



CONTENIDO

UNIDAD I. HERENCIA Y POLIMORFISMO	2
1.1 Definición de Herencia.....	3
1.1.1 Superclases y subclases	6
1.1.2 Tipos de Herencia	9
1.1.3 Ventajas de la herencia	10
1.2 Efectos de los modificadores en la herencia	12
1.2.1 Modificadores de acceso (private, protected, public, por defecto).....	12
1.3 Métodos sobrecargados en la herencia	20
1.4 Métodos o datos sobre escritos y constructores en la herencia	21
1.4.1 Utilizar super para acceso a miembros de la superclase:	21
1.4.2 Utilización de super para acceder a constructores de la superclase	23

UNIDAD I. HERENCIA Y POLIMORFISMO



Poliformismo-desarrollo de [software](#)

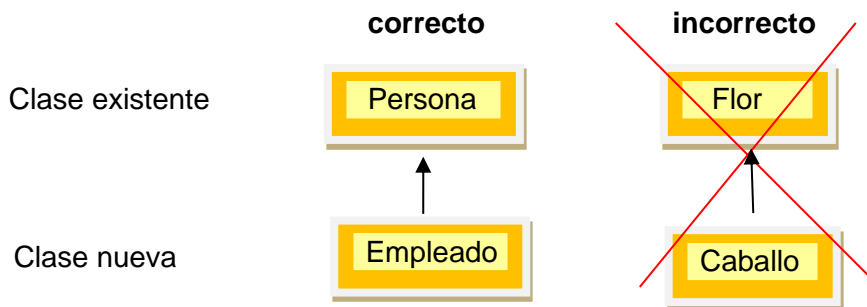
Iniciamos la unidad I con el tema de herencia y polimorfismo. Las cuales representan una de las características más importantes de la metodología de la Programación Orientada a Objetos. Gracias a la herencia y el polimorfismo por ser uno de los mecanismos más utilizado para alcanzar algunos de los objetivos más poderosos en el desarrollo de software como lo son la reutilización de código en la creación de nuevas clases, evitando la duplicidad de código y seguridad en los miembros de una clase.

Por su importancia los invito a que participen activamente en el desarrollo de esta unidad porque se requiere de un análisis profundo para poder utilizar las ventajas que nos ofrece la herencia y polimorfismo en el desarrollo de software.

1.1 Definición de Herencia

La herencia es una de las 3 características fundamentales de la POO. Porque permite crear una nueva clase a partir de una clase existente sin modificarla y **considerando las características comunes y la relación jerárquica entre ellas.**

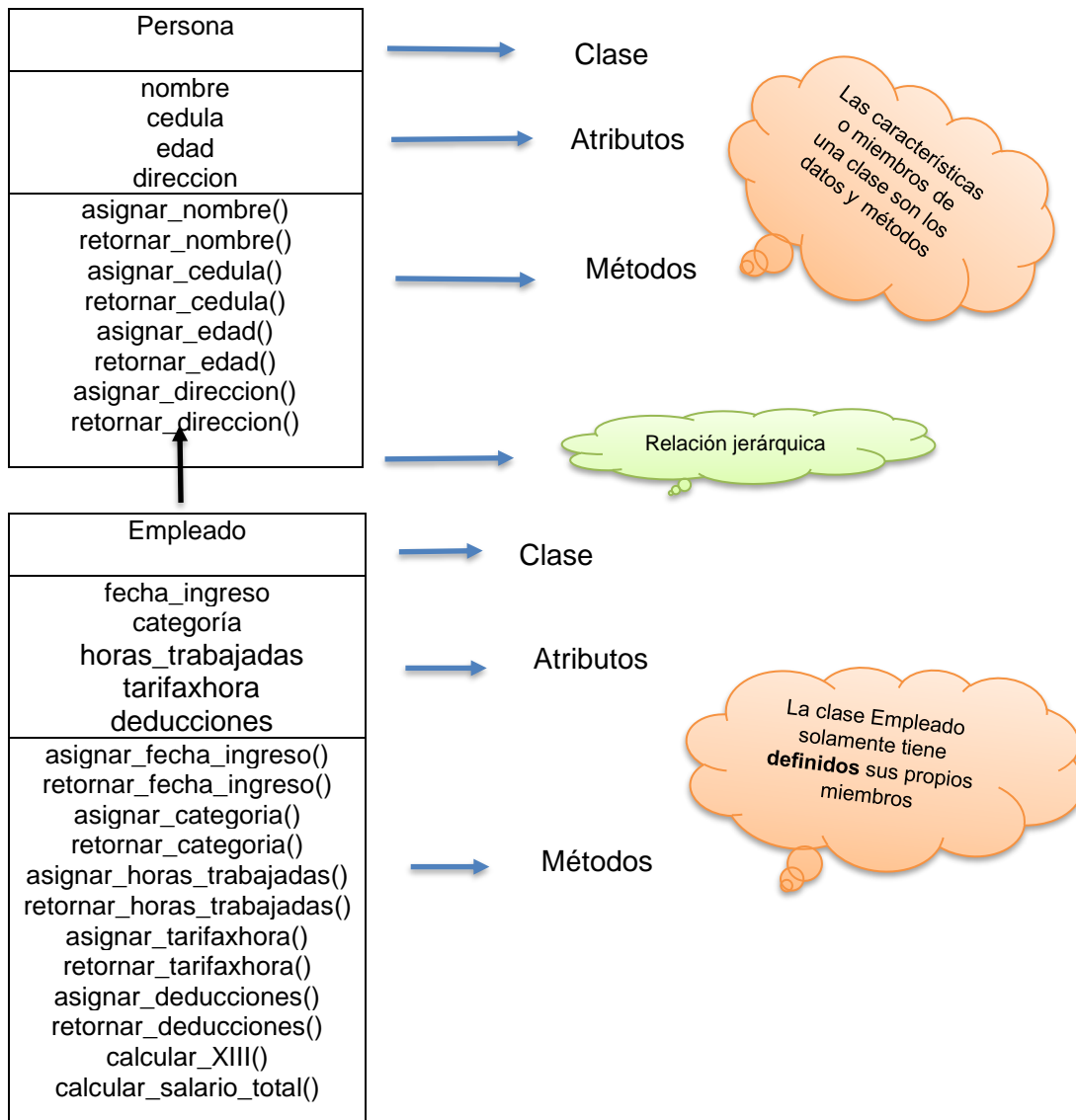
Veamos un ejemplo donde observamos las características comunes y **la relación jerárquica** entre clases.



