

**UNIVERSIDAD PRIVADA
FRANZ TAMAYO**

**CARRERA
INGENIERIA DE SISTEMAS**

Defensa De Hito 2

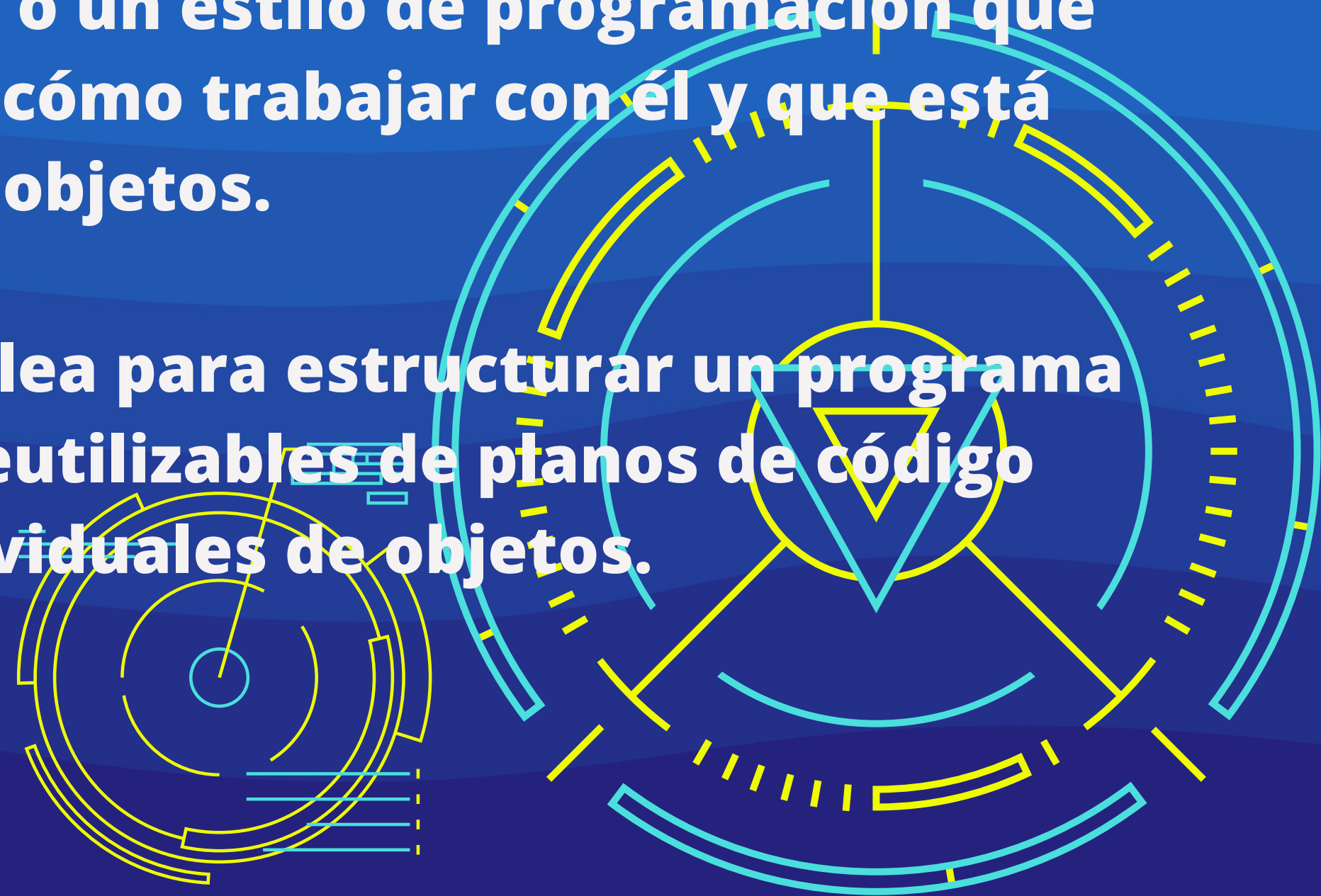
Estudiante:

Michael R. Alcon Villca



1.¿A que se refiere cuando se habla de POO?

- **La Programación Orientada a Objetos (POO) es un paradigma de programación, esto es, un modelo o un estilo de programación que proporciona unas guías acerca de cómo trabajar con él y que está basado en el concepto de clases y objetos.**
- **Este tipo de programación se emplea para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.**



¿Cuáles son los 4 componentes que componen

POO?



Clases:

Las clases pueden ser definidas como un molde que contendrá todas las características y acciones con las cuales podemos construir N cantidad de objetos.

Propiedades:

Las propiedades son las características de una clase, tomando como ejemplo la clase humanos, las propiedades podrían ser: nombre, el género, la altura, color de cabello, color de piel, etc.

Métodos:

Los métodos son las acciones que una clase puede realizar, siguiendo el mismo ejemplo anterior, estas podrían ser: caminar, comer, dormir, soñar, respirar, nadar, etc.

Objetos:

Son aquellos que tienen propiedades y comportamientos, estos pueden ser físicos o conceptuales. Técnicamente, los objetos son instancias de una clase, vendría siendo cuando ya le colocamos un "nombre" a nuestras clases (molde). Por ejemplo: El objeto "Arnell", quien es una instancia de la clase humanos.

3. ¿CUÁLES SON LOS PILARES DE POO?

4 principios de la Programación Orientada a Objetos

- La encapsulación

La encapsulación presenta toda la información importante de un objeto dentro del mismo y solo expone la información elegida al mundo exterior. Esta propiedad permite asegurar que la información de un objeto esté oculta para el mundo exterior, agrupando en una clase las características o atributos que tienen un acceso privado, y los comportamientos o métodos que cuenta con un acceso público.

