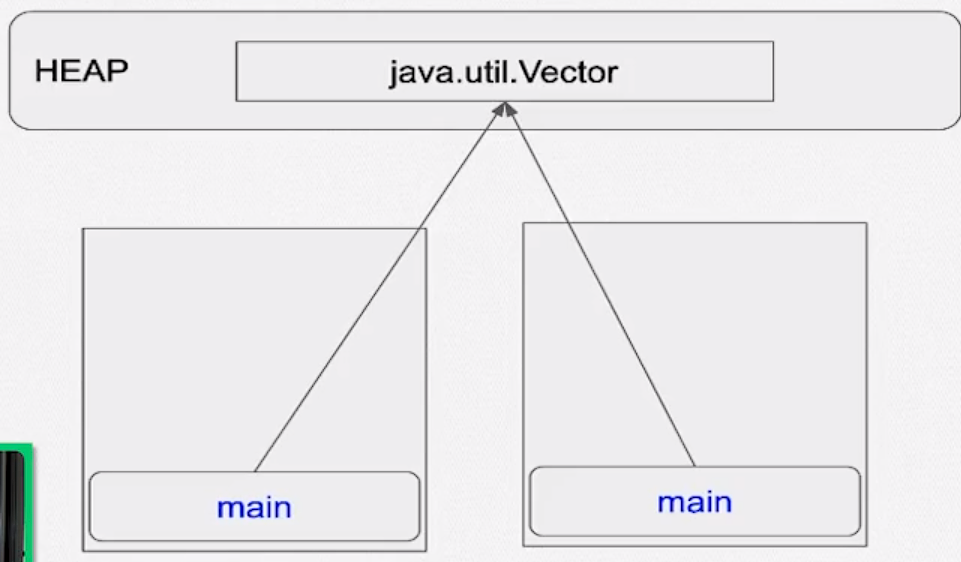
En esta clase aprendemos:

* Cómo implementar el método *equals* para definir la igualdad.
* Que el método *equals* es usado por las listas.
* Que hay más de una lista, *java.util.LinkedList*
* La diferencia entre *ArrayList* y *LinkedList*
* La interfaz *java.util.List* que define los métodos de la lista

