Bloque 3

Colecciones y Mapas

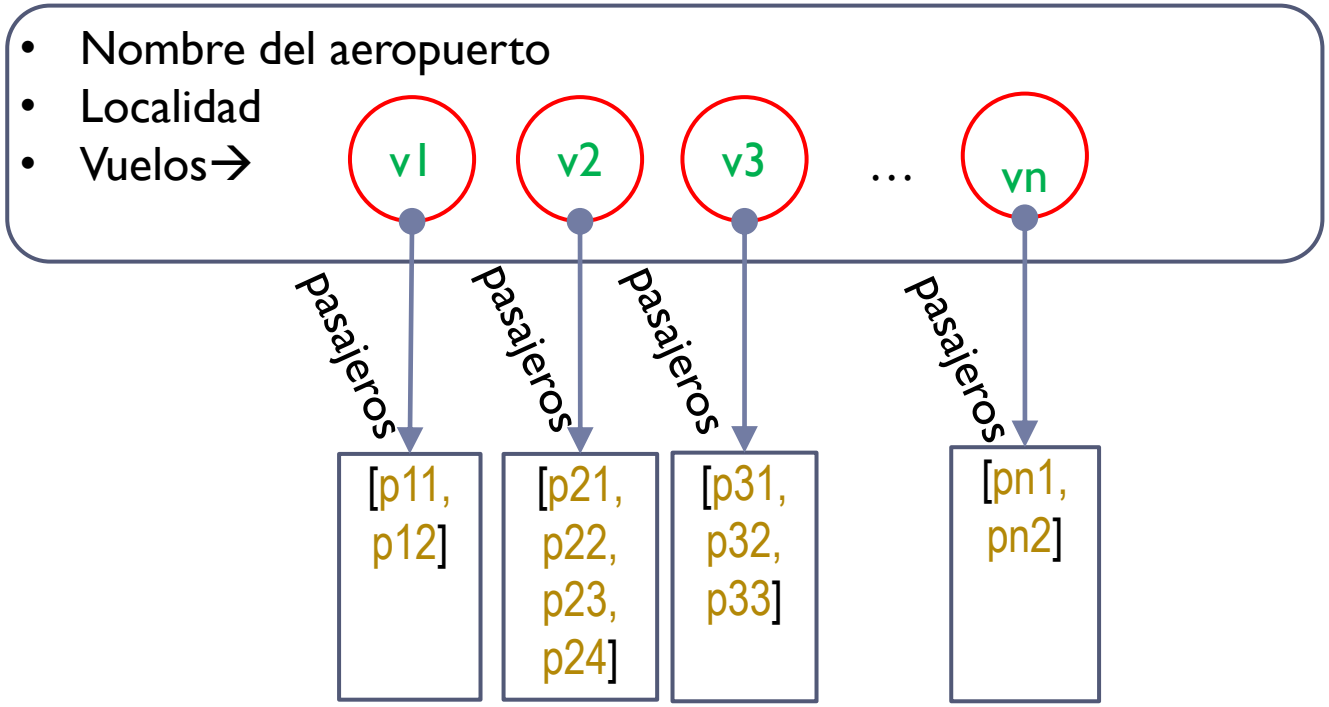
List, Set, SortedSet y Map

Fundamentos de Programación
Departamento de Lenguajes y Sistemas Informáticos

2023/2024

Tipos Contenedores

Aeropuerto



Listas con objetos Persona

Aeropuerto es un contenedor de objetos Vuelo y Vuelo, a su vez, es un contenedor de objetos Persona

