



**Universidad Nacional Autónoma de  
México**  
**Facultad de Ingeniería**



Laboratorio de docencia  
Laboratorios de computación  
Laboratorios de computación A y B  
Práctica #7 Herencia

---

*Profesor:* Tista García Edgar

*Asignatura:* Programación Orientada a Objetos

*Grupo:* 3

*No de Práctica(s):* 7

*Integrante(s):* Félix Flores Paul Jaime

*Semestre:* 2019-2

*Fecha de entrega:* 28-03-2019

*Observaciones:*

CALIFICACIÓN: \_\_\_\_\_

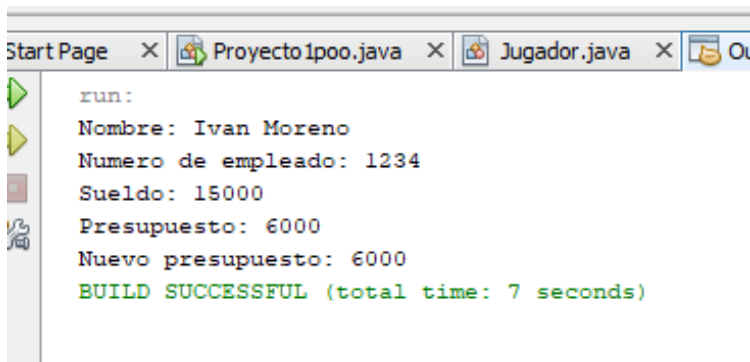
## OBJETIVO.

Implementar los conceptos de herencia en un lenguaje de programación orientado a objetos.

## DESARROLLO.

Ejercicio 1) Manual de laboratorio.

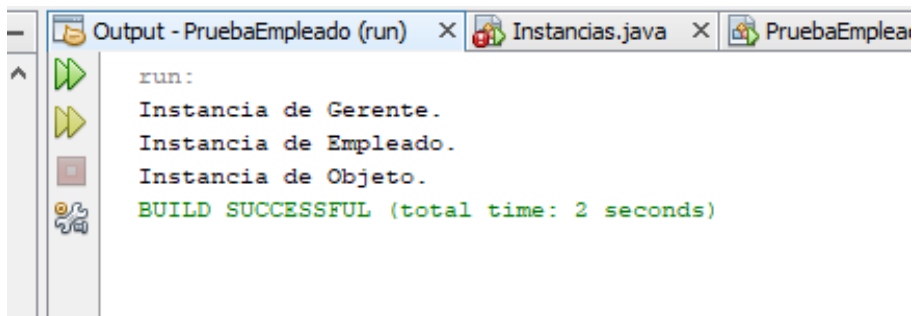
En esta primera parte creamos una clase principal llamada Empleado, a la cual le íbamos asignar atributos, así como crearle sus setters y getters. Una vez hecho esto creamos una clase hija llamada Gerente a la cual le íbamos heredar lo de nuestra clase principal Empleado, por consiguiente, en el menú principal mandamos asignábamos el sueldo y el presupuesto y lo mandamos a llamar mediante los métodos.



```
run:
Nombre: Ivan Moreno
Numero de empleado: 1234
Sueldo: 15000
Presupuesto: 6000
Nuevo presupuesto: 6000
BUILD SUCCESSFUL (total time: 7 seconds)
```

Ejercicio 2) Manual de laboratorio

En esta parte vimos el método *toString* es definido dentro de la clase *Object*. Este método es utilizado para mostrar información de un objeto. Por ejemplo, la clase *Empleado* posee los atributos *nombre*, *numEmpleado* y *sueldo*, los cuales se desean mostrar al momento de imprimir un objeto de esta clase, sin embargo, como el método *toString* está definido en la clase *Object* y ésta no conoce los atributos de *Empleado*, dichos atributos no se van a imprimir. Por lo tanto, para mostrar la información deseada es necesario sobrescribir el método.



