

# Modulo 1

Docente: Henry Antonio Mendoza Puerta  
MitoCode

# Paradigma

# Pilares de la POO



# Abstracción

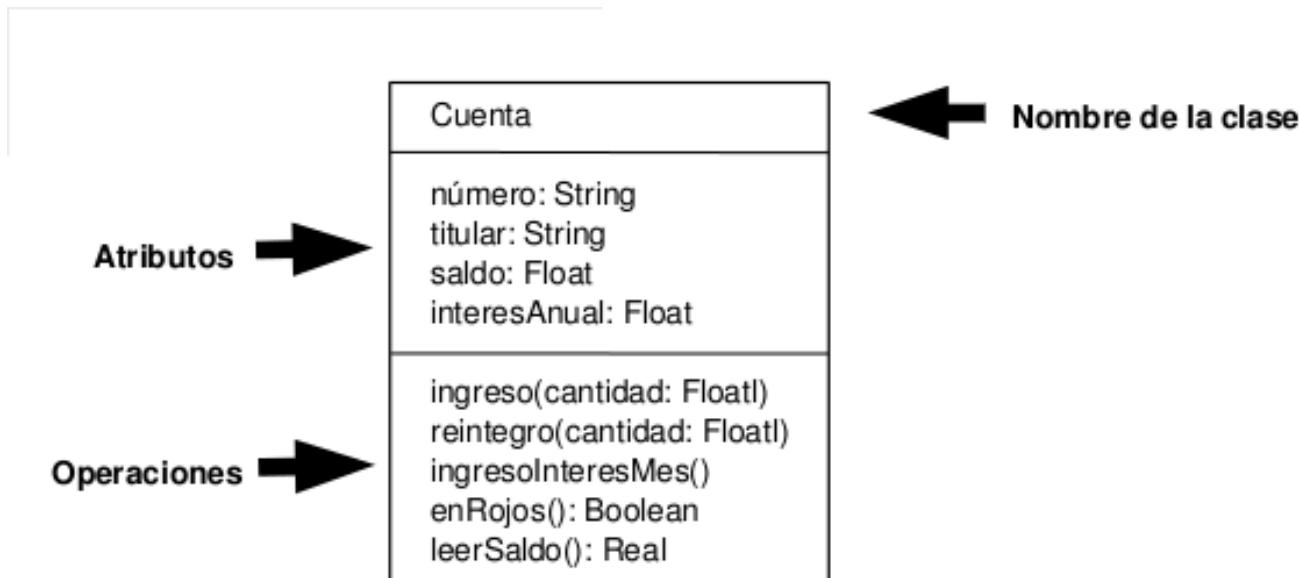
# Encapsulamiento

# Herencia

# Polimorfismo

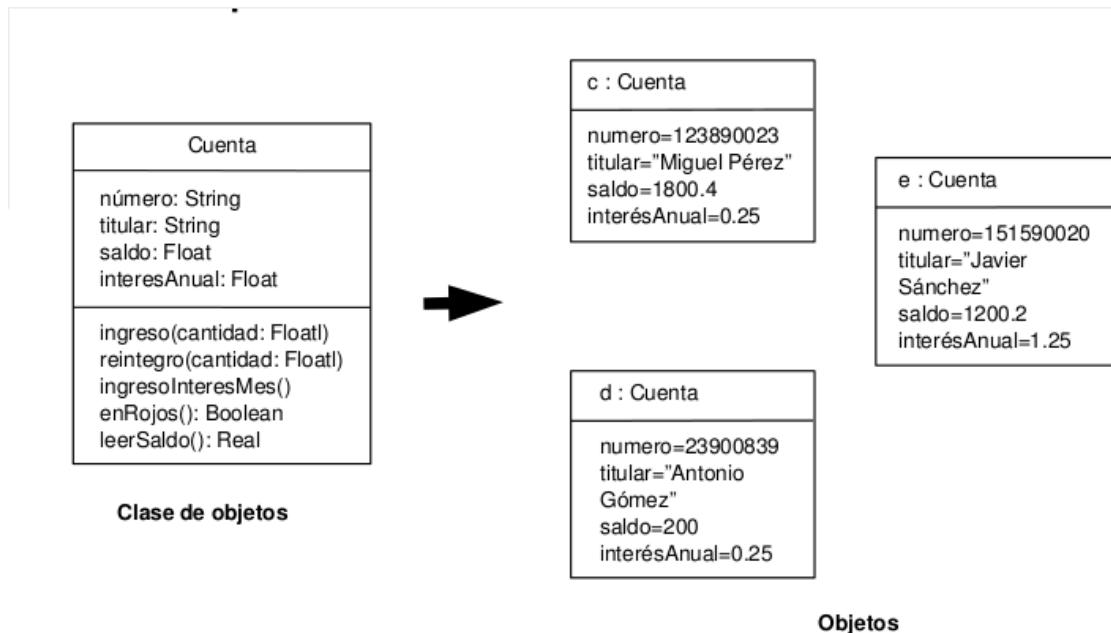
# Polimorfismo

# Clase



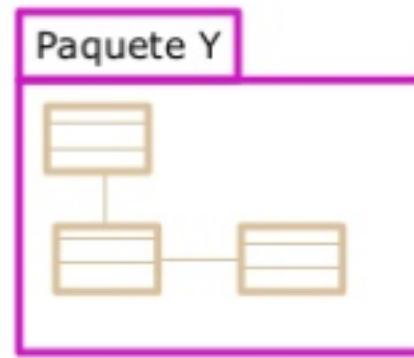
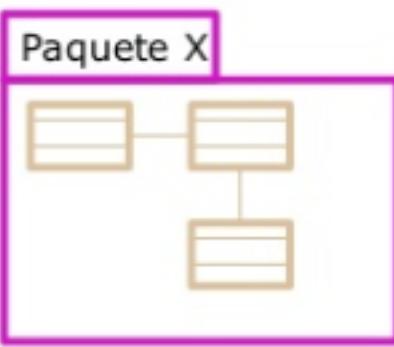
Atributos (propiedades)  
Métodos (comportamiento)

# Objeto (instancia de una clase)



# Getters y Setters (Encapsulamiento)

# Paquetes



# Modificadores

<https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>

# Modificadores

- **public:** Puede ser accedido desde cualquier lugar.
- **private:** Puede ser accedido solo desde su propia clase.
- **default:** Puede ser accedido desde su clase, las clases que están en el mismo paquete y las clases que hereden en el mismo paquete.
- **protected:** Puede ser accedido desde su clase, las clases que están en el mismo paquete y desde las clases que hereden si estén en otras clases.



# Ejercicio

En un banco, para registrar una cuenta bancaria, es necesario informar el número de la cuenta, el nombre del titular de la cuenta, y el valor de depósito inicial que el titular depositó al abrir la cuenta. Este valor de depósito sin embargo, es opcional, es decir: si el titular no tiene dinero que depositar en el momento de abrir su cuenta el depósito inicial no se hará y el saldo inicial de la cuenta será, naturalmente, cero. Importante: una vez que se ha abierto una cuenta bancaria, no se puede cambiar el número de cuenta. ya el nombre del titular puede modificarse (ya que una persona puede cambiar de nombre con motivo de matrimonio, por ejemplo). Por último, el saldo de la cuenta no puede ser modificado libremente. Es necesario un mecanismo para proteger ella. El saldo sólo aumenta por medio de depósitos, y sólo disminuye por medio de retiros. Para cada saque el banco cobra una tasa de \$ 5.00. Nota: la cuenta puede quedar con saldo negativo si el saldo no es suficiente para realizar el saque y / o pagar la tasa. Usted debe hacer un programa que realice el registro de una cuenta, dando opción para que sea o no informado del valor de depósito inicial.

# Ejercicio

```
Enter account number: 8532
```

```
Enter account holder: Alex Green
```

```
Is there na initial deposit (y/n)? y
```

```
Enter initial deposit value: 500.00
```

```
Account data:
```

```
Account 8532, Holder: Alex Green, Balance: $ 500.00
```

```
Enter a deposit value: 200.00
```

```
Updated account data:
```

```
Account 8532, Holder: Alex Green, Balance: $ 700.00
```

```
Enter a withdraw value: 300.00
```

```
Updated account data:
```

```
Account 8532, Holder: Alex Green, Balance: $ 395.00
```

# Ejercicio

---

```
Enter account number: 7801
Enter account holder: Maria Brown
Is there na initial deposit (y/n)? n
```

Account data:

```
Account 7801, Holder: Maria Brown, Balance: $ 0.00
```

```
Enter a deposit value: 200.00
```

Updated account data:

```
Account 7801, Holder: Maria Brown, Balance: $ 200.00
```

```
Enter a withdraw value: 198.00
```

Updated account data:

```
Account 7801, Holder: Maria Brown, Balance: $ -3.00
```

