Laboratorio de Computación
Salas A y B

Profesor(a):	JOSE ANTONIO AYALA BARBOSA
Asignatura:	Programación Orientada a Objetos
Grupo:	2
No de Práctica(s):	07 Herencia
Integrante(s):	**Santos Carmona Valeria Sofia**
	Hernández Reyes Rebeca Sarai
	Almaguer Miranda Michel Santiago
No. de lista o brigada:	
Semestre:	2026-1
Fecha de entrega:	16 de Octubre 2025
Observaciones:	

CALIFICACIÓN: _____

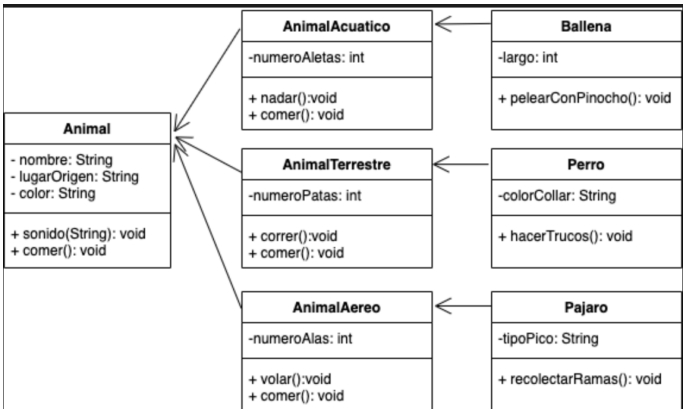
PRÁCTICA 7 Herencia

Integrantes : Santos Carmona Valeria Sofia | Hernandez Reyes Rebeca Sarai | Almaguer Miranda Michel Santiago

I. OBJETIVO

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

II. PREVIO



ACTIVIDADES

Crear clases que implementen herencia.

*Generar una jerarquía de clases.

Actividades extra

*Implementar clases generadas en el diagrama de clases e instanciarlas.

CÓDIGO FUENTE Y DESARROLLO

El proyecto se desarrolló en Java utilizando Apache NetBeans.

Clase empleado

```java
public class Empleado extends Object{
    private String nombre;
    private int numEmpleado;
    private double sueldo;

    //Sobrecarga
    public Empleado() {
    }

    public Empleado(String nombre, int numEmpleado, double sueldo) {
        this.nombre = nombre;
        this.numEmpleado = numEmpleado;
        this.sueldo = sueldo;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getNumEmpleado() {
        return numEmpleado;
    }
```

```java
    public void setNumEmpleado(int numEmpleado) {
        this.numEmpleado = numEmpleado;
    }

    public double getSueldo() {
        return sueldo;
    }

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

    public void aumentarSueldo(int porcentaje){
        // sueldo = sueldo + sueldo*porcentaje/100;
        sueldo += sueldo*porcentaje/100;
    }

    //Sobreescritura
    @Override
    public String toString(){
        return "nombre="+nombre+" numEmpleado="+numEmpleado+" sueldo="+sueldo;
    }
}
```

En esta primera clase se define el modelo base de un empleado en el cual se declaran atributos como lo es el nombre del empleado y su sueldo.

En esta se incluyen constructores que reciben estos datos y los asigna a las variables correspondientes (por ejemplo, this.nombre = nombre;).

La clase tiene varios métodos para poder mostrar esta información uno de ellos es un método para mostrar el nombre y el

sueldo, para imprimir los datos del empleado en pantalla.

Esta clase no depende de otra y toda la información es gestionada por ella En resumen: la clase Empleado solo guarda y muestra los datos básicos de un empleado crea un objeto con nombre y sueldo y permite consultarlo o imprimirlo.

Override se utiliza para indicar que se está sobrescribiendo (redefiniendo) un método que ya existe en una clase superior. En este caso, el método toString() no se crea desde cero, sino que ya existe en la clase base Object (todas las clases en Java heredan de Object).

Dentro del toString() redefinido, se construye y devuelve una cadena que contiene la información del empleado.

Clase gerente

```
11  public class Gerente extends Empleado{
12      private double presupuesto;
13
14      public Gerente() {
15      }
16
17      public Gerente(double presupuesto, String nombre, int numEmpleado, double sueldo) {
18          super(nombre, numEmpleado, sueldo);
19          this.presupuesto = presupuesto;
20      }
21
22      public double getPresupuesto() {
23          return presupuesto;
24      }
25
26      public void setPresupuesto(double presupuesto) {
27          this.presupuesto = presupuesto;
28      }
29
30      //Azucar sintactica
31      public void asignarPresupuesto(double presupuesto){
32          setPresupuesto(presupuesto);
33      }
34
35      @Override
36      public String toString() {
37          return super.toString()+"Gerente{" + "presupuesto=" + presupuesto + '}';
38      }
```

Gerente hereda estructura y comportamiento del empleado. Todo lo definido en Emplead está disponible en Gerente (directamente o mediante sobreescritura).

