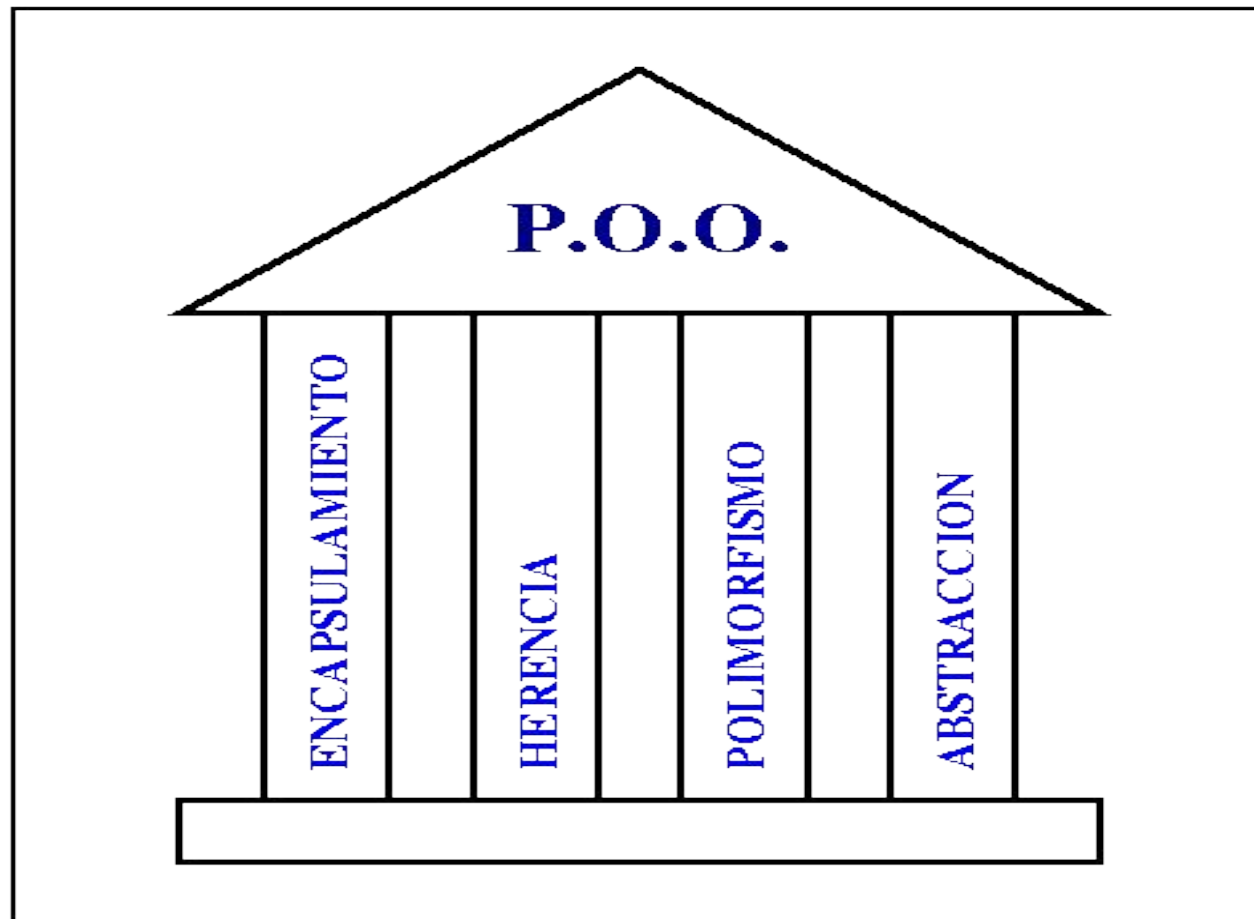




Argentina
programa
4.0

Pilares de la POO

“Desarrollador Java Inicial”