# Miembros Estáticos (static)



# Ejercicio

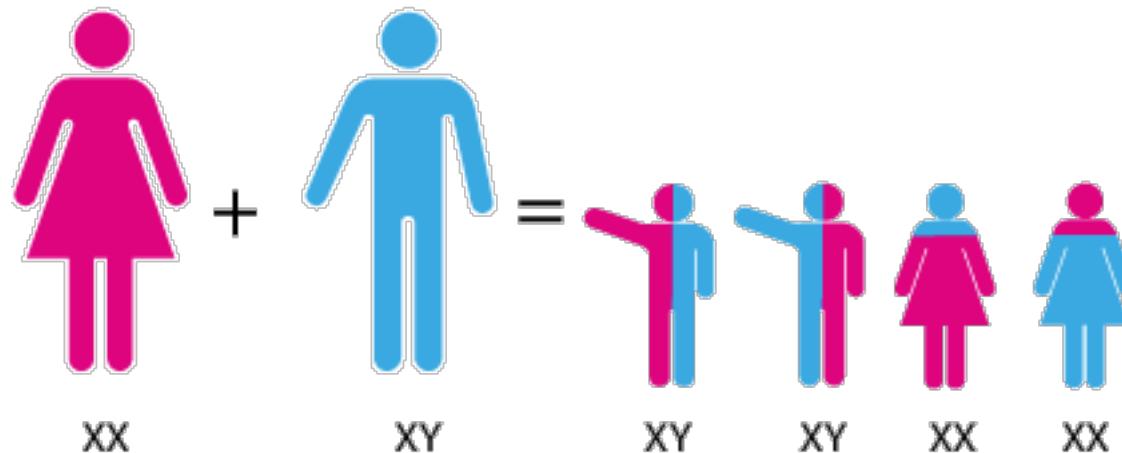
Haga un programa para leer la cotización del dólar, y luego un valor en dólares a ser comprado por una persona en reales. Informar cuántos reales la persona va a pagar por los dólares, considerando aún que la persona tendrá que pagar el 6% de IOF sobre el valor en dólar. Crear una clase CurrencyConverter para ser responsable de los cálculos.

```
What is the dollar price? 3.10
```

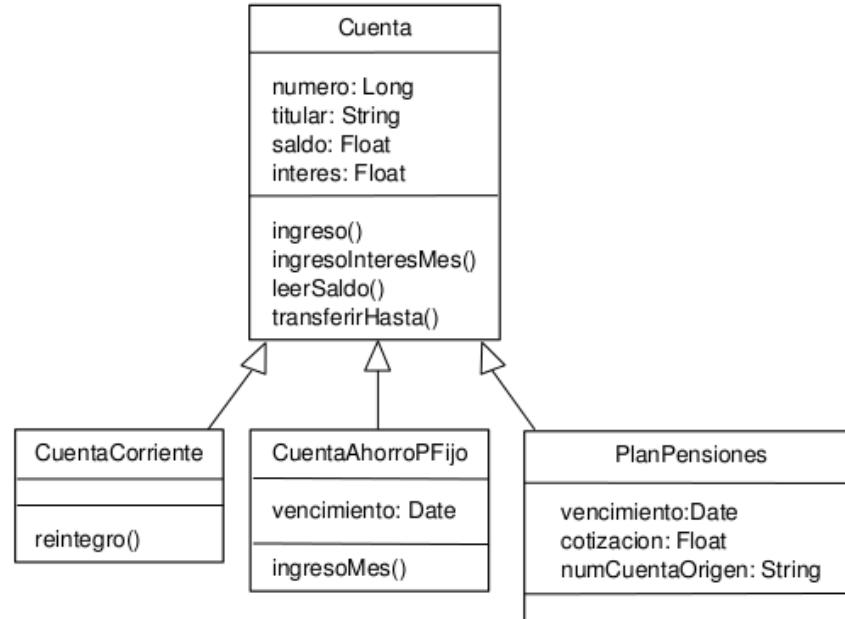
```
How many dollars will be bought? 200.00
```

```
Amount to be paid in reais = 657.20
```

# Herencia



# Herencia





# Ejercicio

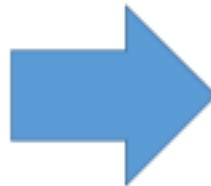
Suponga un negocio de banco que tiene una cuenta común y una cuenta para las empresas, siendo que la cuenta para empresa posee todos los miembros de la cuenta común, más un límite de préstamo y una operación de préstamo.