Es necesario recordar que cuando hablamos de características de una clase nos referimos a los miembros de una clase como lo son los atributos y métodos.



En el ejemplo anterior la clase Persona tiene características comunes a la clase Empleado que puede heredar. Sin embargo, la clase Flor no tiene características comunes con la clase Caballo por lo tanto usted como programador tendrá que validar esa situación cuando plantea la herencia entre clases porque el compilador solo verifica sintáxis de las instrucciones y no la lógica.



Las clases Persona y Empleado tienen sus propias características. La clase Empleado puede utilizar por herencia todas características de la clase Persona porque son comunes y tienen una relación jerárquica.



Por ejemplo el método `calcular_salario_total()` en su **implementación** puede utilizar la edad para realizar algún cálculo aritmético para generar el salario total si el enunciado del problema lo requiere o lo amerita.

Por lo explicado antes la herencia permite organizar las clases en una estructura jerárquica formando jerarquía de clases. Es importante resaltar que a pesar de que la clase Empleado hereda de Persona, estas clases son independientes una de las otras.

Por lo significativo de la independencia de clases al desarrollar aplicaciones tengo que mencionar que la herencia no elimina esta independencia.

- **Independencia de clases**

Una clase base contiene sus atributos (datos) y métodos propios, que la hacen una clase completamente independiente y auto suficiente.

Persona

Se puede instanciar un objeto a la clase Persona

Empleado

Se puede instanciar un objeto a la clase Empleado

Si `Persona obj1 = new Persona();` entonces obj1 tiene los miembros de la
clase Persona

Si `Empleado obj2 = new Empleado();` entonces obj2 tiene los miembros de la
clase Empleado



La herencia maneja dos elementos básicos para trabajarla y se le denominan super clase y subclase. La superclase identifica a la clase existente y la subclase a la clase nueva.

1.1.1 Superclases y subclases

La superclase es una clase existente. Que define características comunes o genéricas a un conjunto de elementos relacionados. Es la clase cuyas características se pueden heredar por otra clase.

La subclase es una nueva clase a la que se le definirán sus **propios miembros y podrá utilizar por herencia** los miembros de la superclase. La subclase hereda todos los miembros definidos por la superclase dependiendo de algunas reglas que veremos más adelante.

Formato a utilizar en Java para aplicar la herencia es el siguiente:

```
class Super_clase {  
  
}  
  
class Sub_clase extends Super_clase {  
  
}
```

Super_Clase: Es el nombre de la clase de la cual se va a extender una clase derivada o subclase.

Sub_Clase: Es una clase que posee sus propios miembros y podrá utilizar los miembros de la superclase.

- **Sinónimos para Super_Clase y Sub_clase**

Al buscar información en libros o sitios Web, así como al mantener una comunicación con profesionales en el área veremos diferentes términos para

referirse a las clases que participan en la herencia. Los sinónimos empleados para referirse a las clases en la herencia son los siguientes:

Super clase = clase base = clase existente = clase padre

Subclase = clase derivada = clase extendida = clase hija

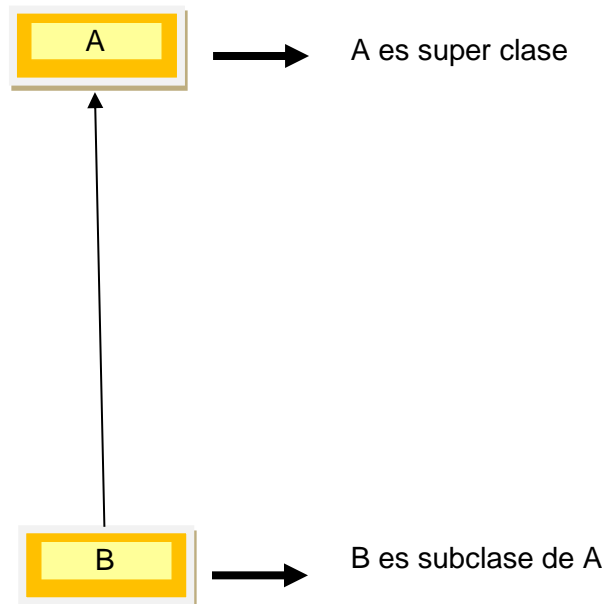
Veamos cómo trabajan los miembros de las clases cuando se define la palabra reservada `extends` para definir que una clase hereda.