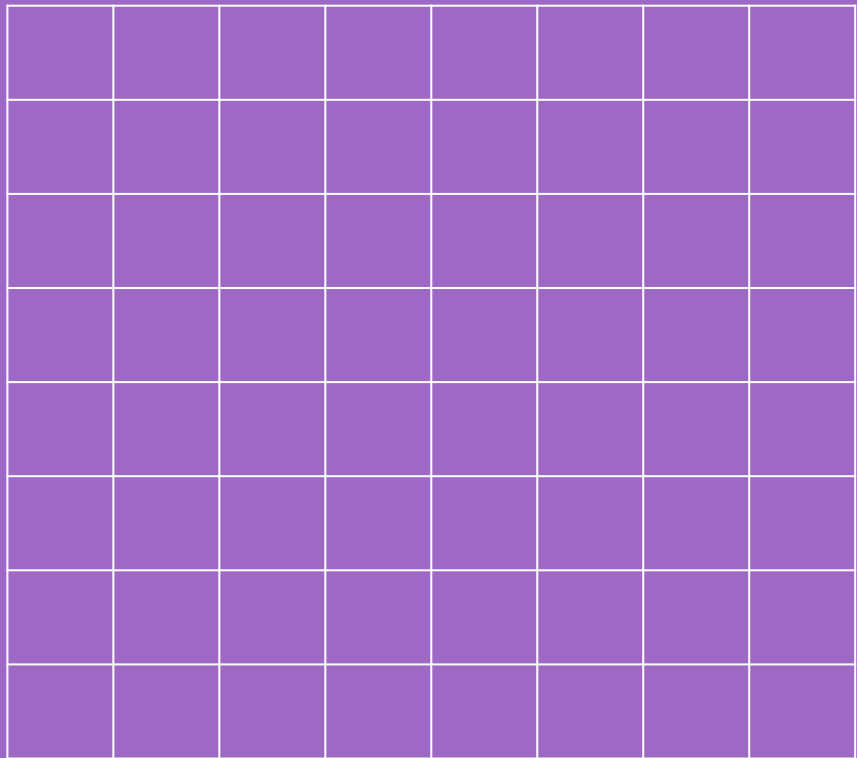


Agenda

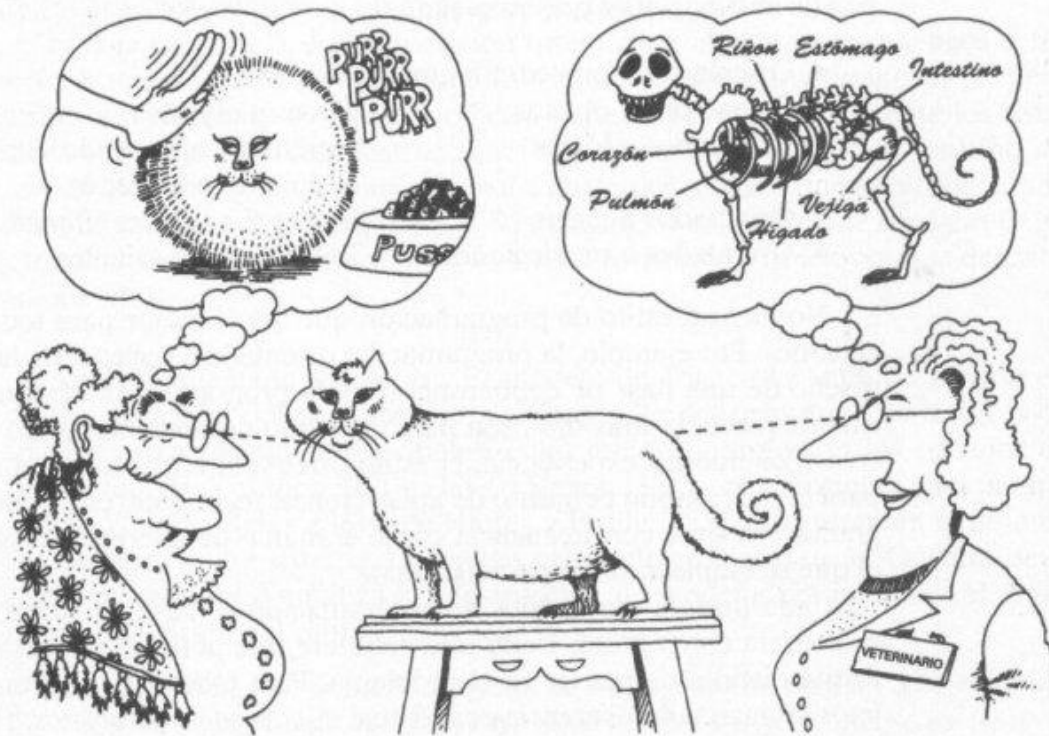
- Abstracción
- Encapsulamiento
- Herencia
 - Concepto
 - Reglas
 - Clase Object y Wrappers
 - Concepto de interfaz
- Polimorfismo
 - Diferencias con Sobrecarga
 - Diferencias con Herencia
- Incorporando objetos a la solución de un problema