Account
- number : Integer
- holder : String
- balance : Double
+ withdraw(amount : Double) : void
+ deposit(amount : Double) : void

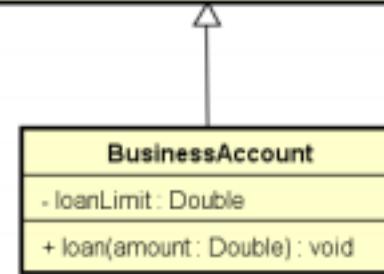
BusinessAccount
- number : Integer
- holder : String
- balance : Double
- loanLimit : Double
+ withdraw(amount : Double) : void
+ deposit(amount : Double) : void
+ loan(amount : Double) : void

# Ejercicio

Account
- number : Integer
- holder : String
- balance : Double
+ withdraw(amount : Double) : void
+ deposit(amount : Double) : void



Account
- number : Integer
- holder : String
- balance : Double
+ withdraw(amount : Double) : void
+ deposit(amount : Double) : void



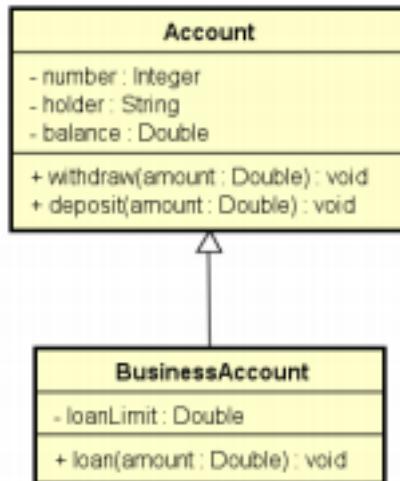
# Modificador de acceso protected

Different class but same package	Different package but subclass	Unrelated class but same module	Different module and p1 not exported
<pre>package p1; class A {     private int i;     int j;     protected int k;     public int l; }</pre>	<pre>package p1; class B {     }</pre>	<pre>package p2; class C extends A {     }</pre>	<pre>package p2; class D {     }</pre>

Accessible   Inaccessible

# Ejercicio

Supongamos que, para realizar un préstamo, es descontado una tasa por valor de 10.0  
Esto produce un error:



```
public void loan(double amount) {
    if (amount <= loanLimit) {
        balance += amount - 10.0;
    }
}
```

# Upcasting y Dowcasting

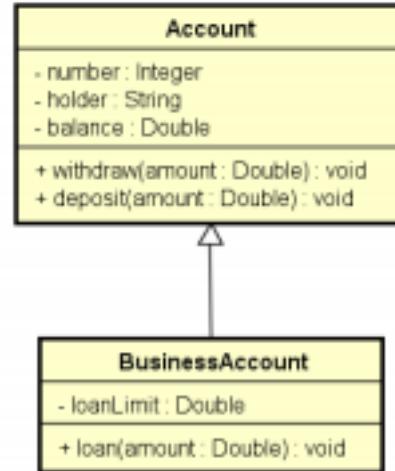
# Checklist

## Upcasting

- Casting de la subclase para la superclase
- Uso común: polimorfismo

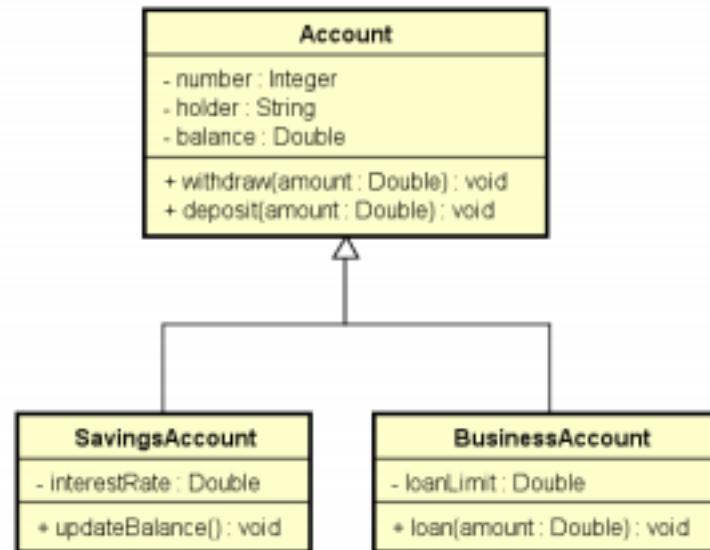
## Downcasting

- Casting de la superclase para subclase
- Palabra instanceof
- Uso común: métodos que reciben parámetros genéricos (por ejemplo, Equals)





# Ejercicio



# Sobrescritura y anotación @Override

# Sobrescritura

Es la implementación de un método de una superclase en la subclase

- Es muy recomendable utilizar la anotación @Override en un método sobrescrito.
- Facilita la lectura y la comprensión código.
- Avisamos al compilador (buena práctica)

Palabra super

# Palabra super

Es posible llamar a la implementación de la superclase usando la palabra super.

