

# CURSO DE PROGRAMACIÓN CON JAVA

## CASTING DE OBJETOS EN JAVA



Por el experto: Ing. Ubaldo Acosta

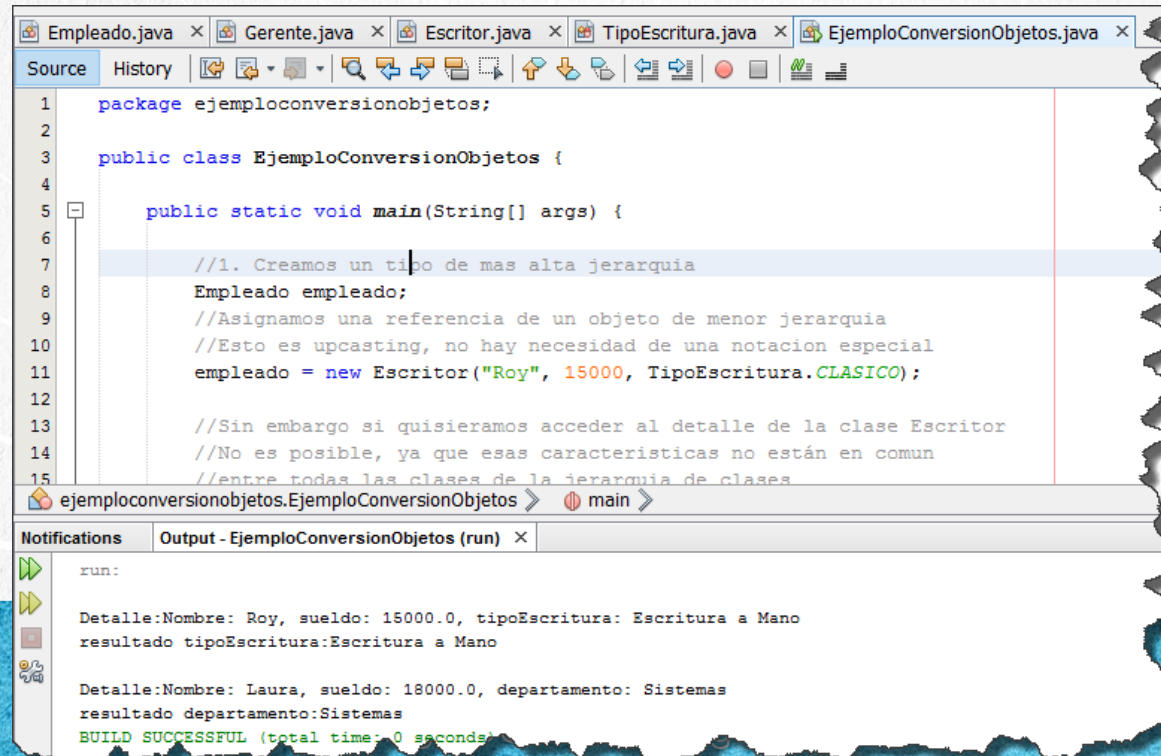


Experiencia y Conocimiento para tu vida



# OBJETIVO DEL EJERCICIO

Poner en práctica el concepto de casting o conversión de objetos en Java. Al finalizar deberemos observar lo siguiente:



The screenshot shows an IDE with several tabs: Empleado.java, Gerente.java, Escritor.java, TipoEscritura.java, and EjemploConversionObjetos.java. The active tab is EjemploConversionObjetos.java, displaying the following code:

```
1 package ejemploconversionobjetos;
2
3 public class EjemploConversionObjetos {
4
5     public static void main(String[] args) {
6
7         //1. Creamos un tipo de mas alta jerarquia
8         Empleado empleado;
9         //Asignamos una referencia de un objeto de menor jerarquia
10        //Esto es upcasting, no hay necesidad de una notacion especial
11        empleado = new Escritor("Roy", 15000, TipoEscritura.CLASICO);
12
13        //Sin embargo si quisieramos acceder al detalle de la clase Escritor
14        //No es posible, ya que esas características no están en comun
15        //entre todas las clases de la jerarquia de clases
16    }
17 }
```

The IDE also shows the output of the program in the 'Output - EjemploConversionObjetos (run)' window:

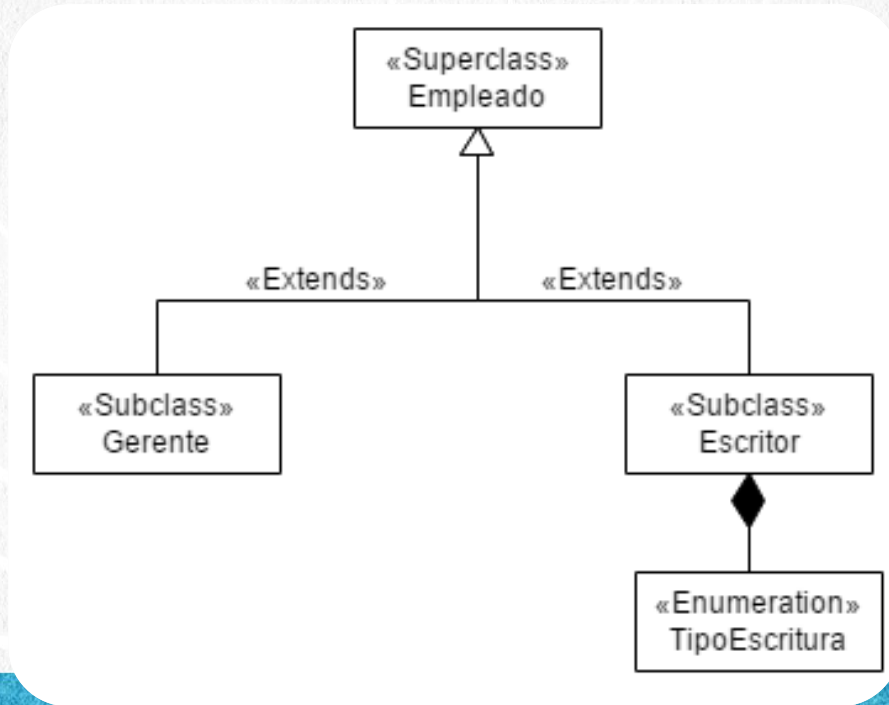
```
run:
Detalle:Nombre: Roy, sueldo: 15000.0, tipoEscritura: Escritura a Mano
resultado tipoEscritura:Escritura a Mano

Detalle:Nombre: Laura, sueldo: 18000.0, departamento: Sistemas
resultado departamento:Sistemas
BUILD SUCCESSFUL (total time: 0 seconds)
```



# DIAGRAMA UML DEL EJERCICIO

Este es el [diagrama UML](#) del ejercicio:



**CURSO DE PROGRAMACIÓN CON JAVA**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)

# PASO 1. CREACIÓN DEL PROYECTO

Vamos a crear el proyecto:

**New Java Application**

**Steps**

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class

# PASO 2. CREACIÓN CLASE EMPLEADO

Vamos a crear la clase Empleado:

New Java Class

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File: [:jemploConversionObjetos\src\ejemploconversionobjetos\Empleado.java]

< Back   Next >   **Finish**   Cancel   Help



# PASO 3. CREACIÓN CLASE GERENTE

Vamos a crear la clase Gerente:

New Java Class

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:

< Back   Next >   **Finish**   Cancel   Help

# PASO 4. CREACIÓN CLASE TIPOESCRITURA

Vamos a crear la clase TipoEscritura:

New Java Class

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name: TipoEscritura

Project: EjemploConversionObjetos

Location: Source Packages

Package: ejemploconversionobjetos

Created File: nploConversionObjetos\src\ejemploconversionobjetos\TipoEscritura.java

< Back Next > **Finish** Cancel Help

# PASO 5. CREACIÓN CLASE ESCRITOR

Vamos a crear la clase Escritor:

**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

Created File:



# PASO 6. MODIFICAMOS EL CÓDIGO

## Archivo Empleado.java:

```
package ejemploconversionobjetos;

public class Empleado {
    protected String nombre;
    protected double sueldo;

    protected Empleado(String nombre, double sueldo){
        this.nombre = nombre;
        this.sueldo = sueldo;
    }

    public String obtenerDetalles(){
        return "Nombre: " + nombre +
            ", sueldo: " + sueldo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public double getSueldo() {
        return sueldo;
    }

    public void setSueldo(double sueldo) {
        this.sueldo = sueldo;
    }
}
```