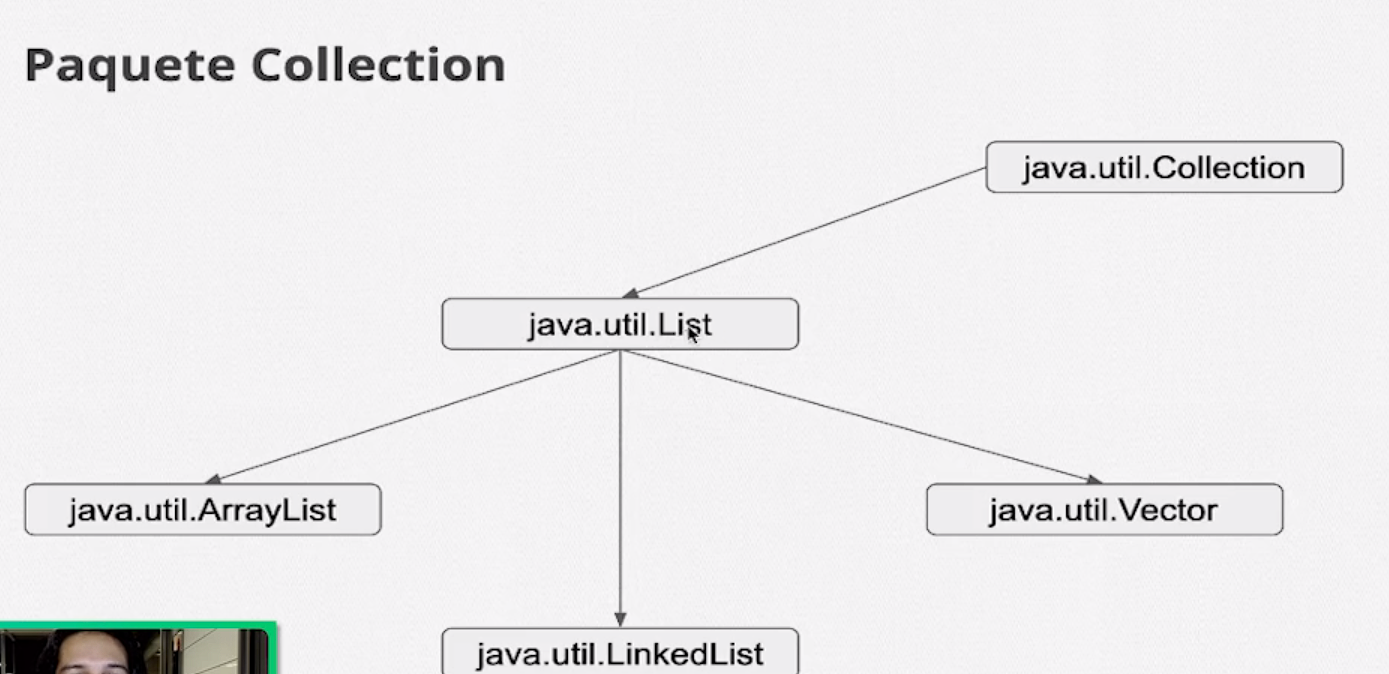
**Vector e Interfaz Collection:**

Vectores:





Superclase Collection:



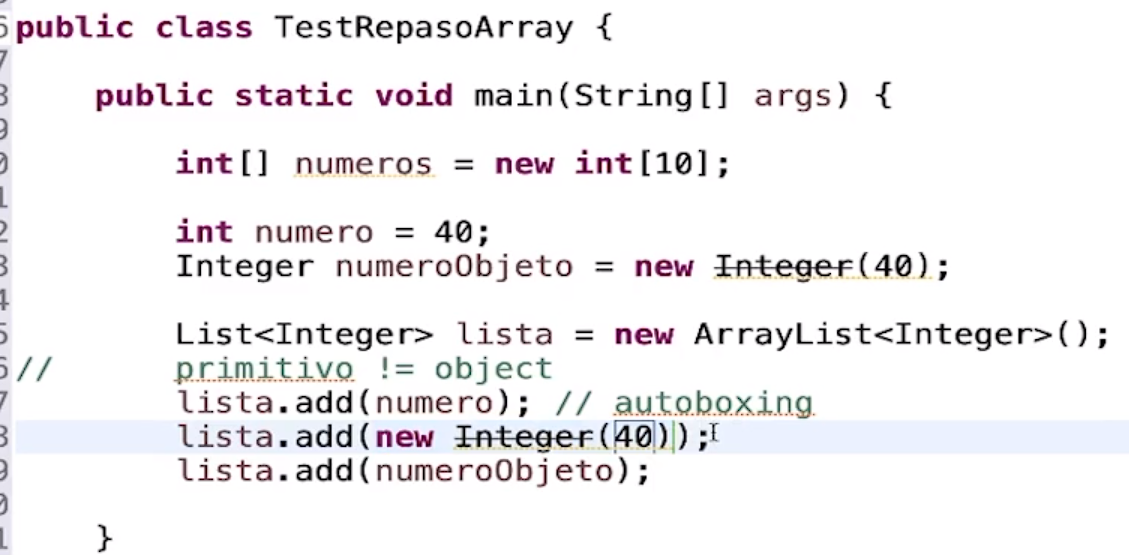
Lo que Aprendimos:

En esta clase vimos:

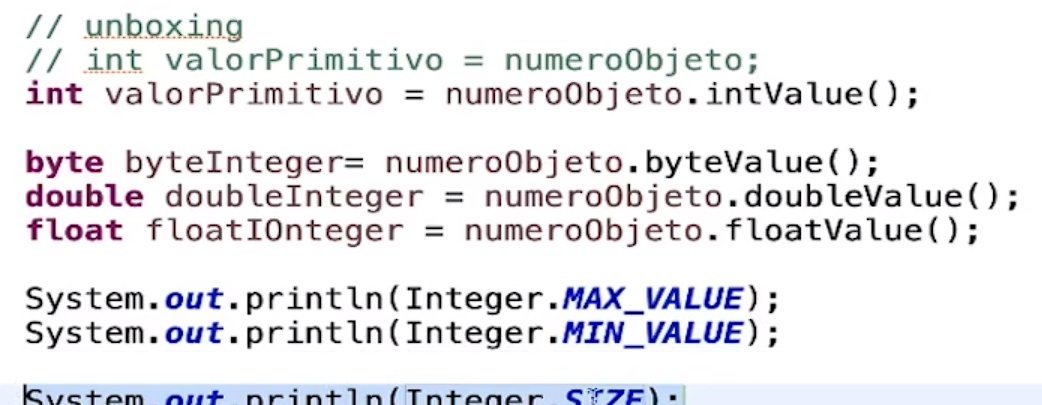
* *java.util.Vector*, que es un *ArrayList* threadsafe.
* La interfaz *java.util.Collection* que es la interfaz de todas las colecciones.
* Las listas son secuencias que aceptan elementos duplicados.
* Los conjuntos (*java.util.Set*) también son colecciones, pero no aceptan duplicados ni listas.

**Las clases Wrappers:**

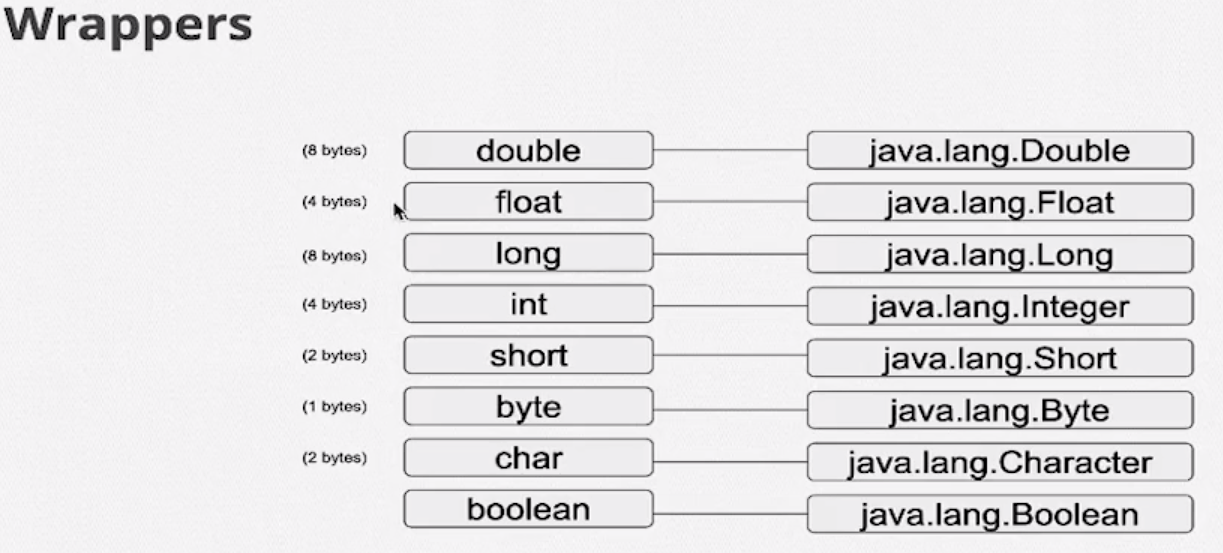
Integer:

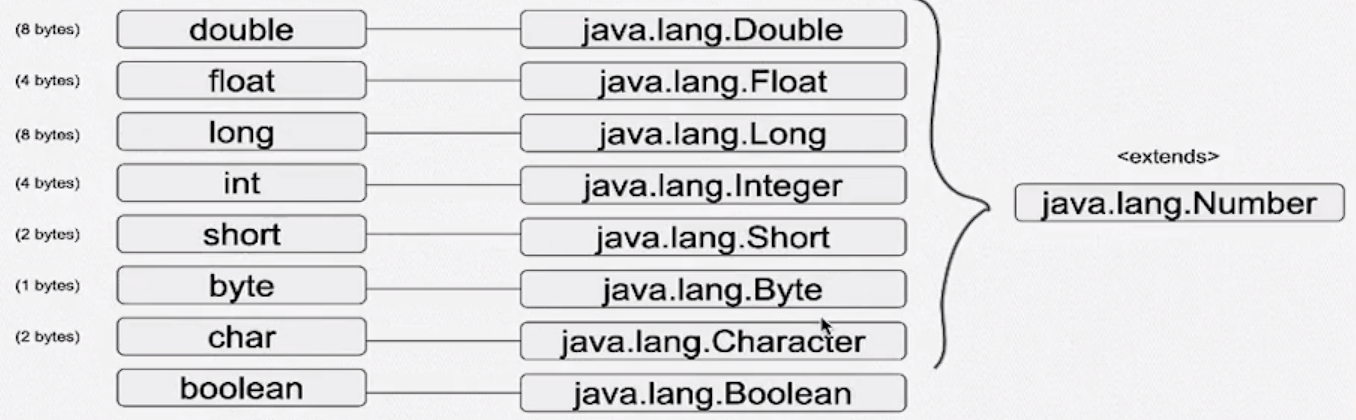


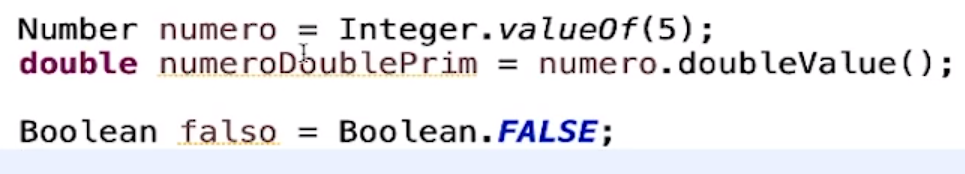
Autoboxing:



Wrappers:







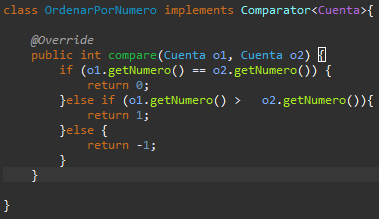
Lo que Aprendimos:

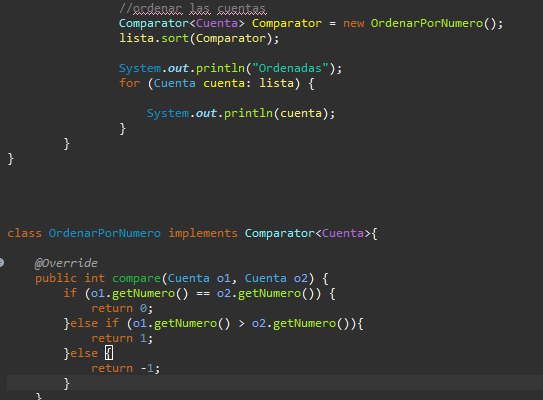
En esta clase nos enfocamos en las clases de*WRAPPERS* y aprendimos que.

