



EXPERIENCIA DE APRENDIZAJE 2 HERENCIA Y COLECCIONES

Colecciones POO

DSY1102-Desarrollo Orientado a Objetos



CONTENIDO

01

COLECCIONES

04

MÉTODOS

02

TIPOS DE
COLECCIONES

05

PROBLEMA

03

ARRAYLIST

01

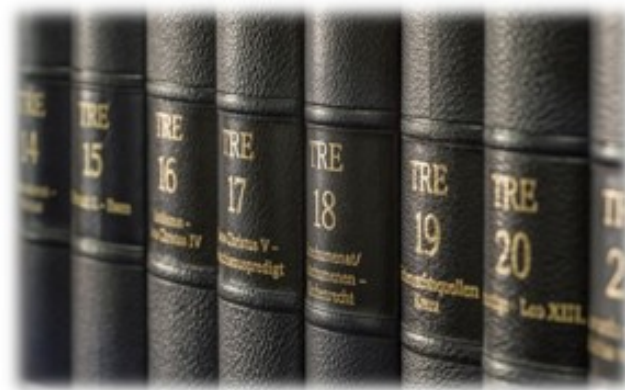
COLECCIONES

Colecciones

Son estructuras que permiten almacenar en la memoria grupos de objetos relacionados ofreciendo mayores capacidades que los arreglos. Las colecciones son reutilizables, confiables, poderosas y eficientes.

Ejemplos:

