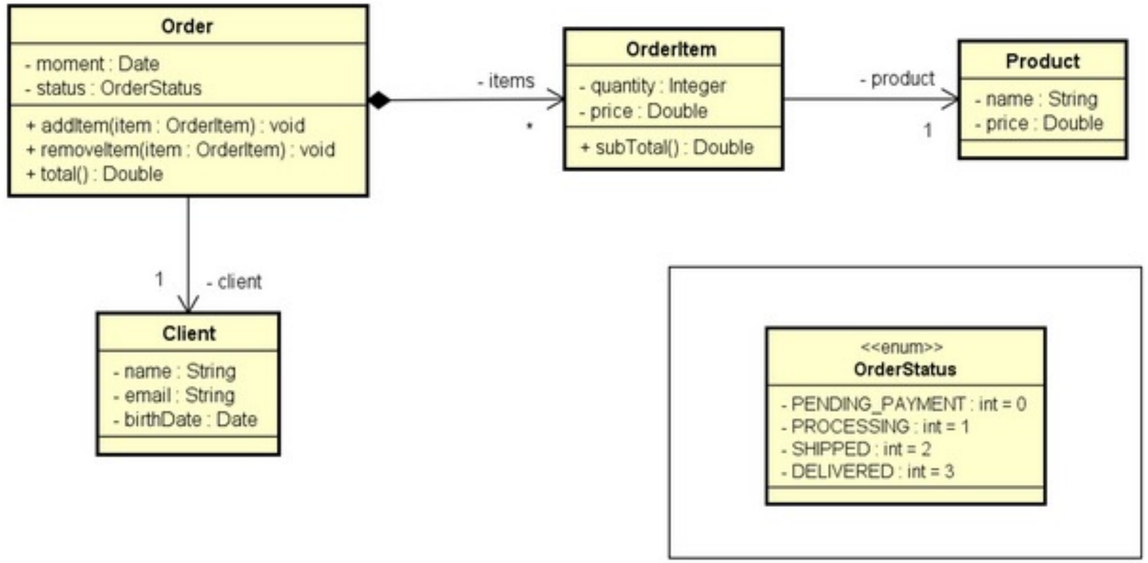
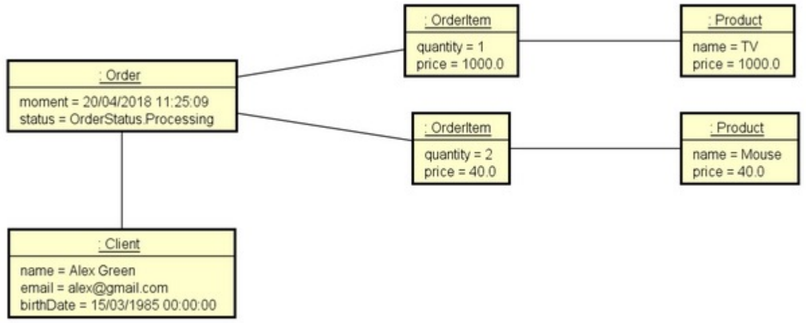
Fundamentos de POO y Maven