2023/24

Ejercicio Aeropuerto

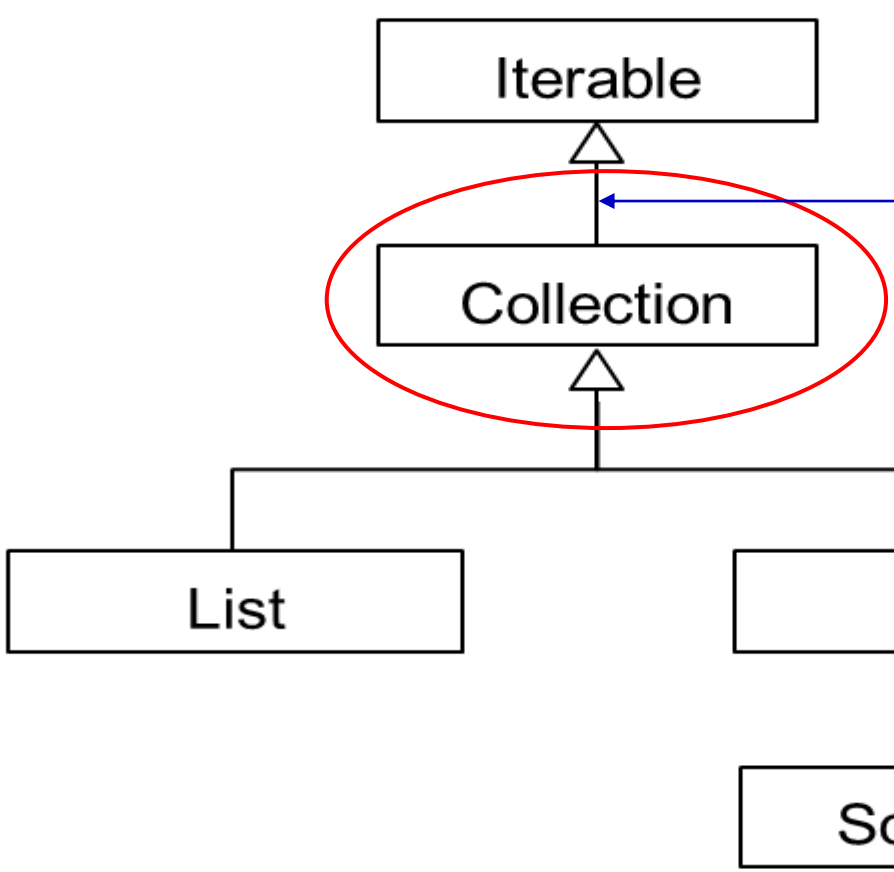
1. Descargue el archivo *ProyectoAeropuerto.zip*.
2. Descargue el documento "*ExportarImportarUnProyecto.pdf*"
3. Siga las instrucciones del documento e importe el proyecto a Eclipse.
4. En la carpeta *doc* tiene el *enunciadoAeropuerto01.pdf*
5. Realice lo que se pide en el enunciado

Nota.- Este proyecto irá creciendo hasta llegar aproximadamente hasta el enunciado 09 o 10, ya que servirá de base para practicar los conceptos que se expliquen en las clases de teoría.

Por lo que es importante llevarlo al día.

Introducción a Collection

Jerarquía de interfaces.



Esto implica que las colecciones se pueden recorrer elemento a elemento (*recordar el for*)

2023/24

Introducción a Collection (métodos)

Métodos de la Interfaz *Collection*:

Nomenclaturas que aparecen a continuación:

- *E*: Tipo de los objetos(por ejemplo: *Vuelo, Animal, Persona, Canción*, etc)
- *e*: Un objeto ya creado del tipo E (Por ejemplo: un tren, un evento, un libro, una canción, etc).
- *c*: una colección de objetos. Si aparece como tipo "*?*" (símbolo comodín) indica que los objetos son del mismo tipo de la colección que invoca al método (de tipo *E*) o un subtipo suyo.
  Por ejemplo, a una *Colección* de objetos Persona le podemos añadir otra colección de tipo Estudiante, siempre que se haya definido Estudiante como subtipo de Persona.

```
public class Estudiante extends Persona {..
```

2023/24

Tipo Collection (recorrido)

Recorrer una colección mediante for extendido:

Si se dispone de una Colección de objetos de tipo E, se recorre con la sentencia *for* con la siguiente sintaxis:

```
Collection<E> miColección=  new ArrayList<E>();
                            new LinkedList<E>();
---se añaden objetos <E>---
                            new HashSet<E>();
                            new TreeSet<E>();
for (E e: miColección){
    tratamiento del objeto e;
}
```

Nota.- Donde dice *Collection<E>* puede ser, y normalmente es, *List <E>* / *Set <E>* / *SortedSet <E>* según el constructor con el que se cree la colección.

2023/24

Introducción a Collection (métodos)

- boolean ***add*** (*E e*): Añade un objeto "e" de tipo "E" a la colección, devuelve *true* si se añade el objeto, ya que se ha modificado la colección.
- boolean ***addAll*** (*Collection<? extends E> c*):Añade todos los objetos de la colección "c" a la colección que invoca (*los elementos de c deben ser del mismo tipo o hijos de los elementos de la colección que invoca*). Es el operador unión. Devuelve *true* si la colección original se modifica.
- void ***clear*** ():Borra todos los objetos de la colección.
- boolean ***contains*** (*Object o*): Devuelve *true* si el objeto "o" está en la colección que invoca.