Colección de Libros



Colección de Personas

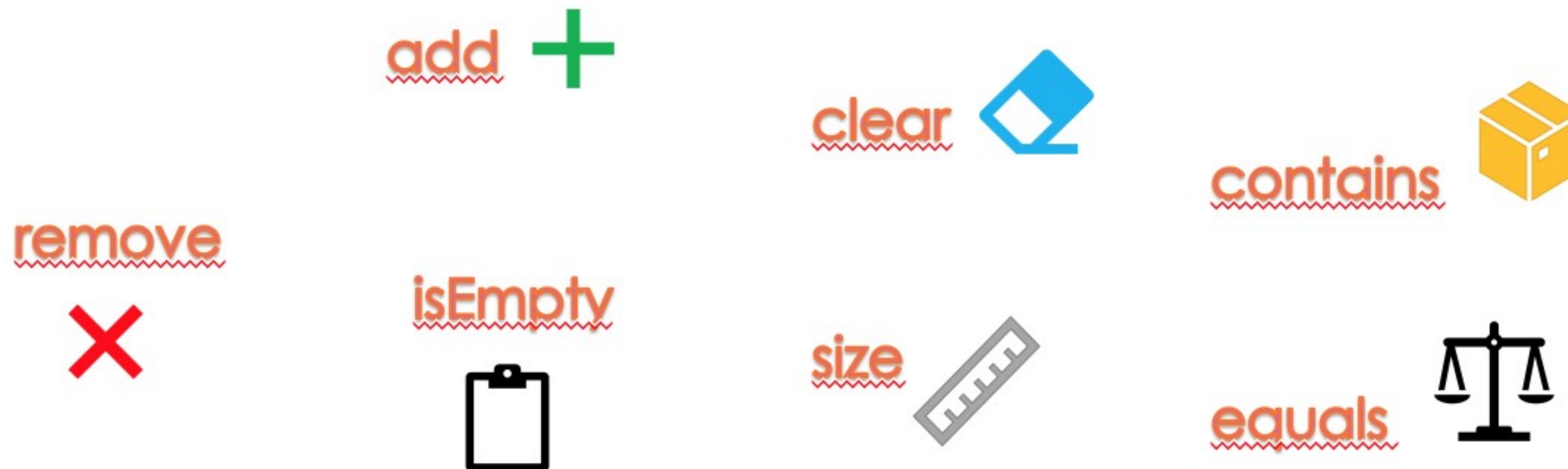


Colección de Casas



Colecciones

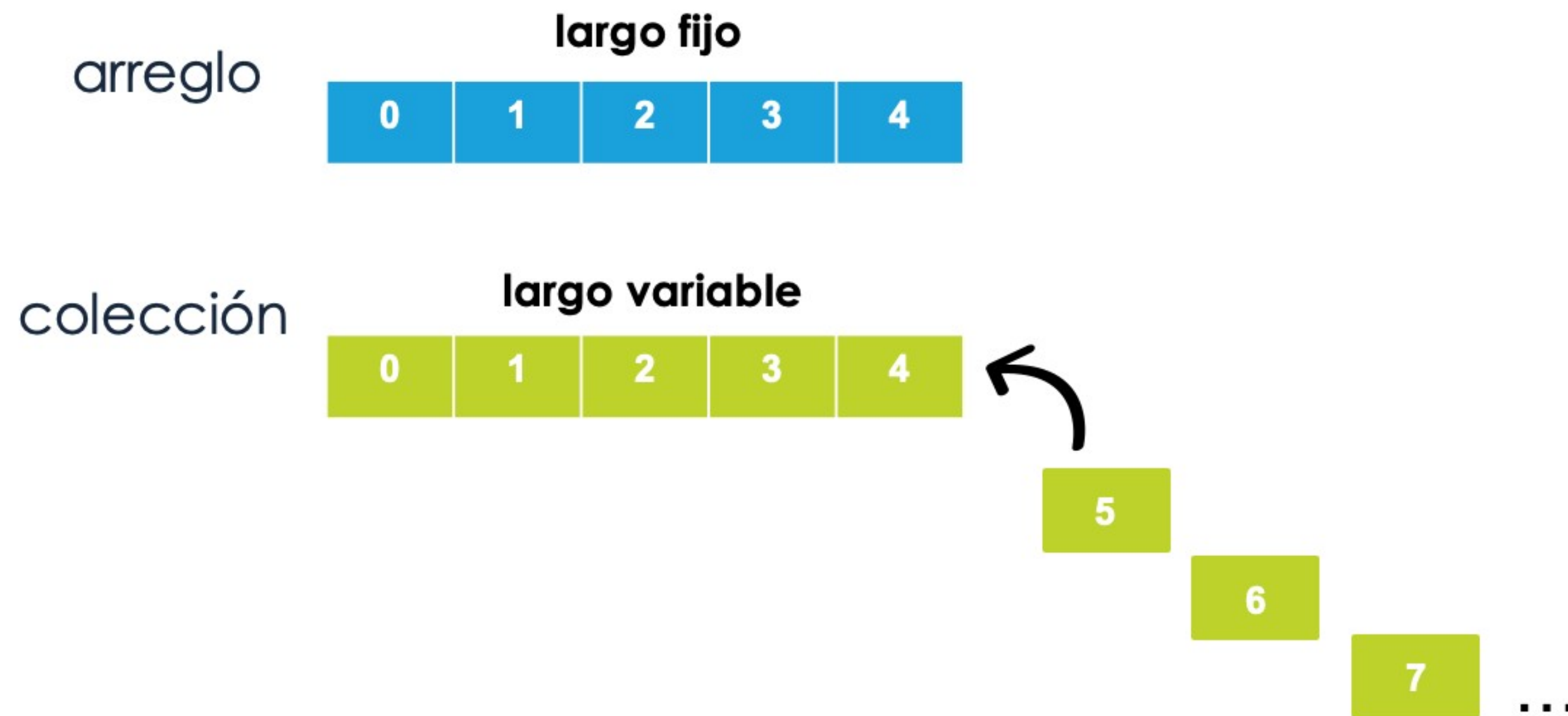
La API de Java, provee las estructuras de datos predefinidas, conocidas como colecciones. Estas, tienen **métodos** eficientes permitiendo organizar, almacenar y obtener los datos sin necesidad de saber cómo se almacenan. Gracias a ello, se reduce el tiempo de desarrollo de aplicaciones.



<https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html>

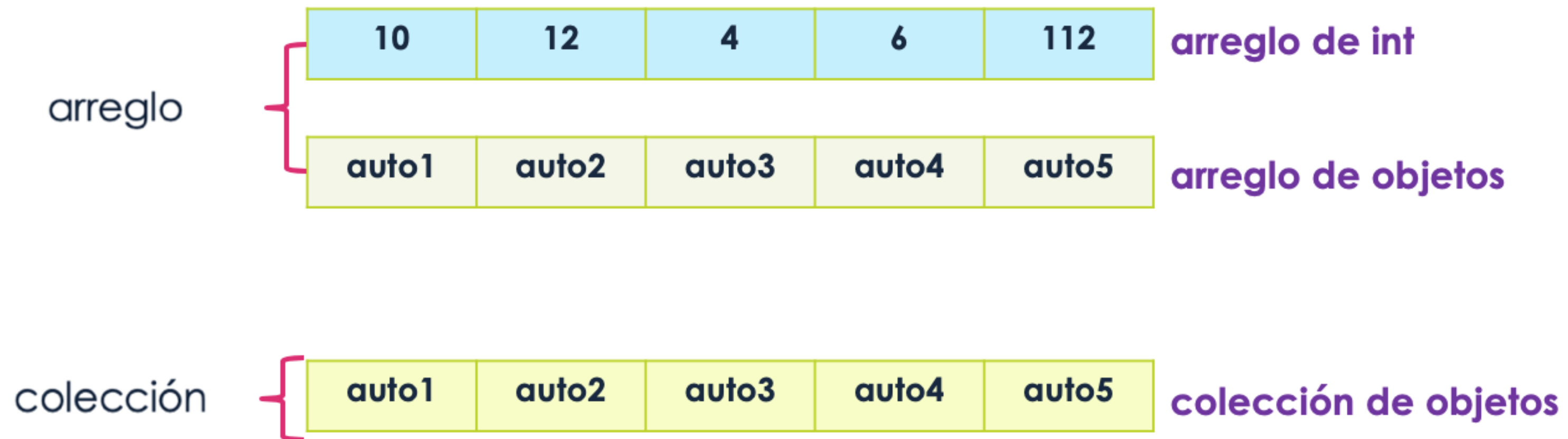
Colecciones v/s Arreglos

Los **arreglos**, tienen un **tamaño fijo**, no cambian automáticamente su tamaño en tiempo de ejecución para dar espacio a nuevos elementos, en cambio, la **colección** `ArrayList<T>` provee una solución a este problema, ya que puede **cambiar** su **tamaño** en forma dinámica para dar espacio a más elementos.