Dentro de gerente adicionamos atributos que no comparte con empleado como lo es presupuesto y no reemplaza nada de la clase padre solo lo complementa.

En el constructor de Gerente hay una línea con super(...), que sirve para enviar los datos del empleado(nombre, sueldo) al constructor de la clase base (Empleado).Después de eso, el constructor asigna los nuevos datos propios del gerente.

También hay un método para mostrar los datos, pero este método imprime tanto la información del empleado como la del gerente, combinando lo que viene de Empleado con lo que se agregó en Gerente.

El @Override garantiza que el método exista en la jerarquía y el uso de super.toString() → reutiliza la implementación de la superclase y la extiende con la información propia. Esto evita repetir la lógica de formato que ya existe en Empleado.

Instancias de ambas clases

```
12  public class POOP7 {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19
20          Empleado empl = new Empleado("Juan",18,30000);
21          System.out.println(empl.getNombre());
22          System.out.println(empl.toString());
23
24          Gerente emp2 = new Gerente(200000,"Luis",20,40000);
25          System.out.println(emp2.toString());
```

Se reserva espacio en memoria para un objeto de tipo empleado el constructor de empleado se ejecuta, recibiendo los

parámetros indicados por el usuario.Dentro del constructor, se asignan los valores a los atributos el objeto emp1 ahora posee su propio estado interno, con esos valores guardados.
Se ejecuta el método toString() sobrescrito en Empleado, y el resultado es una cadena en la que se reserva memoria para un objeto de tipo Gerente este objeto contiene también la parte heredada de Empleado, porque Gerente extiende a Empleado.
Se ejecuta primero el constructor de Empleado mediante la llamada super(nombre, sueldo);
y estoma su vez inicializa la parte "Empleado" del objeto (nombre y sueldo).
Luego el constructor de Gerente asigna sus propios atributos:
Se ejecuta el método toString() sobrescrito en Gerente, que llama a super.toString() (para obtener los datos del empleado) y le añade los suyos.

Proyecto 2

Clase Animal

```java
package animales;

/**
 *
 * @author ABJ
 */
public class Animal {
    private String nombre;
    private String lugarOrigen;
    private String color;

    public Animal() {
    }

    public Animal(String nombre, String lugarOrigen, String color) {
        this.nombre = nombre;
        this.lugarOrigen = lugarOrigen;
        this.color = color;
    }

    public String getNombre() {
        return nombre;
    }

    public String getLugarOrigen() {
        return lugarOrigen;
    }

    public String getColor() {
        return color;
    }

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

    public void setLugarOrigen(String lugarOrigen) {
        this.lugarOrigen = lugarOrigen;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public void sonido(){
        System.out.println("auuuuu");
    }
    public void comer(){
        System.out.println("comiendo");
    }
    }

    @Override
    public String toString() {
        return super.toString()+"Animal(" + "nombre=" + nombre + ", lugarOrigen=" + lugarOrigen + ", color="
    }

}
```

En la clase animal definimos los atributos en primer lugar como lo son el nombre o lugar de origen después los constructores donde primero tenemos el constructor animal vacío por defecto y luego Animal(String nombre, String lugarOrigen, String color): Constructor que inicializa todos los atributos de la clase.

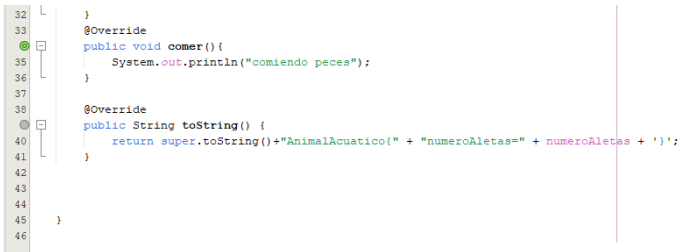
En los métodos existen get y set: Para cada uno de los atributos (nombre, lugarOrigen, color).
sonido(): Imprime el sonido que hace el animal
comer(): Imprime la acción genérica "comiendo".
toString() (Sobrescritura): Retorna una representación en cadena de los atributos de Animal

El Override indica que este método está destinado a sobrescribir un método de la superclase. En este caso, la superclase es Object
return super.toString() + "Animal("...: El método construye y devuelve una cadena de texto:

- super.toString(): Primero llama al método toString() de su superclase inmediata (que es Object). Esto es inusual para una clase base, pero en las clases derivadas es útil para incluir los atributos de los padres.
- "Animal(": Le añade la cadena literal "Animal(".
- "nombre=" + nombre: Incluye la cadena "nombre=" seguida del valor del atributo nombre.
- ", lugarOrigen=" + lugarOrigen: Le añade la cadena ", lugarOrigen=" seguida del valor del atributo lugarOrigen.
- ", color=": Le añade la cadena ", color=" (el resto del valor del atributo está cortado en la imagen).