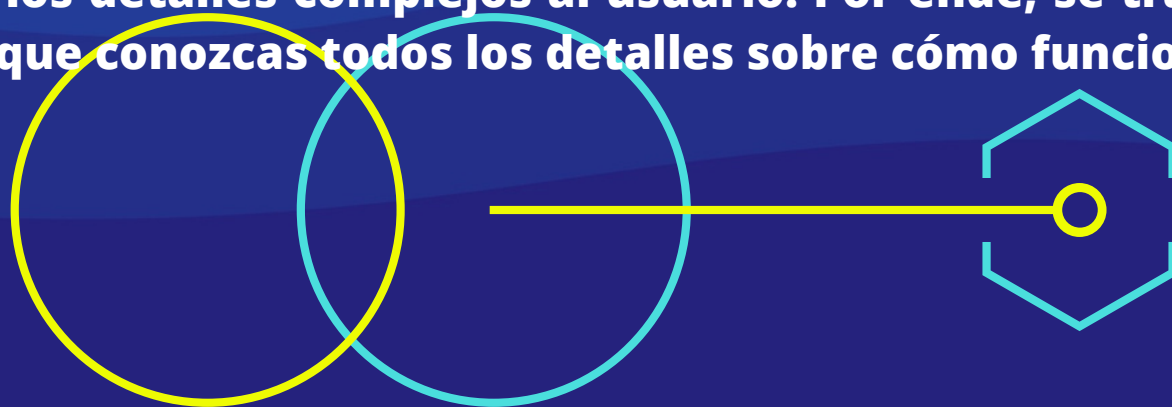
La encapsulación de cada objeto es responsable tanto de su información y de su estado. El único modo en la que esta se puede modificar es a través los propios métodos del objeto. De esta manera, los atributos internos de un objeto son ser inaccesibles desde fuera, pudiéndose modificar sólo llamando a las funciones correspondientes. Así se consigue mantener el estado a salvo de usos indebidos o que puedan resultar inesperados. Para explicar la encapsulación usaremos un coche de ejemplo. El coche comparte información pública mediante las luces de freno o intermitentes para indicar los giros (interfaz pública). Por contra, la interfaz interna, el mecanismo propulsor del coche, está oculto bajo el capó. Al conducir un automóvil es necesario indicar a otros conductores los movimientos, pero no exponer datos privados sobre el tipo de combustible o la temperatura del motor, ya que son muchos datos, lo que confundiría a los demás conductores.

- La abstracción

Otro de los principios de la Programación Orientada a Objetos es la abstracción, que se produce cuando el usuario interactúa solo con los atributos y métodos seleccionados de un objeto, usando herramientas simplificadas de alto nivel para acceder a un objeto complejo.

En la POO, los programas suelen ser muy grandes y los objetos se comunican bastante entre sí. De este modo, la abstracción facilita el mantenimiento de un código de gran tamaño, donde pueden surgir distintos cambios con el paso del tiempo.

Así, la abstracción está basada en utilizar cosas simples para representar la complejidad. Los objetos y las clases representan código subyacente, ocultando los detalles complejos al usuario. Por ende, se trata de una extensión de la encapsulación. Continuando con el ejemplo anterior, no es necesario que conozcas todos los detalles sobre cómo funciona el motor de un coche para poder conducirlo.





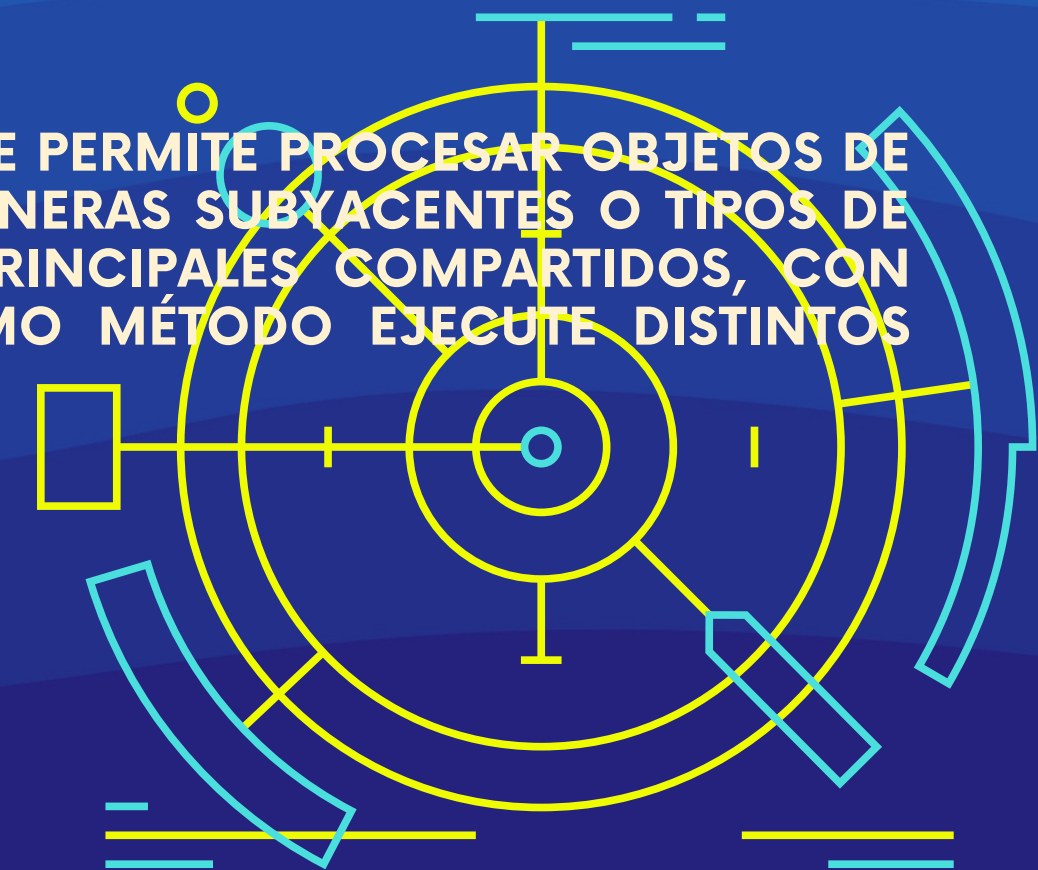
• LA HERENCIA

LA HERENCIA DEFINE RELACIONES JERÁRQUICAS ENTRE CLASES, DE MODO QUE ATRIBUTOS Y MÉTODOS COMUNES PUEDAN SER REUTILIZADOS. LAS CLASES PRINCIPALES EXTIENDEN ATRIBUTOS Y COMPORTAMIENTOS A LAS CLASES SECUNDARIAS. MEDIANTE LA DEFINICIÓN EN UNA CLASE DE LOS ATRIBUTOS Y COMPORTAMIENTOS BÁSICOS, PUEDEN CREARSE CLASES SECUNDARIAS, AMPLIANDO LA FUNCIONALIDAD DE LA CLASE PRINCIPAL Y AÑADIENDO ATRIBUTOS Y COMPORTAMIENTOS EXTRA. ES UNA DE LAS CLAVES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

SIGUIENDO EL EJEMPLO DE LOS ANIMALES, PUEDE EMPLEARSE UNA ÚNICA CLASE DE ANIMAL Y AÑADIR UN ATRIBUTO DE TIPO DE ANIMAL QUE ESPECIFIQUE EL TIPO DE ANIMAL. LOS DISTINTOS TIPOS DE ANIMALES REQUERIRÁN DIFERENTES MÉTODOS, POR EJEMPLO, LOS REPTILES DEBEN PODER PONER HUEVOS Y LOS PECES NADAR. INCLUSO SI LOS ANIMALES DISPONEN DE UN MÉTODO EN COMÚN, COMO MOVERSE, LA IMPLEMENTACIÓN REQUERIRÍA MUCHAS DECLARACIONES “SI” PARA GARANTIZAR EL COMPORTAMIENTO DE MOVIMIENTO IDÓNEO. POR EJEMPLO, LAS RANAS SALTAN, MIENTRAS QUE LAS SERPIENTES SE DESLIZAN. EL PRINCIPIO DE HERENCIA PERMITE SOLUCIONAR DICHO PROBLEMA.