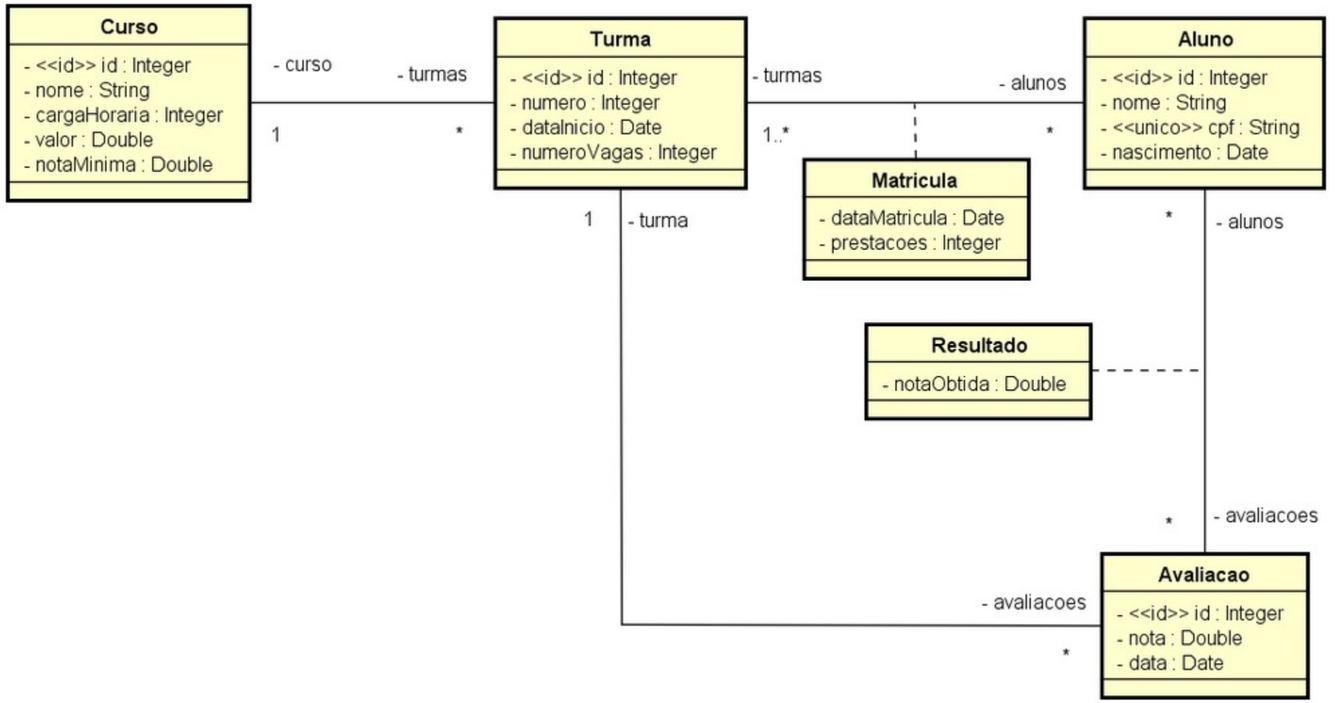
Sesión 2

Diagramas de Clase

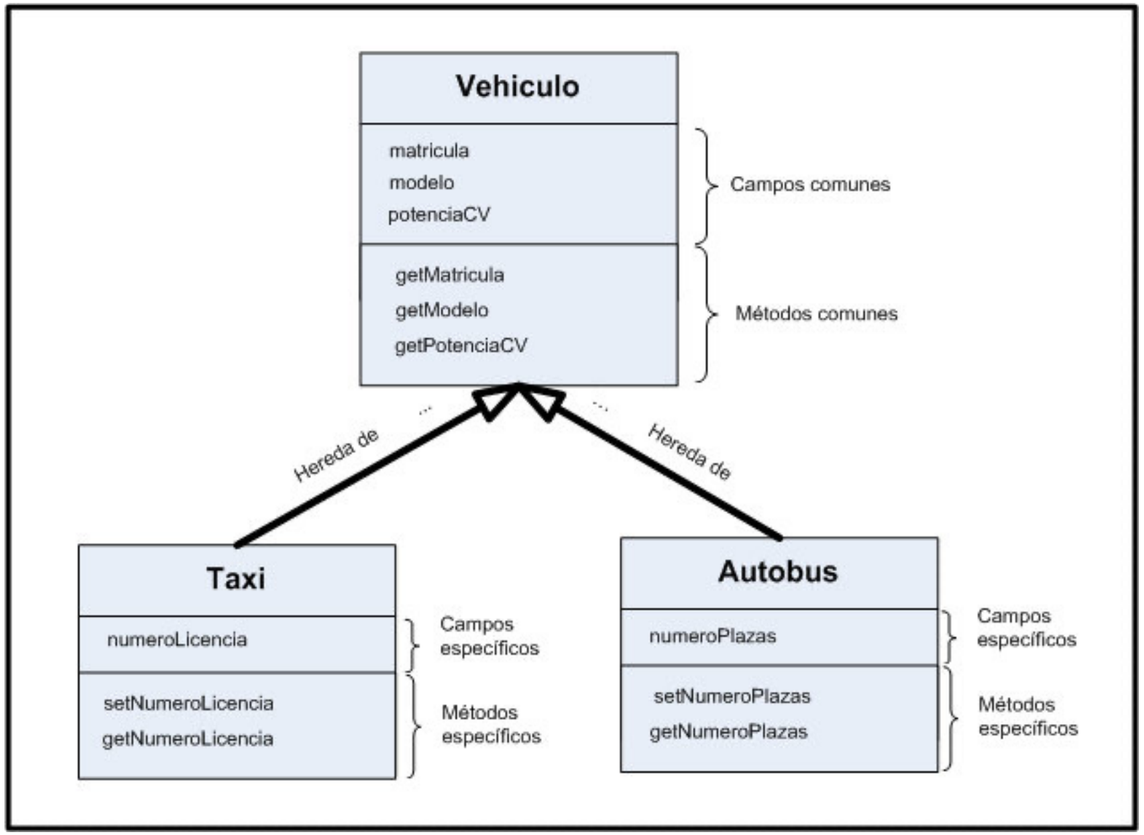




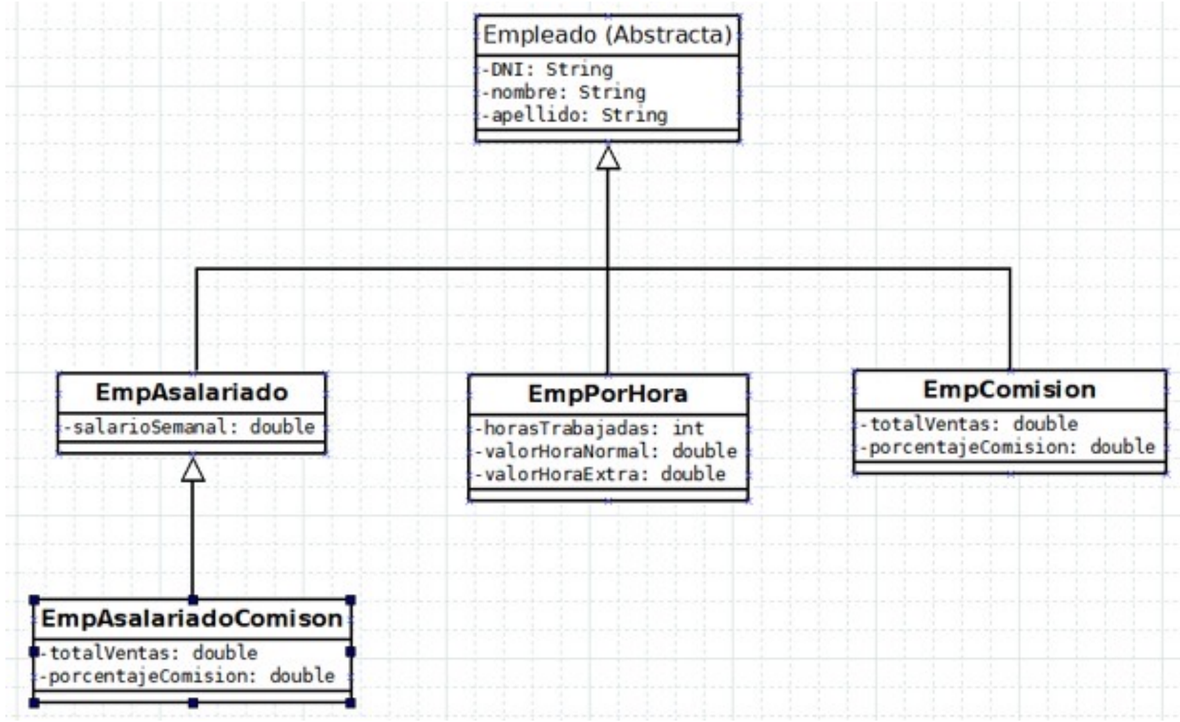
Diagramas de Clase



