El futuro digital es de todos

MinTIC

Universidad Pontificia Bolivariana

Vigilada Mineducación

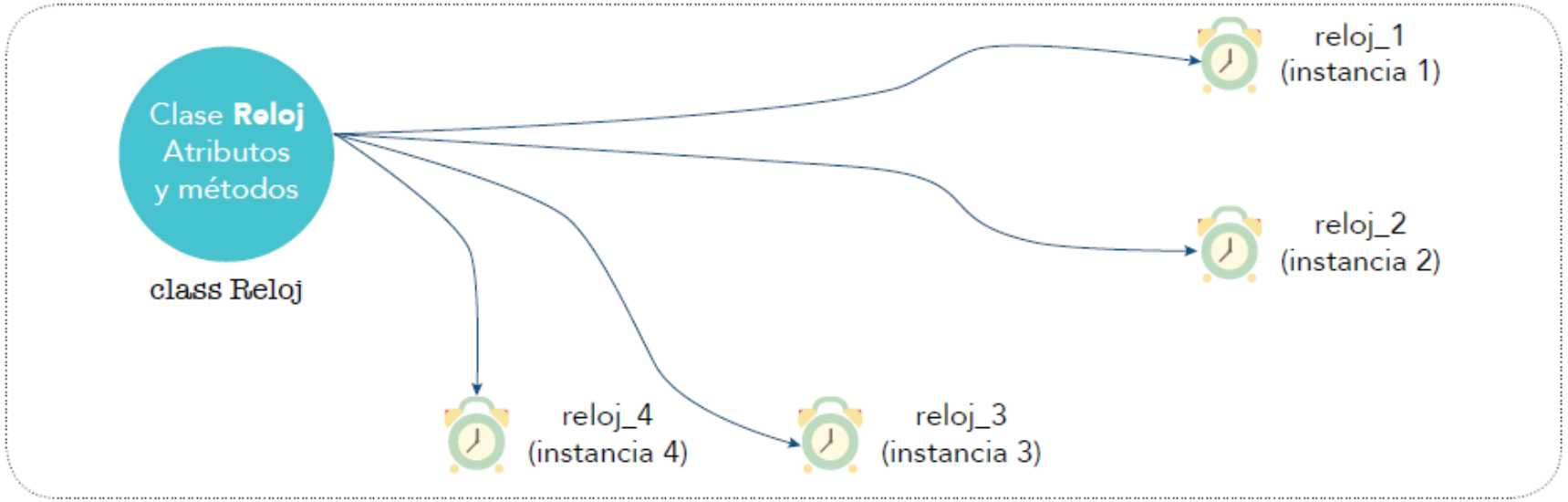
Mision TIC 2022

Clases, Objetos, Métodos, atributos

Las clases

Una clase es un tipo de dato definido por el programador específicamente para crear objetos. Se dice que cada objeto es una *instancia* particular de alguna *clase* de objetos. La clase define las propiedades comunes de un conjunto de objetos. El programador define una clase como lo hace con un tipo de dato compuesto y le da un nombre. Una vez definida la clase, los objetos se crean a partir de ésta.

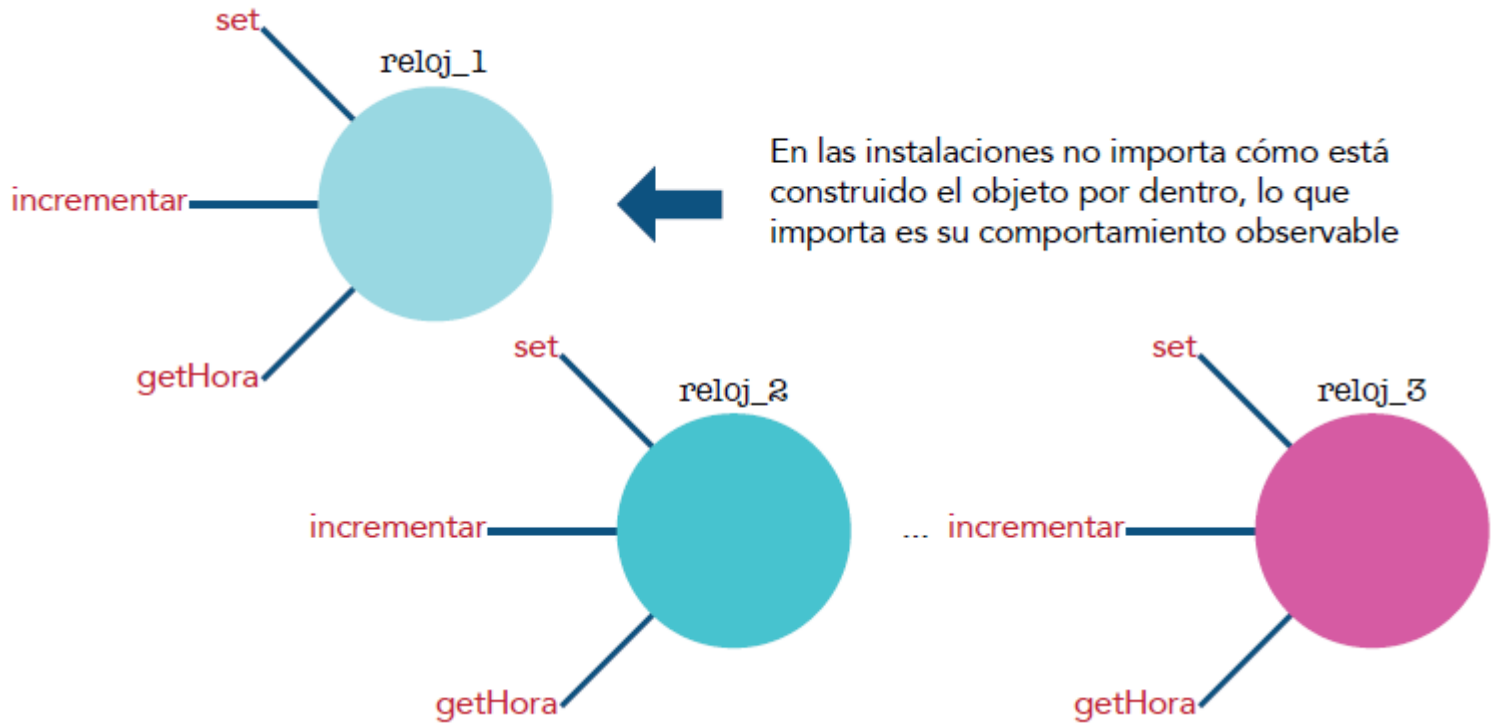
En la siguiente figura se ilustra que se pueden crear varios objetos reloj a partir de una misma clase.



