Leo una vez hago primer resumen genetral

Luego hago resumen del resumen

# Programacion Orientada a objeto

Es una técnica de progamacion que respresenta a objetos o acciones de la vida real

Otra: estructurada

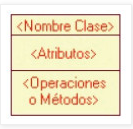
# Clase:

estructura que representa a un objeto de la vida real .

Nombre clase

Atributos: propiedades de la clase

Metodos: comportamiento de la clase



/\*\*Principal define el nombre de la Clase\*/

public class Principal {

 public String atributo="Esto es un atributo";

 /\*\*Esto es un método, donde se definen las operaciones\*/

 public void metodo(){

  /\*\*aqui van las sentencias que definen

   \* el comportamiento del método\*/

 }

}

# Objeto

Identidad de una clase concreta o abstracta , se llama instancia de la clase

Estado: propiedades-atributos de los objetos (características, valores concretos)

Comportamiento: métodos

Identidiad: propiedad única de un objeto que lo diferencia del resto.

Ej:molde es una clase



Se inicia el objeto dela clase con la palabra NEW , para hacer uso de los métodos y de los atributos.

/\*\*Creamos el objeto como instancia de la clase Principal\*/

    Principal miObjeto= new Principal();

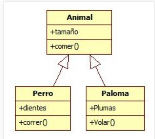
    miObjeto.atributo="Este es el nuevo valor del atributo para el objeto";

    miObjeto.metodo();

atributo-caracteristicas-estado

método-comportamiento-funcionalidad

# HERENCIA



Concepto por el cual una clase a partir de otra clase , heredando atributos y métodos

Se representa mediante la palabra extends

public class Animal{

 public String tamaño;

 public void comer(){

  /\*\*Comportamiento.....\*/

 }

}

 class Perro extends Animal{

 public int dientes;

 public void correr(){

  /\*\*Comportamiento.....\*/

 }

}

 class Paloma extends Animal{

 public int plumas;

 public void volar(){

  /\*\*Comportamiento.....\*/

 }

}

# Encapsulamiento

Encapsulamiento es la forma de proteger nuestros datos en un sistema( niveles de acceso, visibilidad de la información)

Evitar que desde otra clase se modifiquen los atributos y comportamientos . Se usan los métodos getter y setter

Nivel:

Protegido: solo se accede a los atributos y a los métodos de la misma jerarquía de clases.

Publico: accede a todos los atributos y a los métodos

Privado: solo se accede a los atributos y a los métodos dentro de la misma clase

public class Principal{

  public String atributo1;

  private String atributo2;

  protected String atributo3;

  /\*\*Métodos Set y Get para trabajar con las variables\*/

  public String getAtributo2() {

 return atributo2;

  }

  public void setAtributo2(String atributo2) {

 this.atributo2 = atributo2;

  }

}

# Clases abstractas:

Abstracción permite mostrar lo mas representativo de algo. Ej: persona : cabeza, tronco, piernas , brazos ( sin fijarse en el color de ojos , pelo etc).

Abstraccion permite crear clases con métodos generales que las clases concretas usan sin importar sus atributos y sus métodos propios.

Clase abstracta que posee método abstracto que no se implementa, el comportamiento de las acciones lo definen las clases concretas.

Usa la palabra Abstract

public abstract class Principal{

    /\*\*Método Abstracto sin implementación\*/

    public abstract void metodoAbstracto();

}

class subClase extends Principal{

 @Override

 public void metodoAbstracto() {

  /\*\*Implementación definida por la clase concreta\*/

 }

}

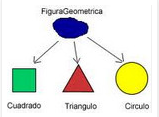
# Interface

Mecanismo que usa java para simular la herencia multiple.

Es una clase abstracta , todos sus métodos no se implementan. No usa la palabra abstract sino la palabra implement .Las clases concretas implementan todos los métodos que la compongan

|  |
| --- |
| interface InterfacePrincipal {     public void metodoAbstracto();     public String otroMetodoAbstracto();    }    public class Principal implements InterfacePrincipal{     public void metodoAbstracto() {    /\*\*Implementación definida por la clase concreta\*/   }     public String otroMetodoAbstracto() {    /\*\*Implementación definida por la clase concreta\*/    return "retorno";   }  } |

# Polimorfismo



Aplica

Objetos que comparten la misma superclase

Usa con herencia

Ejemplo: un objeto de la clase figuraGeometrica puede usarse para refererirse a un objeto de una subclase ( cuadrado, triangulo, circulo).

class FiguraGeometrica{

}

class Cuadrado extends FiguraGeometrica {

}

class Triangulo extends FiguraGeometrica {

}

class Circulo extends FiguraGeometrica{

}

public class Principal{

   public void metodo(){

 /\*\*Puedo crear objetos concretos\*/

 FiguraGeometrica miFiguraGeometrica = new FiguraGeometrica();

 Cuadrado miCuadro=new Cuadrado();

 /\*\*Puedo crear objetos polimorficos\*/

 miFiguraGeometrica=miCuadro;

 /\*\*Objeto Cuadrado de tipo FiguraGeometrica\*/

 FiguraGeometrica miCuadrado= new Cuadrado();

 /\*\*Objeto Circulo de tipo FiguraGeometrica\*/

 FiguraGeometrica miCirculo=new Circulo();

 /\*\*Objeto Triangulo de tipo FiguraGeometrica\*/

 FiguraGeometrica miTriangulo=new Triangulo();

 }

}