Abstracción

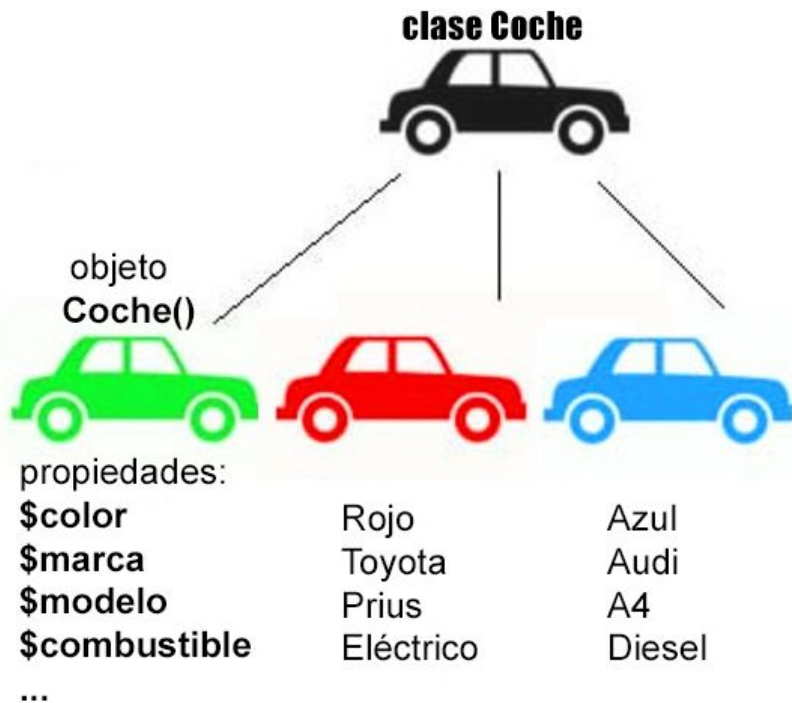


Abstracción



La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

Abstracción

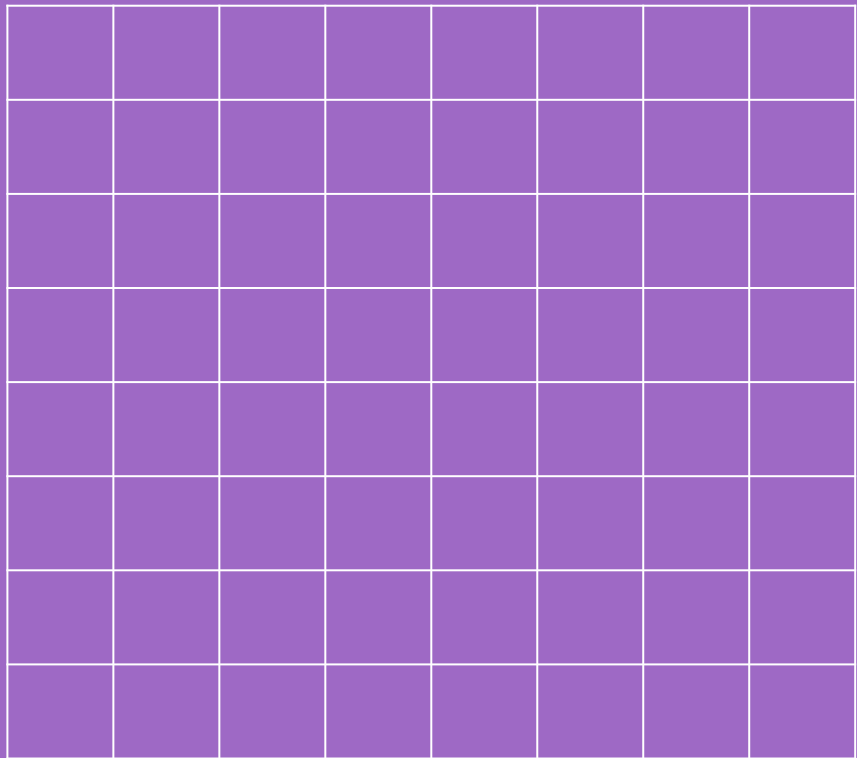


MÉTODOS

llenarDeposito()
arrancarMotor()
frenar()
acelerar()
tocarClaxon()
...

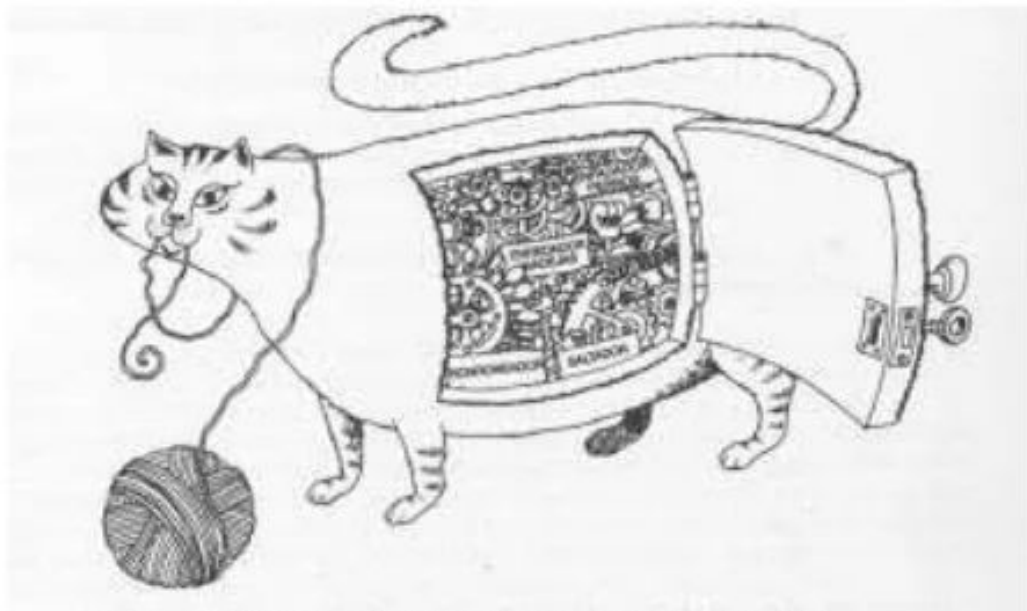


Encapsulamiento




Encapsulamiento

- Facilita el manejo de la complejidad
- Sólo se conoce el comportamiento pero no los detalles internos
- Nos interesa conocer qué hace la Clase pero no saber cómo lo hace



Encapsulamiento



```
class Account{  
    private int account_number;  
    private int account_balance;  
  
    public void show Data(){  
        // code to show data  
    }  
  
    public void deposit(int a){
```

```
class Hacker{  
    Account a= new Account ();  
    a.account_balance= -100;
```

```
class Hacker{  
    Account a= new Account ();  
    a.account_balance= -100;  
  
}
```

Encapsulamiento

```
class Hacker{  
    Account a= new Account  
    a.account_balance= -100  
    a.deposit(-100);  
}
```