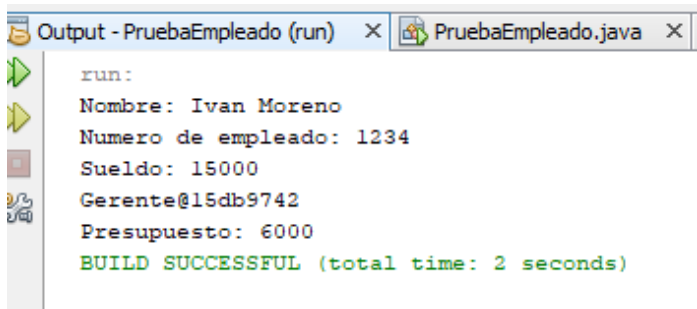
```
run:
Instancia de Gerente.
Instancia de Empleado.
Instancia de Objeto.
BUILD SUCCESSFUL (total time: 2 seconds)
```

### Ejercicio 3) Manual de laboratorio

En esta parte vimos la sobreescritura de los métodos así como el uso del `@override`. Si en la creación del objeto se usa el constructor sin argumentos (*constructor no-args*), entonces se produce una llamada implícita al constructor sin argumentos de la clase base.

Sin embargo, si se quiere utilizar constructores sobrecargados es necesario invocarlos explícitamente.

Así mismo, dentro de una clase derivada se puede acceder a los elementos de la clase base a través de la palabra reservada `super`.

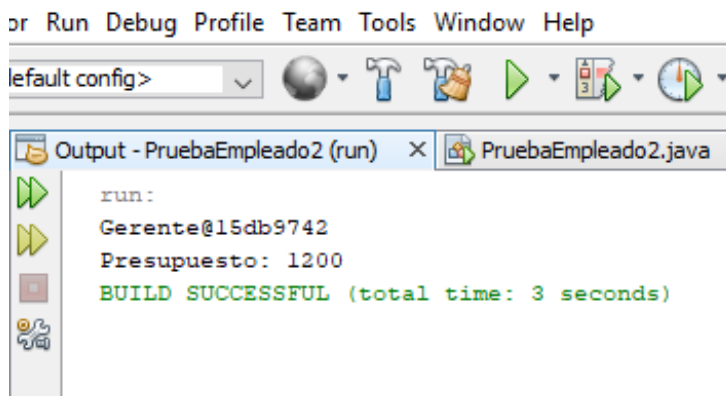


```
run:
Nombre: Ivan Moreno
Numero de empleado: 1234
Sueldo: 15000
Gerente@15db9742
Presupuesto: 6000
BUILD SUCCESSFUL (total time: 2 seconds)
```

#### Ejercicio 4) Manual de laboratorio

En esta parte observamos que, el constructor de *Ejecutivo* invoca directamente al constructor de *Empleado* mediante la palabra reservada *super*. La llamada al constructor de la superclase debe ser la primera sentencia del constructor de la subclase.