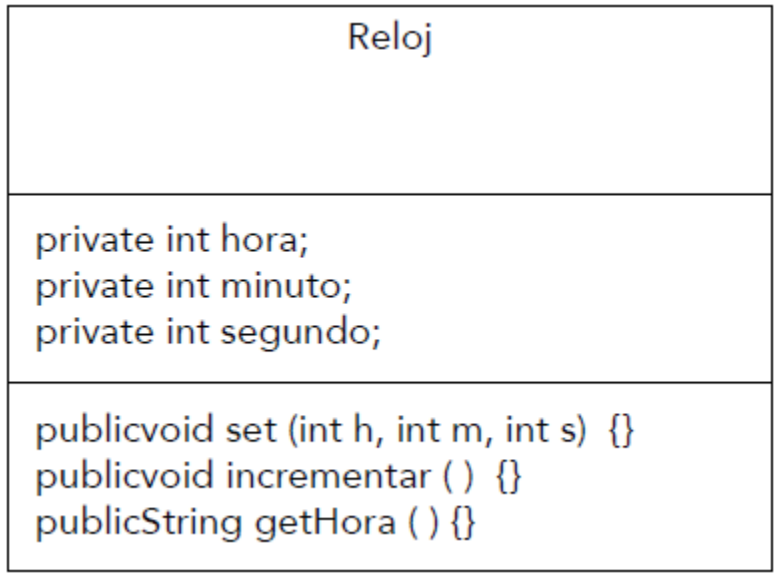
En programación, a las instancias de una clase se les llama *objetos*. A continuación, como ejemplo, definimos la clase ***Reloj***. Por convención, los nombres de las clases comienzan siempre con mayúscula. Un reloj se caracteriza por tener tres indicadores: la hora, los minutos y los segundos; por lo tanto, los atributos de la clase ***Reloj*** son tres números enteros. El comportamiento de este reloj está definido por las siguientes operaciones: inicializar la hora con un valor dado, incrementar la lectura actual en un segundo y obtener la hora. Estas operaciones se definen mediante los métodos ***set***, ***incrementar*** y ***getHora***.

```
public class Reloj {
   private int hora;
   private int minuto;
   private int segundo;

   public void set( int h, int m, int s ) {
      hora    = h % 24;
      minuto  = m % 60;
      segundo = s % 60;
   }

   public void incrementar() {
      segundo = ( segundo + 1 ) % 60;
      if ( segundo == 0 ) {
         minuto = ( minuto + 1 ) & 60;
         if ( minuto == 0 ) {
            hora = ( hora + 1 ) % 24;
         }
      }
   }


   public String getHora() {
      return hora + ":" + minuto + ":" + segundo;
   }
}
```

Como se observa en la figura, al crear los objetos de clase ***Reloj***, lo que importa hacia el exterior es la *interfaz* del objeto, es decir, la manera en la que éste es operado desde el exterior. Por ello los atributos del ejemplo anterior están declarados como ***privados*** y sus métodos como ***públicos***. La idea es que, como en la vida real, los objetos no sean "abiertos" y que sólo se manipulen a través de la interfaz diseñada para hacerlo. De esta forma, se aseguraría que serán operados adecuada y seguramente.