Colecciones v/s Arreglos

Otra diferencia, es que los **arreglos** pueden contener **tipos de datos primitivos y referenciados**, en cambio, las **colecciones** sólo pueden ser de **tipo referenciado**, es decir, no puede contener datos primitivos, sólo objetos.



Colecciones v/s Arreglos

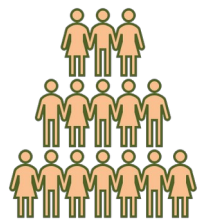
Existen varios tipos de colecciones:



List: Elementos en una secuencia particular que mantienen un orden y permite duplicado. Existen: ArrayList, Vector y LinkedList.



Set: No puede haber duplicados. Cada elemento debe ser único, por lo que si existe uno duplicado, no se agrega.



Queue: Colección ordenada con extracción por el principio e inserción por el principio (LIFO – Last Input, First Output) o por el final (FIFO – First Input, First Output). Se permiten elementos duplicados.



Map: Un grupo de pares objeto clave-valor, que no permite duplicados en sus claves. Existen: HashMap, HashTable, LinkedHashMap y TreeMap.



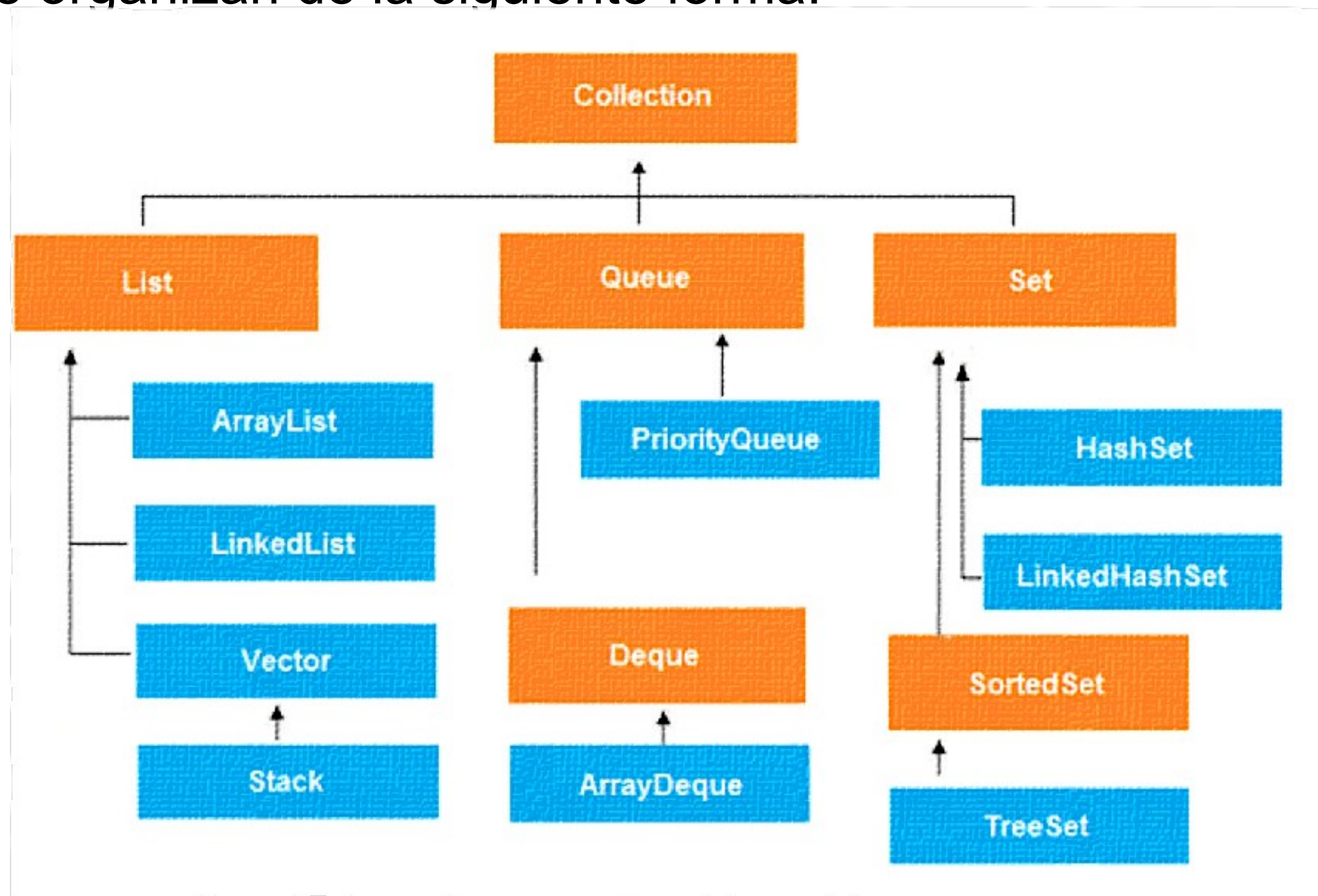
02

TIPOS DE COLECCIONES

TI

Tipos de Colecciones

Las colecciones se organizan de la siguiente forma:



Te invitamos a investigar las características y diferencias de las colecciones

<https://www.aprenderaprogramar.com/>

03

ARRAYLIST

ArrayList

Nos enfocaremos en la interfaz List, para la cual se utilizará la implementación con ArrayList. Estos, son similares a los arreglos, sólo que proporcionan una funcionalidad mejorada, el ajuste de tamaño dinámico, que permite alojar una cantidad mayor o menor de elementos.