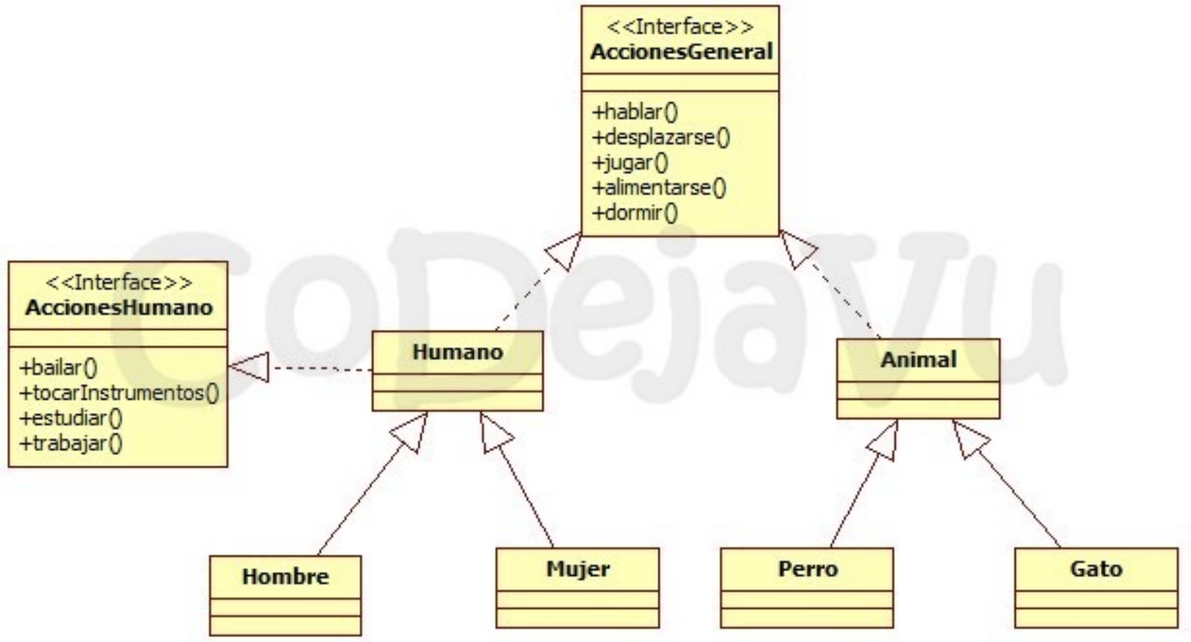
Herencia


Vehiculo
matricula
modelo
potenciaCV
getMatricula
getModelo
getPotenciaCV
Campos comunes
Métodos comunes
Hereda de
Hereda de
Taxi
numeroLicencia
setNumeroLicencia
getNumeroLicencia
Campos específicos
Métodos específicos
Autobus
numeroPlazas
setNumeroPlazas
getNumeroPlazas
Campos específicos
Métodos específicos