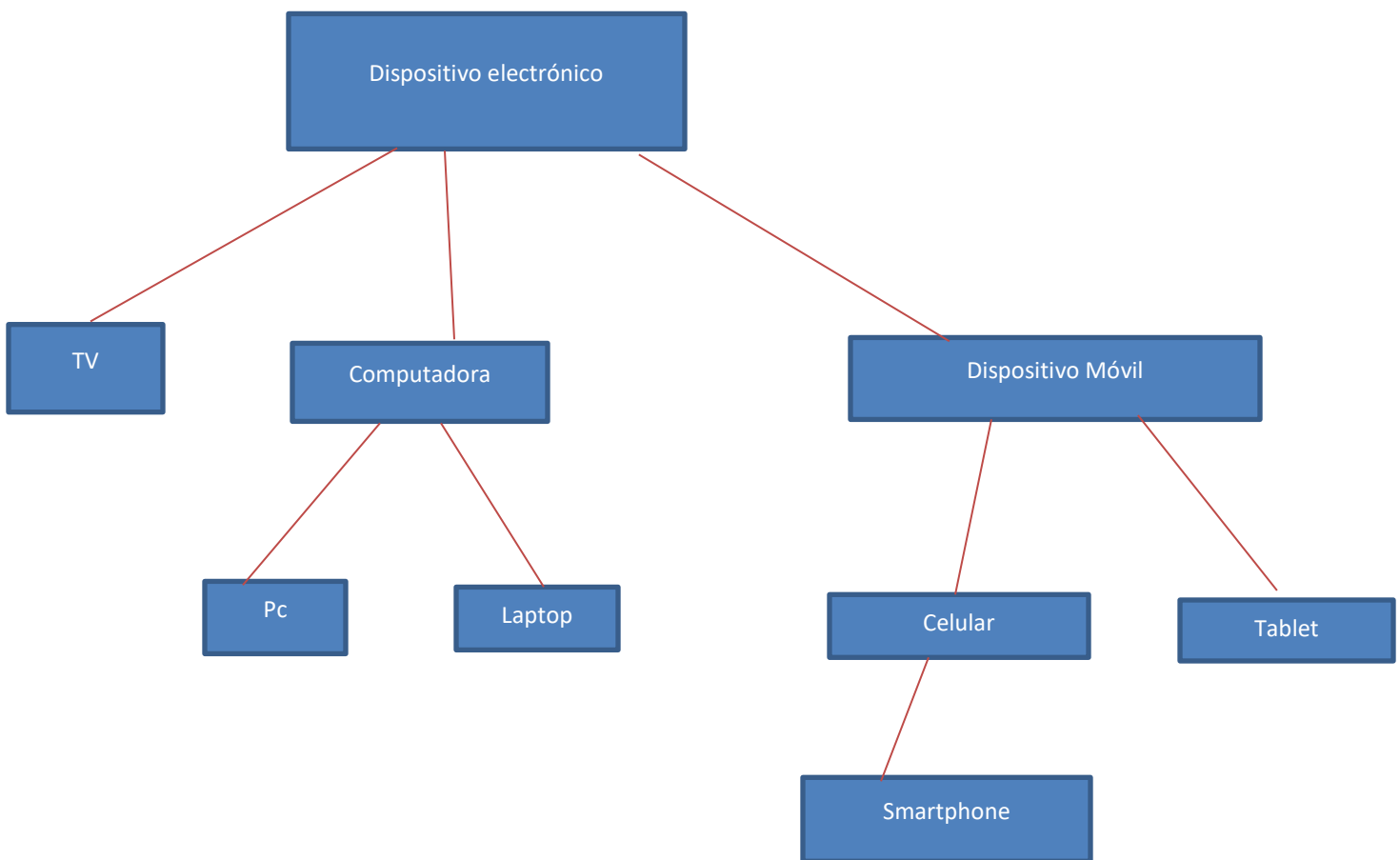
Debido a que el método *toString* de la clase *Empleado* muestra los atributos deseados, se invoca explícitamente y se agrega el atributo presupuesto. La clase *PruebaEmpleado* ahora sí mostrará toda la información del gerente.



```
or Run Debug Profile Team Tools Window Help
default config>
Output - PruebaEmpleado2 (run) x PruebaEmpleado2.java
run:
Gerente@15db9742
Presupuesto: 1200
BUILD SUCCESSFUL (total time: 3 seconds)
```

## Ejercicio 1)

Para resolver mi ejercicio decidí hacer mi jerarquía de clases de la siguiente manera, de esta manera pudo usar abstracción, polimorfismo y herencia al mismo tiempo, de echo quise llevar un poco más allá práctica. Usé esto ya que habíamos llegados hasta estos temas, así que no vi ningún inconveniente en juntarlos.



Lo que hice mi programa ,fue crear a “Dispositivo electronico “ como la Super clase y esta iba a tener sus atributos que iban a ser heredados a las otras clases ,hice la clase abstracta para que las demás clases y por lo cual cree un método abstracto , al igual tuve que ponerle el método abstracto, como sabemos un clase abstracta tiene metodos que no pueden ser desarrollados por falta de informacion concreta ,es decir, metodos abstractos.

Hice el método abstracto “Dispositivo electronico “,¿Por que iba hacer una clase abstracta ?,por la sencilla razón de que tendrá un método abstracto que nos va a decir cuánto vamos a pagar ,y ¿Por qué es abstracto ? ,por que las clases hijas darán sus propias instrucciones para decir cuanto van a pagar.

Para mejorar mi programa decidi hacer lo siguiente:

- Por cada 10,000 de compra en una tv el cliente recibira un descuento del 10 %

- Por cada 10,000 de compra en una tablet el cliente recibira un descuento del 5 %