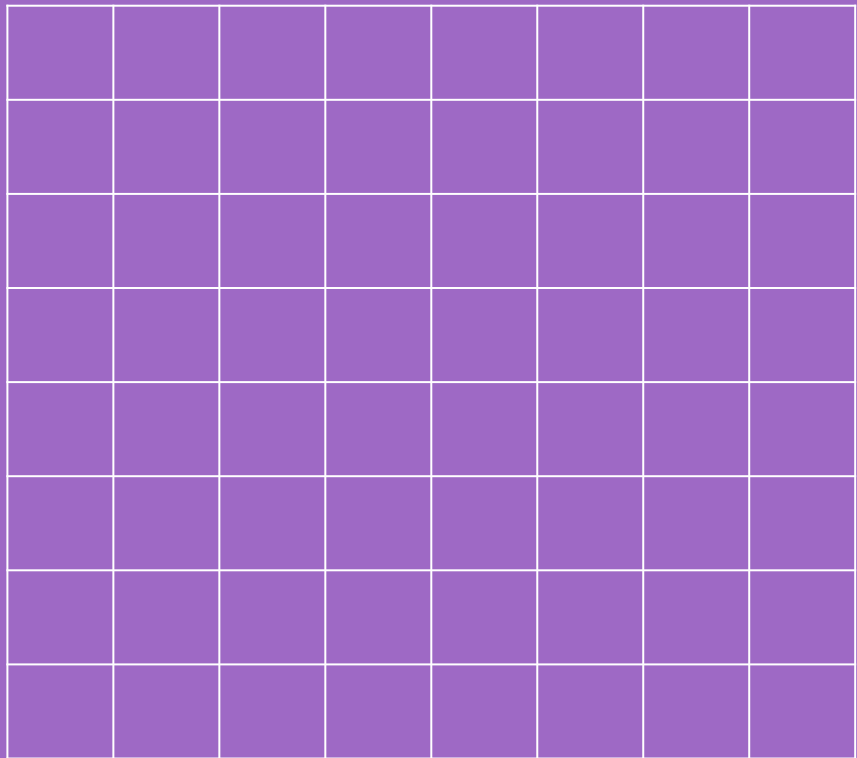


```
public void deposit(int a){  
    if (a<0){  
        //show error  
    }  
    else  
        account_balance=  
    }  
}
```

la implementación
del método ha
verificado valores
negativos (-100) y
arroja un

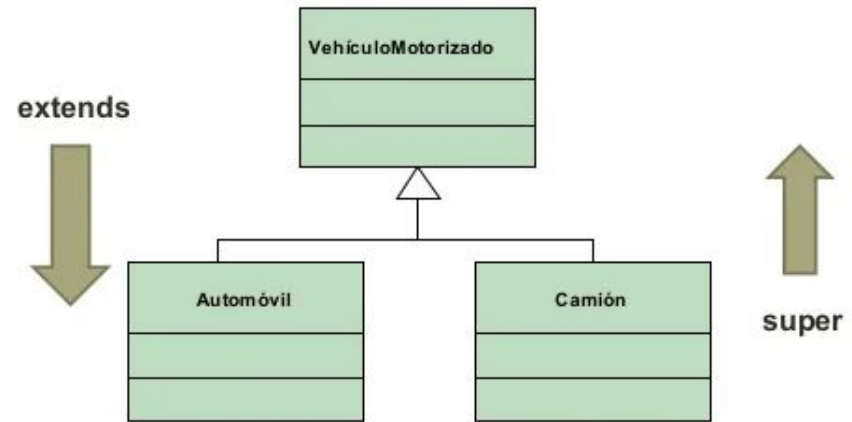


Herencia e interfaces



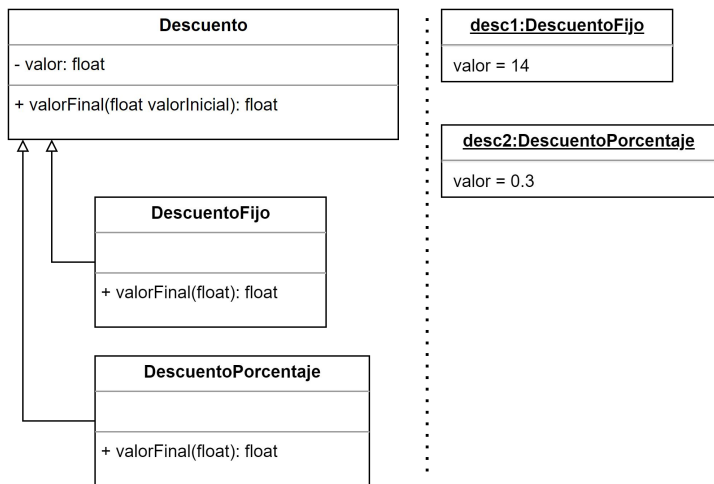
Clase - Herencia

- Concepto: Si X hereda de Y, “una instancia de X es también una instancia de Y”. Por ejemplo:
 - Un Perro es un Mamífero
 - Un Docente es un Empleado y un Empleado es una Persona
- Herencia Simple
 - Una clase puede heredar sólo de una
- En Java:
 - Se heredan los atributos y métodos de instancia
 - Todas las clases heredan de la clase Object
 - Los atributos / métodos de clase **no** se heredan
 - Los constructores **no** se heredan



Clase - Herencia - Ejemplo

Por ejemplo, aquí tenemos una jerarquía de descuentos. Hay 2 descuentos, el “fijo” y por “porcentaje”. Los objetos pueden ser instancia de uno u de otro, no de los 2.



```
public abstract class Descuento {
    private float valor;
    public float getValorDesc() {
        return valor;
    }
    public void setValorDesc(float valor) {
        this.valor = valor;
    }
    public abstract float valorFinal(
        float valorInicial);
}

-----

public class DescuentoFijo extends Descuento {
    @Override
    public float valorFinal(float valorInicial) {
        return valorInicial - this.getValorDesc();
    }
}

-----

public class DescuentoPorcentaje extends Descuento {
    @Override
    public float valorFinal(float valorInicial) {
        return valorInicial - (valorInicial *
            this.getValorDesc());
    }
}
```

