Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

upiita-ipn

Ingeniería Mecatrónica
Programación Avanzada

REPORTE DE LA PRÁCTICA No. 3
Herencia

Nombre del Alumno: Zarazua Aguilar Luis Fernando
Grupo: 2MV4 **Fecha:** 05/Septiembre/2017

Objetivo: Identificar las características, el uso y las ventajas que posee la programación orientada a objetos al momento de tener Herencia.

Resumen

En el presente documento se habla sobre las características, el uso y las ventajas que se tienen al momento de usar la Herencia como una herramienta principalmente de reutilización de código, partiendo sobre de la necesidad de implementar la Herencia cuando se tienen datos en común entre distintas clases con la finalidad de no volver a escribir el mismo código en varias partes del programa en repetidas ocasiones, ejemplificando esa problemática con un ejemplo de una clasificación taxonómica y como se ordena esta por medio del uso de la Herencia.

Posteriormente se habla sobre la ventaja de usar código ya compilado para reducir el tiempo de programación y aumentar la eficacia en su ejecución, para luego introducir los elementos característicos que tiene la Herencia en java como lo son "extends" y "super", además de las características que un objeto con herencia posee como lo es poder ser tratado como el objeto base, pero sin perder la información de todo el objeto con herencia.

Introducción

En la programación orientada a objetos las clases son las entidades principales con los que trabajamos y desarrollamos la aplicación deseada, en base a estas podemos tener clases que contengan a otras clases dentro de ellas como un tipo de dato, sin embargo al momento de querer realizar con clases que comparten datos del mismo tipo estaríamos repitiendo código que en una parte ya fue usado, con lo cual la ventaja de tener una entidad como la clase que nos sirve para no tener que repetir código para objeto distinto se vería de cierta forma sin utilidad, ya que el programador al final estaría repitiendo código para cada objeto con características similares, es por este motivo que se tiene la Herencia como una herramienta para evitar estas situaciones. [1]

Planteamiento del problema.

En el momento en él se requiere programar con un cierto diagrama de clases, ya especificado el cual contiene clases con atributos y métodos en común tal como lo puede ser una clasificación taxonómica es necesario la utilización de la Herencia como una herramienta de reutilización de código, en la cual se comparten atributos y métodos que se tienen en común, de esta manera suponiendo una clase que contenga distintos animales, existirán algunos que sean mamíferos, reptiles u aves entre otros, de los mamíferos todos son vivíparos y de los reptiles todos son ovíparos, por lo tanto se crean las clases de ovíparos y vivíparos como clases base y como la de reptiles como una superclase que contenga a ovíparos, al igual que la clase aves, de igual manera se hace con los demás atributos y métodos de clase animales, conteniendo la mayor cantidad de características en común pero teniendo la posibilidad de agregar características específicas. [2]

En adición la herencia también posee otro importante uso el cual radica en la reutilización del código que se creó como una clase previamente compilada sobre la cual no se tiene acceso y solo conocimiento de sus atributos y sobre lo que devuelve cada método, siendo la herencia ahora una herramienta por la cual en lugar de reescribir el código y probablemente no tener tan buen desempeño como en el original, solamente se toma dicha clase como una clase base y se complementa con los métodos y atributos que logren definir completamente la clase que deseamos, un ejemplo así es que el que se propone en el desarrollo.

Desarrollo

Para ejemplificar el uso de la Herencia se crearon las clases "CarRegistro" y "ParkingAcceso".

CarRegistro.java

```
package aplicacion01;
public class CarRegistro extends Carro{
  protected boolean Acceso;
  CarRegistro(){
   super();
   Acceso=false;
  }
CarRegistro(String M1, String M2, int O){
   super(M1,M2,O);
   Acceso=true;
  }
    public boolean isAcceso() {
        return Acceso;
    }
    public void setAcceso(boolean Acceso) {
        this.Acceso = Acceso;
    }
}
```

Observaciones: Para crear la clase "CarRegistro" se usó como como clase base la clase "Carro" con la palabra reservada "extends" con lo cual se indica que "CarRegistro" tiene como herencia a "Carro", por lo tanto puede contener todos miembros y métodos de "Carro", sin embargo como tiene nuevos atributos esta clase como lo es Acceso, necesita un constructor que indique los

valores iniciales que tiene la clase con herencia, proponiendo para este ejemplo que cuando el constructor sea vacío, el Acceso sea falso, y cuando se le ingresan elementos el Acceso sea verdadero.