Miembros de A

a
datos
b
p1()
métodos
p2()

Miembros de B

c
datos
d
p3()
métodos
p4()



La clase B tiene sus miembros propios como:
Datos **c, d** y métodos **p3() , p4()**

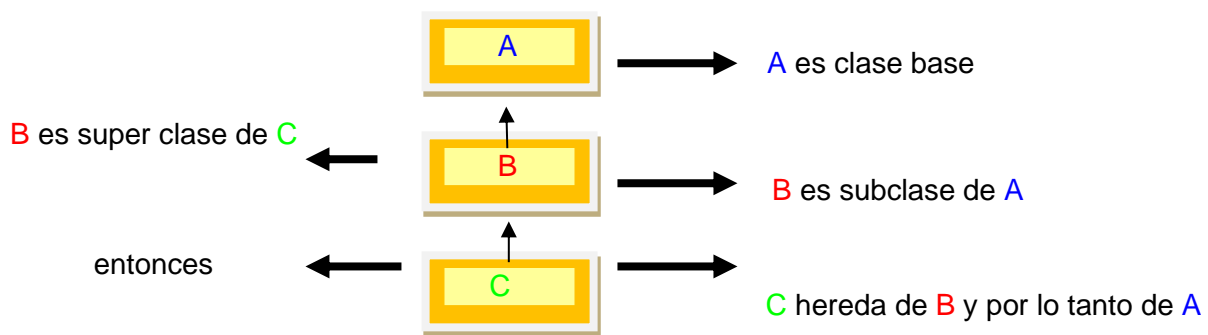
Y además puede reutilizar los miembros de la clase A porque los hereda. Los miembros heredados son: Datos **a, b** y Métodos **p1(), p2()**

En la herencia se mencionó la jerarquía clases ahora es importante destacar que la jerarquía se pueden dar diferentes niveles a esto se le llama transitividad en la herencia.

- **Transitividad en la herencia**

La herencia puede ser **transitiva**, de modo que una clase puede heredar características de superclases de muchos niveles.

Notemos la transitividad en la representación jerárquica de la herencia en el siguiente ejemplo.



La clase B hereda los miembros de la clase A.

La clase C hereda los miembros de la clase B y como B hereda los miembros de A entonces C hereda también los miembros de la clase A.

Cuando hablamos que B hereda A. Nos tenemos que referir a B como la subclase y de A como superclase.

Y cuando decimos que la clase C hereda de B. Nos tenemos que referir a C como subclase de B, B como superclase de C, B como subclase de A y A como superclase de B.

Como mencione antes que a la superclase también se le puede dar el nombre de clase padre, este término si lo manejamos biológicamente nos da que pensar que una subclase solamente puede tener una superclase y no varias. Por lo tanto, Java no permite la herencia múltiple. Usted puede ver el formato de la herencia para el lenguaje Java presentado antes que la palabra reservada `extends` solo permite especificar el nombre de una subclase y no varias.

En el mundo de la programación existen dos tipos de herencia, como lo son la herencia simple y múltiple.



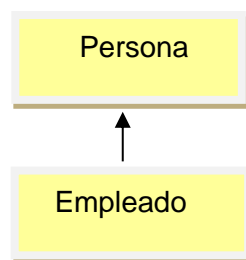
1.1.2 Tipos de Herencia

Como mencione antes, la herencia nos permite crear nuevas clases a partir de clases existentes y cada lenguaje lo implementa según lo soporta o fue definido. El lenguaje Java solamente soporta la herencia simple y permite simular la herencia múltiple. Cabe mencionar que otros lenguajes que permiten implementar la herencia múltiple meramente, es decir sin simulación. Sin embargo, es importante que como la herencia es una de las características de la programación orientada a objeto es valioso que usted este claro en que consisten cada una a pesar de que el lenguaje Java la herencia múltiple la simula. Con esto quiero decir que no la trabajan como otros lenguajes de programación porque, este tema lo trató genéticamente. Por ejemplo, genéticamente un hijo solo tiene un papá y no muchos papas.

- **Herencia simple**

Una clase puede heredar de una única clase. Recuerde que este tipo de herencia es la única que permite Java.

Representación jerárquica de la herencia múltiple

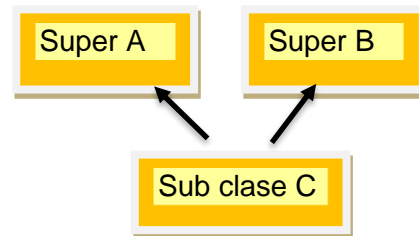


- **Herencia múltiple**

Una clase puede heredar de varias clases.

Como mencioné antes la herencia múltiple en Java no es permitida. No se permite que una subclase herede de múltiples superclases con la palabra `extends`.