Clase AnimalAcuatico

```java
package animales;

/**
 *
 * @author ABJ
 */
public class AnimalAcuatico extends Animal{
    private int numeroAletas;

    public AnimalAcuatico() {
    }

    public AnimalAcuatico(int numeroAletas, String nombre, String lugarOrigen, String color) {
        super(nombre, lugarOrigen, color);
        this.numeroAletas = numeroAletas;
    }

    public int getNumeroAletas() {
        return numeroAletas;
    }

    public void setNumeroAletas(int numeroAletas) {
        this.numeroAletas = numeroAletas;
    }

    public void nadar(){
        System.out.println("animal nadando");
    }
```

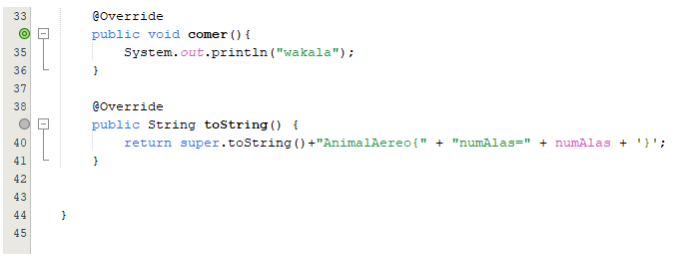


```java
    }
    @Override
    public void comer(){
        System.out.println("comiendo peces");
    }

    @Override
    public String toString() {
        return super.toString()+"AnimalAcuatico{" + "numeroAletas=" + numeroAletas + '}';
    }


}
```

Hereda de: Animal.
Atributo propio: numeroAletas (int).
Constructor: Inicializa su atributo propio y llama al constructor de la clase padre (super).
Método propio: nadar() (Imprime "animal nadando").
comer() (Sobrescritura): Imprime "comiendo peces".
toString() (Sobrescritura): Incluye la información de la clase padre más el numeroAletas

Clase AnimalAereo

```java
package animales;

/**
 *
 * @author ABJ
 */
public class AnimalAereo extends Animal {
    private int numAlas;

    public AnimalAereo() {
    }

    public AnimalAereo(int numAlas, String nombre, String lugarOrigen, String color) {
        super(nombre, lugarOrigen, color);
        this.numAlas = numAlas;
    }

    public int getNumAlas() {
        return numAlas;
    }

    public void setNumAlas(int numAlas) {
        this.numAlas = numAlas;
    }

    public void volar(){
        System.out.println("volandoo");
    }
```



```java
    @Override
    public void comer(){
        System.out.println("wakala");
    }

    @Override
    public String toString() {
        return super.toString()+"AnimalAereo{" + "numAlas=" + numAlas + '}';
    }


}
```

Hereda de: Animal.
Atributo propio: numAlas (int).
Constructor: Inicializa su atributo propio y llama al constructor de la clase padre (super).
Método propio: volar() (Imprime "volandoo").
comer() (Sobrescritura): Imprime "wakala".
toString() (Sobrescritura): Incluye la información de la clase padre más el numAlas.

Clase AnimalTerrestre

```java
package animales;

/**
 *
 * @author ABJ
 */
public class AnimalTerrestre extends Animal {
    private int numPatas;

    public AnimalTerrestre() {
    }

    public AnimalTerrestre(int numPatas, String nombre, String lugarOrigen, String color) {
        super(nombre, lugarOrigen, color);
        this.numPatas = numPatas;
    }

    public int getNumPatas() {
        return numPatas;
    }

    public void setNumPatas(int numPatas) {
        this.numPatas = numPatas;
    }

    public void correr(){
        System.out.println("fium fium");
    }
    @Override
    public void comer(){
        System.out.println("comiendo pasto");
    }

    @Override
    public String toString() {
        return super.toString()+"AnimalTerrestre{" + "numPatas=" + numPatas + '}';
    }


}
```

Hereda de: Animal.
Atributo propio: numPatas (int).
Constructor: Inicializa su atributo propio y llama al constructor de la clase padre (super).
Método propio: correr() (Imprime "fium fium").
comer() (Sobrescritura): Imprime "comiendo pasto".
toString() (Sobrescritura): Incluye la información de la clase padre más el numPatas.