Ejemplo:

ArrayList<T>, donde T, es el tipo del objeto.

ArrayList<String> **lista**;

lista

"juan"	"pedro"	"tomás"	"judas"	"pablo"
0	1	2	3	4

Declara **lista** como una colección del tipo **ArrayList** que sólo puede almacenar objetos del tipo String. Esta colección es de **tipo genérica**, ya que tiene un tipo de dato definido: String.

ArrayList

La colección ArrayList se **declara** (1) y se **crea** (2).

- 1 List<String> lista;
- 2 lista = new ArrayList<String>();

Pertenece al paquete **java.util**, por lo que para utilizarla se debe importar.



```
import java.util.ArrayList;
import java.util.List;

public class PruebaColecciones {

    public static void main(String[] args) {

        String nombre;

        List<String> listaNombres = new ArrayList<>();
        listaNombres.add("hugo");
        listaNombres.add("paco");
        listaNombres.add("luis");

        for (String tmp : listaNombres) {
            nombre = tmp;
        }

    }

}
```

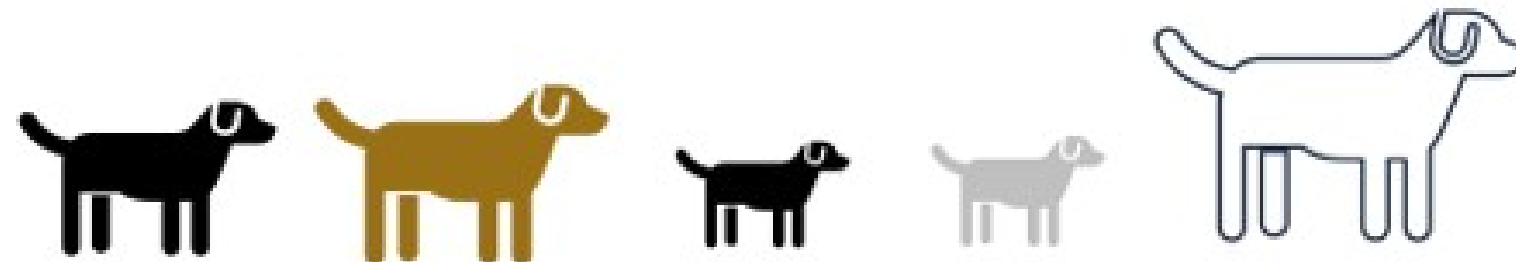


Otra opción:
1 y 2 en una sola línea

ArrayList

Los **tipos genéricos** permiten forzar la seguridad de los tipos de datos, en tiempo de compilación.

Usando tipo genérico:



```
String nombre;
```

```
List<String> listaNombres = new ArrayList<>();  
listaNombres.add("hugo");  
listaNombres.add("paco");  
listaNombres.add("luis");
```

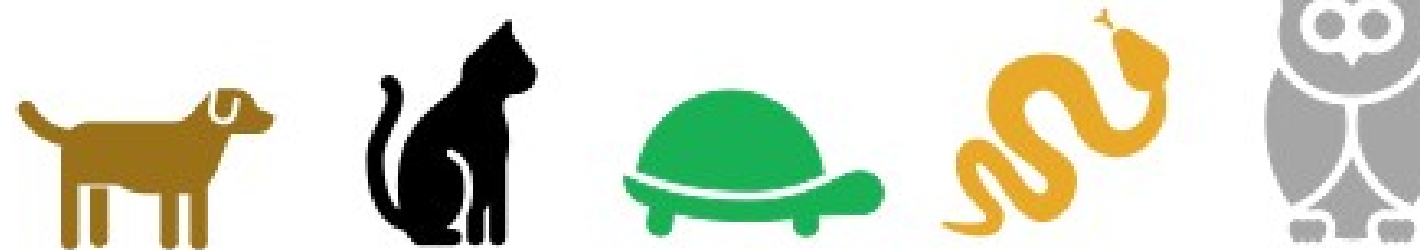
```
for (String tmp : listaNombres) {  
    nombre = tmp;  
}
```

ArrayList<Auto> listaAutos; ✓

ArrayList<int> listaEdades; ✗

ArrayList

Sin colecciones genéricas:



```
String nombre;  
  
List listaNombres = new ArrayList();  
listaNombres.add("hugo");  
listaNombres.add("paco");  
listaNombres.add("luis");  
  
for (Object tmp : listaNombres) {  
    nombre = (String) tmp;  
}
```

Cuando se utilizan colecciones **genéricas**, no es necesario castear los objetos, ya que **se sabe el tipo de dato**. En este ejemplo, puede venir cualquier objeto, String, Auto, Alumno, etc...



04 MÉTODOS

Métodos

ArrayList tiene varios **métodos**, que facilitan la tarea del programador . Algunos de ellos son:

add



Agrega un elemento al final del ArrayList.

clear



Elimina todos los elementos del ArrayList.

contains



Devuelve true si el ArrayList contiene el elemento especificado, en caso contrario, devuelve false.

get



Devuelve el elemento en el índice especificado.

indexOf



Devuelve el índice de la primera ocurrencia del elemento especificado en el ArrayList.