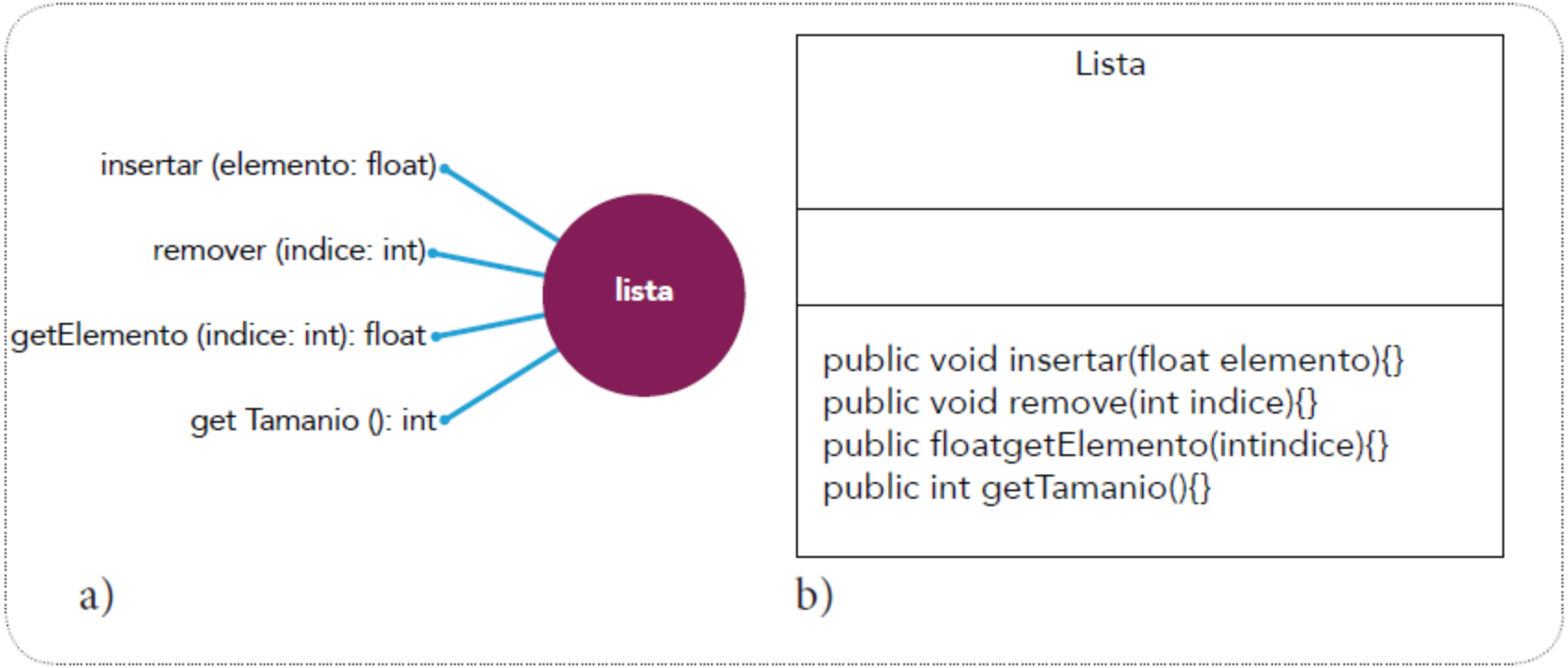
La clase es la descripción de un conjunto de objetos similares, es decir, que comparten los mismos atributos y los mismos métodos. La clase es el bloque constructor más importante de cualquier lenguaje de POO. El Unified Modeling Language, UML (Lenguaje Unificado de Modelado) (Booch *et al.,* 1998) se utiliza para el modelado orientado a objetos, el cual es tema de un curso posterior. Sin embargo, para comenzar a familiarizar al estudiante con este lenguaje, ilustraremos las clases con notación UML. En la figura se muestra la representación UML de la clase Reloj.



En el primer recuadro se indica el nombre de la clase; en el segundo, los atributos que contiene, mientras que en el tercero, sus métodos.

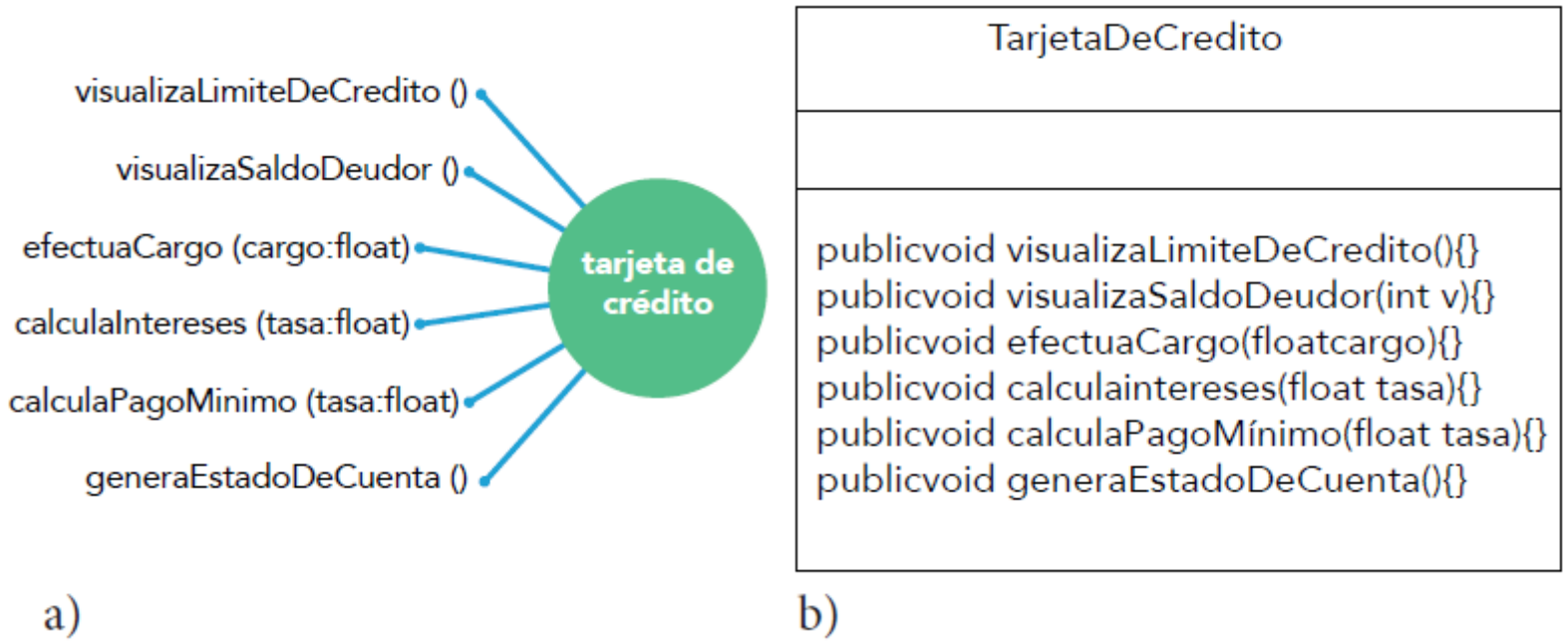
Ejemplo de objetos: Modelando una lista como objeto

En la figura, ***lista*** es un objeto con los métodos ***insertar()***, con el que se agrega un nuevo elemento en la lista; *remover()*, con el que se elimina el elemento en la posición índice de la lista; *getElemento()*, con el que se obtiene el elemento en la posición ***indice*** de la lista, y ***getTamanio()***, con el que se obtiene el número de elementos de la lista.