• EL POLIMORFISMO

EL POLIMORFISMO RESIDE EN DISEÑAR OBJETOS PARA COMPARTIR COMPORTAMIENTOS, LO QUE PERMITE PROCESAR OBJETOS DE DISTINTOS MODOS. ES LA CAPACIDAD DE PRESENTAR LA MISMA INTERFAZ PARA DISTINTAS MANERAS SUBYACENTES O TIPOS DE DATOS. AL USAR LA HERENCIA, LOS OBJETOS PUEDEN ANULAR LOS COMPORTAMIENTOS PRINCIPALES COMPARTIDOS, CON COMPORTAMIENTOS SECUNDARIOS ESPECÍFICOS. EL POLIMORFISMO PERMITE QUE EL MISMO MÉTODO EJECUTE DISTINTOS COMPORTAMIENTOS DE DOS MODOS: ANULACIÓN DE MÉTODO Y SOBRECARGA DE MÉTODO.



4. ¿QUÉ ES ENCAPSULAMIENTO Y MUESTRE UN EJEMPLO?

- DECIMOS QUE EL ENCAPSULAMIENTO EN LA PROGRAMACIÓN ORIENTADA A OBJETOS ES CUANDO LIMITAMOS EL ACCESO O DAMOS UN ACCESO RESTRINGIDO DE UNA PROPIEDAD A LOS ELEMENTOS QUE NECESITA UN MIEMBRO Y NO A NINGUNO MÁS.


EL ELEMENTO MÁS COMÚN DE ENCAPSULAMIENTO SON LAS CLASES, DONDE ENCAPSULAMOS Y ENGLOBAMOS TANTO MÉTODOS COMO PROPIEDADES.

OTRO EJEMPLO MUY COMÚN DE ENCAPSULAMIENTO SON LOS GETTERS Y SETTERS DE LAS PROPIEDADES DENTRO DE UNA CLASE. POR DEFECTO NOS DAN EL VALOR "NORMAL" PERO PODEMOS MODIFICARLOS PARA QUE CAMBIE.


EJEMPLO.-

```
private decimal _velocidadActual { get; set; }  
public decimal VelocidadActual  
{  
    get{  
        return _velocidadActual + 2;  
    }  
    set{  
        _velocidadActual = value;  
    }  
}
```





```
private decimal _velocidadActual { get; set; }
public decimal VelocidadActual
{
    get{
        return _velocidadActual + 2;
    }
    set{
        _velocidadActual = value;
    }
}
```



EN EL EJEMPLO QUE ACABAMOS DE VER TENEMOS DOS PROPIEDADES, AMBAS HACEN REFERENCIA A LA VELOCIDAD ACTUAL, PERO HAY LIGERAS DIFERENCIAS.

UNA ES PRIVADA, POR LO QUE DESDE FUERA DE LA CLASE NO PODEMOS ACCEDER A SU VALOR.

LA SEGUNDA ES PÚBLICA Y ACCEDE A LA PRIVADA ANTERIORMENTE MENCIONADA. PORQUÉ HACEMOS ESTO?

EL EJEMPLO QUE HEMOS VISTO ES UN CASO REAL, LOS COCHES SUELEN MARCAR UN PAR DE KILÓMETROS POR HORA MAS DE LOS REALES. POR LO QUE ENCAPSULAMOS ESA LÓGICA DENTRO DEL SETTER, EL CUAL ESTA OCULTO PARA EL CONSUMIDOR QUE VERÍA EN SU CUENTAQUILÓMETROS LA VELOCIDAD CON LOS DOS KILÓMETROS POR HORA EXTRA.

PODEMOS DECIR QUE ENCAPSULAMIENTO ES UNA FORMA DE OCULTACIÓN DE INFORMACIÓN ENTRE ENTIDADES, MOSTRÁNDOSE ENTRE ELLAS SOLO LA INFORMACIÓN MÁS NECESARIA.



5. ¿Qué es Abstracción y muestre un ejemplo?

OTRO DE LOS PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS ES LA ABSTRACCIÓN, QUE SE PRODUCE CUANDO EL USUARIO INTERACTÚA SOLO CON LOS ATRIBUTOS Y MÉTODOS SELECCIONADOS DE UN OBJETO, USANDO HERRAMIENTAS SIMPLIFICADAS DE ALTO NIVEL PARA ACCEDER A UN OBJETO COMPLEJO.

EN LA POO, LOS PROGRAMAS SUELEN SER MUY GRANDES Y LOS OBJETOS SE COMUNICAN BASTANTE ENTRE SÍ. DE ESTE MODO, LA ABSTRACCIÓN FACILITA EL MANTENIMIENTO DE UN CÓDIGO DE GRAN TAMAÑO, DONDE PUEDEN SURGIR DISTINTOS CAMBIOS CON EL PASO DEL TIEMPO.

ASÍ, LA ABSTRACCIÓN ESTÁ BASADA EN UTILIZAR COSAS SIMPLES PARA REPRESENTAR LA COMPLEJIDAD. LOS OBJETOS Y LAS CLASES REPRESENTAN CÓDIGO SUBYACENTE, OCULTANDO LOS DETALLES COMPLEJOS AL USUARIO. POR ENDE, SE TRATA DE UNA EXTENSIÓN DE LA ENCAPSULACIÓN. CONTINUANDO CON EL EJEMPLO ANTERIOR, NO ES NECESARIO QUE CONOZCAS TODOS LOS DETALLES SOBRE CÓMO FUNCIONA EL MOTOR DE UN COCHE PARA PODER CONDUCIRLO.