Los métodos que posee "CarRegistro" son "isAcceso" que nos regresa el valor de Acceso y "setAcceso" que nos permite darle el valor a Acceso.

ParkingAcceso.java

```
package aplicacion01;
public class ParkingAcceso extends Parking{
 ParkingAcceso(){//Constructor
  super();
 }
 public boolean GetAcceso(CarRegistro C){
    return C.isAcceso();
 }
}
```

Observaciones: Se usa la palabra reservada "extends" para crear una clase con Herencia de Parking teniendo así en ParkingAcceso una clase que contiene todo lo de Parking, pero además tiene un método más que nos permite obtener el valor de Acceso de un determinado "CarRegistro". El constructor lleva adentro la palabra reservada "super" que indica que está realizando la construcción del objeto heredado en este caso sin parámetros, aunque también puede contener parámetros como se da el caso en "CarRegistro".

Aplicacion01.java

```
package aplicacion01;
import java.util.Scanner;
public class Aplicacion01 {
   public static void main(String[] args) {
    Scanner Opcion;
    ParkingAcceso UPIITA;
    UPIITA=new ParkingAcceso();
    int opc;
    Opcion=new Scanner(System.in);
    do
    {
     System.out.println("Bienvenido al Estacionamiento UPIITA");
     System.out.println("1-Agregar Automovil");
     System.out.println("2-Listar Automoviles");
     System.out.println("3-Verificar Acceso");
     System.out.println("5-Salir");
     opc=Opcion.nextInt();
     //IngresaCoche(UPIITA);
     switch(opc)
     {
        case 1: IngresaCoche(UPIITA);//Pasa un ParkingAcceso que contiene un Parking.
              break;
        case 2: ListarCoches(UPIITA);
```

```
            break;
       case 3: VerificarCoche(UPIITA);
            break;
    }
  }while(opc!=5);
 }
 public static boolean IngresaCoche(Parking P)
 {
  if (P==null)
  {System.out.println("Error al crear objeto");
     return false;}
  else
  {
   String Propietario;
   String Marca;
   int Modelo;
   Scanner Texto;
   Texto=new Scanner(System.in);
   System.out.println("Desea agregar información adicional (S/N)");
   if (Texto.nextLine().equals("S")){
   System.out.println("Ingrese el nombre del propietario:");
   Propietario=Texto.nextLine();
   System.out.println("Ingrese la marca del automovil:");
   Marca=Texto.nextLine();
   System.out.println("Ingrese el modelo");
   Modelo=Texto.nextInt();
   CarRegistro c;
   c=new CarRegistro(Propietario, Marca, Modelo);
   P.addCar(c);}
   else{
   CarRegistro c;
   c=new CarRegistro();
   P.addCar(c);
   }
   return true;
  }
 }
  public static void ListarCoches(Parking p)//Acepta un objeto tipo Parking
 {
  CarRegistro Aux;
  int i;
  int n=p.getNumCar();
  for (i=0;i<n;i++)
  {
    Aux=(CarRegistro)p.getCar(i);
    System.out.println(i+".-"+Aux.getModelo());
    System.out.println("Propietario: "+Aux.getPropietario());
    System.out.println("Marca: "+Aux.getMarca());
  }
 }
 public static void VerificarCoche(ParkingAcceso P)
 {
  Scanner Texto;
```

```
    Texto=new Scanner(System.in);
    System.out.println("Ingrese el numero de coches a verificar");
    int n=Texto.nextInt();
    CarRegistro C=(CarRegistro)P.getCar(n);
    if (C!=null){
       System.out.println("El acceso es: ");
       System.out.println(C.isAcceso());
    }
   }
}
```