Ejemplo de objetos: ***Modelando una tarjeta de crédito como objeto***

En la figura, *tarjetaDeCredito* es un objeto con los métodos *getLimiteCredito(), getSaldoDeudor(), efectuarCargo(),* el cual es positivo para un abono y negativo para un retiro; *calcularIntereses(),* con el que se calculan los nuevos intereses generados; *calcularPagoMinimo(),* que calcula el pago mínimo que debe efectuarse entre la fecha de corte y la fecha de pago, y *generarEstadoDeCuenta(),* que genera toda la información producida al momento del corte.



Las clases en Java

La sintaxis para definir una clase en Java es la siguiente:

```
public class <NombreClase> {

   <definicion de los atributos>

   <definicion de los constructores>

   <definicion de los metodos>
}
```

Por convención, el nombre de una clase debe tener inicial mayúscula y el resto en minúsculas. En el caso de nombres compuestos, se utilizan las iniciales de cada palabra en mayúscula.

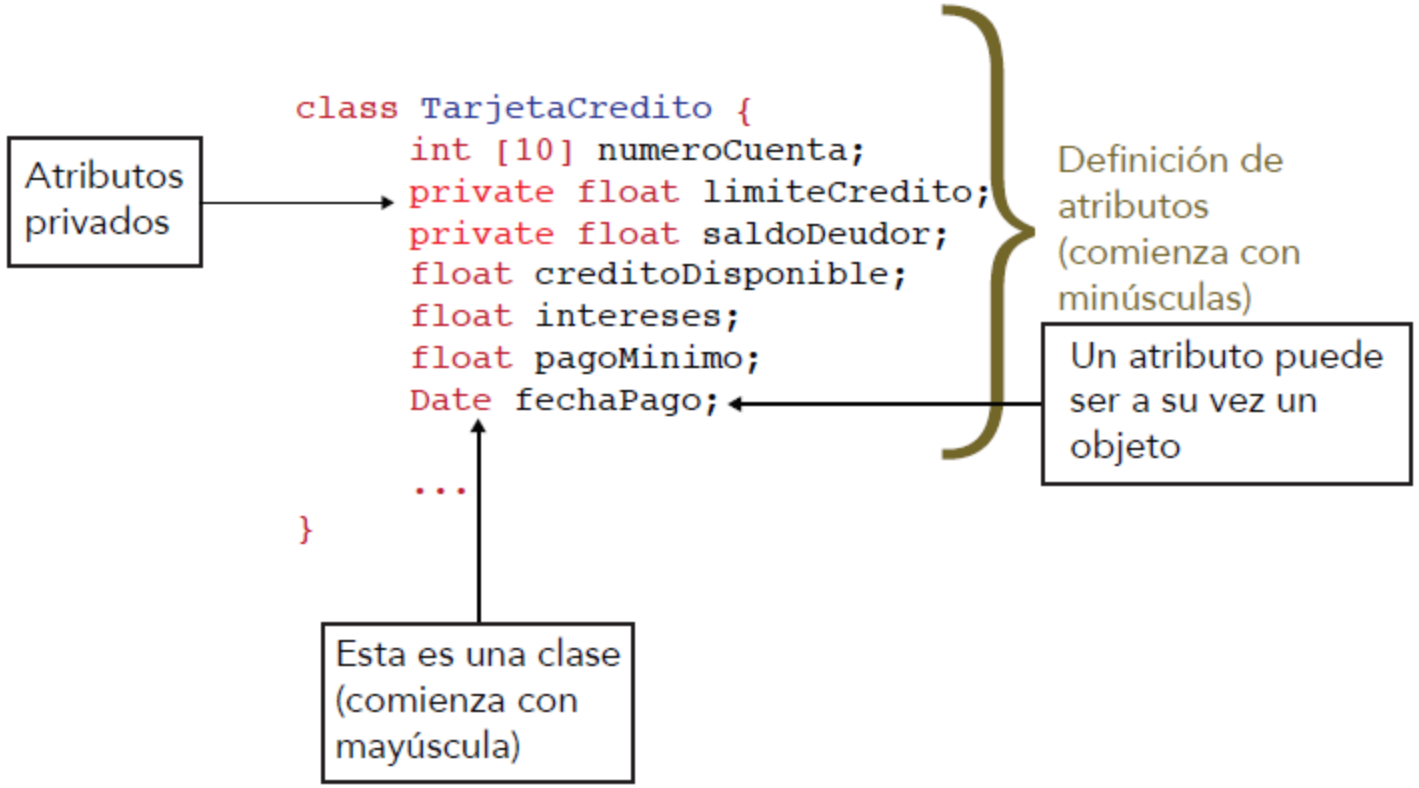
Los atributos

Los atributos definen la estructura de datos de la clase, los cuales, por omisión, son *público*s, es decir, accesibles desde otras clases, lo que significa que se modifican desde afuera del objeto. Es altamente recomendable declarar todos los atributos con el modificador *private* y solamente cambiarlo a *public* o *protected* cuando sea absolutamente necesario. En los ejemplos se verá más claramente el uso de estos modificadores.

Por convención, los nombres de los atributos deben escribirse en minúsculas. En el caso de nombres compuestos, cada palabra intermedia debe iniciar con mayúscula. Por ejemplo:

```
private int promedio;
public float saldoCuentaCredito;
```

La siguiente figura muestra la clase *TarjetaCredito*, que contiene varios ejemplos de atributos.