EJEMPLO

LA GERENCIA DE UN TALLER MECÁNICO NECESITA UN SISTEMA PARA CONTROLAR LOS VEHÍCULOS QUE INGRESAN A SUS INSTALACIONES. EN ESTE CASO, LAS CARACTERÍSTICAS ESENCIALES DE LA CLASE VEHÍCULO SON: MARCA, MODELO, COLOR, FALLA DETECTADA, NOMBRE DEL PROPIETARIO, DIRECCIÓN DEL PROPIETARIO, TELÉFONO DEL PROPIETARIO...

A ESTO SE LE LLAMA ABSTRACCIÓN. EN GENERAL UN PROGRAMA NO ES MÁS QUE UNA DESCRIPCIÓN ABSTRACTA DE UN PROCEDIMIENTO O FENÓMENO QUE EXISTE O SUCEDE EN EL MUNDO REAL.

LA ABSTRACCIÓN ES CRUCIAL PARA COMPRENDER ESTE COMPLEJO MUNDO.

LA ABSTRACCIÓN ES ESENCIAL PARA EL FUNCIONAMIENTO DE UNA MENTE HUMANA NORMAL Y ES UNA HERRAMIENTA MUY POTENTE PARA TRATAR LA COMPLEJIDAD.

LA ABSTRACCIÓN ES CLAVE PARA DISEÑAR UN BUEN SOFTWARE.



Abstracción

Programación orientada a objetos (POO)

6. ¿QUE ES HERENCIA Y MUESTRE UN EJEMPLO?

LA HERENCIA DEFINE RELACIONES JERÁRQUICAS ENTRE CLASES, DE MODO QUE ATRIBUTOS Y MÉTODOS COMUNES PUEDAN SER REUTILIZADOS. LAS CLASES PRINCIPALES EXTIENDEN ATRIBUTOS Y COMPORTAMIENTOS A LAS CLASES SECUNDARIAS. MEDIANTE LA DEFINICIÓN EN UNA CLASE DE LOS ATRIBUTOS Y COMPORTAMIENTOS BÁSICOS, PUEDEN CREARSE CLASES SECUNDARIAS, AMPLIANDO LA FUNCIONALIDAD DE LA CLASE PRINCIPAL Y AÑADIENDO ATRIBUTOS Y COMPORTAMIENTOS EXTRA. ES UNA DE LAS CLAVES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

SIGUIENDO EL EJEMPLO DE LOS ANIMALES,

PUEDE EMPLEARSE UNA ÚNICA CLASE DE ANIMAL Y AÑADIR UN ATRIBUTO DE TIPO DE ANIMAL QUE ESPECIFIQUE EL TIPO DE ANIMAL. LOS DISTINTOS TIPOS DE ANIMALES REQUERIRÁN DIFERENTES MÉTODOS, POR EJEMPLO, LOS REPTILES DEBEN PODER PONER HUEVOS Y LOS PECES NADAR. INCLUSO SI LOS ANIMALES DISPONEN DE UN MÉTODO EN COMÚN, COMO MOVERSE, LA IMPLEMENTACIÓN REQUERIRÍA MUCHAS DECLARACIONES "SI" PARA GARANTIZAR EL COMPORTAMIENTO DE MOVIMIENTO IDÓNEO. POR EJEMPLO, LAS RANAS SALTAN, MIENTRAS QUE LAS SERPIENTES SE DESLIZAN. EL PRINCIPIO DE HERENCIA PERMITE SOLUCIONAR DICHO PROBLEMA.

7. ¿QUÉ ES POLIMORFISMO Y MUESTRE UN EJEMPLO?

EL POLIMORFISMO RESIDE EN DISEÑAR OBJETOS PARA COMPARTIR COMPORTAMIENTOS, LO QUE PERMITE PROCESAR OBJETOS DE DISTINTOS MODOS. ES LA CAPACIDAD DE PRESENTAR LA MISMA INTERFAZ PARA DISTINTAS MANERAS SUBYACENTES O TIPOS DE DATOS. AL USAR LA HERENCIA, LOS OBJETOS PUEDEN ANULAR LOS COMPORTAMIENTOS PRINCIPALES COMPARTIDOS, CON COMPORTAMIENTOS SECUNDARIOS ESPECÍFICOS. EL POLIMORFISMO PERMITE QUE EL MISMO MÉTODO EJECUTE DISTINTOS COMPORTAMIENTOS DE DOS MODOS: ANULACIÓN DE MÉTODO Y SOBRECARGA DE MÉTODO.

```

class Animal {
    public void makeSound() {
        System.out.println("Grr...");
    }
}

class Cat extends Animal {
    public void makeSound() {
        System.out.println("Meow");
    }
}

class Dog extends Animal {
    public void makeSound() {
        System.out.println("Woof");
    }
}

```

**COMO TODOS LOS
OBJETOS GATO Y
PERRO SON OBJETOS
ANIMALES, PODEMOS
HACER LO SIGUIENTE**

```

public static void main(String[] args) {
    Animal a = new Dog();
    Animal b = new Cat();
}

```

**CREAMOS DOS
VARIABLES DE
REFERENCIA DE TIPO
ANIMAL Y LAS
APUNTAMOS A LOS
OBJETOS GATO Y
PERRO. AHORA,
PODEMOS LLAMAR A
LOS MÉTODOS
MAKESOUND().**

```

a.makeSound();
//Outputs "Woof"

b.makeSound();
//Outputs "Meow"

```

UN EJEMPLO :

CLÁSICO DE POLIFORMISMO ES EL SIGUIENTE. PODEMOS CREAR DOS CLASES DISTINTAS: GATO Y PERRO, QUE HEREDAN DE LA SUPERCLASE ANIMAL. LA CLASE ANIMAL TIENE EL MÉTODO ABSTRACTO MAKESOUND() QUE SE IMPLEMENTA DE FORMA DISTINTA EN CADA UNA DE LAS SUBCLASES (GATOS Y PERROS SUENAN DE FORMA DISTINTA). ENTONCES, UN TERCER OBJETO PUEDE ENVIAR EL MENSAJE DE HACER SONIDO A UN GRUPO DE OBJETOS GATO Y PERRO POR MEDIO DE UNA VARIABLE DE REFERENCIA DE CLASE ANIMAL, HACIENDO ASÍ UN USO POLIMÓRFICO DE DICHOS OBJETOS RESPECTO DEL MENSAJE MOVER.

8.- QUE ES UN ARRAY

PARA DEFINIR QUÉ ES UN ARRAY EN JAVA DEBEMOS PENSAR EN ELLOS COMO UNA ESTRUCTURA DE DATOS QUE NOS PERMITE ALMACENAR UNA SERIE DE DATOS DE UN MISMO TIPO. ADEMÁS, EL TAMAÑO DE UN ARRAY EN JAVA ES FIJO Y SE DECLARA EN UN PRIMER MOMENTO, NO PUDIENDO CAMBIARSE MÁS ADELANTE COMO SÍ OCURRE EN OTROS LENGUAJES. ESTO QUE PARECE UNA LIMITACIÓN NO LO ES TANTO, NOS PERMITE TRABAJAR DE UN MODO MÁS ORDENADO Y CLARO.