Ejemplo: suponga que, en la clase BusinessAccount, la regla de saque es realizar el registro el servicio normalmente de la superclase, y descontar más 2.0

# Palabra super

Super en constructor

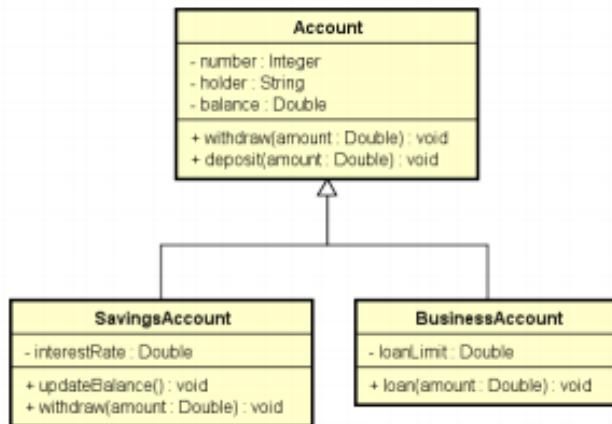
```
public class Account {  
  
    private Integer number;  
    private String holder;  
    private Double balance;  
  
    public Account(Integer number, String holder, Double balance) {  
        this.number = number;  
        this.holder = holder;  
        this.balance = balance;  
    }  
    (...)
```

```
public class BusinessAccount extends Account {  
  
    private double loanLimit;  
  
    public BusinessAccount(Integer number, String holder, Double balance, double loanLimit) {  
        super(number, holder, balance);  
        this.loanLimit = loanLimit;  
    }  
    (...)
```



# Ejercicio

Supongamos que la operación de servicio tiene una tasa en el valor de 5.0. Sin embargo, para la cuenta del tipo de ahorro (SavingsAccount), esta tasa no debe cobrarse. ¿Cómo resolver esto? Respuesta: sobrescribiendo método de extracción en la subclase SavingsAccount



# Ejercicio

## Account:

```
public void withdraw(double amount) {  
    balance -= amount + 5.0;  
}
```

## SavingsAccount:

```
@Override  
public void withdraw(double amount) {  
    balance -= amount;  
}
```

# Clases y métodos final

# Clases y métodos final

- **Clase**: evita que la clase sea heredada.

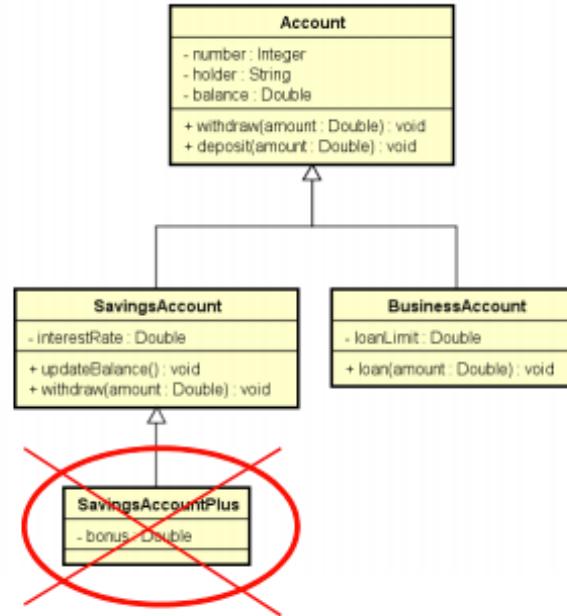
```
public final class SavingsAccount {
```

- **Método**: evita que el método bajo se sobreescriba.

# Clases y métodos final

**Clase:** Suponga que usted desea evitar que se creen subclases de SavingsAccount.

```
public final class SavingsAccount {  
    (...)
```



# Clases y métodos final

**Método:** Suponga que no desea que el método Withdraw de SavingsAccount se sobreescriba.

```
@Override  
public final void withdraw(double amount) {  
    balance -= amount;  
}
```

# Clases y métodos final

## Para que:

Seguridad: dependiendo de las reglas del negocio, a veces es deseable garantizar que una clase no se hereda, o que un método no se superpone.

- Generalmente, conviene añadir final en métodos sobrescritos, pues sobrescrituras múltiples pueden ser un puerto de entrada para inconsistencias.

