Clases abstractas

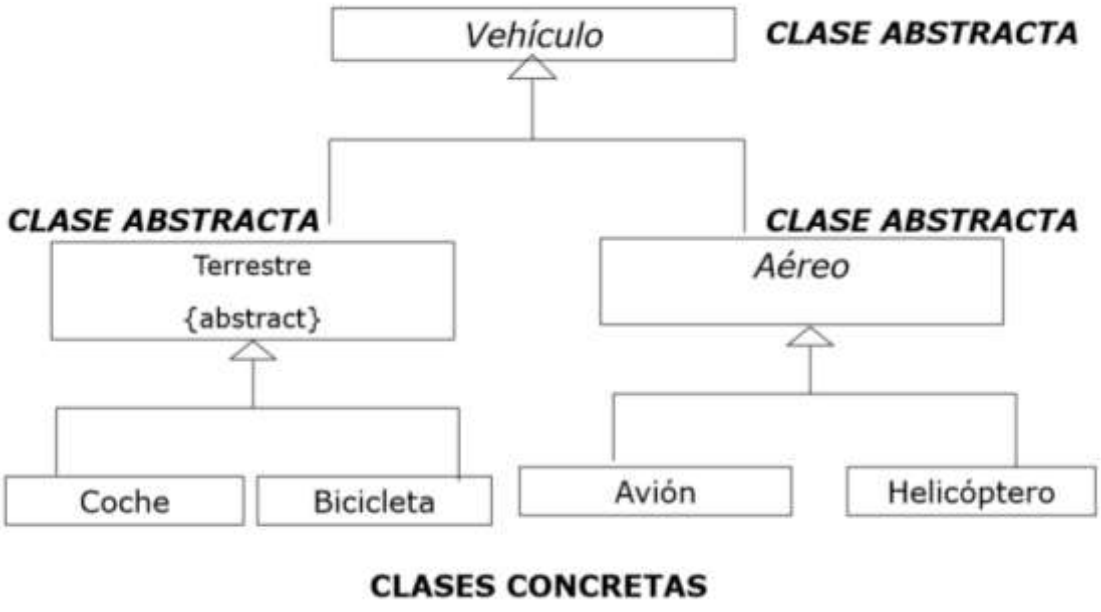
Una clase que declara la existencia de métodos pero no la implementación de dichos métodos (o sea, las llaves { } y las sentencias entre ellas), se considera una clase abstracta.

Una clase abstracta puede contener métodos **no-abstractos** pero al menos uno de los métodos debe ser declarado abstracta.

Para declarar una clase o un método como abstractos, se utiliza la palabra reservada **abstract**.

```
abstract class Drawing
{
  abstract void miMetodo(int var1, int var2);
  String miOtroMetodo( ){ ... }
}
```

Una clase abstracta no se puede instanciar pero si se puede heredar y las clases hijas serán las encargadas de agregar la funcionalidad a los métodos abstractos. Si no lo hacen así, las clases hijas deben ser también abstractas.



Interfaces

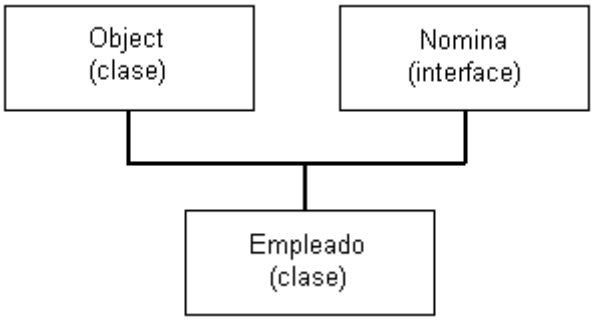
Una interface es una variante de una clase abstracta con la condición de que todos sus métodos deben ser abstractos. Si la interface va a tener atributos, éstos deben llevar las palabras reservadas **static final** y con un valor inicial ya que funcionan como constantes por lo que, por convención, su nombre va en mayúsculas.

```
interface Nomina
{
 public static final String EMPRESA = "Patito, S. A.";
 public void detalleDeEmpleado(Nomina obj);
}
```

Una clase *implementa* una o más interfaces (separadas con comas ",") con la palabra reservada **implements**. Con el uso de interfaces se puede "simular" la herencia múltiple que Java no soporta.

```
class Empleado implements Nomina
{
  ...
}
```

En este ejemplo, la clase Empleado tiene una clase padre llamada Object (implícitamente) e implementa a la interface Nomina, quedando el diagrama de clases de la siguiente manera:



La clase que implementa una interface tiene dos opciones:

1) Implementar todos los métodos de la interface.

