Acosta Porcayo Alan Omar, Gutiérrez Grimaldo Alejandro, Medina Villa Samuel

Práctica 7. Herencia

Resumen

El objetivo es implementar los conceptos de herencia en un lenguaje de programación orientado a objetos, lo que implica crear clases que se basan en otras, estableciendo una jerarquía de clases. En esencia, se busca aprovechar la capacidad de una clase secundaria para heredar propiedades y métodos de una clase principal, lo que permite una organización y reutilización eficiente del código en un programa orientado a objetos.

Introducción

La herencia es omnipresente en la programación orientada a objetos, ya que permite que las clases hereden propiedades y métodos de otras clases. Esto fomenta la reutilización de código y facilita la organización de un programa.

Jerarquía de Clases

En la herencia, se establece una jerarquía de clases, con una "superclase" o "clase base" que se utiliza como base para crear "subclases" o "clases derivadas". Las clases derivadas heredan todas las propiedades y métodos de la clase base y pueden agregar los suyos propios.

Relación *IS-A* y *HAS-A*

En la herencia, se establece la relación *"IS-A"*, lo que significa que una clase derivada es un tipo específico de la clase base. También se menciona la relación *"HAS-A"*, que se basa en la composición y se refiere a la inclusión de una referencia de una clase en otra.

Acosta Porcayo Alan Omar Ing. en Computación 320206102
Gutiérrez Grimaldo Alejandro Ing. en Computación 320282098
Medina Villa Samuel Ing. en Computación 320249538

Clase *Object*

En lenguajes de programación orientados a objetos como Java, todas las clases heredan implícitamente de la clase *Object*. Esto significa que todas las clases poseen métodos como *clone, equals, hashCode, notify* y otros definidos en *Object*.

Sobrescritura (*Overriding*)

La sobrescritura se refiere a la capacidad de una subclase para redefinir el comportamiento de un método heredado de la clase base. Esto permite ajustar el comportamiento de un método de acuerdo con las necesidades de cada clase.

Constructores en la Herencia

Los constructores son métodos especiales utilizados para inicializar los atributos de un objeto. Cuando se crea un objeto de una clase derivada, se crea implícitamente un objeto de la clase base con su constructor correspondiente. Se explica cómo se pueden invocar constructores sobrecargados y acceder a elementos de la clase base utilizando la palabra reservada "super".

Sobrecarga (*Overloading*) vs. Sobrescritura (*Overriding*)

Se destacan las diferencias entre la sobrecarga y la sobrescritura. La sobrecarga implica definir varios métodos con el mismo nombre en una clase, pero con diferentes tipos y números de parámetros, mientras que la sobrescritura se aplica a la herencia y permite redefinir métodos heredados en subclases.

Objetivos

- Crear una estructura de clases que muestre una relación jerárquica sólida, con una clase base y múltiples clases derivadas, para demostrar cómo

la herencia puede facilitar la organización del código.

- Utilizar la herencia para reutilizar propiedades y métodos de la clase base en las clases derivadas, mostrando cómo esta técnica simplifica el desarrollo al evitar la duplicación de código.
- Implementar los conceptos de herencia en un lenguaje de programación orientado a objetos.

Metodología

Ejercicio realizado por el profesor

Ejemplo de herencia **Clase *Vehiculo***

```
package practica7;

public class Vehiculo {
  private String numSerie = "";

  public String getNumSerie() {
    return numSerie;
  }

  public void setNumSerie(String a) {
    this.numSerie = a;
  }

  @Override
  public String toString() {
    return "Vehiculo{" + "numSerie=" + numSerie + '}';
  }
}
```

Clase *Automovil*

```
package practica7;

public class Automovil extends Vehiculo {
  public Automovil() {
    super();
  }
}
```

Clase *AutoElectrico*

```
package practica7;

public class AutoElectrico extends Automovil{

}
```

Clase *Avion*

```
package practica7;

public class Avion extends Vehiculo{

}
```