# Polimorfismo

# Polimorfismo

- La asociación del tipo específico con el tipo genérico se realiza en tiempo de ejecución (upcasting).
- El compilador no sabe para qué tipo específico es la llamada del método Withdraw (sólo sabe que son dos variables de tipo de cuenta):

```
Account x = new Account(1020, "Alex", 1000.0);
Account y = new SavingsAccount(1023, "Maria", 1000.0, 0.01);

x.withdraw(50.0);
y.withdraw(50.0);
```

## Account:

```
public void withdraw(double amount) {
    balance -= amount + 5.0;
}
```

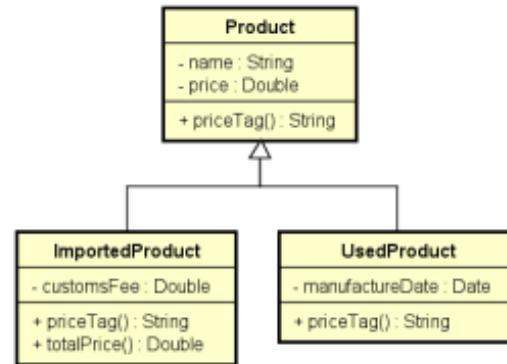
## SavingsAccount:

```
@Override
public void withdraw(double amount) {
    balance -= amount;
}
```



# Ejercicio

Hacer un programa para leer los datos de N productos (N proporcionado por el usuario). Al final, mostrar la etiqueta de precio de cada producto en la el mismo orden en que se hayan introducido. Todo producto tiene nombre y precio. productos importados tienen una tasa de aduana, y los productos usados tienen fecha de fabricación. Estos datos específicos deben añadidos en la etiqueta de precio conforme (página siguiente). Para productos importados, la tasa y la aduana debe añadida al precio final del producto.



# Ejercicio

```
Enter the number of products: 3
```

```
Product #1 data:
```

```
Common, used or imported (c/u/i)? i
```

```
Name: Tablet
```

```
Price: 260.00
```

```
Customs fee: 20.00
```

```
Product #2 data:
```

```
Common, used or imported (c/u/i)? c
```

```
Name: Notebook
```

```
Price: 1100.00
```

```
Product #3 data:
```

```
Common, used or imported (c/u/i)? u
```

```
Name: Iphone
```

```
Price: 400.00
```

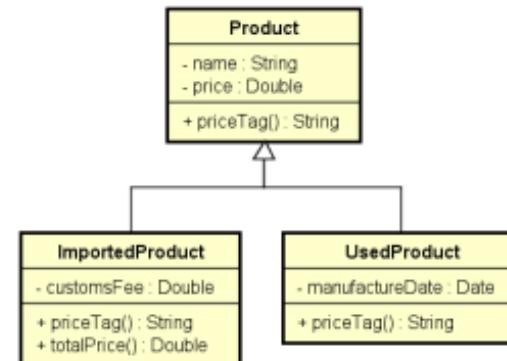
```
Manufacture date (DD/MM/YYYY): 15/03/2017
```

PRICE TAGS:

Tablet \$ 280.00 (Customs fee: \$ 20.00)

Notebook \$ 1100.00

Iphone (used) \$ 400.00 (Manufacture date: 15/03/2017)



# Clases abstractas

# Clases abstractas

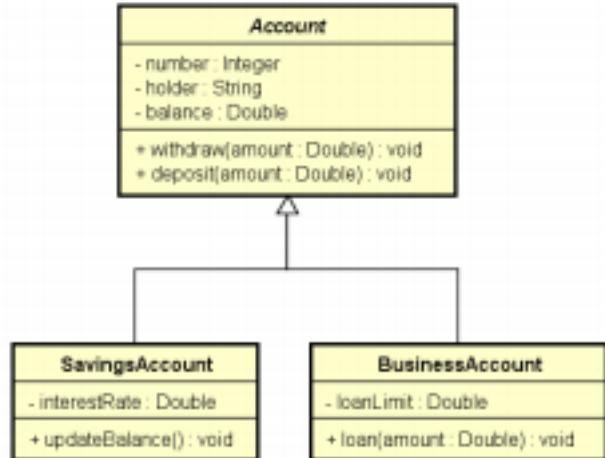
- Son clases que no pueden ser instanciadas
- Es una forma de garantizar la herencia total: sólo subclases no abstractos pueden ser instanciadas, pero nunca la superclase abstracta



# Ejercicio

Suponga que en un negocio relacionado con el banco, sólo las cuentas de ahorro (SavingsAccount) y las cuentas para negocio (BusinessAccount) están permitidas. No existe una cuenta común. Para garantizar que las cuentas comunes no puedan ser instanciadas, basta con añadir la palabra "abstract" en la declaración de clase..

```
public abstract class Account {  
    (...)
```