2) Implementar sólo algunos de los métodos de la interface pero esa clase debe ser una clase abstracta (debe declararse con la palabra **abstract**).

Paquetes /Package

Para hacer que una clase sea más fácil de localizar y utilizar así como evitar conflictos de nombres y controlar el acceso a los miembros de una clase, las clases se agrupan en paquetes.

- **Paquete**

  Un paquete es un conjunto de clases e interfaces relacionadas.

La forma general de la declaración **package** es la siguiente:
package nombrePaquete;
donde nombrePaquete puede constar de una sola palabra o de una lista de nombres de paquetes separados por puntos.
Ejemplo

```
package miPaquete;

class MiClase
{
  ...
}
```

Ejemplo

```
package nombre1.nombre2.miPaquete;

class TuClase
{
  ...
}
```

Los nombres de los paquetes se corresponden con el nombre de directorios en el sistema de archivos. De esta manera, cuando se requiera hacer uso de estas clases se tendrán que importar de la siguiente manera.
Ejemplo

```
import miPaquete.MiClase;
import nombre1.nombre2.miPaquete.TuClase;

class OtraClase
{
  /* Aqui se hace uso de la clase 'Miclase' y de la
  clase 'TuClase' */
  ...
}
```

Para importar todas las clases que están en un paquete, se utiliza el asterisco (*).

Ejemplo

```
import miPaquete.*;
```

otros ejemplos

```
package arraylist011;
import java.util.ArrayList;

package ejer10gui2;
import java.io.*;
```

Si no se utiliza la sentencia package para indicar a que paquete pertenece una clase, ésta terminará en el package por default, el cual es un paquete que no tiene nombre.

Ejemplo

```
/* Usuario5.java */

package paquete1;

class Usuario5
{
   static char MAS = 'm';
   static char FEM = 'f';

   String nombre;
   int edad;
   String direccion;
   char sexo;

   Usuario5( )
   {
     nombre = null;
     edad = 0;
     direccion = null;
     sexo = '\0';
   }

   Usuario5(String nombre, int edad, String direccion,char sexo)
   {
     this.nombre = nombre;
     this.edad = edad;
     this.direccion = direccion;
     this.sexo = sexo;
   }

   Usuario5(Usuario5 usr)
   {
     nombre = usr.getNombre();
     edad = usr.getEdad();
     direccion = usr.getDireccion();
     sexo = usr.getSexo();
   }
```

```
void setNombre(String n)
{
  nombre = n;
}

String getNombre()
{
  return nombre;
}

void setEdad(int e)
{
  edad = e;
}

int getEdad()
{
  return edad;
}

void setDireccion(String d)
{
  direccion = d;
}

String getDireccion()
{
  return direccion;
}

void setSexo(char s)
{
  sexo = s;
}

char getSexo()
{
  return sexo;
```

```
    }

    public String toString()
    {
      return nombre;
    }
}
```