Clase *Barco*

```
package practica7;

public class Barco extends Vehiculo {
  @Override
  public String toString() {
    //System.out.println(super.toString());
    return "Barco{" + "numSerie=" + getNumSerie() + '}';
  }
}
```

Clase *Fabrica*

```
package practica7;

public class Fabrica {
  public static void main(String[] args) {
    Vehiculo a = new Avion(); //Upcasting

    if (a instanceof Avion) {
      System.out.println("a es una instancia de Avion");
    }
    if (a instanceof Vehiculo) {
      System.out.println("a es una instancia de Vehiculo");
    }
    //Avion b = new Vehiculo();

    Vehiculo b = (Vehiculo) a;
    if (b instanceof Avion) {
      System.out.println("b es una instancia de Avion");
    }
    if (b instanceof Vehiculo) {
      System.out.println("b es una instancia de Vehiculo");
    }

    Barco titanic = new Barco();
    titanic.setNumSerie("KLA0009101");
    System.out.println(titanic);

    Automovil mercedes = new Automovil();
    mercedes.setNumSerie("MNX0009101");
    System.out.println(mercedes);

    Avion boeing = new Avion();
    boeing.setNumSerie("BOE0009101");
    System.out.println(boeing);
  }
}
```

Resultados

Problema 1

Investigue el uso de *ENUM* y aplíquelo en el ejemplo visto en laboratorio para:

a. Crear tipos de aviones supersónicos, militares, turísticos y de carga.

b. Escriba en sus conclusiones de este reporte, a qué se parece la estructura *ENUM*.

Clase *Vehiculo*

```
package practica7.ejercicio1;

public class Vehiculo {
  private String numSerie = "";

  public String getNumSerie() {
    return numSerie;
  }

  public void setNumSerie(String a) {
    this.numSerie = a;
  }

  @Override

  public String toString() {
    return "Vehiculo{" + "numSerie=" + numSerie + '}';
  }
}
```

Clase *Automovil*

```
package practica7.ejercicio1;
```

```
public class Automovil extends Vehiculo {
  public Automovil() {
    super();
  }
}
```

Clase *AutoElectrico*

```
package practica7.ejercicio1;

public class AutoElectrico extends Automovil{

}
```

Clase *Avion*

```
package practica7.ejercicio1;

public class Avion extends Vehiculo{
  //enum some differents kinds of airplanes
  public enum TipoAvion {
    supersonico, militar, turustico, carga
  }

  private TipoAvion tipo;

  public Avion(TipoAvion tipo) {
    this.tipo = tipo;
  }

  public TipoAvion getTipo() {
    return tipo;
  }

  public void setTipo(TipoAvion tipo) {
    this.tipo = tipo;
  }
}
```

Clase *Barco*

```
package practica7.ejercicio1;

public class Barco extends Vehiculo {
  @Override
  public String toString() {
    //System.out.println(super.toString());
    return "Barco{" + "numSerie=" + getNumSerie() + '}';
  }
}
```

Clase *Fabrica*

```
package practica7.ejercicio1;

public class Fabrica {
  public static void main(String[] args) {

    //Avion supersonico
    Avion caza = new Avion(Avion.TipoAvion.supersonico);
    caza.setNumSerie("CAZ0009101");
    System.out.println(caza);
    if (caza instanceof Avion) {
      System.out.println("caza es una instancia de Avion");
    }
    if (caza instanceof Vehiculo) {
      System.out.println("caza es una instancia de Vehiculo");
    }

    // Avion militar
    Avion concorde = new Avion(Avion.TipoAvion.militar);
    concorde.setNumSerie("CON0009101");
    System.out.println(concorde);
    if (concorde instanceof Avion) {
      System.out.println("concorde es una instancia de Avion");
    }
    if (concorde instanceof Vehiculo) {
      System.out.println("concorde es una instancia de Vehiculo");
    }

    // Avion turistico
    Avion boeing = new Avion(Avion.TipoAvion.turustico);
    boeing.setNumSerie("BOE0009101");
    System.out.println(boeing);
    if (boeing instanceof Avion) {
      System.out.println("boeing es una instancia de Avion");
    }
    if (boeing instanceof Vehiculo) {
      System.out.println("boeing es una instancia de Vehiculo");
    }

    //Avion de carga
    Avion airbus = new Avion(Avion.TipoAvion.carga);
    airbus.setNumSerie("AIR0009101");
    System.out.println(airbus);
    if (airbus instanceof Avion) {
      System.out.println("airbus es una instancia de Avion");
    }
    if (airbus instanceof Vehiculo) {
      System.out.println("airbus es una instancia de Vehiculo");
    }
  }
}
```