Vamos a implementar desde

<https://github.com/gitHAMP/CursoJavaWeb/tree/master/Mod1/App5>

# Ejercicio

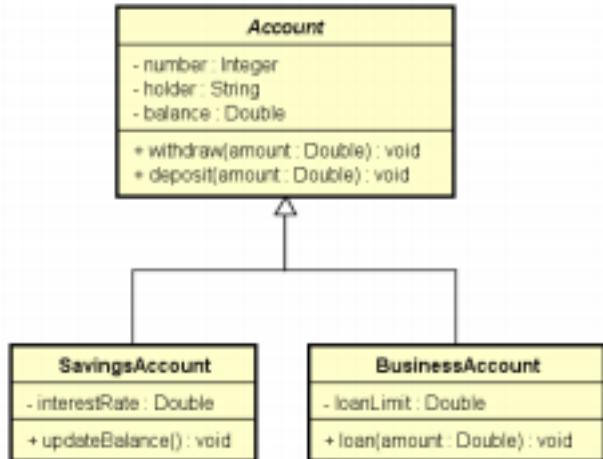
Si la clase Account no se puede crear una instancia, que simplemente no crear solamente SavingsAccount y BusinessAccount?.

Respuesta:

- Reutilización.
- Polimorfismo: la superclase clase genérica nos permite tratar de forma fácil y uniforme todos los tipos de cuenta, incluso con polimorfismo si es el caso (como lo hicimos en los últimos ejercicios). Por ejemplo, puede colocar todos los tipos cuentas en una misma colección.

Demo: suponga que usted desea:

- Totalizar el saldo de todas las cuentas.
- Depositar 10.00 en todas las cuentas.



# Métodos abstractos

# Métodos abstractos

- Son métodos que no tienen implementación.
- Los métodos deben ser abstractos cuando la clase es demasiado genérico para contener su identidad implementación.
- Si una clase tiene al menos un método abstracto, entonces esta clase también es abstracta.



# Ejercicio

Hacer un programa para leer los datos de N contribuyentes (N proporcionado por el usuario), los cuales pueden ser persona física o jurídica, y luego mostrar el valor del impuesto pagado por cada uno, así como el total de impuestos recaudado. Los datos de persona física son: nombre, renta anual y gastos de salud. Los datos de la persona jurídica son el nombre, la renta anual y el número de empleados. Las normas para el cálculo del impuesto son las siguientes:

Persona física: personas cuya renta fue inferior a 20000.00 pagan un 15% de impuesto. Personas con renta de 20000.00 en adelante pagan 25% de impuesto. Si la persona tuvo gastos de salud, el 50% de estos gastos se sacrifican en el impuesto. Ejemplo: una persona cuyo ingreso fue 50000.00 y tuvo 2000.00 en gastos de salud, el impuesto  $(50000 * 25\%) - (2000 * 50\%) = 11500.00$

Persona jurídica: las personas jurídicas pagan el 16% del impuesto. Sin embargo, si la empresa tiene más de 10 los empleados, paga un 14% de impuestos. Ejemplo: una empresa cuya renta fue 400000.00 y posee 25 funcionarios, el impuesto queda:  $400000 * 14\% = 56000.00$

# Ejercicio

Enter the number of tax payers: 3

Tax payer #1 data:

Individual or company (i/c)? i

Name: Alex

Anual income: 50000.00

Health expenditures: 2000.00

Tax payer #2 data:

Individual or company (i/c)? c

Name: SoftTech

Anual income: 400000.00

Number of employees: 25

Tax payer #3 data:

Individual or company (i/c)? i

Name: Bob

Anual income: 120000.00

Health expenditures: 1000.00

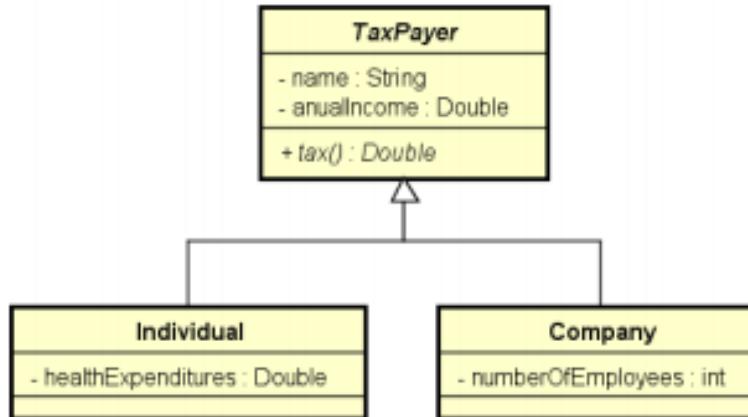
TAXES PAID:

Alex: \$ 11500.00

SoftTech: \$ 56000.00

Bob: \$ 29500.00

TOTAL TAXES: \$ 97000.00



# Enumeraciones

# Enumeraciones

Es un tipo especial que sirve para especificar de forma literal un tipo conjunto de constantes relacionadas

- Palabra clave en Java: enum
- Ventaja: mejor semántica, código más legible y auxiliado por el usuario. Compilado