Clase - Herencia - Ejemplo

Agrandando vamos a suponer que tenemos un producto

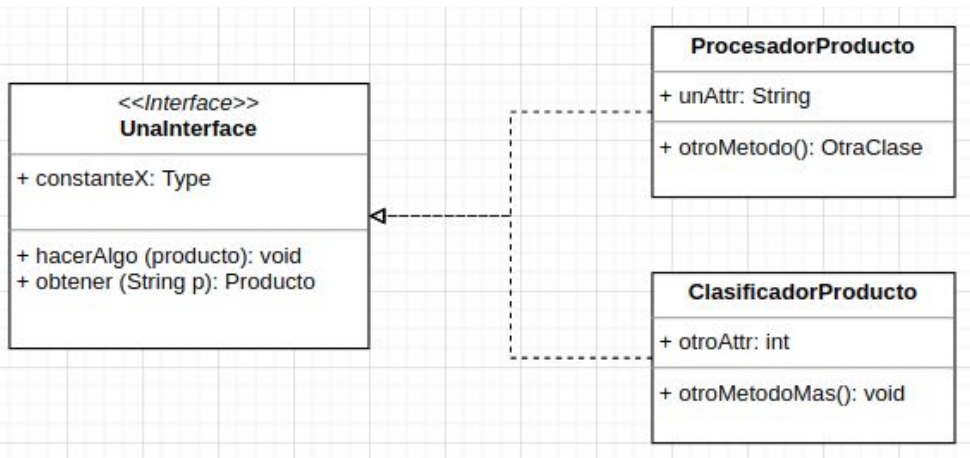
```
Descuento desc1 = new DescuentoFijo();
desc1.setValor(14.0);
System.out.println(desc1.valorFinal(100)); // 86
//-----
Descuento desc2 = new DescuentoPorcentaje();
desc2.setValor(0.3);
System.out.println(desc2.valorFinal(100)); // 70
```

```
public abstract class Descuento {
    private float valor;
    public float getValorDesc() {
        return valor;
    }
    public void setValorDesc(float valor) {
        this.valor = valor;
    }
    public abstract float valorFinal(
        float valorInicial);
}
-----
public class DescuentoFijo extends Descuento {
    @Override
    public float valorFinal(float valorInicial) {
        return valorInicial - this.getValorDesc();
    }
}
-----
public class DescuentoPorcentaje extends Descuento {
    @Override
    public float valorFinal(float valorInicial) {
        return valorInicial - (valorInicial *
            this.getValorDesc());
    }
}
```

Clase Object de Java

- En Java todas las clases heredan de la clase Object
- La misma tiene varios métodos, que por lo que establecimos antes, los van a compartir TODOS los objetos que creamos. En principio hablaremos de estos 2:
 - **toString:**
 - Muestra una representación en String del objeto
 - Orientado al desarrollador
 - X defecto muestra el nombre de la clase y el “identificador” del objeto
 - **equals**
 - Determina si 2 objetos son “iguales”
 - X defecto es una operación de identidad
- En general, una subclase puede “sobreescribir” la implementación de su padre (especializar el comportamiento). En particular es conveniente sobreescribir estos 2 métodos.

Interfaces



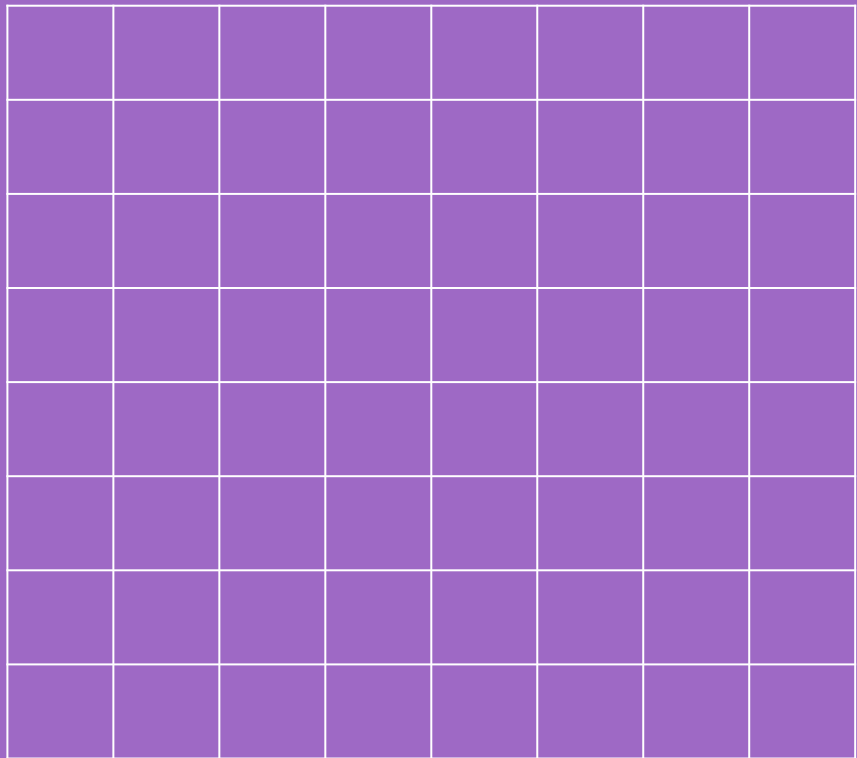
```
public interface UnaInterface {
    public void hacerAlgo (Producto p);
    public Producto obtenerProducto (String nombre);
}
```