Clases Hijas

Clase Perro

```java
package animales;

/**
 *
 * @author ABJ
 */
public class Perro extends AnimalTerrestre{
    private String colorCollar;

    public Perro() {
    }

    public Perro(String colorCollar, int numPatas, String nombre, String lugarOrigen, String color) {
        super(numPatas, nombre, lugarOrigen, color);
        this.colorCollar = colorCollar;
    }

    public String getColorCollar() {
        return colorCollar;
    }

    public void setColorCollar(String colorCollar) {
        this.colorCollar = colorCollar;
    }

    public void hacerTrucos(){
        System.out.println("miren  mi truco");
    }

    @Override
    public String toString() {
        return super.toString()+ "Perro{" + "colorCollar=" + colorCollar + '}';
    }


}
```

Hereda de: AnimalTerrestre.
Atributo propio: colorCollar (String).
Constructor: Llama al constructor de la clase padre (super) para los atributos de AnimalTerrestre y Animal, e inicializa su atributo propio
Método propio: hacerTrucos() (Imprime "miren mi truco").
toString() (Sobrescritura): Incluye la información de la clase padre más el colorCollar.

Clase Pajaro

```java
package animales;

/**
 *
 * @author ABJ
 */
public class Pajaro extends AnimalAereo {
    private String tipoPico;

    public Pajaro() {
    }

    public Pajaro(String tipoPico, int numAlas, String nombre, String lugarOrigen, String color) {
        super(numAlas, nombre, lugarOrigen, color);
        this.tipoPico = tipoPico;
    }

    @Override
    public String toString() {
        return super.toString()+"Pajaro{" + "tipoPico=" + tipoPico + '}';
    }


}
```

Hereda de: AnimalAereo.
Atributo propio: tipoPico (String).
Constructor: Llama al constructor de la clase padre (super) para los atributos de AnimalAereo y Animal, e inicializa su atributo propio.
toString() (Sobrescritura): Incluye la información de la clase padre más el tipoPico

Clase Ballena

```java
package animales;

/**
 *
 * @author ABJ
 */
public class Ballena extends AnimalAcuatico {
    private int largo;

    public Ballena() {
    }

    public Ballena(int largo, int numeroAletas, String nombre, String lugarOrigen, String color) {
        super(numeroAletas, nombre, lugarOrigen, color);
        this.largo = largo;
    }

    public int getLargo() {
        return largo;
    }

    public void setLargo(int largo) {
        this.largo = largo;
    }
```

```java

    public void pelearconPinocho(){
        System.out.println("muere pinocho ");
    }

    @Override
    public String toString() {
        return super.toString()+"Ballena{" + "largo=" + largo + '}';
    }


}
```

Hereda de: AnimalAcuatico.
Atributo propio: largo (int).}
Constructor: Llama al constructor de la clase padre (super) para los atributos de AnimalAcuatico y Animal, e inicializa su atributo propio.
Método propio: pelearconPinocho() (Imprime "muere pinocho ").
toString() (Sobrescritura): Incluye la información de la clase padre más el largo.

Instancias de las clases creadas

```java
        Ballena ballenal = new Ballena(20, 2, "moby", "atlantico", "gris");
        ballenal.pelearconPinocho();
        System.out.println(ballenal.toString());
        ballenal.sonido();
    }

}
```

CONCLUSIONES

Con esta actividad se logró comprender mejor la herencia al construir una jerarquía funcional que promueve la reutilización, el mantenimiento y la especialización del código, cumpliendo plenamente con el objetivo propuesto . La jerarquía organizada nos permite realizar un código lógico donde identificamos a la clase padre o base donde definimos atributos y en el caso de la clase animal los comportamientos básicos , luego subclases donde definimos el hábitat y atributos especiales de cada uno y al final las clases especializadas donde se añaden métodos específicos.
Se utilizó el polimorfismo en el método toString para sobrescribir en cada nivel e incluir los atributos específicos de cada clase junto con los atributos heredados a través de la llamada a super.toString().

REFERENCIAS

Bailón, J. y Baltazar, J. M. (2021). *Características de la POO*. Portal Académico del CCH, UNAM. Recuperado de https://portalacademico.cch.unam.mx/cibernetica1/algoritmos-y-codificacion/caracteristicas-POO

CONALEP Veracruz. (2021). *Programación Orientada a Objetos*. Recuperado de https://www.conalepveracruz.edu.mx/iniciobackup/wp-content/uploads/2021/03/Programaci%C3%B3n-orientada-a-objetos-M%C3%93DULO-PROFESIONAL.pdf

oyanes Aguilar, L. (1996). *Programación orientada a objetos*. McGraw-Hill.