Ejecución

```
Vehiculo{numSerie=CAZ0009101}
caza es una instancia de Avion
caza es una instancia de Vehiculo
Vehiculo{numSerie=CON0009101}
concorde es una instancia de Avion
concorde es una instancia de Vehiculo
Vehiculo{numSerie=BOE0009101}
boeing es una instancia de Avion
boeing es una instancia de Vehiculo
Vehiculo{numSerie=AIR0009101}
airbus es una instancia de Avion
airbus es una instancia de Vehiculo
```

Problema 2

Utilice el código visto en laboratorio para crear una competencia de carreras, considere:

a. La variable velocidad inicial de tipo privada para cada objeto creado.

b. Una variable de aceleración/desaceleración que se determine como un numero aleatorio entre 1-20 $(+/-\ 10)$

c. Realice una simulación de carreras entre tipos de *Vehículo* en una pista de longitud de 1000 km, utilice un ciclo *for* para simular el tiempo t. Calcule la distancia recorrida usando la velocidad inicial y decrementando o incrementando la aceleración del número aleatorio (que será distinta para cada vehículo y para cada instante t).

d. Cree usando *HASH TABLE* que contenga al menos un vehículo por cada tipo.

e. Coloque en consola la sig. Salida (Ejemplo):

```
                MAPA DE LA PISTA
                - = 20 km
Instante de tiempo (Ti) = 10


0                                   1000 meta
--------------------------------------------------
      a
   b
      c
         d
--------------------------------------------------

a -> Objeto tipo Avion: Vi = 80 km/h Distancia recorrida Actualmente = 167 km
b -> Objeto tipo Barco: Vi = 32 km/h Distancia recorrida Actualmente = 84 km
c -> Objeto tipo Automovil:  Vi = 40 km/h Distancia recorrida Actualmente = 121 km
d -> Objeto tipo Avion Supersonico: Vi =110 km/h Distancia recorrida Actualmente = 260 km
```

Clase *Vehiculo*

```java
package practica7.ejercicio2;

import java.util.Random;

public class Vehiculo {
  private String numSerie = "";
  private int velocidadInicial;
  private int aceleracionDesaceleracion;

  public String getNumSerie() {
    return numSerie;
  }

  public void setNumSerie(String a) {
    this.numSerie = a;
  }

  public Vehiculo() {
    Random rand = new Random();
    velocidadInicial = rand.nextInt(100) + 1;
    aceleracionDesaceleracion = rand.nextInt(20) - 10;
  }

  public int getVelocidadInicial() {
    return velocidadInicial;
  }

  public int getAceleracionDesaceleracion() {
    return aceleracionDesaceleracion;
  }

  @Override
  public String toString() {
      return "Vehiculo{" + "numSerie=" + numSerie + '}';
  }
}
```

Clase *Automovil*

```java
package practica7.ejercicio2;

public class Automovil extends Vehiculo {
  public Automovil() {
    super();
  }
}
```

Clase *AutoElectrico*

```java
package practica7.ejercicio2;

import java.util.Random;

public class AutoElectrico extends Automovil {
  private int velocidadInicial;
  private int aceleracionDesaceleracion;

  public AutoElectrico() {
    Random rand = new Random();
    velocidadInicial = rand.nextInt(100) + 1;
    aceleracionDesaceleracion = rand.nextInt(20) - 10;
  }

  public int getVelocidadInicial() {
    return velocidadInicial;
  }

  public int getAceleracionDesaceleracion() {
    return aceleracionDesaceleracion;
  }
}
```