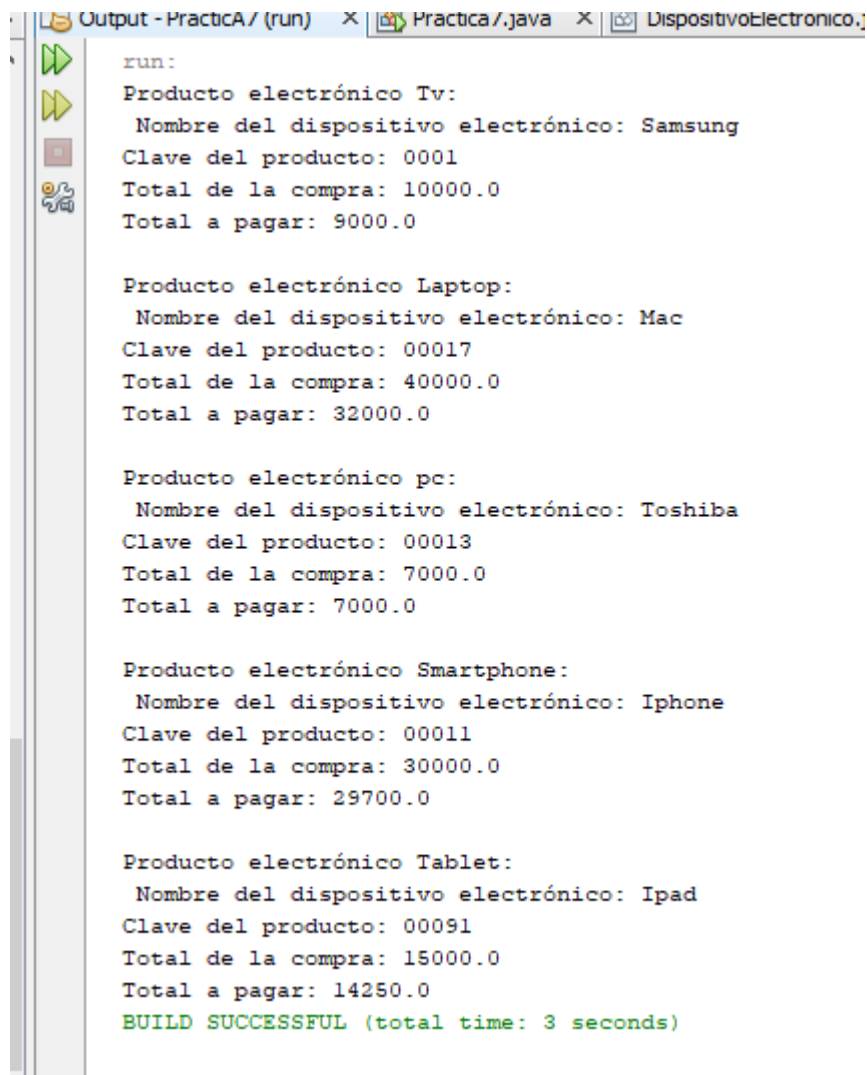
- Por cada 10,000 de compra en una smartphone el cliente recibira un descuento del 10 %

- Por cada 10,000 de compra en una pc el cliente recibira un descuento de 3 %

- Por cada 10,000 de compra en una laptop el cliente recibira un descuento del 20%

(En caso de no tener el pago indicado , no recibiran descuento )

Como vemos cada dispositivo electrónico tendrá su sobreescritura de una manera diferente, para así, poder mandar a imprimir en pantalla su información correspondiente.



```
Output - Practica/ (run) x | Practica/.java x | DispositivoElectronico.java
run:
Producto electrónico Tv:
  Nombre del dispositivo electrónico: Samsung
  Clave del producto: 0001
  Total de la compra: 10000.0
  Total a pagar: 9000.0

Producto electrónico Laptop:
  Nombre del dispositivo electrónico: Mac
  Clave del producto: 00017
  Total de la compra: 40000.0
  Total a pagar: 32000.0

Producto electrónico pc:
  Nombre del dispositivo electrónico: Toshiba
  Clave del producto: 00013
  Total de la compra: 7000.0
  Total a pagar: 7000.0

Producto electrónico Smartphone:
  Nombre del dispositivo electrónico: Iphone
  Clave del producto: 00011
  Total de la compra: 30000.0
  Total a pagar: 29700.0

Producto electrónico Tablet:
  Nombre del dispositivo electrónico: Ipad
  Clave del producto: 00091
  Total de la compra: 15000.0
  Total a pagar: 14250.0
BUILD SUCCESSFUL (total time: 3 seconds)
```

## **Conclusiones.**

En esta práctica se logró repasar el tema de la herencia, así como temas anteriores a este: el tema de los métodos sobrescritos, métodos abstractos, clases abstractas, el uso de getters y setters, invocación de los constructores.

Esta práctica no me costó mucho ya que estuve investigando sobre los temas, me pareció muy interesante el uso de las clases abstractas y con esta práctica pude entender mejor, quise llevarla un poco más allá, de lo básico, ya que pude hacer un programa para recibir un descuento de acuerdo al monto pagado.

Esta práctica me ayudó a reforzar mis conocimientos sobre clases y lo básico en Poo, así como reforzar los conocimientos vistos en clase y con los ejercicios hechos en ella.