Ejemplo

```
/* ProgUsuario5.java */

package paquete1;

import java.util.Vector;

class ProgUsuario5
{
  static int NUM_USUARIOS = 0;

  static Vector usuarios = new Vector();
  /* La siguiente línea sería obligatoria si
     se omitiera la linea import java.util.Vector; */
//   static java.util.Vector usuarios = new java.util.Vector();

  String nombreObj = null;

  ProgUsuario5(String nombre)
  {
    this.nombreObj = nombre;
  }

  static int getNumUsuarios()
  {
    return NUM_USUARIOS;
  }

  static void imprimeUsuario(Usuario5 usr)
  {
```

```
    System.out.println("\nNombre: " + usr.nombre );
    System.out.println("Edad: " + usr.getEdad() );
    System.out.println("Sexo: " + usr.getSexo() );
    System.out.println("Direccion: " + usr.getDireccion()
);
  }

  void addUsuario(Usuario5 usr)
  {
    usuarios.addElement(usr);
    System.out.print(usr.toString( )+ " agregado por el "+ this.toString()
+",");
    NUM_USUARIOS ++;
  }

  void delUsuario(Usuario5 usr)
  {
    boolean b = usuarios.removeElement(usr);
    if( b == true )
    {
      NUM_USUARIOS--;
      System.out.print(usr.toString( )+ " eliminado por el "+ this.toString()
+",");
    }
    else System.out.println("No se pudo eliminar al usuario.");
  }

  public String toString()
  {
    return nombreObj;
  }

  public static void main(String args[])
  {

    ProgUsuario5 obj1 = new ProgUsuario5("objeto1");
    ProgUsuario5 obj2 = new ProgUsuario5("objeto2");

    Usuario5 usr1,usr2,usr3,usr4;
```

```
    usr1 = new Usuario5( );
    usr2 = new Usuario5("Usuario B",24,"La direccion A",Usuario5.FEM);
    usr1 = new Usuario5(usr2);
    usr1.setNombre("Usuario A");
    usr3 = new Usuario5("Usuario C",35,"La direccion C",Usuario5.MAS);

    usr4 = new Usuario5("Usuario D",15,"La direccion D",Usuario5.MAS);


    obj1.addUsuario(usr1);
    System.out.println( "\t Total: " +ProgUsuario5.getNumUsuarios()
);
    obj2.addUsuario(usr2);
    System.out.println( "\t Total: " +obj1.getNumUsuarios()
);
    obj1.addUsuario(usr3);
    System.out.println( "\t Total: " +ProgUsuario5.NUM_USUARIOS
);
    obj2.addUsuario(usr4);
    System.out.println( "\t Total: " +getNumUsuarios() +"\n");

    obj2.delUsuario(usr4);
    System.out.println( "\t Total: " +ProgUsuario5.getNumUsuarios()
);
    obj1.delUsuario(usr3);
    System.out.println( "\t Total: " +obj1.getNumUsuarios()
);
    obj2.delUsuario(usr2);
    System.out.println( "\t Total: " +ProgUsuario5.NUM_USUARIOS
);
    obj1.delUsuario(usr1);
    System.out.println( "\t Total: " +getNumUsuarios()
+"\n");
  }
}
```

Control de acceso a miembros de una clase

Los modificadores más importantes desde el punto de vista del diseño de clases y objetos, son los que permiten controlar la visibilidad y acceso a los métodos y variables que están dentro de una clase.

Uno de los beneficios de las clases, es que pueden proteger a sus variables y métodos (tanto de instancia como de clase) del acceso desde otras clases.

Java soporta cuatro niveles de acceso a variables y métodos. En orden, del más público al menos público son: público (public), protegido (protected), sin modificador (también conocido como(*package*) y privado (private).

La siguiente tabla muestra el nivel de acceso permitido por cada modificador:

	public	**protected**	**(sin modificador)**	**private**
Clase	SI	SI	SI	SI
Subclase en el mismo paquete	SI	SI	SI	NO
No-Subclase en el mismo paquete	SI	SI	SI	NO
Subclase en diferente paquete	SI	SI/NO **(*)**	NO	NO

No-Subclase en diferente paquete (Universo)	SI	NO	NO	NO

(*) Los miembros (variables y métodos) de clase (static) si son visibles. Los miembros de instancia no son visibles.

Como se observa de la tabla anterior, una clase se ve a ella misma todo tipo de variables y métodos (desde los public hasta los private); las demas clases del mismo paquete (ya sean subclases o no) tienen acceso a los miembros desde los public hasta los sin-modificador. Las subclases de otros paquetes pueden ver los miembros public y a los miembros protected, éstos últimos siempre que sean static ya de no ser así no serán visibles en la subclase (Esto se explica en la siguiente página). El resto del universo de clases (que no sean ni del mismo paquete ni subclases) pueden ver sólo los miembros public.