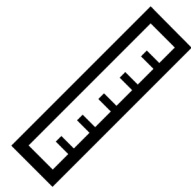
Métodos

remove



Elimina la primera ocurrencia del valor especificado o del elemento en el subíndice especificado.

size



Devuelve el número de elementos almacenados en el ArrayList.

Puedes encontrar todos los métodos en:

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

Profundiza la interfaz List:

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

05

PROBLEMA

Problema

El uso del scooter, ha pasado a ser un medio de transporte preferido, por su accesibilidad en relación a los otros medios de transporte y por disminuir los tiempos de traslado, mejorando la calidad de vida.

Como ha crecido el aumento de ventas de este medio, es que las empresa chilenas han tenido que contratar más empleados para la venta de este vehículo.

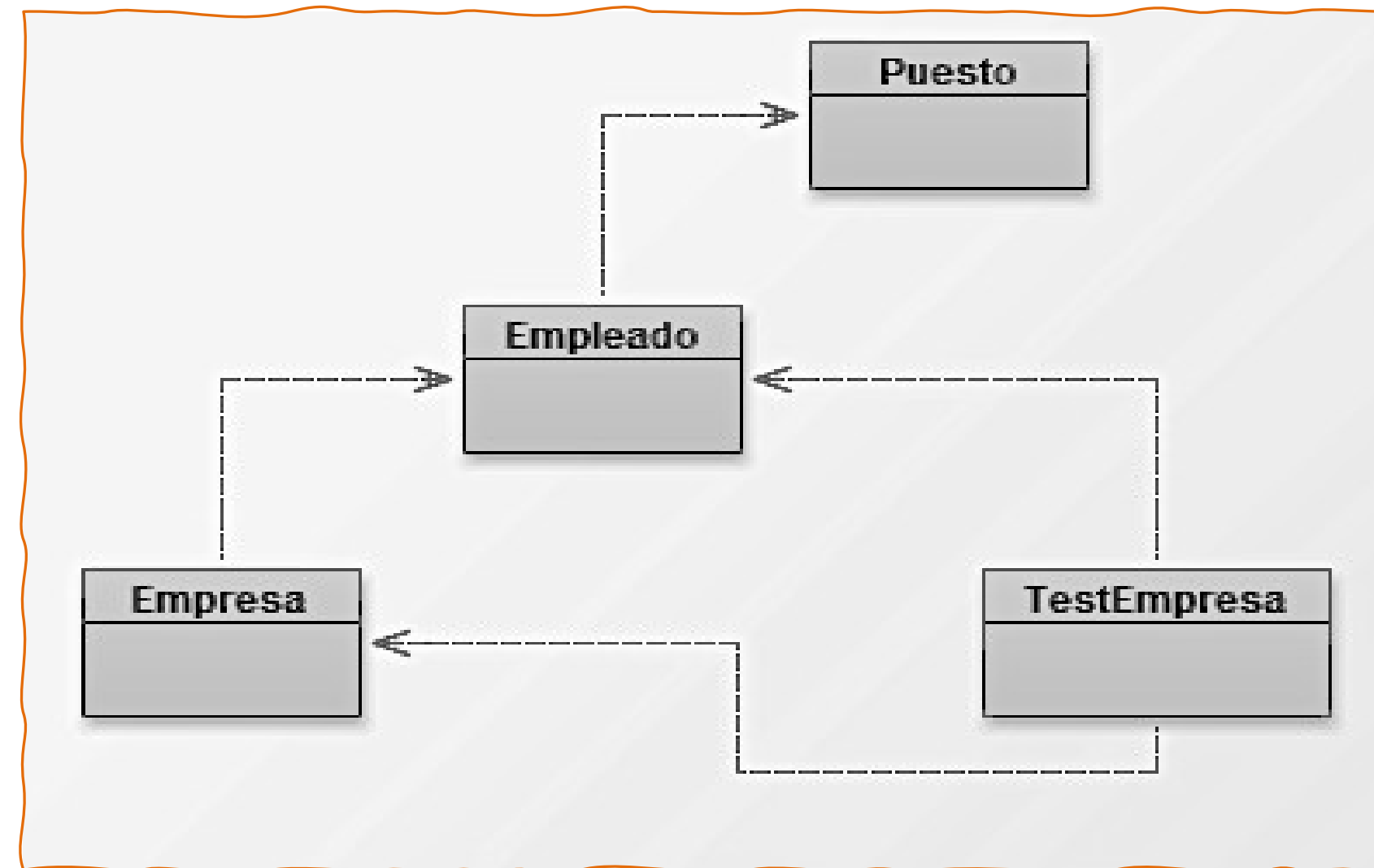
Se le ha solicitado Ingresar, Eliminar y Listar los datos de los empleados pertenecientes a una empresa de venta de scooter llamada “scootin”.