- Contrato a cumplir
- Sin estado
- Se pueden implementar múltiples
- Se pueden definir constantes
- Semánticamente != heredar

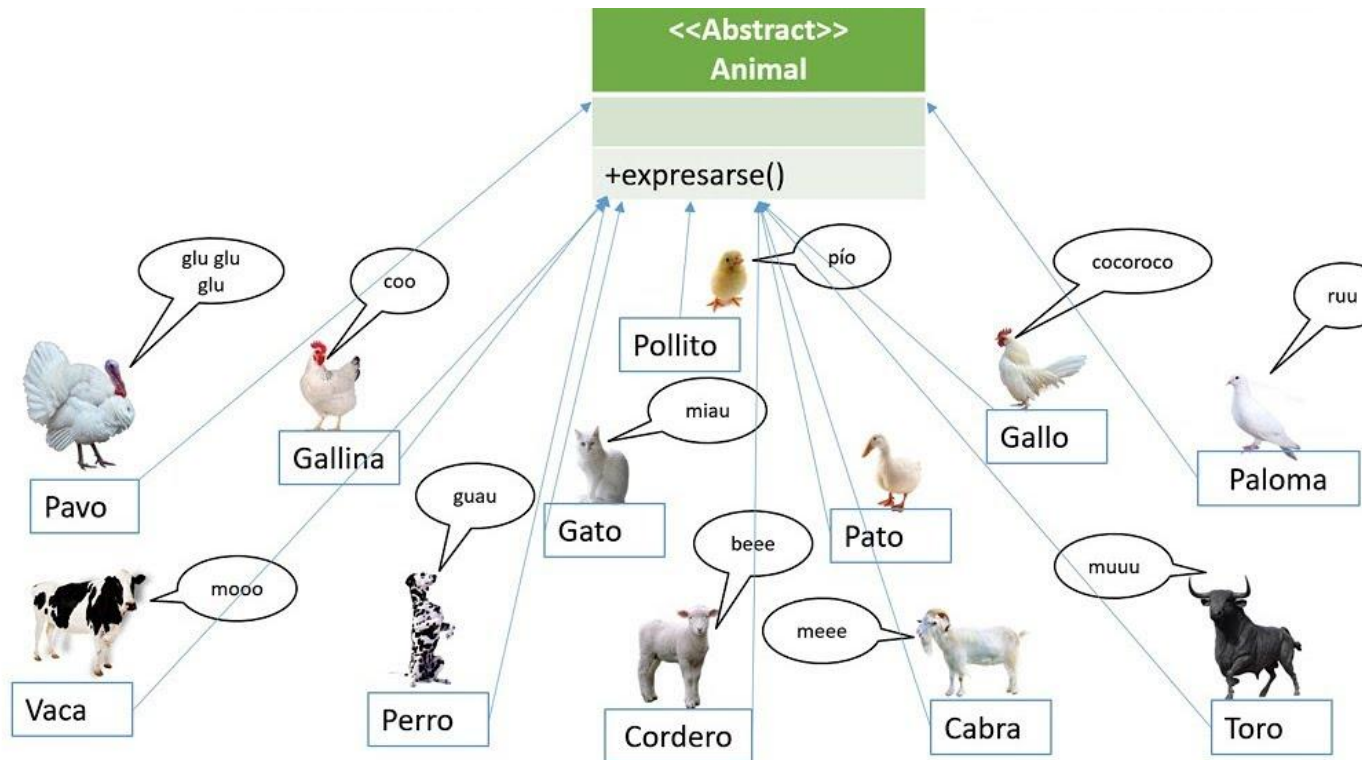
```
public class ProcesadorProducto
    implements UnaInterface {
    @Override
    public void hacerAlgo(Producto p) {
        //hacer algo...
    }
    @Override
    public Producto obtenerProducto(String nombre) {
        //retornar algo
        return algo;
    }
}
```