Introducción a Collection (métodos)

- boolean *containsAll* (*Collection<E> c*): Devuelve *true* si la colección que invoca al método contiene todos los objetos de la colección c (*los objetos de c y de la colección que invoca deben ser del mismo tipo*).
- boolean *isEmpty* (): Devuelve *true* si la colección que invoca a método no tiene objetos.
- boolean *remove* (*Object o*): Borra el objeto "o" de la colección que invoca; si no estuviera se devuelve *false* ya que la colección no se ha modificado.
- boolean *removeAll* (*Collection<E> c*): Borra todos los objetos de la colección que invoca que estén en la colección c (*el tipo de objetos de la colección c debe ser igual que la de la colección que invoca*). Es el operador diferencia. Devuelve *true* si la colección original se modifica.

2023/24

Introducción a Collection (métodos)

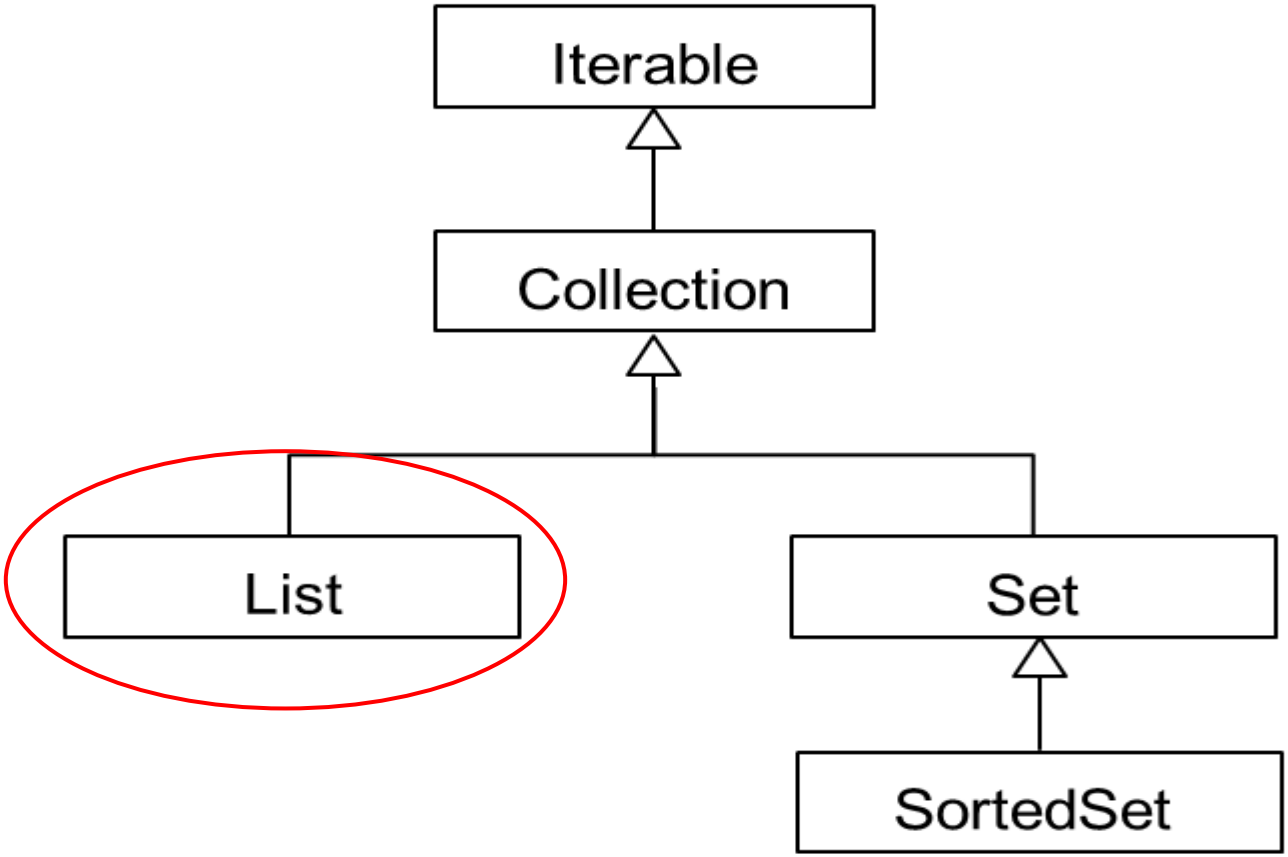
- boolean *retainAll* (*Collection<E> c*): La colección que invoca se queda con los objetos que están en la colección c (*el tipo de objetos de la colección c debe ser igual que la de la colección que invoca*). Por tanto, es la intersección entre ambas colecciones. Devuelve true si la colección original se modifica.
- int *size* ():Devuelve el número de elementos de la colección que invoca al método.

¡¡Los 10 métodos vistos anteriormente se pueden utilizar para manejar listas, conjuntos y conjuntos ordenados!!