# PASO 7. MODIFICAMOS EL CÓDIGO

## Archivo Gerente.java:

```
package ejemploconversionobjetos;

public class Gerente extends Empleado{

    private String departamento;

    public Gerente(String nombre, double sueldo, String departamento) {
        super(nombre, sueldo);
        this.departamento = departamento;
    }

    //Sobreescribimos el metodo padre heredado
    public String obtenerDetalles(){
        //Observamos que para no repetir codigo, podemos utilizar
        //el metodo del padre y solo agregar a este metodo de la clase hija
        //esto es invocar un metodo sobreescrito
        return super.obtenerDetalles() + ", departamento: " + departamento;
    }

    public String getDepartamento() {
        return departamento;
    }

    public void setDepartamento(String departamento) {
        this.departamento = departamento;
    }
}
```

# PASO 8. MODIFICAMOS EL CÓDIGO

## Archivo TipoEscritura.java:

```
package ejemploconversionobjetos;

public enum TipoEscritura {

    CLASICO("Escritura a Mano"),
    MODERNO("Escritura digital");

    private final String descripcion;

    private TipoEscritura(String descripcion) {
        this.descripcion = descripcion;
    }

    public String getDescripcion() {
        return descripcion;
    }
}
```

**CURSO DE PROGRAMACIÓN CON JAVA**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)



# PASO 9. MODIFICAMOS EL CÓDIGO

## Archivo Escritor.java:

```
package ejemploconversionobjetos;

public class Escritor extends Empleado {

    //Utilizamos una enumeracion para las opciones de tipo de escritura
    final TipoEscritura tipoEscritura;

    public Escritor(String nombre, double sueldo, TipoEscritura tipoEscritura) {
        super(nombre, sueldo);
        this.tipoEscritura = tipoEscritura;
    }

    public String obtenerDetalles(){
        //Observamos que para no repetir codigo, podemos utilizar
        //el metodo del padre y solo agregar a este metodo lo de la clase hija
        //esto es invocar un metodo sobreescrito
        return super.obtenerDetalles() + ", tipoEscritura: " + tipoEscritura.getDescripcion();
    }

    public TipoEscritura getTipoEscritura() {
        return tipoEscritura;
    }

    public String getTipoDeEscrituraEnTexto() {
        return tipoEscritura.getDescripcion();
    }
}
```

# PASO 10. MODIFICAMOS EL CÓDIGO

## Archivo EjemploConversionObjetos.java:

```
package ejemploconversionobjetos;

public class EjemploConversionObjetos {

    public static void main(String[] args) {

        //1. Creamos un tipo de mas alta jerarquia
        Empleado empleado;
        //Asignamos una referencia de un objeto de menor jerarquia
        //Esto es upcasting, no hay necesidad de una notacion especial
        empleado = new Escritor("Roy", 15000, TipoEscritura.CLASICO);

        //Sin embargo si quisieramos acceder al detalle de la clase Escritor
        //No es posible, ya que esas características no están en comun
        //entre todas las clases de la jerarquia de clases
        //emp.getTipoDeEscrituraEnTexto();

        //Imprimimos los detalles en un metodo generico
        imprimirDetalles(empleado);

        //2. Podemos hacer lo mismo con la clase Gerente
        //Esto es upcasting, por lo que no requiere una sintaxis especial
        empleado = new Gerente("Laura", 18000, "Sistemas");

        //Pero si queremos acceder directo por la variable empleado
        //al detalle del objeto Gerente no es posible, se pierde el acceso
        //empleado.getDepartamento();

        //Utilizamos el mismo metodo para imprimir los detalles
        imprimirDetalles(empleado);
    }
}
```