¿Qué son las colecciones de Objetos ?

Una colección representa un grupo de objetos. Esto objetos son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos. En Java, se emplea la interfaz genérica **Collection** para este propósito. Gracias a esta interfaz, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser: añadir, eliminar, obtener el tamaño de la colección… Partiendo de la interfaz genérica **Collection** extienden otra serie de interfaces genéricas. Estas subinterfaces aportan distintas funcionalidades sobre la interfaz anterior.

Por ejemplo una muy utilizada como por ejemplos el **List (listas) y los iteradores.**

Las listas en Java son variables que permiten almacenar grandes cantidades de datos. Son similares a los Arrays o a las Matrices**.**

Trabajando con arrays es frecuente cometer errores al utilizar los índices; por ejemplo al intentar guardar un elemento en una posición que no existe (índice fuera de rango). Aunque las colecciones permiten el uso de índices, no es necesario indicar los siempre. Por ejemplo, en una colección del tipo ArrayList, cuando hay que añadir el elemento "Amapola", se puede hacer simplemente flores.add("Amapola"). Al no especificar índice, el elemento "Amapola" se añadiría justo al final de flores independientemente del tamaño y del número de elementos que se hayan introducido ya. La clase ArrayList es muy similar a la clase Vector. Ésta última está obsoleta y, por lo tanto, no se recomienda su uso.

Entonces… ¿cómo declaramos una lista? seguimos la siguiente estructura:

List ejemploLista = new ArrayList();

Este tipo de lista puede almacenar cualquier tipo de dato, pero este enfoque ya ha quedado obsoleto. Se prefiere que se designe el tipo de dato que se va a almacenar. Entonces para declarar una lista donde guardaremos datos tipo **String**, hacemos lo siguiente:

List<String> ejemploLista = new ArrayList<String>();

Con nuestra lista creada podemos empezar a introducir datos en ella.

Supongamos que queremos **agregar** los siguientes nombres: **Juan, Pedro, José, María, Sofía**.

Hacemos lo siguiente:

```
List<String> ejemploLista = new ArrayList<String>();
    ejemploLista.add("Juan");
    ejemploLista.add("Pedro");
    ejemploLista.add("José");
    ejemploLista.add("María");
    ejemploLista.add("Sofía");
```

Es posible además agregar el índice en el que queremos agregar dicho elemento. Podemos **obtener la cantidad de elementos** que posee la lista:

```
ejemploLista.size();
```

Para consultarla haríamos así:

```
ejemploLista.get(0);
```

Donde 0 es el índice en el que se encuentra la información que queremos. En este caso, el índice 0 vendría siendo **Pedro**.

Si queremos **eliminar** determinado elemento:

```
ejemploLista.remove(0);
```

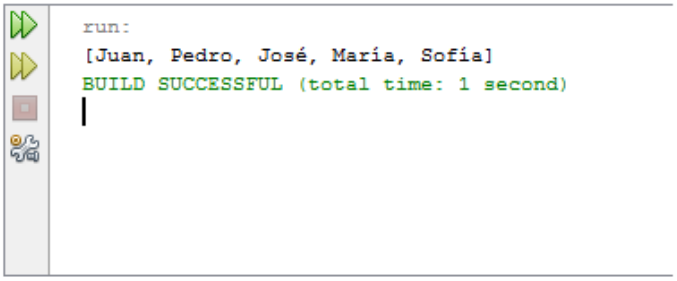
Donde nuevamente el 0 representa el índice que queremos eliminar. Otra forma de eliminar un registro es por su nombre:

```
ejemploLista.remove("Juan");
```

Si deseamos **imprimir en consola** los elementos de la lista:

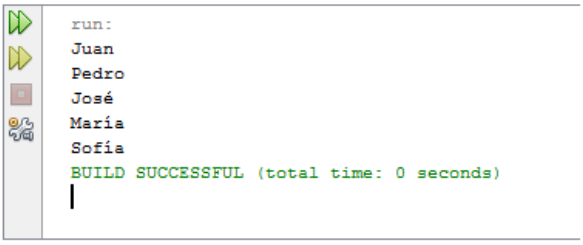
```
System.out.println(ejemploLista);
```

Y veríamos esto….