* Para cada primitivo hay una clase llamada *Wrapper*.
* Para almacenar un primitivo en una colección, necesita crear un objeto que envuelva el valor.
* La creación del objeto *Wrapper* se llama *autoboxing*.
* La extracción del valor primitivo del objeto *Wrapper* se llama *unboxing*.
* El *autoboxing* y *unboxing* ocurren automáticamente.
* Las clases *wrapper* tienen varios métodos auxiliares, por ejemplo para el *parsing*.
* Todas las clases *wrappers* que representan un valor numérico tienen la clase *java.lang.Number* como madre

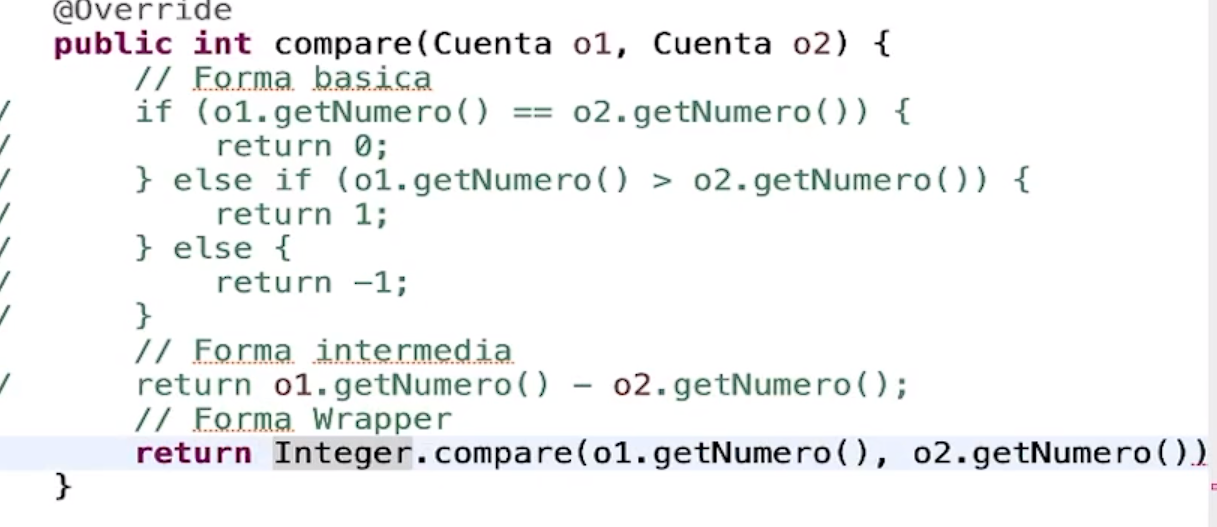
**Ordenando Listas:**

Ordenando Listas:

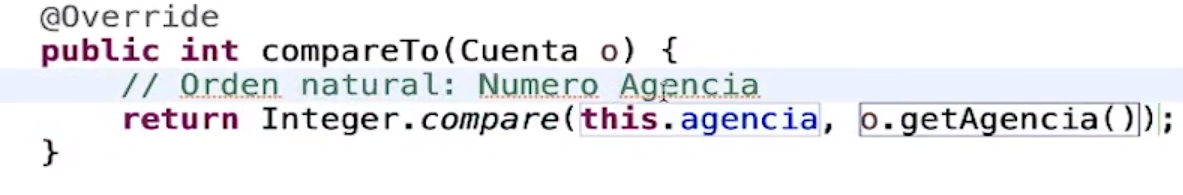


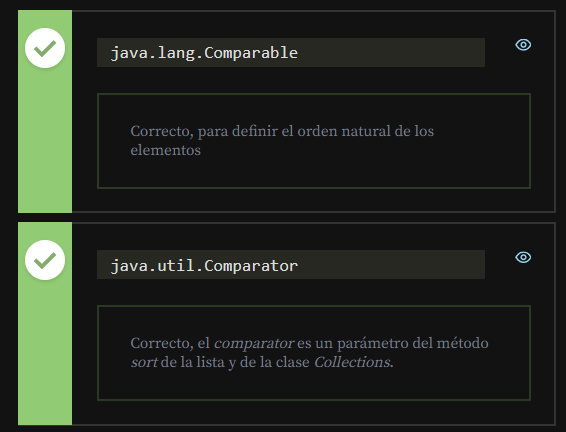


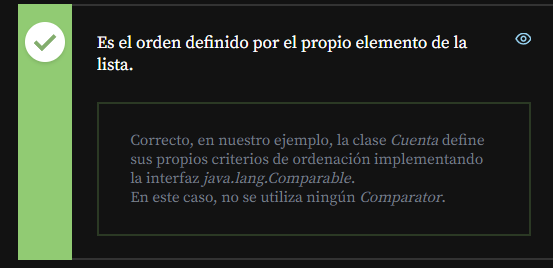
Usando Wrappers:



Orden Natural:







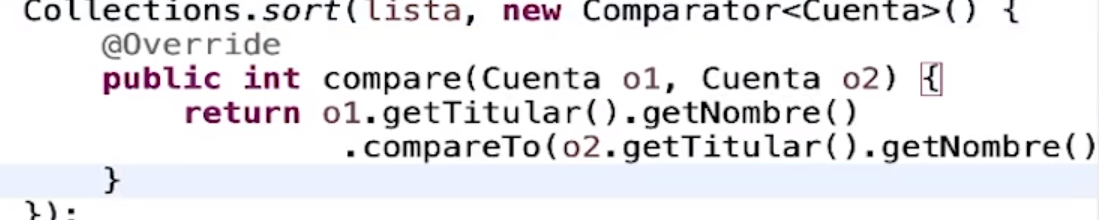
Lo que Aprendimos:

En esta clase fundamental e importante aprendimos que:

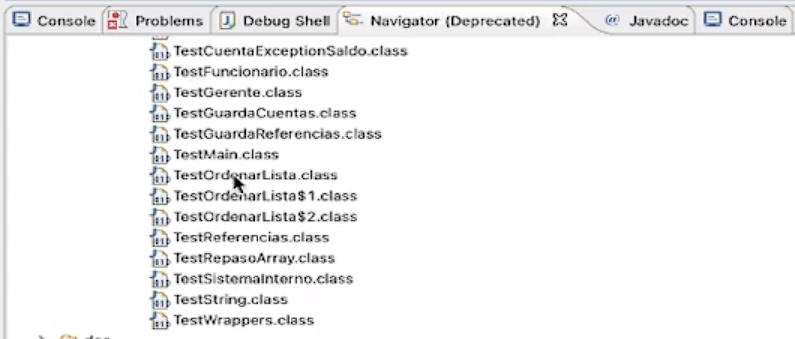
* Para ordenar una lista necesita definir un criterio de ordenación
* Hay dos formas de definir este criterio
* A través de la interfaz del *comparador*
* A través de la interfaz *Comparable* (*orden natural*)
* El algoritmo de ordenación ya se ha implementado
* En la lista en el método de *sort*
* En la clase *Collections* por el método *sort*
* La clase *Collections* es una fachada con varios métodos auxiliares para trabajar con colecciones, principalmente listas

**Clases anónimas y Lambdas:**

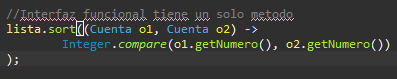
Clases anónimas:

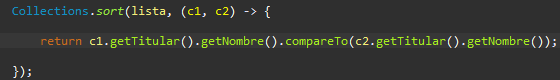


Crea una clase atrás.



Lambdas:





Conclusión:

* Arreglos estructura más básica para agrupar, se crean con tamaño fijo, trabajan en base a índices (empieza en 0),
* ArrayList, Lista donde interactuamos con un objeto con métodos, Es hijo de interfaz List.
* LinkedList tiene referencia de sus nodos anterior y posterior.
* Vector es threadsafe, dos pilas de pueden escribir pero no al tiempo.
* Clase anónima no es creada por nosotros, es creada por Java.
* Lambdas funciones donde enviamos parámetros por medio ->, no necesitamos especificar el objeto.