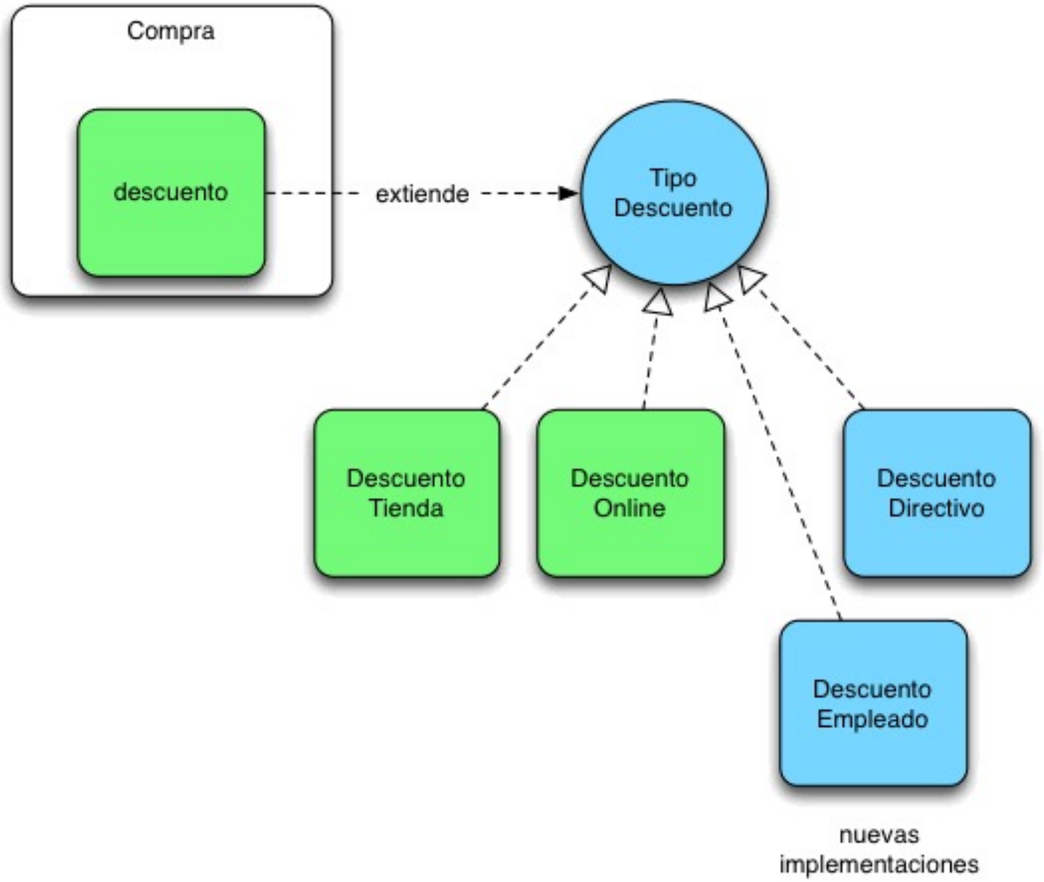
Herencia Clase Abstracta



Interfaces


<<Interface>>
AccionesGeneral
+hablar()
+desplazarse()
+jugar()
+alimentarse()
+dormir()
<<Interface>>
AccionesHumano
+bailar()
+tocarInstrumentos()
+estudiar()
+trabajar()
Humano
Animal
Hombre
Mujer
Perro
Gato

Interfaces


Compra
descuento
extiende
Tipo Descuento
Descuento Tienda
Descuento Online
Descuento Directivo
Descuento Empleado
nuevas implementaciones

Generic

Generics permiten que las clases, interfaces y métodos puedan ser parametrizados por tipo. Sus beneficios son

- Reuso
- Type safety
- Performance

Uso común : Colecciones

Convención de nombres

Dado que se trata de tipos genéricos, su nomenclatura no afecta su comportamiento y podría designarse cualquier nombre para un genérico en Java. Sin embargo, existen convenciones para los nombres de estos tipos cuyo objetivo es mejorar la legibilidad e interpretación del código. Algunas de importancia son:

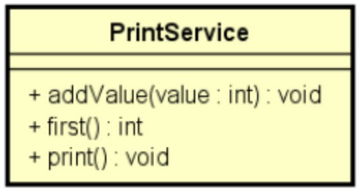
- E – Element.
- K – Key.
- V – Value.
- N – Number.
- T – Type.
- S, U, V, y así sucesivamente, para más tipos.