DuocUC[®]

Problema

El analista ha entregado el siguiente diagrama de clases:



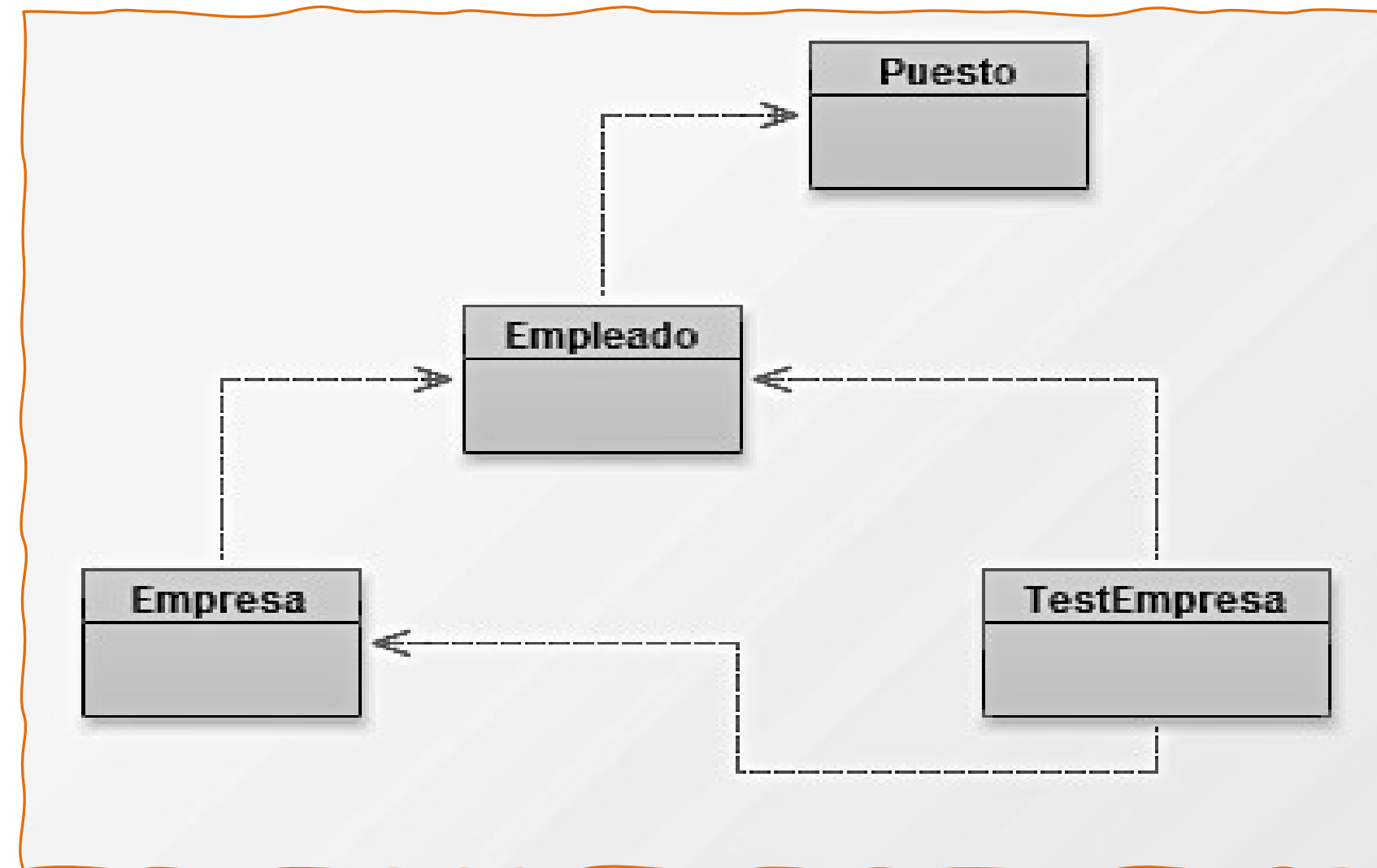
Se requiere almacenar:

- Puesto: código, nombre
- Empleado: rut, nombre completo, genero, años de servicio, edad y puesto
- Empresa: contiene colección de Empleados (ArrayList)

Solución

Si analizamos el diagrama, ¿qué clase se debiera implementar primero?