Representación jerárquica de la herencia múltiple

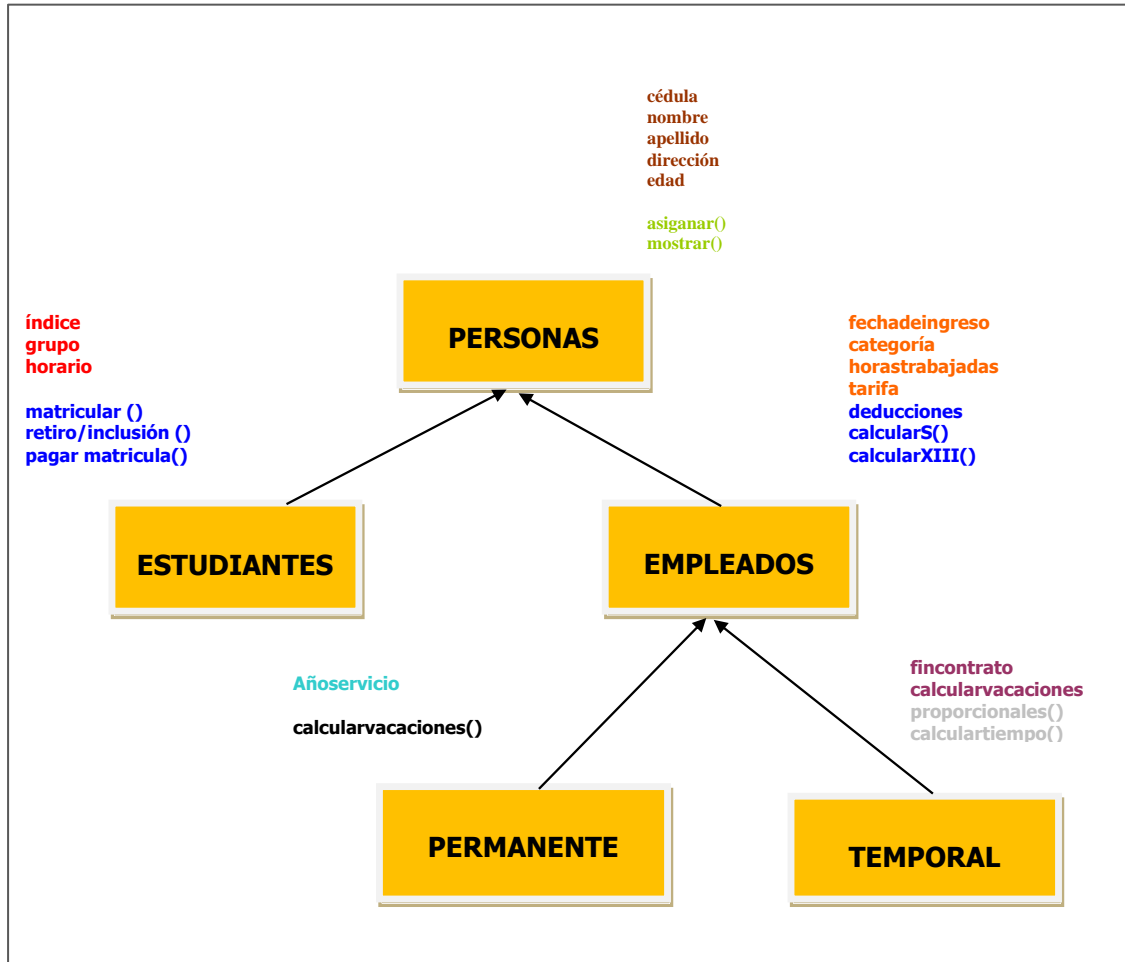


1.1.3 Ventajas de la herencia

La herencia nos ofrece muchas ventajas muy importantes al desarrollar aplicaciones empleando el lenguaje Java, tales como:

- Mecanismo importante de extensión que permite la **reutilización de código existente**.
 - **Evita la modificación del código existente** permitiendo añadir nuevas características o cambiar características.
 - **Evita duplicación de código idéntico o similar** en clases diferentes aplicando relación jerárquica.
- **Simplifica la escritura de código.**

A continuación, muestro un diagrama de jerárquica de clases para representar la herencia donde participan diferentes clases.



Antes introducir a ver que se hereda considerando el efecto que tienen los modificadores de acceso, la sobrecarga, sobreescritura y constructores empleando la característica o propiedad de la herencia tenga presente que el código de una superclase no debe ser modificado jamás, así son las reglas por seguridad porque podríamos afectar lógicamente a otras clases que están reutilizando esa superclase.

Por experiencia, de ahora en adelante hay que procurar no olvidar que toda clase debe tener métodos individualmente que permitan **retorna el valor de los atributos de una clase**. El nombre que le demos a esos métodos puede ser cualquiera, pero es necesario que usted conozca que el estándar utilizado para esos nombre es `get_nombre_del_atributo()`.



También cabe señalar que cuando se les inicio a enseñar Java empleando la metodología orientada a objetos en el curso de Desarrollo de software II se usa un solo método para asignar los valores a cada uno de los atributos de una clase, eso funciona. Pero a medida que usted va aprendiendo más de programación va a observar que en una clase ya no utilizaremos un solo método asignar para darle o almacenar valores a los atributos. Lo que escribiremos será un método `set_nombre_de_atributo()`.