Observaciones: En el método "VerificarCoche" se usa un objeto con herencia el cual es P, que es un "ParkingAcceso", por medio de este método se obtiene que valor de Acceso tiene un determinado "Carro", o si existe ese objeto de tipo "Carro", usando un método que originalmente tiene "Parking", pero como "ParkingAcceso" la tiene como herencia puede obtener el "Carro", estos elementos de tipo "Carro" a su vez fueron añadidos como "CarRegistro" con el método "addCar" el cual puede añadir cualquier tipo de "Carro" incluyendo a "CarRegistro", entonces puede realizar el cast para en vez de obtener un "Carro" con el método "getCar" obtenga un "CarRegistro" sobre el cual podemos leer su acceso. Ejemplificando así que los métodos pueden trabajar correctamente con objetos más grandes que el original para el cual fue programado para recibir, sin perder la información como se ve con el método "addCar", posteriormente para volver a obtener esa información con un método que regresa solo el objeto base, basta con hacerle el casting para de nuevo tener el super objeto.

```
Bienvenido al Estacionamiento UPIITA
1-Agregar Automovil
2-Listar Automoviles
3-Verificar Acceso
5-Salir
1
Desea agregar información adicional (S/N)
N
```

```
Bienvenido al Estacionamiento UPIITA
1-Agregar Automovil
2-Listar Automoviles
3-Verificar Acceso
5-Salir
1
Desea agregar información adicional (S/N)
S
Ingrese el nombre del propietario:
Luis Fernando
Ingrese la marca del automovil:
Subaru
Ingrese el modelo
2014
```

Figuras 1 y 2. Ejecución del programa con ingreso de datos.

```
Bienvenido al Estacionamiento UPIITA
1-Agregar Automovil
2-Listar Automoviles
3-Verificar Acceso
5-Salir
3
Ingrese el numero de coches a verificar
0
El acceso es:
false
```

```
Bienvenido al Estacionamiento UPIITA
1-Agregar Automovil
2-Listar Automoviles
3-Verificar Acceso
5-Salir
3
Ingrese el numero de coches a verificar
1
El acceso es:
true
```

Figuras 3 y 4. Lectura de los datos y finalización del programa.

Resultados y conclusiones

"ParkingAcceso" por tener de Herencia a Parking, puede guardar los objetos de tipo "Carro" que le sean ingresados.

Cuando se crea un objeto de tipo "CarRegistro" y este es ingresado como un elemento de "Parking" por medio del método "addCar" lo ingresa sin problemas, ya que se trata finalmente de un "Carro".

Una vez que se eligió si el automóvil tiene información adicional o no, este es creado por el constructor de "CarRegistro", el cual indica que si no tiene información su acceso es falso y en caso de poseer información su acceso es verdadero, en cada caso con su constructor se ejecuta por medio de "super" equivaliendo a un constructor "Carro".

Al momento de leer el acceso se usa el método "getCar" que regresa un "Carro", por lo tanto como "ParkingAcceso" contiene elementos "CarRegistro", puede devolver el "Carro" sin problemas, pero usando el cast también puede devolver el "CarRegistro" completo.

En conclusión las clases y objetos con herencia permiten reutilizar el código ya escrito anteriormente, brindando la posibilidad de una mejor ejecución del programa, además de poder ampliar la funcionalidad que se quiere dar en una clase, todo esto sin tener que crear los objetos con código repetido de nuevo para cada uno de ellos y en cambio tratarlos como un nuevo objeto más grande que puede ser manipulado con las funciones de la clase base sin perder información.

Referencias

[1] H. M. Deitel y P. J. Deitel, Cómo programar en C/C++ y Java, Pearson Educación, 2004.

[2] E. Peñaloza Romero, Fundamentos de Programación, México: Alfaomega, 2004.