La empresa requiere empleados



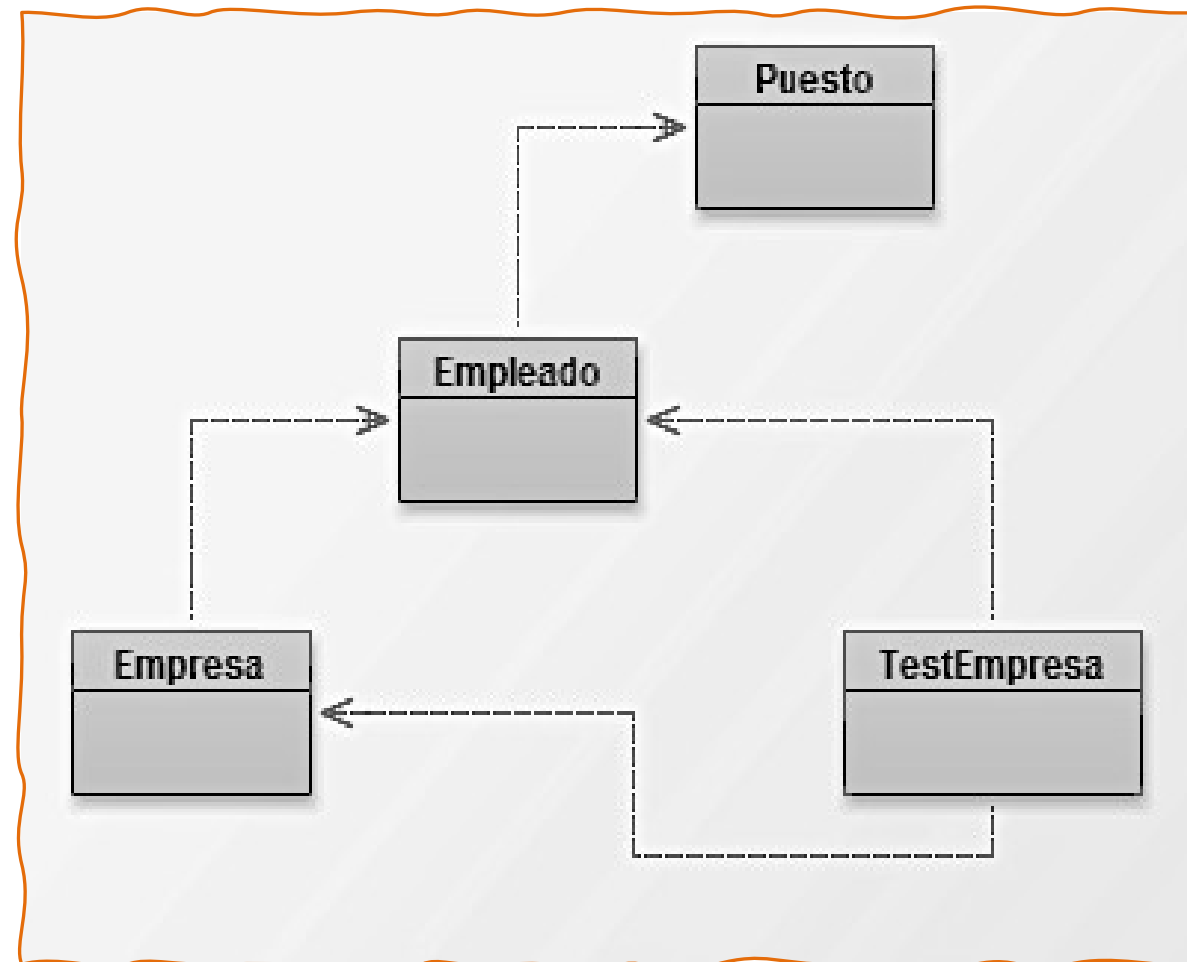
Los empleados necesitan un cargo

Solución

¡El puesto!

Dado que no necesita de otras clases, luego continuamos con el empleado y finalmente la empresa.

El Test, es la clase que contiene el main y se programará después de todas las clases.



¡Comencemos!

Solución

Puesto

```
public class Puesto {  
  
    private int codigo;  
    private String nombrePuesto;  
  
    public Puesto() {  
    }  
  
    public Puesto(int codigo, String nombrePuesto) {  
        this.codigo = codigo;  
        this.nombrePuesto = nombrePuesto;  
    }  
  
    public int getCodigo() {  
        return codigo;  
    }  
  
    public void setCodigo(int codigo) {  
        this.codigo = codigo;  
    }  
  
    public String getNombrePuesto() {  
        return nombrePuesto;  
    }  
  
    public void setNombrePuesto(String nombrePuesto) {  
        this.nombrePuesto = nombrePuesto;  
    }  
  
    @Override  
    public String toString() {  
        return "Puesto{" + "codigo=" + codigo + ", nombrePuesto=" + nombrePuesto + '}';  
    }  
}
```

Solución

Empleado

atributo puesto (objeto)
del tipo de dato Puesto
(clase)

Inicializar el objeto

```
public class Empleado {  
  
    private String rut, nombreEmpleado;  
    private char genero; //M:masculino F:femenino  
    private int acno, edad;  
    private Puesto puesto;  
  
    public Empleado() {  
        puesto = new Puesto();  
    }  
  
    public Empleado(String rut, String nombreEmpleado, char genero, int acno, int edad, Puesto puesto) {  
        this.rut = rut;  
        this.nombreEmpleado = nombreEmpleado;  
        this.genero = genero;  
        this.acno = acno;  
        this.edad = edad;  
        this.puesto = puesto;  
    }  
  
    public String getRut() {...3 lines }  
  
    public void setRut(String rut) {...3 lines }  
  
    public String getNombreEmpleado() {...3 lines }  
  
    public void setNombreEmpleado(String nombreEmpleado) {...3 lines }
```

Solución

Empresa

Se declara la colección del tipo ArrayList listaEmpleados, que va a ser llenada con objetos de la clase Empleado

Inicializar la colección listaEmpleados

Recibe por parámetro un objeto emple de la clase Empleado y con el método add lo agrega a la colección

Recibe por parámetro el rut del empleado y con el ciclo for recorre la colección buscando al empleado que tiene ese rut

```
public class Empresa {  
  
    private ArrayList<Empleado> listaEmpleados;  
  
    public Empresa() {  
        listaEmpleados = new ArrayList<Empleado>();  
    }  
  
    // agregar  
    public boolean agregar(Empleado emple) {  
        return listaEmpleados.add(emple);  
    }  
  
    // buscar  
    public boolean buscarEmpleado(String rut) {  
        for (Empleado emple : listaEmpleados) {  
            if (emple.getRut().equals(rut)) {  
                return true;  
            }  
        }  
        return false;  
    }  
}
```


Solución

Empresa continuación...