Veamos cómo trabajan los miembros de las clases en un programa cuando se define la palabra reservada `extends` para definir una clase que hereda y cuál es el efecto de los modificadores de acceso como el por defecto, `public`, `protected` y `private`.

1.2 Efectos de los modificadores en la herencia

Los modificadores de acceso nos dan la posibilidad de poder establecer la **visibilidad** de los atributos y métodos de una clase y es una forma de establecer los permisos que tendrán los **objetos para acceder a los atributos y métodos** de la clase como se explicó en el curso de Desarrollo de Soft. II.

A continuación, explicare cómo funcionan los modificadores de acceso en la **herencia**.

1.2.1 Modificadores de acceso (`private`, `protected`, `public`, por defecto)

Como es de su conocimiento, usted estudio el efecto de los modificadores de acceso en una clase y en un objeto en la asignatura Desarrollo de Software II que el prerequisite para la asignatura Desarrollo de Software III. Esos conceptos estudiados antes no cambian para esta asignatura. Ahora entonces estudiaremos cual será el efecto de los modificadores en la herencia. Primero veremos los que se heredan.

Los miembros que se heredan por una **subclase** o en el **objeto** instanciado a la misma son aquellos declarados en la superclase:



- por defecto
- public
- protected

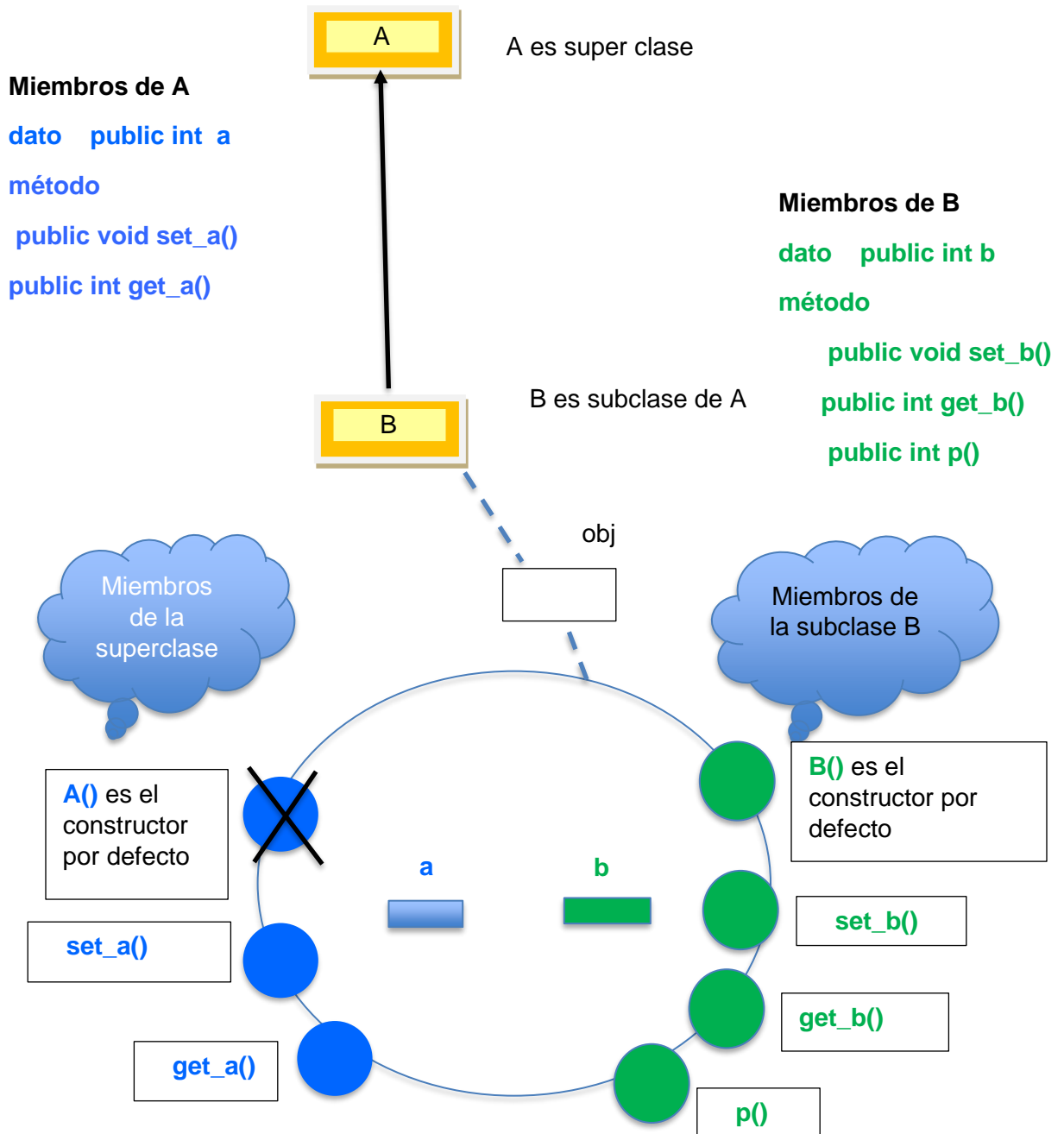


Primero observe el siguiente diagrama en donde se identifican 2 clases la super clase A y la subclase B. Respectivamente manejan los siguientes miembros propios para A como: Dato **a** y métodos **A()**, **set_a ()** y los miembros propios de B: Dato **b** y métodos **B()**, **p()**. Y la subclase B **puede reutilizar (no duplicar código) los miembros de la superclase A** porque los hereda.