Generic

Reuso

Se desea hacer un programa que lee una cantidad N, y luego N números enteros. Al final, imprima estos números de forma organizada de acuerdo ejemplo. A continuación, indicar cuál fue el primer valor informado.

```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```

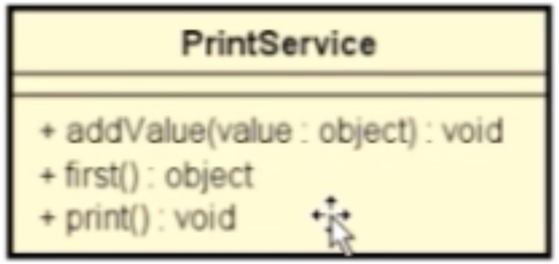


Type safety & performace

Generic

Se desea hacer un programa que lee una cantidad N, y luego N números enteros. Al final, imprima estos números de forma organizada de acuerdo ejemplo. A continuación, indicar cuál fue el primer valor informado.

```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```

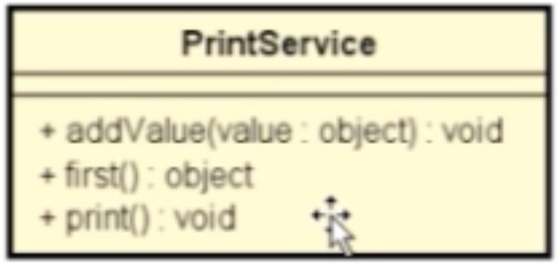


Type safety & performace

Generic

Se desea hacer un programa que lee una cantidad N, y luego N números enteros. Al final, imprima estos números de forma organizada de acuerdo ejemplo. A continuación, indicar cuál fue el primer valor informado .

```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```

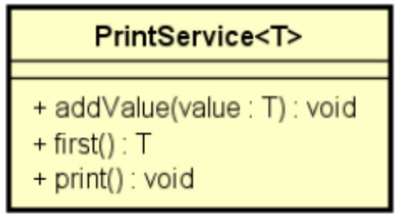


Generic

Solución con Generics

Se desea hacer un programa que lee una cantidad N, y luego N números enteros. Al final, imprima estos números de forma organizada de acuerdo ejemplo. A continuación, indicar cuál fue el primer valor informado.

```
How many values? 3
10
8
23
[10, 8, 23]
First: 10
```

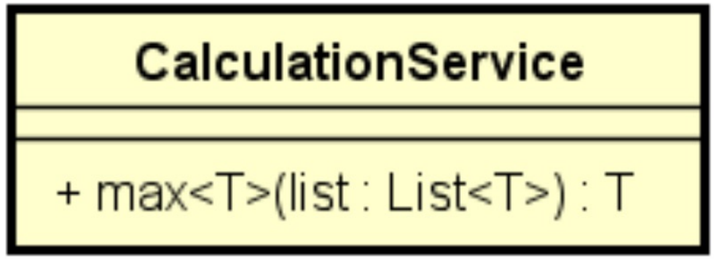


Generic

Problema

Una empresa de consultoría quiere evaluar el desempeño de productos, empleados, entre otras cosas. Uno de los cálculos que necesita es encontrar el máximo valor de un conjunto de elementos. Cree un programa que lea un conjunto de productos de un archivo, como ejemplo, y luego muestre el mayor precio de ellos.

```
Computer,890.50
IPhone X,910.00
Tablet,550.00
Most expensive:
IPhone, 910.00
```



Generic

Wildcard types (tipo comodin)

Generic son invariantes
List <Object> no es el supertipo de ningún tipo de lista:

```
List<Object> myObjs = new ArrayList<Object>();
List<Integer> myNumbers = new ArrayList<Integer>();
myObjs = myNumbers; // error de compilación
```

El supertipo de cualquier tipo de lista es List <?>. Este es un tipo de comodín:

```
List<?> myObjs = new ArrayList<Object>();
List<Integer> myNumbers = new ArrayList<Integer>();
myObjs = myNumbers;
```

Generic

Con los tipos de comodines podemos crear métodos que reciban un genérico de "cualquier tipo":