SE ACCEDE A CADA ELEMENTO INDIVIDUAL DEL ARRAY MEDIANTE UN NÚMERO ENTERO DENOMINADO ÍNDICE. 0 ES EL ÍNDICE DEL PRIMER ELEMENTO Y N-1 ES EL ÍNDICE DEL ÚLTIMO ELEMENTO, SIENDO N, LA DIMENSIÓN DEL ARRAY. Y DICHO ESTO, SEGURO QUE AHORA ENTIENDES EL PORQUÉ DE LA BROMA EN DONDE LOS INFORMÁTICOS COMENZAMOS A CONTAR POR 0 Y NO EN 1 COMO EL RESTO DE LAS PERSONAS. ;)

LA DECLARACIÓN DE UN ARRAY EN JAVA Y SU INICIALIZACIÓN SE REALIZA DE LA SIGUIENTE MANERA:

```
tipo_dato nombre_array[];  
nombre_array = new tipo_dato[tamano];
```

POR EJEMPLO, PODRÍAMOS DECLARAR UN ARRAY DE CARACTERES E INICIALIZARLO DE LA SIGUIENTE MANERA:

```
char arrayCaracteres[];  
arrayCaracteres = new char[10];
```

9.-Qué son Paquetes en Java

- LOS PAQUETES SIRVEN PARA AGRUPAR CLASES RELACIONADAS Y DEFINEN UN ESPACIO DE NOMBRES (NAMESPACE) PARA LAS CLASES QUE CONTIENEN.
- EN GENERAL, CUANDO NOMBRA UNA CLASE, ESTÁ ASIGNANDO UN NOMBRE DEL NAMESPACE. UN NAMESPACE DEFINE UNA REGIÓN DECLARATIVA. EN JAVA, NO HAY DOS CLASES QUE PUEDAN USAR EL MISMO NOMBRE DEL MISMO NAMESPACE. POR LO TANTO, DENTRO DE UN NAMESPACE DADO, CADA NOMBRE DE CLASE DEBE SER ÚNICO.
- EN PROGRAMAS GRANDES, ENCONTRAR NOMBRES ÚNICOS PARA CADA CLASE PUEDE SER DIFÍCIL. ADEMÁS, DEBE EVITAR LAS COLISIONES DE NOMBRES CON CÓDIGO CREADO POR OTROS PROGRAMADORES QUE TRABAJAN EN EL MISMO PROYECTO Y CON LA BIBLIOTECA DE JAVA. LA SOLUCIÓN A ESTOS PROBLEMAS ES EL PAQUETE PORQUE LE DA UNA MANERA DE PARTICIONAR EL ESPACIO DE NOMBRES(NAMESPACE). CUANDO SE DEFINE UNA CLASE DENTRO DE UN PAQUETE, EL NOMBRE DE ESE PAQUETE SE ADJUNTA A CADA CLASE, EVITANDO ASÍ LAS COLISIONES DE NOMBRES CON OTRAS CLASES QUE TIENEN EL MISMO NOMBRE, PERO ESTÁN EN OTROS PAQUETES.
- DADO QUE UN PAQUETE GENERALMENTE CONTIENE CLASES RELACIONADAS, JAVA DEFINE DERECHOS DE ACCESO ESPECIALES PARA EL CÓDIGO DENTRO DE UN PAQUETE. EN UN PAQUETE, PUEDE DEFINIR CÓDIGO AL QUE PUEDA ACCEDER OTRO CÓDIGO DENTRO DEL MISMO PAQUETE PERO NO MEDIANTE CÓDIGO FUERA DEL PAQUETE. ESTO LE PERMITE CREAR GRUPOS INDEPENDIENTES DE CLASES RELACIONADAS QUE MANTIENEN SU OPERACIÓN PRIVADA.

10.¿Cómo se define una clase main en JAVA y muestra un ejemplo?

Lo que debes saber en primer lugar es que el método `main()` es el punto de entrada de la aplicación, es decir, es el punto en el que comienza la ejecución de esta. Es por ello que ha de ser `public` y `static`.

Ha de ser público y estático

- **public:**

Un método público es accesible desde fuera de la clase.

- **static:**

Un método estático es aquel que se puede ejecutar sin una instancia de la clase.

Al ser el punto de entrada, ha de ser accesible desde fuera de la clase en la que se encuentra. Además, al ser lo primero que se ejecuta, ha de ser posible su ejecución antes de instanciar un objeto.

Además, ha de tener un tipo de devolución `void`

Como consecuencia directa de ser la primera línea de código que se ejecuta, no tiene sentido que tenga un tipo de devolución distinto de `void`, ya que no hay un código anterior que pueda hacer algo con ese valor. El método `main()` en Java siempre tiene un tipo de devolución `void`.

A continuación, ha de llamarse `main()`, en minúscula. El método `main()` ha de llamarse `main`, en minúscula. A la hora de ejecutar el código, el entorno de ejecución de Java (JRE) busca el punto de entrada predeterminado. El JRE solo sabe que es un método llamado `main`. Si no lo encuentra, no puede acceder. Es por ello que puedes considerar `main` como una palabra clave de acceso. Si la cambias, tu aplicación no se ejecutará.

Por último, ha de aceptar un único parámetro: una matriz `String[]`

El método `main()` acepta un parámetro (y solo uno): una matriz de tipo `String`. Esta matriz recoge los valores que introduzcas a la hora de ejecutar tu aplicación desde la línea de comandos. Da igual el valor que introduzcas; el JRE lo transformará a `String`

```
package es.javautodidacta;

import java.util.*;

public class Robot {
    public static void main(String[] args) {
        System.out.println(Arrays.toString(args));
    }
}
```

PARTE PRACTICA

11. Generar la clase Provincia.

Provincia			
+ nombre: String			
Provincia() => Constructor			
gets() => todos los gets de la clase			
sets() => todos los sets de la clase			
muestraProvincia()			

Project

Tarea_hito3 C:\Users\MICHAEL\IdeaProjects

- .idea
- out
- src
 - defensahito2
 - Departamento
 - Pais
 - Provincia
 - PilaDeClientes
 - polimorfismo
 - Main.java
 - .gitignore
 - Tarea_hito3.iml

External Libraries

Scratches and Consoles

Main.java × Provincia.java × Departamento.java × Pais.java ×

```
1 package defensahito2;
2
3 public class Provincia {
4     4 usages
5     private String NombreProvincia;
6
7     public Provincia (String NombreProvincia) {
8         this.NombreProvincia = NombreProvincia;
9     }
10
11     public Provincia() {
12         this.NombreProvincia = "";
13     }
14
15     public void setNombreProvincia(String NuevoNombre) {
16         NombreProvincia = NuevoNombre;
17     }
18     1 usage
19
20     public String getNombreProvincia() {
21         return this.NombreProvincia;
22     }
23     1 usage
24
25     public void mostrarProvincia() {
26         System.out.println("Mostrando datos de la provincia");
27         System.out.println("Nombre Provincia: " + this.getNombreProvincia());
28         System.out.println("\n");
29     }
30 }
```

12. Generar la clase Departamento.

Diseno.