Los miembros heredados son: Dato **a** y métodos **set_a ()**, **get_a()**. El método **p()** puede reutilizar lo que hereda de la superclase A. El método **A()** es el constructor por defecto y no se hereda, la explicación sobre los constructores y la herencia lo explicare en el tema llamado constructores y la herencia.

Recuerde que en el curso prerequisite usted estudió el método constructor por defecto y esto quiere decir que, aunque no esté definido en la clase toda clase tiene un constructor por defecto.

La tarea de un método constructor es el de crear el objeto e inicializar los datos o atributos del objeto.





La  denota que no se hereda, no es visible o se oculta. Como es el constructor lo explicaré en el punto los constructores y la herencia.



En el diagrama también puede observar al objeto que se crea o instancia a la subclase B. Recuerde que el elemento principal de la programación orientada a objetos es el objeto. En la herencia el objeto tendrá los miembros propios que son los de la subclase B y por herencia tendrá los de la superclase A que, según las reglas de los modificadores de acceso a los miembros declarados por defecto, public y protected son heredados o visibles.

Ejemplo: Son válidas las instrucciones obj. **set_a()**, obj. **get_a()**, obj. **set_b()**, obj. **get_b()** y obj. **p()** **porque para el objeto son visibles por ser métodos públicos.**

Igual ocurrirá si son métodos por defecto o protegidos.

Realizar los laboratorios: 1.1, 1.2
(Herencia1.java, Herencia2.java)

Ya vimos los miembros que se heredan, a continuación, veremos los que no se heredan.



Al decir no se heredan significa que miembros (datos o métodos) no son visibles, están ocultos o no se está autorizado para utilizar.

El termino no utilizar quiere decir en programación que no se le autoriza de usarlo solamente, pero sin olvidar que están allí y que no desaparecen.

Reflexionamos sobre el termino no se hereda en el mundo real. Un padre tiene un carro y el hijo tiene un carro. Si el padre fallece y el padre no lo nombra heredero. Entonces el hijo no hereda el carro de su padre, sin embargo, el carro está allí, tan solo que no lo puede utilizar.

Los miembros que **no se heredan** (**se ocultan**) en una **subclase** y **en el objeto** instanciado a la misma son aquellos declarados en la superclase `private`.

- `private`

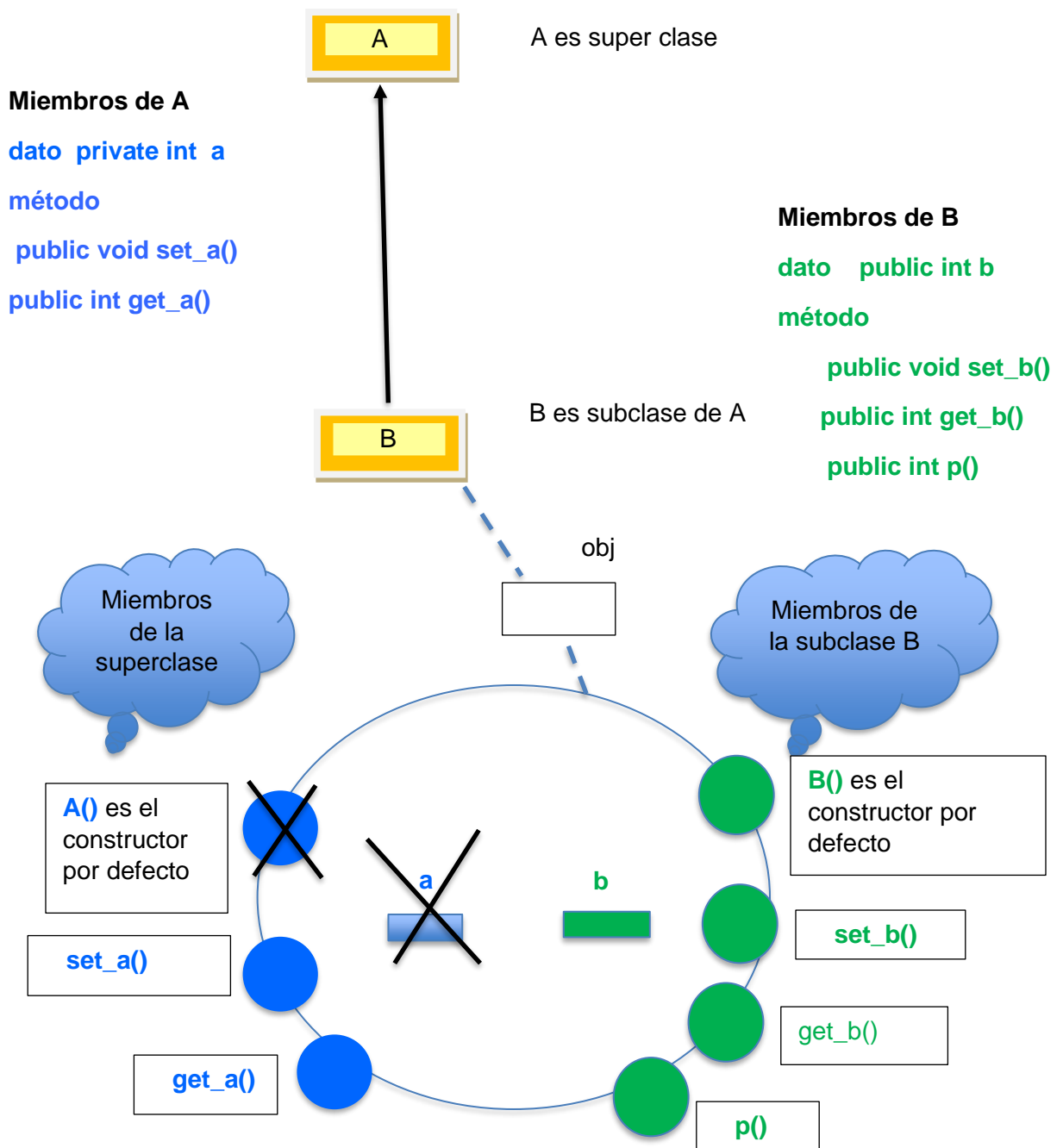
Para tener acceso a los miembros privados que se ocultan de la superclase en la subclase se utiliza el mecanismo de método intermediario, que es un método público que retorna el contenido del dato privado.