```
package application;
import java.util.Arrays;
import java.util.List;
public class Program {
	public static void main(String[] args) {
		List<Integer> myInts = Arrays.asList(5, 2, 10);
		printList(myInts);
		List<String> myStrs = Arrays.asList(“Maria”,”Alex”);
		printList(myStrs);
	}

	public static void printList(List<?> list) {
		for (Object obj : list) {
		  System.out.println(obj);
		}
	}
}
```

Generic

Sin embargo, no es posible agregar datos a una colección de comodines

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

public class Program {
	public static void main(String[] args) {
		List<?> list = new ArrayList<Integer>();
		list.add(3); // error compilación
	}
}
```

El compilador no sabe cuál es el tipo específico del que se creó una instancia de la lista.

Generic

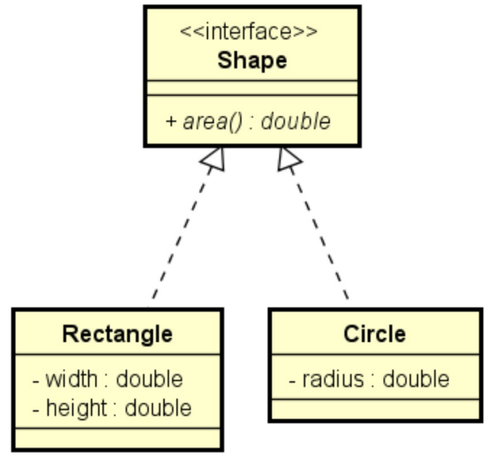
Bounded wildcards (tipo comodin delimitado)

Vamos a hacer un método para devolver la suma de las áreas de una lista de figuras.

Nota 1: soluciones inadecuadas:

```
public double totalArea(List<Shape> list)
public double totalArea(List<?> list)
```

Nota 2: no podremos agregar elementos a la lista de métodos



Equals

Método que compara si el objeto es igual a otro, devolviendo verdadero o falso.

```
String a = "Maria";
String b = "Alex";

System.out.println(a.equals(b));
```

HashCode

Método que devuelve un número entero que representa un código generado a partir de la información del objeto.

```
String a = "Maria";
String b = "Alex";

System.out.println(a.hashCode());
System.out.println(b.hashCode());
```

HashCode

Si el código hash de dos objetos es diferente, entonces los dos objetos son diferentes

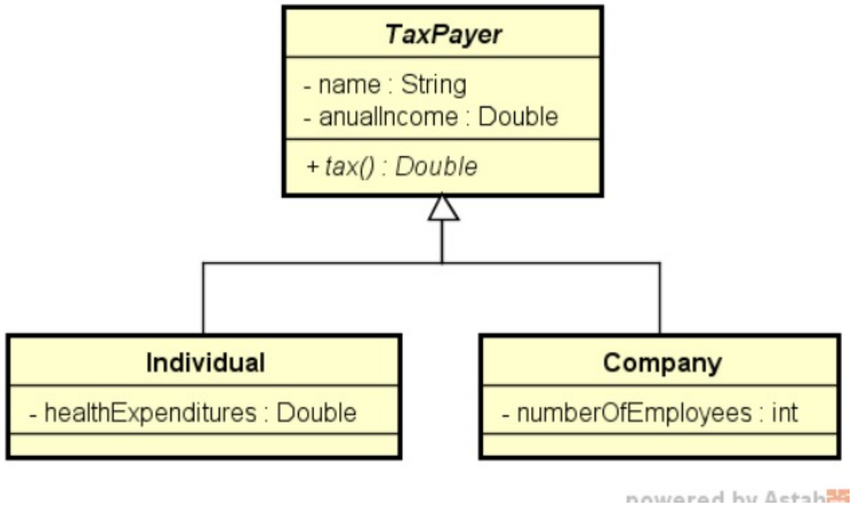
Si el código de dos objetos es el mismo, lo más probable es que los objetos sean iguales

Colecciones

http://www.cpcppw.es/java-colecciones.html
https://www.baeldung.com/java-collections