También podemos imprimir todos los elementos de la lista de forma individual con la ayuda de un ciclo for.

```
for (int i = 0; i <= ejemploLista.size() - 1; i++) {
        System.out.println(ejemploLista.get(i));
}
```



Ordenacióndeun ArrayList Los elementos de una lista se pueden ordenar con el método **sort**. El formato es el siguiente:

Collections.sort(lista);

Observa que sort es un método de clase que está definido en Collections. Para poder utilizar este método es necesario incluir la línea

import java.util.Collections;

al principio del programa. A continuación se muestra un ejemplo del uso de sort

```
import java.util.Collections;

import java.util.ArrayList;

public class EjemploArrayList071 {
```

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

ArrayList<Integer> a = new ArrayList<Integer>();

a.add(67);

a.add(78);

a.add(10);

a.add(4);

System.out.println("\nNúmeros en el orden original:");

for (int numero: a) {

 System.out.println(numero);

}

Collections.sort(a);

System.out.println("\nNúmeros ordenados:");

for (int numero: a) {

System.out.println(numero);

}

}

}
```

Con Iterator:

También podríamos usar un iterador para recorrer la lista e imprimir todos sus valores:

```
1       Iterator i = ejemploLista.iterator();
```

```
2       while(i.hasNext())
3       {
4          System.out.println(i.next());
5       }
```

Para eliminar todos los elementos de la lista usamos:

```
1 ejemploLista.clear();
```

Si deseamos saber si nuestra lista contiene algún elemento utilizamos:

ejemploLista.isEmpty();

Esto nos devolverá un **true** o un **false.** En caso de que contenga algún elemento podemos verificar si entre esos elementos tenemos alguno en específico. Por ejemplo si queremos saber si en nuestra lista está escrito el nombre de José, utilizamos:

ejemploLista.contains("José");

Esto también nos devolverá un **true** o un **false**. Y si por alguna razón queremos modificar algún dato de nuestra lista, por ejemplo el índice 1 que contiene el nombre de Pedro, utilizamos el siguiente método:

```
ejemploLista.set(1, "Félix");
```

Habremos cambiado el nombre en el índice 1 (Pedro) por el nuevo nombre (Félix).

Si queremos extraer una lista que contenga los nombres entre un índice y otro podemos utilizar:

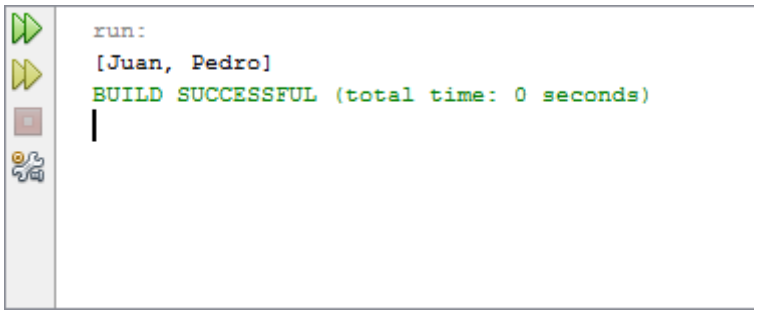
ejemploLista.subList(0, 2)

Veamos el siguiente ejemplo:

```
List<String> ejemploLista = new ArrayList<String>();
ejemploLista.add("Juan");
ejemploLista.add("Pedro");
ejemploLista.add("José");
ejemploLista.add("María");
ejemploLista.add("Sofía");
```

```
System.out.println(ejemploLista.subList(0, 2));
```

El resultado de esto es:



Nótese que se transforman los elementos desde el índice inicial (0) hasta el elemento anterior al índice final (2), despreciando el elemento que se encuentra en el índice final.

Estos son los usos elementales que le podemos dar a las listas