Departamento
+ nombre: String + nroDeProvincias[]: Provincia
 Departamento() => constructor gets() => todos los gets de la clase sets() => todos los sets de la clase muestraDepartamento() agregaNuevaProvincia()

- Crear una clase MAIN (Utilizar el MAIN del anterior ejercicio)
 - Crear todos los gets y sets de la clase.
 - El constructor no recibe parámetros.
 - Crear una instancia de la clase Departamento.
 - Omitir el método **agregaNuevaProvincia()**
 - Mostrar los datos de los departamentos.
- Adjuntar el código JAVA generado.

The screenshot shows an IDE with the following components:

- Project Structure (Left Sidebar):**
 - Project: Tarea_hito3 (C:\Users\MICHAEL\IdeaProjects\)
 - Folder: .idea
 - Folder: out
 - Folder: src
 - Folder: defensahito2
 - File: Departamento
 - File: Pais
 - File: Provincia
 - Folder: PilaDeClientes
 - Folder: polimorfismo
 - File: Main.java
 - File: .gitignore
 - File: Tarea_hito3.iml
 - External Libraries
 - Scratches and Consoles
- Main Editor (Right):**
 - File: Departamento.java
 - Line 1: `package defensahito2;`
 - Line 2: (blank)
 - Line 3: `public class Departamento {`
 - Line 4: `private String nombreDepartamento;`
 - Line 5: `private int noProvincias;`
 - Line 6: `private Provincia[] provincias;`
 - Line 7: (blank)
 - Line 8: `public Departamento(String nombreDepartamento, int noProvincias, Provincia[] provincias) {`
 - Line 9: `this.nombreDepartamento = nombreDepartamento;`
 - Line 10: `this.noProvincias = noProvincias;`
 - Line 11: `this.provincias = provincias;`
 - Line 12: `}`
 - Line 13: `public Departamento () {}`
 - Line 14: `public String getNombreDepartamento() {`
 - Line 15: `return this.nombreDepartamento;`
 - Line 16: `}`
 - Line 17: `public Provincia[] getProvincias() {`
 - Line 18: `return this.provincias;`
 - Line 19: `}`
 - Line 20: (blank)
 - Line 21: `public int getNoProvincias() {`

> PilaDeClientes
> polimorfismo
> Main.java
.gitignore

Tarea_hito3.iml

> External Libraries
Scratches and Consoles

```
21 public int getNoProvincias() {  
22     return noProvincias;  
23 }  
24  
25 public void setNoProvincias(int noProvincias) {  
26     this.noProvincias = noProvincias;  
27 }  
28  
29 public void setNombreDepartamento(String nombreDepartamento) {  
30     this.nombreDepartamento = nombreDepartamento;  
31 }  
32  
33 public void setProvincias(Provincia[] provincias) {  
34     this.provincias = provincias;  
35 }  
36  
37 1 usage  
38 public void mostrarDepartamento() {  
39     System.out.println("\nMOSTRANDO DATOS DEL DEPARTAMENTO");  
40     System.out.println("Nombre Departamento: " + this.getNombreDepartamento());  
41     System.out.println("No Provincias: " + this.noProvincias);  
42     for (int i=0; i < this.noProvincias; i++) {  
43         this.getProvincias()[i].mostrarProvincia();  
44     }  
45 }
```

13. Generar la clase País.

País

+ nombre: String
+ nroDepartamentos: Int
+ departamentos[]: Departamento

País() => Constructor

gets() => todos los gets de la clase

sets() => todos los sets de la clase

muestraPaís()

agregaNuevoDepartamento()

Crear una clase MAIN (Utilizar el MAIN del anterior ejercicio)

- Crear una instancia de la clase País
- El constructor no recibe parámetros.
- Crear una instancia de la clase Departamento.
- Omitir el método **agregaNuevoDepartamento()**
- Mostrar los datos del País.

Project ▾

⊕ ⊖ ⚙ —

▼ Tarea_hito3 C:\Users\MICHAEL\IdeaProjects\

- > .idea
- > out
- ▼ src
 - ▼ defensahito2
 - Departamento
 - Pais**
 - Provincia
 - > PilaDeClientes
 - > polimorfismo
 - > Main.java
 - .gitignore
 - Tarea_hito3.iml
- > External Libraries
- Scratches and Consoles

Main.java × Provincia.java × Departamento.java × Pais.java ×

1 package defensahito2;

2

3 public class Pais {

4 3 usages
private String nombrePais;

5 4 usages
private int noDepartamentos;

6 3 usages
private Departamento[] departamentos;

7

8 public Pais(String nombrePais, int noDepartamentos, Departamento[] departamentos) {

9 this.nombrePais = nombrePais;

10 this.noDepartamentos = noDepartamentos;

11 this.departamentos = departamentos;

12 }

13 1 usage
public String getNombrePais() {

14 return this.nombrePais;

15 }

16 1 usage
public Departamento[] getDepartamentos() {

17 return this.departamentos;

18 }

19 public int getNoDepartamentos() {

20 return noDepartamentos;