Recorre la colección listaEmpleados y muestra por cada empleado sus atributos invocando al método toString que está implementado en la clase Empleado

Recibe por parámetro el rut del empleado y con el ciclo for recorre la colección buscando al empleado que tiene ese rut para eliminarlo con el método remove()



Investiga los métodos equals y hashCode

```
// listar empleados
public void listarEmpleados() {
    for (Empleado emple : listaEmpleados) {
        System.out.println(emple.toString());
    }
}

// eliminar empleado
public boolean eliminarEmpleado(String rut) {
    for (Empleado emple : listaEmpleados) {
        if (emple.getRut().equals(rut)) {
            listaEmpleados.remove(emple);
            return true;
        }
    }
    return false;
}
```

Solución

Desde la clase que contiene el **main**

Al crear la empresa, se ejecuta su constructor, el cual crea la colección

Si el empleado no existe (método buscarEmpleado) se agrega el empleado1 a La colección

```
public static void main (String[] args) {  
  
    Puesto pueston = new Puesto (1, "gerente");  
    Puesto puestito = new Puesto (2, "ejecutivo");  
  
    Empleado empleado1 = new Empleado ("1-9", "john", 'M', 10, 40, pueston);  
    Empleado empleado2 = new Empleado ("2-7", "juanita", 'F', 2, 25, puestito);  
  
    Empresa empresa = new Empresa(); // crea la colección en el constructor  
  
    // agregar empleado 1  
    if (empresa.buscarEmpleado("1-9") == false) {  
        empresa.agregar(empleado1);  
        System.out.println("Se agregó empleado " + empleado1.getNombreEmpleado());  
    } else {  
        System.out.println("Empleado existe");  
    }  
}
```


Solución

Otra forma de ver:

```
public class VentaScooter {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main (String[] args) {  
  
        Puesto puesto = new Puesto(1, "gerente");  
        Puesto puesto = new Puesto(2, "ejecutivo");  
  
        Empleado empleado1 = new Empleado("1-9", "john", 'M', 10, 40, puesto);  
        Empleado empleado2 = new Empleado("2-7", "juanita", 'F', 2, 25, puesto);  
  
        Empresa empresa = new Empresa(); // crea la colección en el constructor  
  
        // agregar empleado 1  
        if (empresa.buscarEmpleado("1-9") == false) {  
            empresa.agregar(empleado1);  
            System.out.println("Se agregó empleado " + empleado1.getNombreEmpleado());  
        } else {  
            System.out.println("Empleado existe");  
        }  
    }  
}
```

```
14 public class Empresa {  
15  
16     private ArrayList<Empleado> listaEmpleados;  
17  
18     public Empresa() {  
19         listaEmpleados = new ArrayList<Empleado>();  
20     }  
21  
22     // agregar  
23     public boolean agregar(Empleado emple) {  
24         return listaEmpleados.add(emple);  
25     }  
26  
27     // buscar  
28     public boolean buscarEmpleado (String rut) {  
29         for (Empleado emple : listaEmpleados) {  
30             if (emple.getRut().equals(rut)) {  
31                 return true;  
32             }  
33         }  
34         return false;  
35     }  
36 }
```

Solución

Continuación del main...

Invoca al método que lista los empleados

Elimina a un empleado si lo encuentra

```
//listar empleados
empresa.listarEmpleados();

// eliminar
if (empresa.eliminarEmpleado("2-7")) {
    System.out.println("Se eliminó empleado " + empleado1.getNombreEmpleado());
} else {
    System.out.println("No se eliminó empleado " + empleado1.getNombreEmpleado());
}

empresa.listarEmpleados();
```

Otra forma de ver:

```
//listar empleados
empresa.listarEmpleados();


// eliminar
if (empresa.eliminarEmpleado("2-7")) {
    System.out.println("Se eliminó empleado " + empleado1.getNombreEmpleado());
} else {
    System.out.println("No se eliminó empleado " + empleado1.getNombreEmpleado());
}

empresa.listarEmpleados();
```

```
36
37 //listar empleados
38 public void listarEmpleados() {
39     for (Empleado emple : listaEmpleados) {
40         System.out.println(emple.toString());
41     }
42 }
43
44 //eliminar empleado
45 public boolean eliminarEmpleado(String rut) {
46     for (Empleado emple : listaEmpleados) {
47         if (emple.getRut().equals(rut)) {
48             listaEmpleados.remove(emple);
49             return true;
50         }
51     }
52     return false;
53 }
```

¿Qué hemos aprendido?

- ✓ Reconocer una colección.
- ✓ Utilizar una colección y sus métodos.
- ✓ Identificar tipos de colecciones.
- ✓ Utilizar ArrayList.
- ✓ Diferenciar entre una colección genérica y una no.



¿Qué te resultó difícil entender?

DuocUC[®]