Clase *Avion*

```java
package practica7.ejercicio2;

import java.util.Random;

public class Avion extends Vehiculo {
  private int velocidadInicial;
  private int aceleracionDesaceleracion;

  public Avion() {
    Random rand = new Random();
    velocidadInicial = rand.nextInt(100) + 1;
    aceleracionDesaceleracion = rand.nextInt(20) - 10;
  }

  public int getVelocidadInicial() {
    return velocidadInicial;
  }

  public int getAceleracionDesaceleracion() {
    return aceleracionDesaceleracion;
  }
}
```

Clase *AvionSupersonico*

```java
package practica7.ejercicio2;

import java.util.Random;

class AvionSupersonico extends Avion{
  private int velocidadInicial;
  private int aceleracionDesaceleracion;

  public AvionSupersonico() {
    Random rand = new Random();
    velocidadInicial = rand.nextInt(100) + 1;
    aceleracionDesaceleracion = rand.nextInt(20) - 10;
  }

  public int getVelocidadInicial() {
    return velocidadInicial;
  }

  public int getAceleracionDesaceleracion() {
    return aceleracionDesaceleracion;
  }
}
```

Clase *Barco*

```java
package practica7.ejercicio2;

import java.util.Random;

public class Barco extends Vehiculo {
  private int velocidadInicial;
  private int aceleracionDesaceleracion;

  public Barco() {
    Random rand = new Random();
    velocidadInicial = rand.nextInt(100) + 1;
    aceleracionDesaceleracion = rand.nextInt(20) - 10;
  }

  public int getVelocidadInicial() {
    return velocidadInicial;
  }

  public int getAceleracionDesaceleracion() {
    return aceleracionDesaceleracion;
  }
}
```

Clase *Fabrica*

```java
package practica7.ejercicio2;

import java.util.Hashtable;

public class Fabrica {
  public static void main(String[] args) {
```

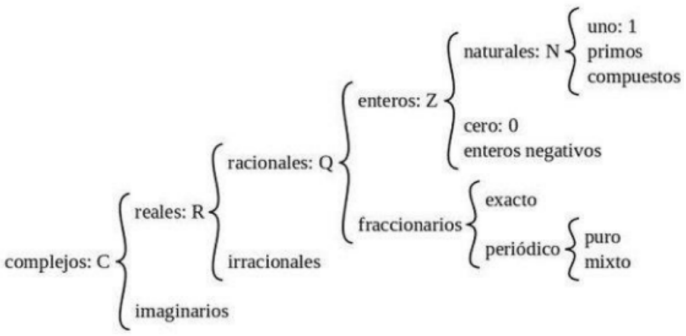
```
    Avion avion = new Avion();
    Barco barco = new Barco();
    AvionSupersonico avionSupersonico = new AvionSupersonico();
    Automovil automovil = new Automovil();

    Vehiculo[] vehiculos = { avion, barco, avionSupersonico, automovil };

    Hashtable<String, Vehiculo> vehiculosHash = new Hashtable<>();

    vehiculosHash.put(avion.getNumSerie(), avion);
    vehiculosHash.put(barco.getNumSerie(), barco);
    vehiculosHash.put(avionSupersonico.getNumSerie(), avionSupersonico);
    vehiculosHash.put(automovil.getNumSerie(), automovil);

    int longitudPista = 20;
    Vehiculo ganador = null;
    int tiempoGanador = Integer.MAX_VALUE;

    int tiempo = 0;

    while (ganador == null) {
      System.out.printf("MAPA DE LA PISTA");
      System.out.println("Instante de tiempo: " + tiempo + " min");
      System.out.println("0.............20 meta");
      System.out.println("------------------------");
      boolean todosLlegaron = true;
      for (int i = 0; i < vehiculos.length; i++) {
        Vehiculo vehiculo = vehiculos[i];

        int aceleracionAleatoria = vehiculo.getAceleracionDesaceleracion();
        int nuevaVelocidad = vehiculo.getVelocidadInicial() + aceleracionAleatoria;

        if (nuevaVelocidad < 0) {
          nuevaVelocidad = 0;
        }

        int distanciaRecorrida = (nuevaVelocidad * tiempo) / 60;

        if (distanciaRecorrida < longitudPista) {
          todosLlegaron = false;
        }

        StringBuilder pista = new StringBuilder(".....................");
        char etiqueta = (char) ('a' + i);
        pista.setCharAt(distanciaRecorrida >= 19 ? 19 : distanciaRecorrida, etiqueta);

        String nombreVehiculo = vehiculo.getClass().getSimpleName();
        String distanciaRecorridaStr = String.valueOf(distanciaRecorrida);
        String velocidadStr = String.valueOf(nuevaVelocidad);
        System.out.printf("%s\n", pista);
        System.out.printf("%c -> Objeto tipo %s: Vi = %s km/h
               Distancia recorrida Actualmente = %s km\n",
               etiqueta, nombreVehiculo, velocidadStr, distanciaRecorridaStr);

        if (distanciaRecorrida >= longitudPista) {
          if (tiempo < tiempoGanador) {
            ganador = vehiculo;
            tiempoGanador = tiempo;
          }
        }
      }

      if (todosLlegaron) {
        break;
      }

      tiempo += 1;
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      System.out.print("\033[H\033[2J");
      System.out.flush();
    }

    System.out.println("------------------------");
    if (ganador != null) {
      System.out.println("El ganador es: " + ganador.getClass().getSimpleName());
      System.out.println("Tiempo del ganador: " + tiempoGanador + " min");
    }
  }
}
```