Observar que, en ningún caso, se ha hecho referencia a una posición concreta de un elemento dentro de la colección

Introducción a Collection

Jerarquía de interfaces.



- Colecciones: Listas, Conjuntos y Conjuntos Ordenados

2023/24

Tipo List (*Definición y Construcción*)

El tipo ***List*** es una interfaz del paquete "java.util" que se implementa mediante dos clases (*al igual que nuestro tipo Evento*). Las dos clases que lo implementa son: ***ArrayList*** y ***LinkedList***.

- ***Construcción***: Para crear una lista vacía que almacene objetos de tipo E se utiliza la siguiente sintaxis según la implementación que se elija
  - *List*<E> lista1=new *ArrayList*<E>(); Esta implementación accede rápido a los elementos de la lista, pero la inserción o supresión de un objeto en determinada posición de la lista puede ser lento (*pensar en una estantería de libros en la que no puede haber huecos*).
  - *List*<E> lista2=new *LinkedList*<E>(); Esta implementación permite la inserción o supresión de un objeto de manera rápida pero su localización es más lenta que la anterior (*pensar en una cadena de eslabones que para llegar a uno hay que pasar por los anteriores*).

2023/24

Tipo List (*Definición y Construcción*)

El tipo ***List*** es una interfaz del paquete "java.util" que se implementa mediante dos clases (*al igual que nuestro tipo Evento*). Las dos clases que lo implementa son: ***ArrayList*** y ***LinkedList***.

- ***Construcción***: Para crear una lista vacía que almacene objetos de tipo E se utiliza la siguiente sintaxis según la implementación que se elija
  - *List*<E> lista1=new *ArrayList*<E>(); Esta implementación accede rápido a los elementos de la lista pero la inserción o supresión de un objeto en determinada posición de la lista puede ser lento (*pensar en una estantería de libros en la que no puede haber huecos*).
  - *List*<E> lista2=new *LinkedList*<E>(); Esta implementación permite la inserción o supresión de un objeto de manera rápida pero su localización es más lenta que la anterior (*pensar en una cadena de eslabones que para llegar a uno hay que pasar por los anteriores*).

Tipo List (*métodos específicos*)

Además de los 10 métodos de *Collection* vistos anteriormente las listas tienen algunos *otros métodos* en los que interviene la *posición*. Los 8 más relevantes son:

- boolean ***add*** (*int p, E e*): Inserta el objeto "e" en la posición "p", desplazando "*hacia la derecha*" los objetos desde la posición p. Con 0≤p<size(). Si p>=size() lo insertará en la última posición. Devuelve *true* si se añade.
- boolean ***addAll*** (*int p, Collection<? extends E> c*): Inserta todos los elementos de la colección c en la lista que invoca al método en la posición especificada, desplazando "*hacia la derecha*" los objetos desde la posición p, c.size() posiciones. Con 0≤ p<size(). Si p>=size() la insertarán desde la última posición. Los objetos de c deben ser del mismo tipo o hijos de los elementos de la lista que invoca. Devuelve *true* si se añaden.

Tipo List (*métodos específicos*)

- E *get* (*int p*): Devuelve el elemento de la lista que invoca al método en la posición especificada. Con 0 ≤ p < size().
- int *indexOf* (*Object o*): Devuelve el índice donde se encuentra por primera vez el objeto "o" (si no está devuelve -1).
- int *lastIndexOf* (*Object o*): Devuelve el índice donde se encuentra por última vez el elemento "o" (si no estuviera devuelve -1).
- E *remove* (*int p*): Borra el objeto en la posición p. Con 0 ≤ p < size(). Desplaza "hacia la izquierda" una posición los elementos desde p+1 en adelante. Devuelve el elemento borrado.
- E *set* (*int p, E e*): Reemplaza el objeto de la posición p por el objeto "e". Con 0 ≤ p < size(). Devuelve el objeto reemplazado.

Tipo List (*métodos específicos*)

- List<E> *subList* (*int p, int q*): Devuelve una "*vista*" de la porción (una sublista) desde p (incluido) hasta q (sin incluir) [p,q).
  - Una vista significa que si se modifica la sublista se está modificando la original y viceversa. Con 0 ≤ p ≤ q ≤ size().
  - Si no se quiere vinculación entre la lista original y la sublista, debe crearse una copia de esta con la siguiente sintaxis:

```
List<E> miSublista=new ArrayList<E>();
miSublista.addAll (listaOriginal.subList(p,q));
```

2022/23

Ejercicio

- Realice los ejercicios del ***EnunciadoAeropuerto01***