Los métodos

Los métodos constituyen el comportamiento de los objetos de la clase. La definición de un método es muy similar a la de una función. Los métodos públicos son las *operaciones* que los objetos externos realizan con el objeto en cuestión. Los métodos privados son las *operaciones internas* que no se pueden invocar desde el exterior, pero sí desde otro método dentro de la clase.

La sintaxis para escribir un método público es la siguiente:

```
public <TipoDatoRetorno> <NombreMetodo> ( <listaParametros> ) {
  …
}
```

```
private <TipoDatoRetorno> <NombreMétodo> ( <listaParametros> ) {
  …
}
```

Los métodos privados son auxiliares de los públicos. Se escriben para estructurar mejor el código de la clase. Por convención, los nombres de los métodos son verbos en infinitivo y escritos en minúsculas. En el caso de nombres compuestos, se usan mayúsculas para la inicial de cada palabra intermedia.

En el siguiente código se declara la clase *Ejemplo*, con los atributos *x* y *a*. Esta clase también contiene los métodos *hacerAlgo(), suma() e imprimir().*

```
public class Ejemplo {
  public int x;
  public int a;

  public void hacerAlgo() {
    x = 1 + a * 3;
  }

  public int suma() {
    return x + a;
  }

  public void imprimir() {
    System.out.println( "x= " + x + " a= " + a + "\n" );
  }
}
```

Crear un objeto

Declaremos que los objetos e y f son de la clase **Ejemplo:**

```
Ejemplo e;
Ejemplo f;
```

Declarar las variables *e* y *f* de la clase *Ejemplo* no implica que se crearon los objetos. En realidad, hasta aquí, sólo se crean apuntadores a objetos de esta clase. Para crear un objeto, se usa el operador *new*. El operador new crea un nuevo objeto de la clase especificada (alojando suficiente memoria para almacenar los datos del objeto) y regresa una *referencia* al objeto que se creó. Esta *referencia* es, en realidad, un *apuntador oculto.* En Java, el manejo de apuntadores y el desalojo de memoria se hacen automáticamente para evitar olvidos de los programadores y, por ende, el uso explícito de apuntadores se omite. El código para crear las instancias de los objetos de la clase *Ejemplo* sería:

```
Ejemplo e;
Ejemplo f;
e = new Ejemplo();
f = new Ejemplo();
```

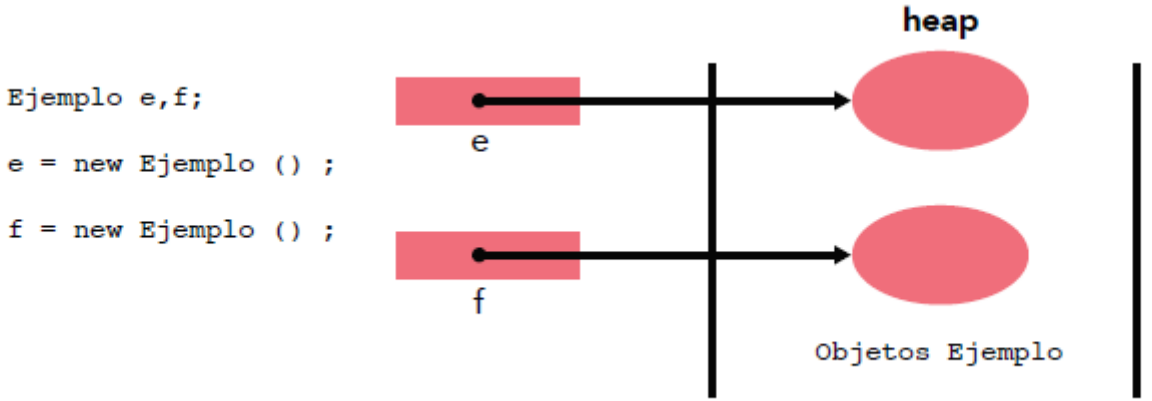
En general, la manera de crear un objeto de una clase X es la siguiente:

```
// primero declarar el objeto indicando su clase
<Clase> <nombreObjeto>;
// después crear el objeto
<nombreObjeto> = new <Clase>();
```

Lo anterior se crea en un solo enunciado:

```
<Clase> <nombreObjeto> = new <Clase>();
```

La figura ilustra lo que es una *referencia* al objeto. Con new se deposita en las variables e y f un apuntador a la clase Ejemplo. Se dice que las variables e y f son una *referencia* al objeto o, dicho de otra manera, a la dirección de memoria que le fue asignada por el sistema operativo de la memoria *heap* (o



En la práctica, sólo se dice que e y f son objetos de la clase Ejemplo. También podemos crear los objetos e y f en un solo paso:

```
Ejemplo e = new Ejemplo();
Ejemplo f = new Ejemplo();
```

Para tener acceso a los atributos y a los métodos del objeto, se utiliza el operador punto ".". La sintaxis de la notación punto de Java para acceder a un método es la siguiente:

```
<referenciaObjeto>.<nombreMetodo>( <listaParametros> )
```

```
nombreObjeto.nombreMetodo();
```

Entonces, la forma de ejecutar un método del objeto es:

```
e.hacerAlgo();
```

Por ejemplo, para llamar al método hacerAlgo( ) del objeto e se escribe:

A la clase que tiene el método *main* se le llama *clase principal*. En el siguiente ejemplo tenemos la clase *PruebaEjemplo*, cuyo método *main()* hace uso de dos objetos de la clase Ejemplo,

```
public class PruebaEjemplo {
  public static void main( String[] args ) {
    Ejemplo e;
    Ejemplo f;
    e = new Ejemplo(); // instancia "e" de la clase Ejemplo
    f = new Ejemplo(); // instancia "f" de la clase Ejemplo
    e.a = 7;
    e.hacerAlgo();
    f.x = e.suma();
    f.a = f.x + e.a;
    e.a = f.suma();
    e.imprimir();
    f.imprimir();
  }
}
```

El método *main()* debe ser siempre estático. La palabra *static* en la definición de un método implica que no es necesario crear el objeto para llamar al método. Así, la JVM puede invocarlo. Si queremos evitar que algunos datos de la clase se modifiquen desde afuera de ésta, conviene declarar a los atributos en cuestión como *privados*. La sintaxis para declarar un atributo como privado, es la siguiente:

```
private <tipo> <nombreAtributo>;

public class Ejemplo {
  int x;
  private int a; // el atributo "a" es privado

  public void hacerAlgo() {
    x = 1 + a * 3;
  }

  public int suma() {
    return x + a;
  }

  public void imprimir() {
    System.out.println( "x= " + x + " a= " + a + "\n" );
  }
}
```

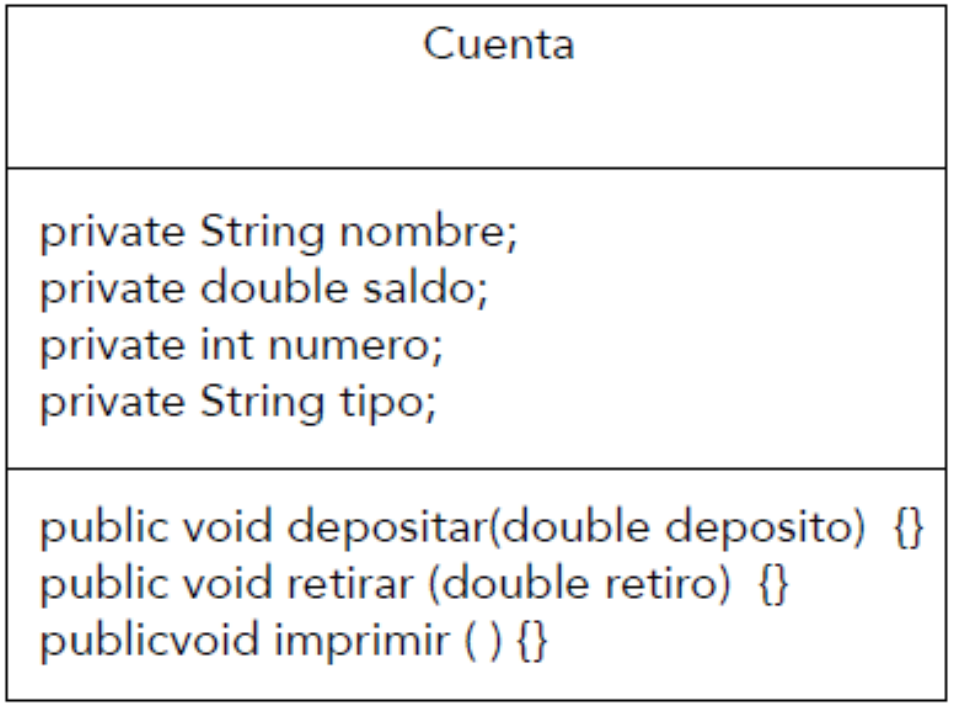
Entonces, no será posible tener acceso al atributo a desde el exterior de la clase y sólo los métodos de la clase tienen acceso a este atributo. Así es que, cuando los objetos *e* y *f* en la clase *PruebaEjemplo* tratan de leer o modificar el atributo a, ocurre un error de compilación.

```
public class PruebaEjemplo {
  public static void main( String[] args ) {
    Ejemplo e;
    Ejemplo f;
    e = new Ejemplo();
    f = new Ejemplo();
    e.a = 7;
    e.hacerAlgo();
    f.x = e.suma();
    f.a = f.x + e.a;
    e.a = f.suma();
    e.imprimir();
    f.imprimir();
  }
}
```

¡Error! El acceso al atributo a no está permitido.

Ejercicios resueltos: Clase Cuenta

Declarar la clase *Cuenta* con los atributos *nombre, saldo, numero* y *tipo*, y con los métodos *depositar*, que recibe como parámetro la cantidad a depositar, y *retirar*, que recibe como parámetro la cantidad a retirar. Sólo se efectuará el retiro si el saldo es mayor o igual a la cantidad a retirar. Escribir un método que imprima los datos del objeto.



Solución:

```
public class Cuenta {
  public String nombre;
  public double saldo;
  public int numero;
  public String tipo;

  public void depositar( double deposito ) {
    saldo = saldo + deposito;
  }

  public void retirar( double retiro ) {
    if ( saldo >= retiro ) {
      saldo = saldo - retiro;
    }
  }

  public void imprimir() {
    System.out.println( "La cuenta es de: " + nombre
                + ", número: " + numero
                + ". Es una cuenta de " + tipo
                + ", con saldo: " + saldo + "\n" );
  }
}
```

Ejercicios resueltos: Clase Cuenta

Crear los objetos cuentaCredito y cuentaDebito en la clase Principal. Al objeto cuentaCredito ponerlo a nombre de Pedro Sánchez, con saldo de 1500, con número de cuenta 244513, indicando que es una cuenta de crédito. Al objeto cuentaDebito ponerlo a nombre de Pablo García, con saldo de 7800, con número de cuenta 273516, indicando que es una cuenta de débito. ¿Qué es lo que se imprime?