21 }

22

23 public void setNoDepartamentos(int noDepartamentos) {

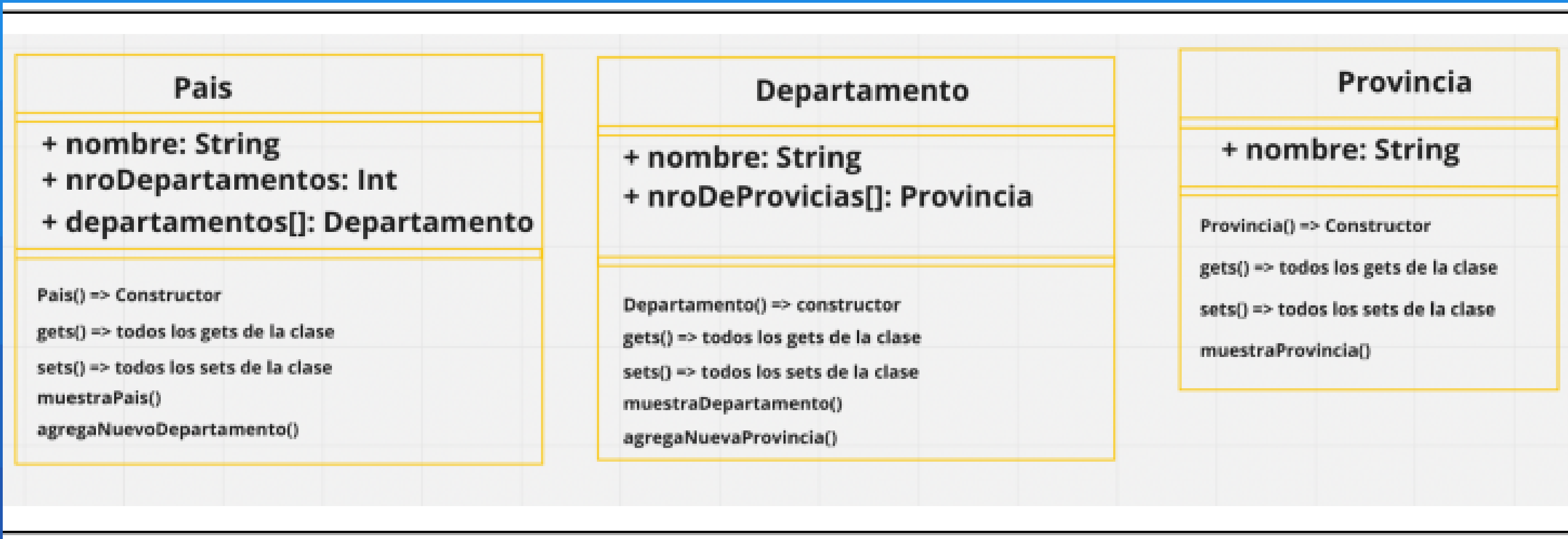
24 this.noDepartamentos = noDepartamentos;

25 }

Provincia
PilaDeClientes
polimorfismo
Main.java
gitignore
Tarea_hito3.iml
ernal Libraries
atches and Consoles

```
1 usage
16 public Departamento[] getDepartamentos() {
17     return this.departamentos;
18 }
19 public int getNoDepartamentos() {
20     return noDepartamentos;
21 }
22
23 public void setNoDepartamentos(int noDepartamentos) {
24     this.noDepartamentos = noDepartamentos;
25 }
26
27 public void setNombrePais(String nombrePais) {
28     this.nombrePais = nombrePais;
29 }
30
31 public void setDepartamentos(Departamento[] departamentos) {
32     this.departamentos = departamentos;
33 }
34 public void mostrarPais() {
35     System.out.println("\nMOSTRANDO DATOS DEL PAIS");
36     System.out.println("Nombre Pais: " + this.getNombrePais());
37     for (int i = 0; i < this.noDepartamentos; i++) {
38         this.getDepartamentos()[i].mostrarDepartamento();
39     }
40 }
41 }
```


14. Crear el diseño completo de las clases.



- Crear todos gets y sets de cada clase.
- Implementar los métodos **agregarNuevoDepartamento()**, **agregarNuevaProvincia()**, es decir todos los métodos.
- El método **agregarNuevoDepartamento** permite ingresar un nuevo departamento a un país.
- El método **agregarNuevaProvincia** permite ingresar una nueva provincia a un departamento.
- La clase Main debe mostrar lo siguiente:
 - Crear el PAÍS Bolivia
 - Al país Bolivia agregarle 3 departamentos.
 - Cada departamento deberá tener 2 provincias.
- Adjuntar el código JAVA generado.

Project

Tarea_hito3 C:\Users\MICHAEL\IdeaProjects\

.idea

out

src

calculadora

defensahito2

Departamento

Main

Main1

Pais

Provincia

PilaDeClientes

polimorfismo

.gitignore

Tarea_hito3.iml

External Libraries

Scratches and Consoles

Provincia.java

Departamento.java

Main1.java

Main.java

Pais.java

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

```
package defensahito2;

import java.util.Scanner;

public class Main {

    public static void main(String [] args) {

        Scanner leer = new Scanner(System.in);

        String nombreDep, nombreProv;
        int d, p, nDepartamentos = 2, nProvincias = 2;

        Departamento[] departamentos = new Departamento[100];

        for (d = 0; d < nDepartamentos; d = d + 1) {
            System.out.println("Ingrese el nombre del departamento " + (d + 1) + ": ");
            nombreDep = leer.next();
            Provincia[] provincias = new Provincia[100];
            for (p = 0; p < nProvincias; p = p + 1) {
                System.out.println("Ingrese el nombre de la provincia " + (p + 1) + ": ");
                nombreProv = leer.next();

                Provincia prov = new Provincia();
                prov.setNombreProvincia(nombreProv);
                provincias[p] = prov;

            }

            Departamento Dep = new Departamento();
            Dep.setNombreDepartamento(nombreDep);
            Dep.setNoProvincias(nProvincias);
            departamentos[d] = Dep;

        }

    }

}
```

1

Provincia

PilaDeClientes

polimorfismo

.gitignore

Tarea_hito3.iml

External Libraries

Scratches and Consoles

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

```
Provincia prov = new Provincia();
prov.setNombreProvincia(nombreProv);
provincias[p] = prov;

}

Departamento Dep = new Departamento();
Dep.setNombreDepartamento(nombreDep);
Dep.setNoProvincias(nProvincias);
departamentos[d] = Dep;

}

Pais Pa = new Pais( nombrePais: "Bolivia", nDepartamentos, departamentos);
Pa.setNoDepartamentos(nDepartamentos);

Pa.mostrarPais();

}

}
```

Run: Main1 x Main x

↶

↷

⚙

⏏

📷

C:\Users\MICHAEL\.jdk\openjdk-19.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2.4\lib\idea_rt.jar=59760:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2.4\bin\java.exe" -jar C:\Program Files\JetBrains\IntelliJ IDEA 2022.2.4\bin\java.exe