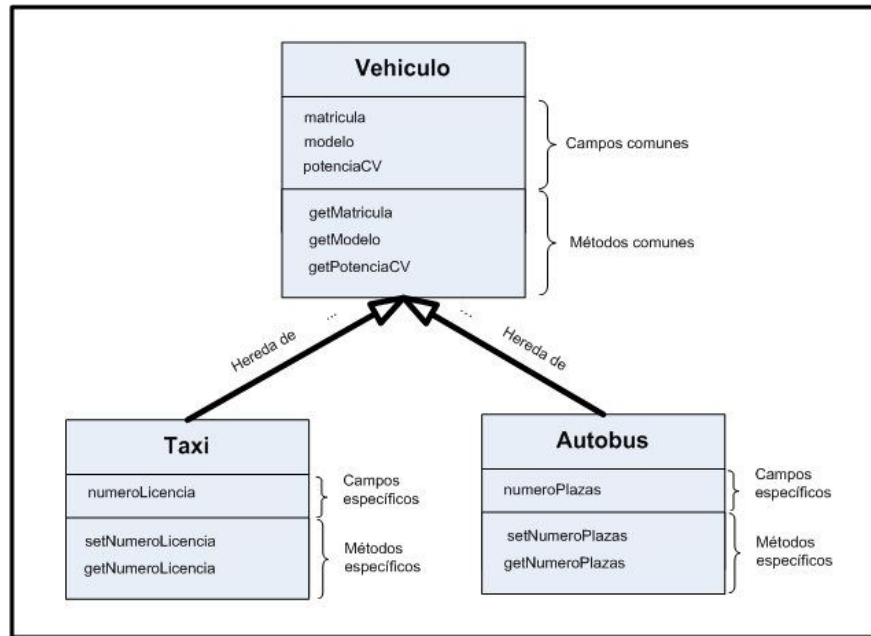
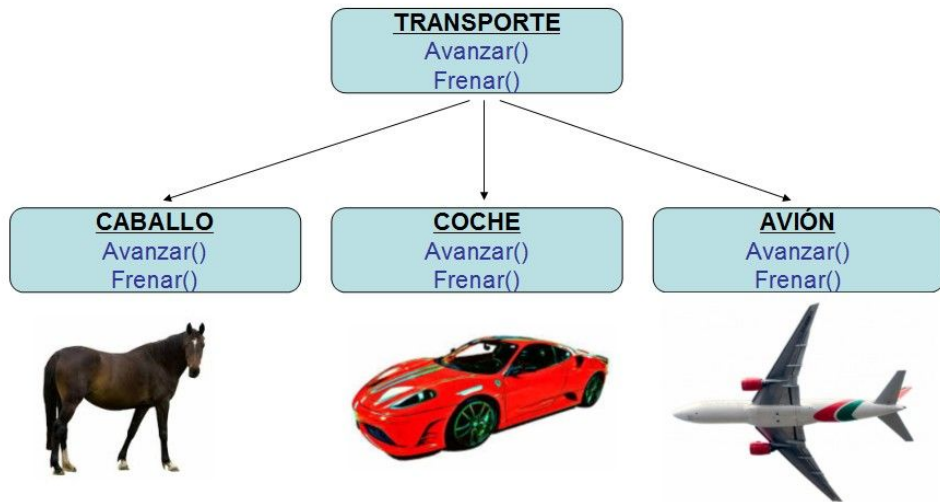

Polimorfismo



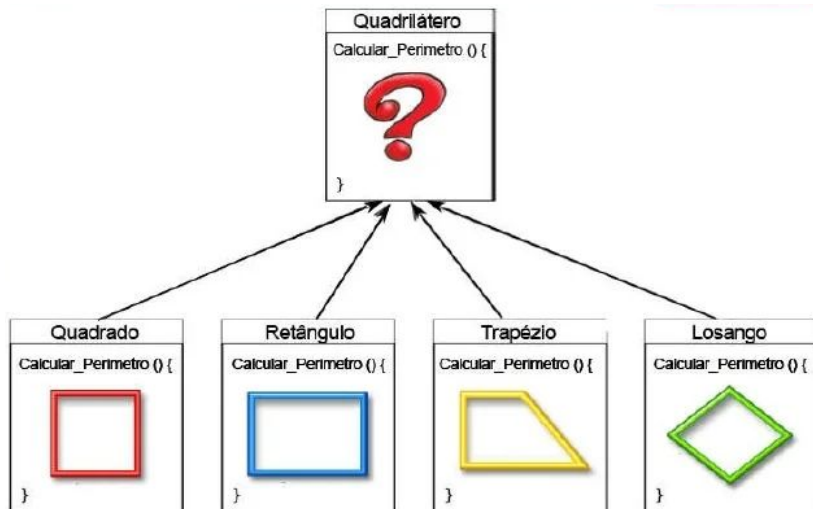
Polimorfismo



Polimorfismo y Herencia



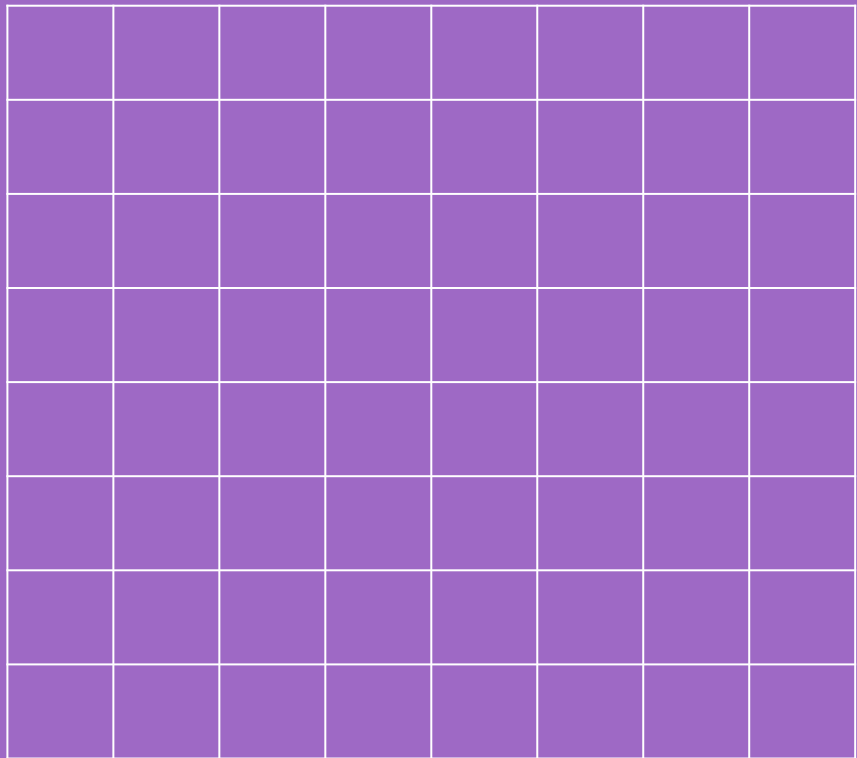
Polimorfismo y Sobrecarga



```
1 public class Calculos{
2
3     public int Suma(int a, int b){
4         return a + b;
5     }
6
7     public double Suma(double a, double b){
8         return a + b;
9     }
10
11     public long Suma(long a, long b){
12         return a + b;
13     }
14
15 }
```



Incorporando enfoque de objetos



Ejemplo - Contexto

Queremos desarrollar un “carrito de compras”. El mismo básicamente consta de los productos que un cliente seleccionó, la cantidad de cada uno, y los descuentos que aplican para dicha compra.