# PASO 10. MODIFICAMOS EL CÓDIGO (CONT)

## Archivo EjemploConversionObjetos.java:

```
//Metodo generico para imprimir los detalles de la jerarquia Empleado
private static void imprimirDetalles(Empleado empleado) {

    String resultado = null;

    //Imprimir detalles es general para todos ya que es por polimorfismo
    //no se necesita de ninguna conversion
    System.out.println("\nDetalle:" + empleado.obtenerDetalles());

    //Pero los detalles de cada clase debemos hacer un downcasting
    if (empleado instanceof Escritor) {
        //Convertimos el objeto al tipo inferior deseado
        //Convierte objeto - Downcasting
        Escritor escritor = (Escritor) empleado;
        //Finalmente podemos acceder a los metodos de la clase Escritor
        resultado = escritor.getTipoDeEscrituraEnTexto();

        //Otra forma es hacer la conversion en la misma linea de codigo.
        //Esto es muy comun encontrarlo en Java
        //para evitar la creacion de variables innecesarias
        resultado = ((Escritor) empleado).tipoEscritura.getDescripcion();

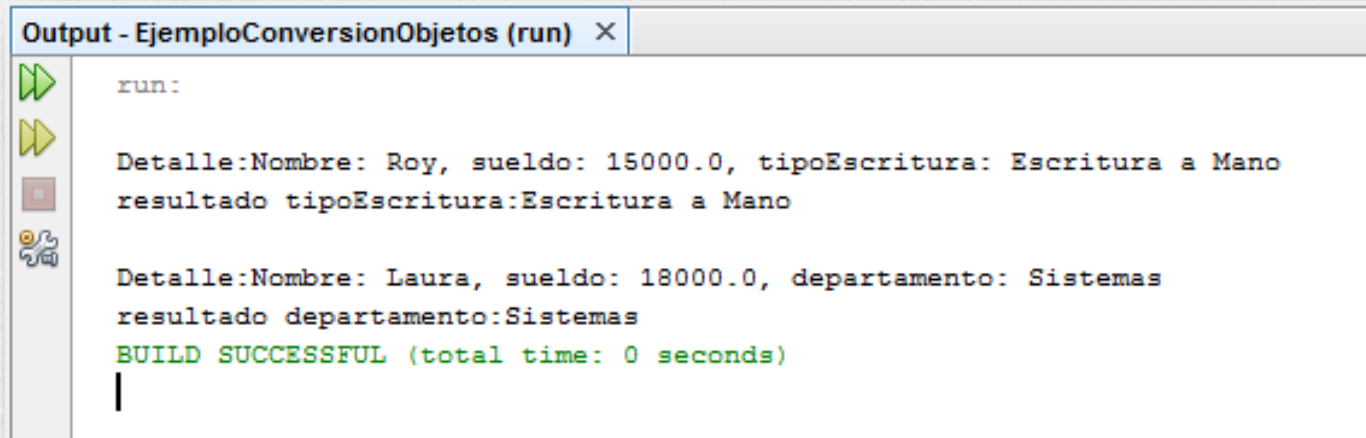
        System.out.println("resultado tipoEscritura:" + resultado);
    } else if (empleado instanceof Gerente) {
        //Hacemos el downcasting en la misma linea de codigo
        //nos ahorramos una variable
        resultado = ((Gerente) empleado).getDepartamento();

        System.out.println("resultado departamento:" + resultado);
    }
}
```



# PASO 11. EJECUCIÓN DEL PROYECTO

Ejecutamos el proyecto:



The screenshot shows an IDE output window titled "Output - EjemploConversionObjetos (run) X". On the left side of the window, there is a vertical toolbar with icons for running (a green play button), stepping through code (a yellow play button), stopping (a red square), and debugging (a magnifying glass over a bug). The output text is as follows:

```
run:
Detalle:Nombre: Roy, sueldo: 15000.0, tipoEscritura: Escritura a Mano
resultado tipoEscritura:Escritura a Mano

Detalle:Nombre: Laura, sueldo: 18000.0, departamento: Sistemas
resultado departamento:Sistemas
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

# TAREAS EXTRA DEL EJERCICIO

- Probar con el modo debug del IDE y verificar paso a paso.
- Se pueden crear más objetos y probar la ejecución del programa invocando al método `imprimirDetalles()`.



# CONCLUSIÓN DEL EJERCICIO

Con este ejercicio hemos puesto en práctica el concepto de conversión de objetos, también conocido como casting.

Hemos visto que el upcasting no necesita de una sintaxis particular, pero el downcasting el compilador si requiere que confirmemos si realmente queremos hacer esa conversión, además de que esa conversión debe estar dentro de la jerarquía de clases que hayamos definido.



**CURSO ONLINE**

# **PROGRAMACIÓN CON JAVA**

---

Por: Ing. Ubaldo Acosta



Experiencia y Conocimiento para tu vida

**CURSO DE PROGRAMACIÓN CON JAVA**

[www.globalmentoring.com.mx](http://www.globalmentoring.com.mx)