Solución:

```
public class Principal {
  public static void main( String[] args ) {
    Cuenta cuentaCredito;
    Cuenta cuentaDebito;

    // Creamos los objetos
    cuentaCredito = new Cuenta();
    cuentaDebito  = new Cuenta();

    …

    cuentaCredito.imprimir();
    cuentaDebito.imprimir();
  }
}
```

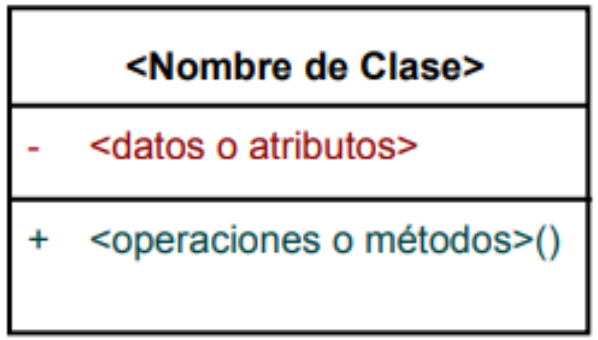
```
Solución:

public class Principal {
  public static void main( String[] args ) {
    Cuenta cuentaCredito;
    Cuenta cuentaDebito;

    // Creamos los objetos
    cuentaCredito = new Cuenta();
    cuentaDebito  = new Cuenta();

    cuentaCredito.nombre = "Pedro Sanchez";
    cuentaCredito.saldo = 1500;
    cuentaCredito.numero = 244513;
    cuentaCredito.tipo = "crédito";

    cuentaDebito.nombre = "Pablo Garcia";
    cuentaDebito.saldo = 7800;
    cuentaDebito.numero = 273516;
    cuentaDebito.tipo = "débito";

    cuentaCredito.imprimir();
    cuentaDebito.imprimir();
  }
}
```

Diagrama de clases

Un diagrama de clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Los diagramas de Clases por definición son estáticos, esto es, representan qué partes interactúan entre sí, no cuando ocurre. Un diagrama de clases está compuesto por clases y relaciones. Las clases agrupan datos y métodos con sus respectivos niveles de ocultación, mientras que las relaciones pueden ser: herencia para crear jerarquías de clases de objetos, asociación entre distintas clases, composición (agregación estática) y agregación dinámica de clases mediante objetos o instancias de otras clases.

El componente CLASE:

Una clase agrupa datos y métodos de un conjunto de objetos de la realidad que comparten características esenciales y comportamientos similares. Una clase es considerada una plantilla para instanciar o crear objetos. En UML, una clase se representa mediante un rectángulo con tres divisiones, tal como se ve en la figura 8, en la parte superior se aprecia el nombre de la clase, en el intermedio se ubican los datos y en la parte inferior los métodos. Tanto a datos como a métodos es posible agregar el especificador de acceso para establecer el nivel de ocultamiento.



Los datos o atributos. Corresponden a las variables asociadas a un tipo de dato (int, double, char, etc), identificadas en función de las características esenciales de objetos comunes de la realidad. Los métodos son las operaciones que representan comportamientos de objetos comunes de la realidad. A través de los métodos se manipulan los datos de la propia clase y es posible invocar a metódos de otros objetos (envío de mensajes).

Los Especificadores de Acceso son los términos private (-), protected (#), package (por defecto sin símbolo) y public (+), sirven para establecer el nivel de ocultamiento de los elementos de una clase. private (-): los miembros con este especificador son visibles sólo para métodos de la propia clase y clases internas. protected (#): los datos y métodos con este especificador son visibles para métodos de la propia clase, clases internas y clases derivadas. package (ningún símbolo): los miembros con este especificador son visibles en métodos de las clases que conforman el paquete. public (+): el dato o método es visible para métodos de todas las clases del programa o aplicación.

Ejemplo:

Una Cuenta de Ahorros que posee como características: número de cuenta, saldo y cliente Y las operaciones:

• Depositar

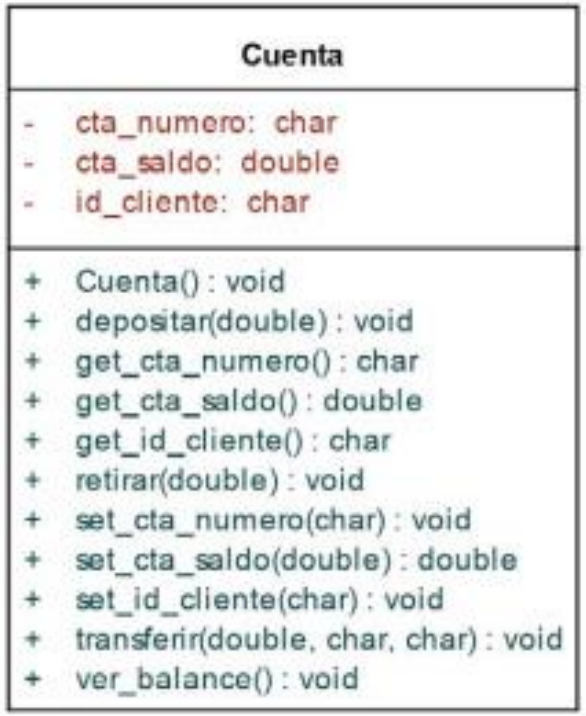
• Retirar

• Transferir

• Ver Balance

• Métodos de lectura (get) y escritura (set)

• Constructor, lleva el mismo nombre de la clase Un ejemplo del diseño asociado se muestra en la figura 9:



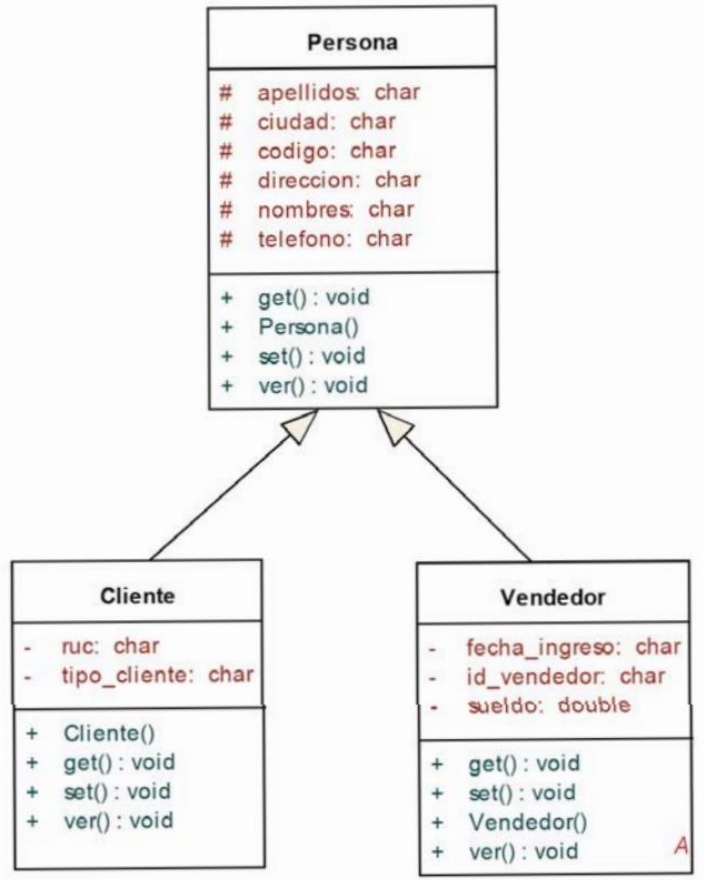
Las relaciones entre clases

Relación de Herencia

(Especialización/Generalización): ⟶ (Es un ó Es una)

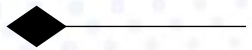
Una clase puede derivarse de otra clase base, heredando datos y métodos públicos, protegidos y de paquete. Los miembros privados no se heredan. La nueva clase puede añadir sus propios datos y métodos o sobrescribir métodos de la clase padre.

En la figura se especifica que las clases Cliente y Vendedor heredan de Persona, es decir, Cliente es una Persona y Vendedor es una Persona. La clase Cliente posee las características de Persona y añade los datos ruc y tipo_cliente, en cambio Vendedor también hereda las características de Persona y añade los datos específicos: fecha_ingreso, id_vendedor y sueldo. Los datos de la clase Persona son tipo protegido y sus métodos de tipo público. Mientras que en las clases derivadas los datos son private y los métodos públicos. Cabe recordar que los datos privados y los constructores no se heredan. Los métodos get y set deben especificarse por cada dato privado o protegido, en el ejemplo se dejó solo expresado, pero hace falta colocar para cada dato



Relación de Agregación:

Cuando se necesita crear clases compuestas por instancias de otras clases es útil la agregación o composición. Por ejemplo un un auto se compone de partes como: 1 motor, 4 ruedas, 1 chasis, etc. A continuación se describe los tipos de agregación:



• Agregación por valor o composición:

Considerada también agregación estática, cuyo tiempo de vida del objeto agregado depende del tiempo de vida del objeto contenedor. En otras palabras, el objeto compuesto (todo) se construye a partir del objetos incluidos (partes). Esta es una relación “todo/parte”). Una aplicación de este tipo de relación se evidencia cuando los datos de una clase son de tipo clase, es decir cuando se cree una instancia de una clase compuesta, algunos de sus datos serán instancias de otras clases.

• Agregación por referencia:

Es un tipo de agregación dinámica, cuyo tiempo de vida de los objetos agregados (partes) es independiente del objeto contenedor (todo). Por lo general en arreglos o colecciones dinámicas de objetos se evidencia este tipo de agregación.

Relación de Asociación:

Este tipo de relación no es una relación fuerte, es para objetos que colaboran con otros objetos. El tiempo de vida de un objeto no depende del objeto que se asocia. Se evidencia este tipo de relación cuando los objetos tienen algún dato en común y mediante sus métodos es posible enviarse mensajes. Por ejemplo un cliente puede tener asociadas muchas facturas de venta, en cambio una factura de venta solo puede tener asociado un cliente.

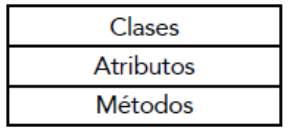
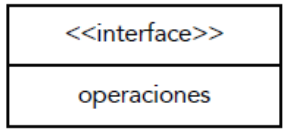
Asociación con multiplicidad:

Representa el grado o nivel de dependencia entre las clases que se relacionan. El tipo de cardinalidad se registra en los dos extremos de la relación y estos pueden ser:

- Cero (0)
- Cero o Uno (0..1)
- Uno (1) • Uno o muchos: (1..*)
- 0 o muchos: (0..*)
- Número fijo: m (m denota un número fijo).

La cardinalidad es aplicable sólo en las relaciones de asociación y agregación. En la relación de herencia no se especifica cardinalidad debido a que es una relación uno a uno.

Glosario de notaciones para diagramas de clases

Símbolo	Significado
	Clase
	interfaz
	Generalización
	Implementación de interfaz
	Asociación
	Asociación con multiplicidad
	Asociación con navegabilidad
	Agregación
	Composición

Un ejemplo de diagrama de clase de un sistema de facturación. La relación de herencia se observa entre Vendedor – Persona y Cliente – Persona, la relación de asociación entre: Vendedor – Factura y Producto – Items, la relación de Agregación estática o Composición entre Factura – Cliente y la relación de Agregación Dinámica entre Factura e Items