Usando el diagrama anterior le cambiaré el modificador de acceso al dato **a** en la superclase para el siguiente diagrama, para que puedan observar y comprender que sucede con los miembros private. Los miembros heredados son: Método **set_a ()** y **get_a()**. El método **p()** puede reutilizar lo que hereda de la superclase A en otras palabras los miembros public, protected o por defecto. En este caso el dato **a** es private y no puede ser utilizado por **p()** porque no se hereda o se oculta causando que no sea visible en la subclase.

Para poder tener acceso al dato **a** hay que tener un método intermediario llamado **get_a()** y puede ser public, protected o por defecto para que sea visible. En realidad, el modificador de acceso usado para tal efecto son el public y protected. Este método **get_a()** retorna el valor de **a** el cual, cualquier método de la subclase B podrá utilizar para establecer una comparación, realizar un cálculo pero no podrá modificar su valor provocando seguridad a el dato a través de la herencia.

Usted no debe pensar que un método **get_nombre_de_atributo()** es el unico método intermediario que existe. Esto dependerá del problema a resolver a veces existirán otros métodos que servirán como métodos intermediarios. Por razonamiento lógico lo vamos a distinguir si un método nos servirá como método intermediario. La lógica nos dirá que cuando un método publico retorna un valor generado de un cálculo o proceso donde utiliza al dato o atributo privado de acceso y requerimos de ese valor podremos utilizar a ese método como mecanismo de accesibilidad al dato privado.





En el diagrama anterior puede apreciar que el atributo **a** como es privado lo denotamos con una cruz y esto representa como mencioné antes que el atributo **a** no es accesible por el objeto o con otras palabras el atributo **a** no visible por el objeto.

Ejemplo: **Es invalida** la instrucción `obj. a`

Realizar el laboratorio 1.3, 1.4
(Herencia3.java, Herencia4.java)

Realizar asignación 1.1

He terminado de explicar todo lo referente a modificadores de acceso, a continuación, veremos que ocurre con la sobrecarga de métodos en la herencia.



1.3 Métodos sobrecargados en la herencia

Es importante que antes de explicar la sobrecarga de métodos y la herencia usted debe tener claro la definición de un método y la identificación de cada uno de sus componentes.

A manera de repaso presento el siguiente ejemplo para recordar la definición de un método, sus componentes y por último un método.

Definición de un método		
Componentes de un método		
Tipo de dato a retornar	Nombre de la función	Parámetros
int	pro	(int x, int y)

Con el ejemplo anterior recordamos los componentes de un método, ahora repasaremos que son métodos sobrecargados.

Los métodos sobrecargados son aquellos que tienen el **mismo nombre** y **parámetros diferentes en cantidad y/o tipos de datos**.

A continuación, ejemplos de parejas de métodos identificándolos si son sobrecargados o no y en caso de ser sobrecargados si es por cantidad de parámetros o tipo de datos.

Metodo1	Método 2	Son métodos sobrecargados	Parámetros
int pro (int x, int y)	int pro (int x, int y)	no	
float pro (int x, int y)	int pro (int x, int y)	no	
int pro (int x, int y)	int pro (int x, char y)	si	tipo de datos diferentes
int pro (int x, int y)	Int pro (int x)	si	cantidad de parámetros diferentes

Si tenemos claro lo explicado anteriormente en el punto 1.3 entonces seguimos a ver que ocurre con los métodos sobrecargados y la herencia.



Las subclases y los objetos **heredan** los métodos sobrecargados de la superclase. Cuando decimos se heredan queremos decir que los métodos son visibles, por lo que se pueden **usar dentro de la clase y el objeto**.

Realizar el laboratorio 2.1
(Php.java)

Realizar asignación 1.2

A continuación, veremos que ocurre con la sobreescritura y los constructores en la Herencia.