Nos enfocaremos en la funcionalidad “calcular precio”:

- el precio base es la suma de los precios de los productos comprados
- luego tengo descuentos que aplican al carrito en cuestión (el descuento está dado, nosotros no nos ocupamos de asignar el que corresponde)
- Los mismos pueden ser:
 - % del total
 - Descuento de monto fijo

Programación estructurada vs orientada a objetos

Programación estructurada

- Variables y Funciones
- Se estructura el flujo a medida que una función llama a otra
- Puedo tener estructuras de datos más complejas llamadas registros

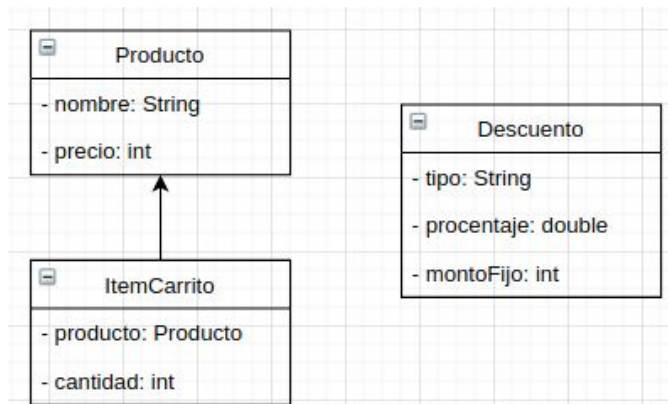
```
double calcularPrecio(int precios[], int cantidades[],String tipoDescuento,  
                     double porcentaje, int montoFijo)  
  
    int base = 0  
    for (int i = 0; i < precios.length ;i++){ base = base + precios[i] * cantidades[i]; }  
    if (tipodescuento == "%") { return base - (base * porcentaje); }  
    if (tipodescuento == "fijo") { return base - montoFijo; }
```

Programación estructurada vs orientada a objetos

Programación estructurada con registros + subrutinas

```
double calcularPrecio(ItemCarrito items[],Descuento desc)
    int base = 0
    for (int i = 0; i < items.length ;i++){ base = base + items[i].producto.precio *
items[i].cantidad; };
    return base - calcularDescuento(base,desc)
```

```
int calcularDescuento(int base,Descuento desc )
    if (desc.tipo == "%") {
        return base * desc.procentaje;    }
    if (desc.tipo == "fijo") {
        return desc.montoFijo;    }
    //error...
```



Programación estructurada vs orientada a objetos

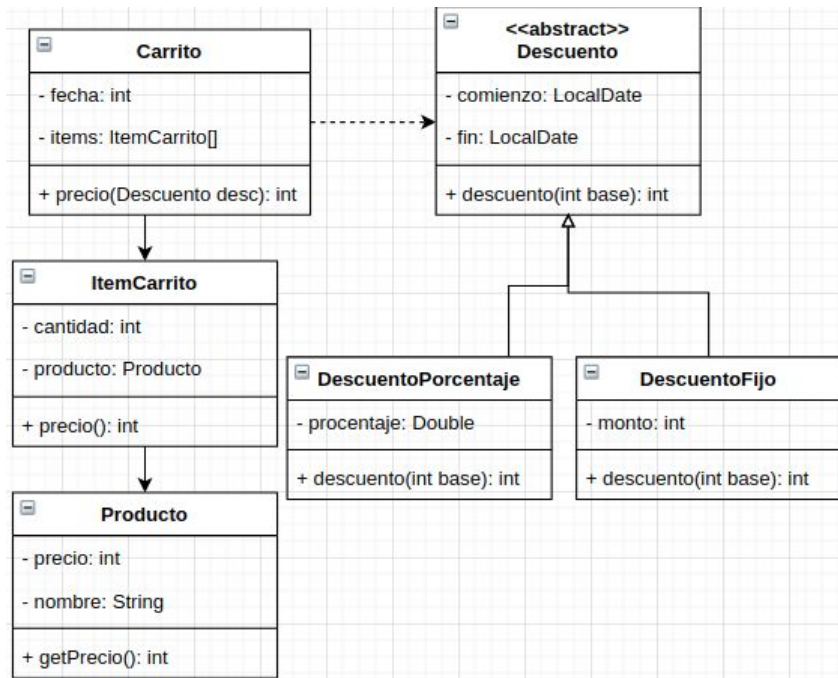
Utilizando Clases y Objetos (Iteración 1)

```
double CarritoCompras#precio(Descuento desc)
    int base = 0
    for (int i = 0; i < items.length ;i++){ base = base
+ item[i].precio() };
    return base - desc.descuento(base)
```

```
int ItemCarrito#precio( ) {
    return cantidad * producto.getPrecio();
}
```

```
int DescuentoPorcentaje#descuento(int base ) {
    return base * porcentaje;
}
```

```
int DescuentoFijo#descuento(int base ){
    return monto;
}
```





**Argentina
programa
4.0**

Gracias!