## Ciclo de vida de un pedido

```
package entities.enums;

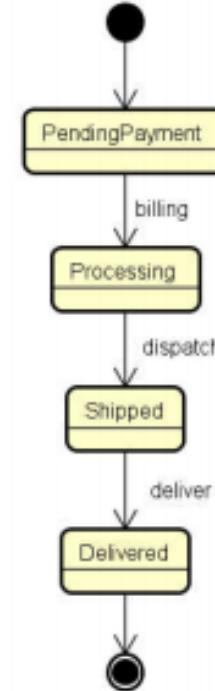
public enum OrderStatus {
    PENDING_PAYMENT,
    PROCESSING,
    SHIPPED,
    DELIVERED;
}
```

```
package entities;

import java.util.Date;
import entities.enums.OrderStatus;

public class Order {
    private Integer id;
    private Date moment;
    private OrderStatus status;
}

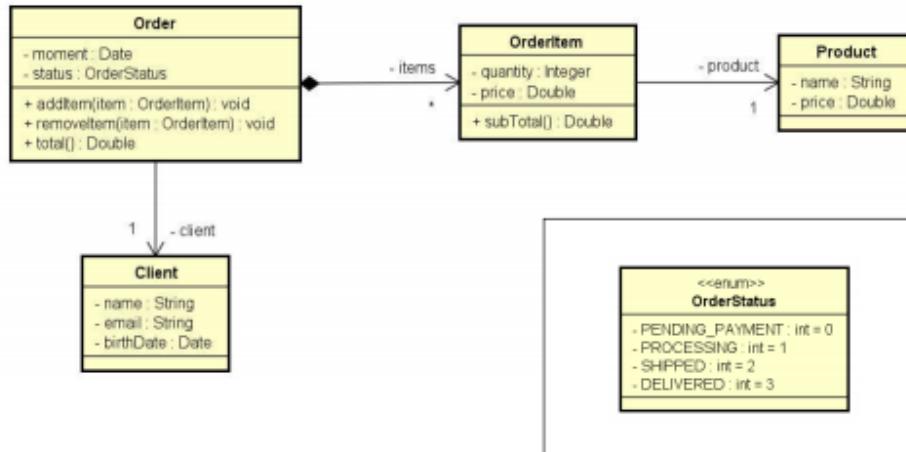
(...)
```





# Ejercicio

Leer los datos de una solicitud con N elementos (N proporcionado por el usuario). A continuación, mostrar un resumen de la solicitud según el ejemplo (siguiente página). Nota: el instante del pedido debe ser el instante del sistema: new Date () .



# Ejercicio

Enter cliente data:

Name: **Alex Green**

Email: **alex@gmail.com**

Birth date (DD/MM/YYYY): **15/03/1985**

Enter order data:

Status: **PROCESSING**

How many items to this order? **2**

Enter #1 item data:

Product name: **TV**

Product price: **1000.00**

Quantity: **1**

Enter #2 item data:

Product name: **Mouse**

Product price: **40.00**

Quantity: **2**

ORDER SUMMARY:

Order moment: **20/04/2018 11:25:09**

Order status: **PROCESSING**

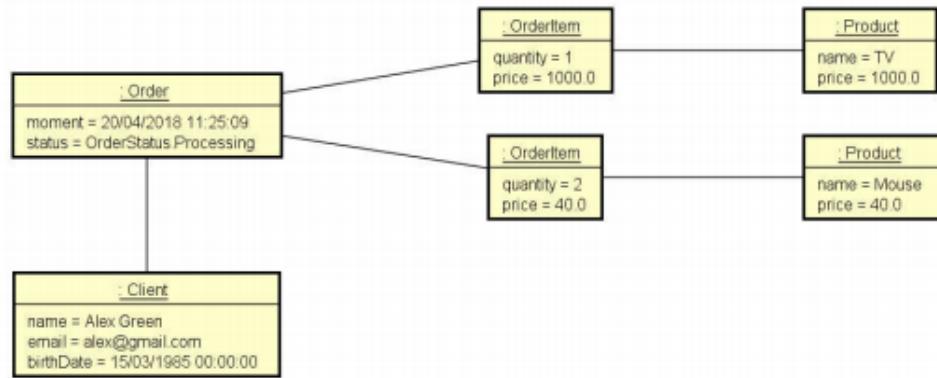
Client: **Alex Green (15/03/1985) - alex@gmail.com**

Order items:

**TV, \$1000.00, Quantity: 1, Subtotal: \$1000.00**

**Mouse, \$40.00, Quantity: 2, Subtotal: \$80.00**

**Total price: \$1080.00**



# MitoCode Network

## Descubre tu potencial

[www.mitocode.com](http://www.mitocode.com)