1.4 Métodos o datos sobre escritos y constructores en la herencia

Antes de ver los efectos de los de la sobrescritura de datos y métodos en la herencia tenemos que estar claros que es sobreescritura de datos y métodos:

- Los datos están sobrescritos cuando tienen el mismo nombre.
- Los métodos están sobrescritos cuando son iguales.
La **sobreescritura de métodos** nos permite redefinir un **método** que heredamos para que este funcione de acuerdo con nuestras necesidades en la subclase y no a lo definido en la superclase.

1.4.1 Utilizar super para acceso a miembros de la superclase:

Las subclases no heredan un miembro de la superclase. O sea, se **ocultan** los miembros de la superclase en la **subclase cuando:**

- Si la subclase declara un miembro **dato** con el mismo nombre que la superclase.
- Si la subclase declara un miembro **método** sobrescrito a la superclase.

Si instanciamos un objeto a la subclase entonces los miembros (datos y métodos) sobrescritos que corresponden a la superclase se ocultan en el objeto.

En este caso el mecanismo para tener acceso a los miembros de la superclase ocultos en la subclase es utilizando la palabra reservada **super** como se muestra a continuación:

Formatos:

- **super.dato;** **//para miembros dato**

- **super.método** (con/sin parámetros); **//para miembros métodos**



Un truco en la herencia es que cuando en la superclase exista un método y que al construir la subclase esta tenga que definir un método que realizará lo mismo, pero de otra forma diferente a la superclase, usted debe darle el mismo nombre al método de la subclase que al de la superclase (sobrescritura de Métodos). Entonces se dará sobrecarga o sobrescritura. Entonces si ocurrió sobrescritura usará el mecanismo correspondiente que es la palabra reservada **super.método** (con/sin parámetros) para lograr la reutilización de código. Si la superclase tiene un propio dato y la superclase también lo tiene del nombre (sobrescritura de datos). Con la sobrescritura de datos hay que tener mucho cuidado porque esto se aplica cuando ambos tienen el mismo dato como propio de sus características de la entidad.

Realizar el laboratorio 3.1, 3.2
(PHp.java,

Realizar asignación 1.3
(Baldosa_Piso)

Realizar asignación 1.4
(Padre_hijo)



1.4.2 Utilización de super para acceder a constructores de la superclase

Las subclases no heredan constructores de la superclase o sea se ocultan para la **subclase y para el objeto.**

Los constructores por defecto y sin parámetros de la super clase **se invocan automáticamente.** Esto se aplica también en la herencia de varios niveles.

El mecanismo para tener acceso al constructor de la superclase ocultado en la subclase es utilizar super como se muestra a continuación:

Formatos :

- `super ();` //constructor sin parámetros
- `super (lista de parámetros);` //constructor con parámetros

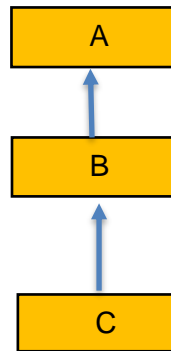
A continuación un ejemplo utilizando super () como mecanismo para referirse al constructor:

```
class A {  
    protected int a;  
    A (int aa) {  
        a=aa;  }  
    int mostrar_a(){  
        return a;  }  
}  
  
class B extends A{  
    private double b;  
    B (int aa, double bb){  
        super(aa);  
        b=bb;  }  
    public double mostrar_b(){  
        return b;  }  }
```

Recuerde que super (...) debe ir después inmediatamente de la declaración del constructor

Así que **super ()** o **super (lista de parámetros)** siempre se refiere a la superclase inmediatamente encima de la subclase que llama. **Esto se aplica incluso en una jerarquía multinivel.**

Ejemplo:



Utilizar **super** en la subclase B se refiere al constructor de la superclase A

Utilizar **super** en la subclase C se refiere al constructor de B

De existir varios constructores (sobrecargados). Entonces el constructor ejecutado será el primero al que correspondan los parámetros en **cantidad y/o tipo de dato**.

Ejemplo:

```
Class A {
```

```
    private int x;
```

```
    private int y;
```

```
    A(int xx, int yy )
```

```
    { x=xx;
      y=yy; }
```

```
    A ( )
```

```
    {x=10;
```

```
      Y=5; }
```

```
    A( int v)
```

```
    {x=y=v; }
```

```
}
```

Si se escribe la instrucción `A obj = new A (2, 3);` el constructor va a ejecutar el constructor que tiene 2 parámetros).



Realizar el laboratorio 4.1
(Pesocajap1.java)

Realizar el laboratorio 4.2
(Pesocajap2.java)

Realizar el laboratorio 4.3
(Pesocajap2.java)

Realizar asignación 1.5
(Limonada)

A continuación, presento un resumen de temerario tratado sobre la herencia:



		Los miembros de la superclase se ocultan en la subclase y en el objeto	Mecanismo para poder utilizar el dato o método en la subclase
Datos o métodos	por defecto, public, protected	no	
	private	si	Método intermediario
Métodos			
	sobrecargados	no	
Datos o métodos			
	sobre escritos	si	super. dato super.método(con/sin parámetros)
Métodos			
	Constructor() //sin parametro	si	Se llaman automáticamente o super()
	Constructor(con parámetros)		super (con parámetros)