INGRESE DATOS DE PROVINCIAS

INGRESE DATOS DE DEPARTAMENTOS

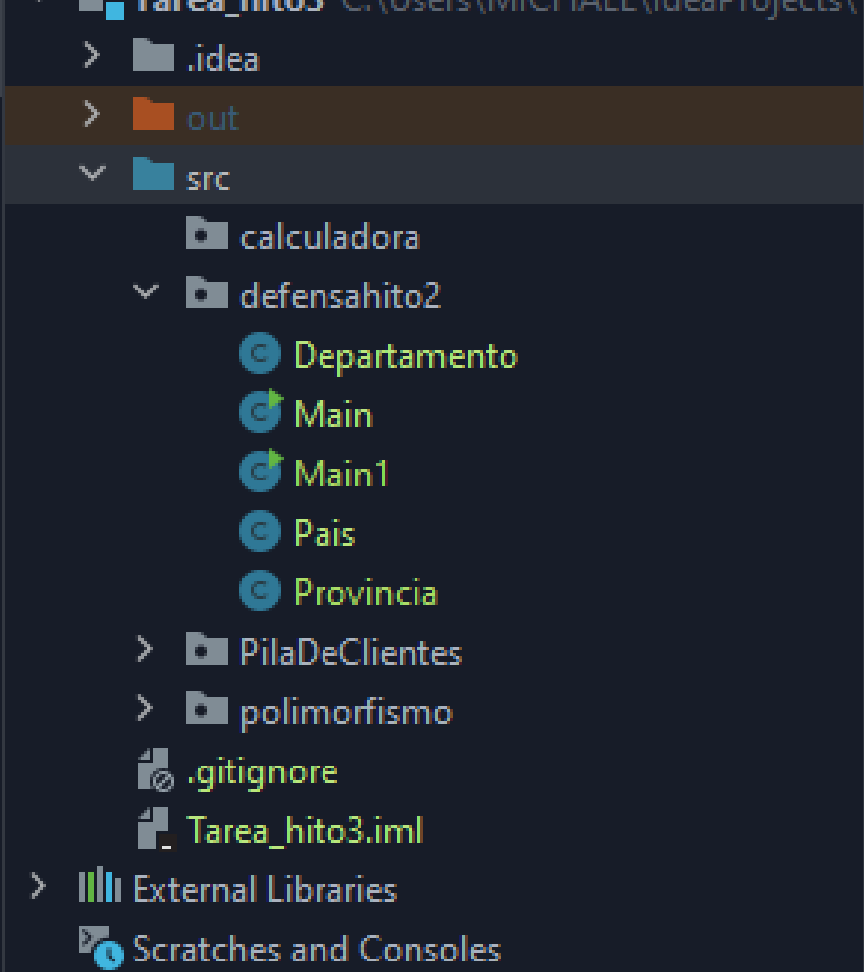
Ingrese el nombre del departamento 1:

micghgh

Ingrese el numero de provincias

Ingrese el nombre de la provincia 1:

```
1 package defensahito2;
2 import java.util.Scanner;
3
4 public class Main1 {
5     public static void main(String [] args) {
6         Scanner leer = new Scanner(System.in);
7
8         System.out.println("INGRESE DATOS DE PROVINCIAS");
9         String nombreProvincia;
10        int i, nProvincias;
11        nProvincias = 2;
12
13        System.out.println("INGRESE DATOS DE DEPARTAMENTOS");
14        String nombreDepartamento;
15        int j, nDepartamentos = 2;
16
17        Departamento[] departamentos = new Departamento[100];
18
19        for (j = 0; j < nDepartamentos; j = j + 1) {
20            System.out.println("Ingrese el nombre del departamento " + (j + 1) + ": ");
21            nombreDepartamento = leer.next();
22            System.out.println("Ingrese el numero de provincias");
23
24            Provincia[] provincias = new Provincia[100];
25
26            for (i = 0; i < nProvincias; i = i + 1) {
27                System.out.println("Ingrese el nombre de la provincia " + (i + 1) + ": ");
28                nombreProvincia = leer.next();
29
30                Provincia prov = new Provincia();
```

```
28         nombreProvincia = leer.next();
29
30         Provincia prov = new Provincia();
31         prov.setNombreProvincia(nombreProvincia);
32
33         provincias[i] = prov;
34     }
35     System.out.println("Ingrese el nombre de la nueva provincia: ");
36     nombreProvincia = leer.next();
37
38     Provincia prov = new Provincia();
39     prov.setNombreProvincia(nombreProvincia);
40
41     provincias[nProvincias] = prov;
42
43     Departamento dep = new Departamento();
44     dep.setNombreDepartamento(nombreDepartamento);
45     dep.setProvincias(provincias);
46     dep.setNoProvincias(nProvincias + 1);
47     departamentos[j] = dep;
48 }
49
50 System.out.println("Ingrese el nombre del nuevo departamento: ");
51 nombreDepartamento = leer.next();
52
53 Departamento dep = new Departamento();
54 dep.setNombreDepartamento(nombreDepartamento);
55
56 departamentos[nDepartamentos] = dep;
57 Pais pais = new Pais( nombrePais: "BOLIVIA", noDepartamentos: nDepartamentos + 1, departamentos);
58 pais.mostrarPais();
59
60 }
61
62 }
```



```
60     }  
61  
62     }  
63
```

Run: Main1 x Main x

```
C:\Users\MICHAEL\.jdk\openjdk-19.0.1\bin\java  
Files\JetBrains\IntelliJ IDEA 2022.2.4\bin" -D  
C:\Users\MICHAEL\IdeaProjects\Tarea hito3\out\  
INGRESE DATOS DE PROVINCIAS  
INGRESE DATOS DE DEPARTAMENTOS  
Ingrese el nombre del departamento 1:  
micghgh  
Ingrese el numero de provincias  
Ingrese el nombre de la provincia 1:  
hgghfhf  
Ingrese el nombre de la provincia 2:  
jhghghjh  
Ingrese el nombre de la nueva provincia:  
jjhbjjhb  
Ingrese el nombre del departamento 2:  
kjjbbj  
Ingrese el numero de provincias  
Ingrese el nombre de la provincia 1:  
jkjkh  
Ingrese el nombre de la provincia 2:
```

```
60     }  
61  
62     }  
63
```

Run: Main1 x Main x

  `C:\Users\MICHAEL\.jdk\openjdk-19.0.1\bin\java.exe` "-javaagent:C:\Program
Files\JetBrains\IntelliJ IDEA 2022.2.4\bin" -Dfile.encoding=UTF-8 -Dsun.
C:\Users\MICHAEL\IdeaProjects\Tarea hito3\out\production\Tarea hito3 def
Ingrese el nombre del departamento 1:

Ingrese el nombre de la provincia 1:
bkbjkjkg
Ingrese el nombre de la provincia 2:
hbjhbb
Ingrese el nombre del departamento 2:
jkbjbb
Ingrese el nombre de la provincia 1:

FIN

GRACIAS