Ejecución

```
MAPA DE LA PISTA
Instante de tiempo: 17 min
0.............20 meta
.............a.....
a -> Objeto tipo Avion: Vi = 53 km/h Distancia recorrida Actualmente = 14 km
...............b..
b -> Objeto tipo Barco: Vi = 64 km/h Distancia recorrida Actualmente = 17 km
.................c
c -> Objeto tipo AvionSupersonico: Vi = 79 km/h Distancia recorrida Actualmente = 21 km
..d..................
d -> Objeto tipo Automovil: Vi = 9 km/h Distancia recorrida Actualmente = 2 km
------------------------
El ganador es: AvionSupersonico
Tiempo del ganador: 16 min
```

Problema 3

Implemente la estructura de los conjuntos numerales que usamos en Matemáticas. Siguiendo la siguiente jerarquía:



Donde se aplique la herencia considerando las atributos y métodos que considere, por ejemplo, considere el atributo de conmutatividad y cerradura y un método existencia de elemento idéntico y obtenga ese elemento. Aplique propiedades propias del polimorfismo dinámico y estático, así como super.

Tome en cuenta lo siguiente:

- $\mathbb{I}$ los números imaginarios se deben modelar con una clase de tipo *final*, dado que ya no puede heredar.
- No considere a los números fraccionarios ni sus divisiones, entiéndase que son racionales.
- No considere las divisiones de lo naturales, solo manéjelos como conjunto general.
- Considere al 0 como una clase final.

Clase *Complejos*

```
public class Complejos {
  private double parteReal;
  private double parteImaginaria;
```

```
  public Complejos() {}

  public Complejos(double parteReal, double parteImaginaria) {
    this.parteReal = parteReal;
    this.parteImaginaria = parteImaginaria;
  }

  public double getParteReal() {
    return parteReal;
  }

  public void setParteReal(double parteReal) {
    this.parteReal = parteReal;
  }

  public double getParteImaginaria() {
    return parteImaginaria;
  }

  public void setParteImaginaria(double parteImaginaria) {
    this.parteImaginaria = parteImaginaria;
  }

  public boolean pertenece() {
    return this instanceof Complejos;
  }

  public String elementoIdentico() {
    return "0.0";
  }

  public String inversoAditivo() {
    return "(" + (-parteReal) + ", " + (-parteImaginaria) + ")";
  }

  public String inversoMultiplicativo() {
    return "(" + (parteReal /
    (parteReal * parteReal + parteImaginaria * parteImaginaria))
    + ", " +
    (-parteImaginaria /
    (parteReal * parteReal + parteImaginaria * parteImaginaria)) + ")";
  }


  public String toString() {
    return "Complejos";
  }
}
```