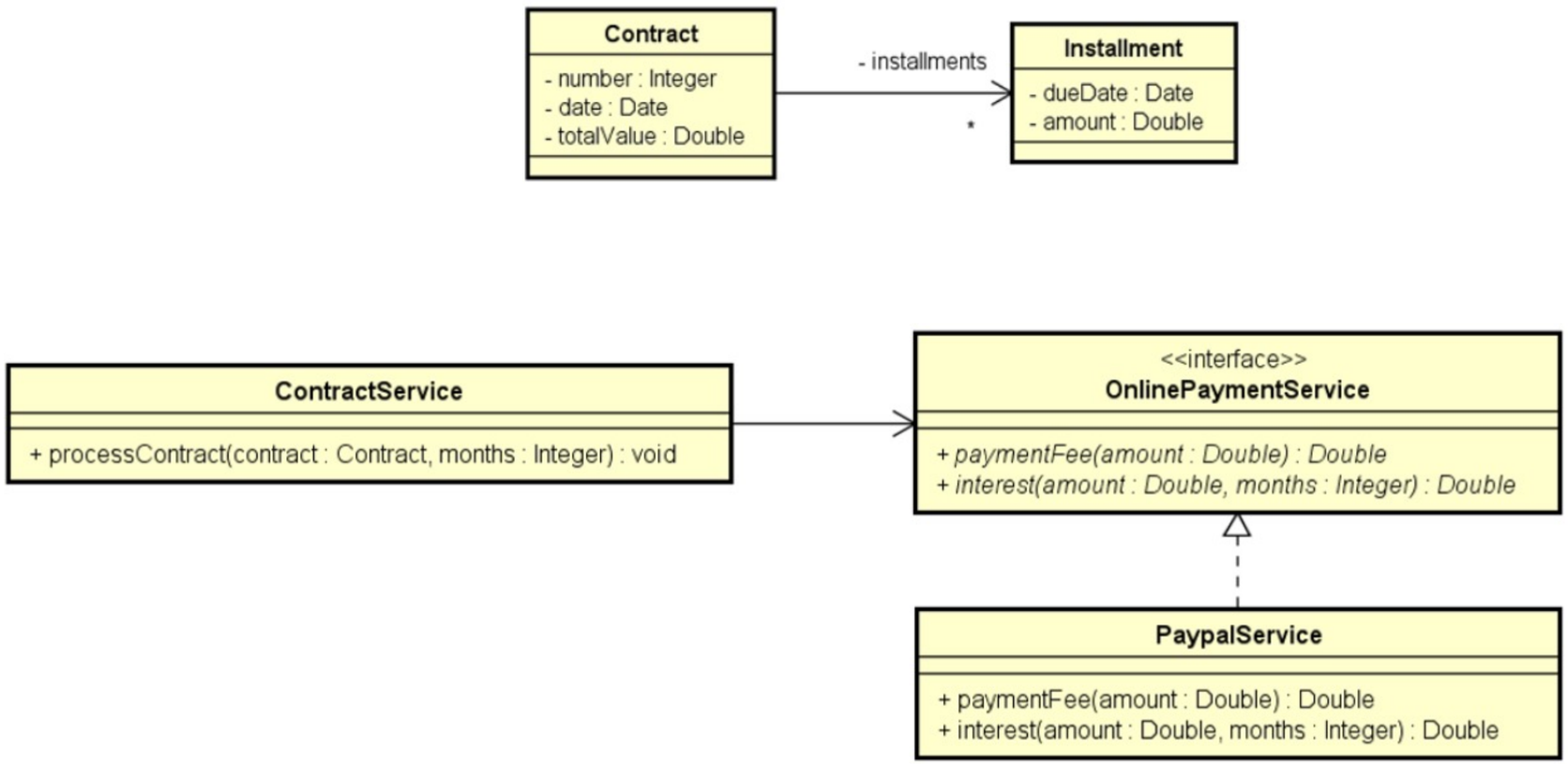
Demo

Herencia – Polimorfismo



Demo

Interfaces



Demo

List