Clase *Reales*

```
public class Reales extends Complejos {
  private double valor;

  public Reales() {
    super();
  }

  public Reales(double valor) {
    super();
    this.valor = valor;
  }

  public double getValor() {
    return valor;
  }

  public void setValor(double valor) {
    this.valor = valor;
  }

  @Override
  public boolean pertenece() {
    return this instanceof Reales;
  }

  @Override
  public String elementoIdentico() {
    return "0.0";
  }

  @Override
  public String inversoAditivo() {
    return "" + (-valor);
  }

  @Override
  public String inversoMultiplicativo() {
    return "" + (1 / valor);
  }

  @Override
  public String toString() {
    return "Reales";
  }
}
```

Clase *Imaginarios*

```
public final class Imaginarios extends Complejos{
  private double parteReal;
  private double parteImaginaria;

  public Imaginarios() {
    super();
  }

  public Imaginarios(double parteReal, double parteImaginaria) {
    super();
    this.parteReal = parteReal;
    this.parteImaginaria = parteImaginaria;
  }

  public double getParteReal() {
    return parteReal;
  }

  public void setParteReal(double parteReal) {
    this.parteReal = parteReal;
  }

  public double getParteImaginaria() {
    return parteImaginaria;
  }

  public void setParteImaginaria(double parteImaginaria) {
    this.parteImaginaria = parteImaginaria;
  }

  @Override
  public boolean pertenece() {
    return this instanceof Imaginarios;
  }

  @Override
  public String elementoIdentico() {
    return "0.0";
  }

  @Override
  public String inversoAditivo() {
    return "(" + (-parteReal) + ", " + (-parteImaginaria) + ")";
  }

  @Override
  public String inversoMultiplicativo() {
    return "(" + (parteReal /
    (parteReal * parteReal + parteImaginaria * parteImaginaria))
    + ", " +
    (-parteImaginaria /
    (parteReal * parteReal + parteImaginaria * parteImaginaria)) + ")";
  }

  @Override
  public String toString() {
    return "Imaginarios";
  }
}
```

Clase *Racionales*

```
public class Racionales extends Reales{
  private double valor;

  public Racionales(){
    super();
  }

  public Racionales(double valor){
    super();
    this.valor = valor;
  }

  public double getValor(){
    return valor;
  }

  public void setValor(double valor){
    this.valor = valor;
  }
```

```
  @Override
  public boolean pertenece(){
    return this instanceof Racionales;
  }

  @Override
  public String elementoIdentico(){
    return "0.0";
  }

  @Override
  public String inversoAditivo(){
    return "" + (-valor);
  }

  @Override
  public String inversoMultiplicativo(){
    return "" + (1/valor);
  }

  @Override
  public String toString(){
    return "Racionales";
  }
}
```

Clase *Irracionales*

```
public final class Irracionales extends Reales{
  private double valor;

  public Irracionales(){
    super();
  }

  public Irracionales(double valor){
    super();
    this.valor = valor;
  }

  public double getValor(){
    return valor;
  }

  public void setValor(double valor){
    this.valor = valor;
  }

  @Override
  public boolean pertenece(){
    return this instanceof Irracionales;
  }

  @Override
  public String elementoIdentico(){
    return "No existe elemento identico";
  }

  @Override
  public String inversoAditivo(){
    return "No existe inverso aditivo";
  }

  @Override
  public String inversoMultiplicativo(){
    return "No existe inverso multiplicativo";
  }

  @Override
  public String toString(){
    return "Irracionales";
  }
}
```

Clase *Enteros*

```
public class Enteros extends Racionales {
  private int valor;

  public Enteros(){
    super();
  }

  public Enteros(int valor){
    super();
    this.valor = valor;
  }

  public int getValorE(){
    return valor;
  }

  public void setValorE(int valor){
    this.valor = valor;
  }

  @Override
  public boolean pertenece(){
    return this instanceof Enteros;
  }

  @Override
  public String elementoIdentico(){
    return "0";
  }

  @Override
  public String inversoAditivo(){
    return "" + (-valor);
  }

  @Override
  public String inversoMultiplicativo(){
    return (this.valor == 1) ? "1" : "No existe inverso multiplicativo";
  }

  @Override
  public String toString(){
    return "Enteros";
  }
}
```

Clase *Naturales*

```
public class Naturales extends Enteros {
  private int valor;

  public Naturales() {
    super();
  }

  public Naturales(int valor) {
    super();
    this.valor = valor;
  }

  public int getValorN() {
    return valor;
  }

  public void setValorN(int valor) {
    this.valor = valor;
  }

  @Override
  public boolean pertenece() {
    return this instanceof Naturales;
  }

  @Override
  public String elementoIdentico() {
    return "1";
  }

  @Override
  public String inversoAditivo() {
    return "No existe inverso aditivo";
  }

  @Override
  public String inversoMultiplicativo() {
    return (this.valor == 1) ? "1" : "No existe inverso multiplicativo";
  }

  @Override
  public String toString() {
    return "Naturales";
  }
}
```

Clase *Cero*

```
public final class Cero extends Enteros {
  final int valor = 0;

  public Cero() {
    super();
  }
```

```
  public int getValorC() {
    return valor;
  }

  @Override
  public boolean pertenece() {
    return this instanceof Cero;
  }

  @Override
  public String elementoIdentico() {
    return "0";
  }

  @Override
  public String inversoAditivo() {
    return "0";
  }

  @Override
  public String inversoMultiplicativo() {
    return "No existe inverso multiplicativo";
  }

  @Override
  public String toString() {
    return "Cero";
  }
}
```

Clase *EnterosNegativos*

```
public class EnterosNegativos extends Enteros{
  private int valor;

  public EnterosNegativos(){
    super();
  }

  public EnterosNegativos(int valor){
    super();
    this.valor = valor;
  }

  public int getValorEN(){
    return valor;
  }

  public void setValorEN(int valor){
    this.valor = valor;
  }

  @Override
  public boolean pertenece(){
    return this instanceof EnterosNegativos;
  }

  @Override
  public String elementoIdentico(){
    return "0";
  }

  @Override
  public String inversoAditivo(){
    return "" + (-valor);
  }

  @Override
  public String inversoMultiplicativo(){
    return (this.valor == -1) ? "-1" : "No existe inverso multiplicativo";
  }

  @Override
  public String toString(){
    return "EnterosNegativos";
  }
}
```

Conclusiones

En resumen, la herencia en el contexto de la Programación Orientada a Objetos (POO) es un principio fundamental que posibilita la formación de una estructura jerárquica de clases, permitiendo que las clases derivadas hereden tanto propiedades como métodos de una clase base.

Esta característica de reutilización de código facilita la organización y la estructuración del software, lo que a su vez mejora la eficiencia en el proceso de desarrollo y en el mantenimiento del sistema. Además, la herencia capacita para modelar relaciones *"IS-A"*, donde una subclase se considera un tipo específico de su superclase, reflejando de esta manera las relaciones entre objetos en el mundo real.

En última instancia, la herencia se erige como una herramienta poderosa que contribuye a la construcción de software más modular, expansible y de comprensión sencilla.

Referencias

Solano, J. (2017, 20 enero). *Manual de prácticas de Programación Orientada a Objetos*. Laboratorio de Computación Salas A y B. http://lcp02.fi-b.unam.mx/