UNIVERSIDAD DEL BÍO-BÍO

Paradigmas de la Programación

Herencia

UNIVERSIDAD DEL BÍO-BÍO

Herencia en la vida real - Biología

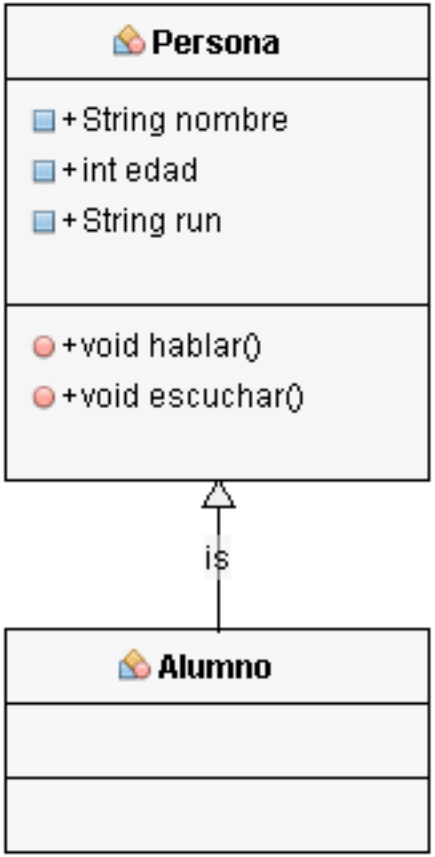
La herencia biológica es el proceso por el cual la descendencia de una célula u organismo adquiere o está predispuesta a adquirir las características (físicas, fisiológicas, morfológicas, bioquímicas o conductuales) de sus progenitores. Esas características pueden transmitirse a la generación siguiente.

En la programación orientada a objetos, se entiende como una propiedad que permite que las clases sean creadas a partir de otras ya existentes, de esta forma heredan las características de su padre.

Estas características pueden ser:

1.- Métodos

2.- Atributos



En la siguiente imagen se muestra como Alumno hereda los métodos y atributos especificados por la clase Padre(Persona/Súper Clase).

Observación 1: Alumno puede tener sus propios métodos y atributos.

Observación 2: Como hereda de Persona podrá acceder a las características y atributos **visibles** de su clase padre(public/protected).

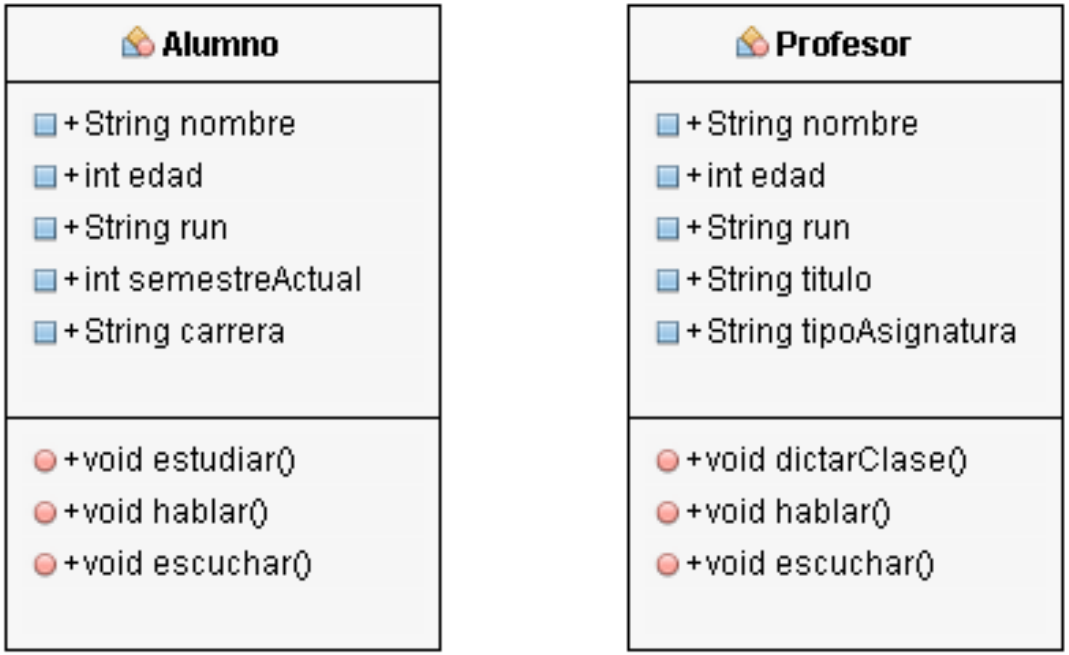
UNIVERSIDAD DEL BÍO-BÍO

Ventajas de la herencia

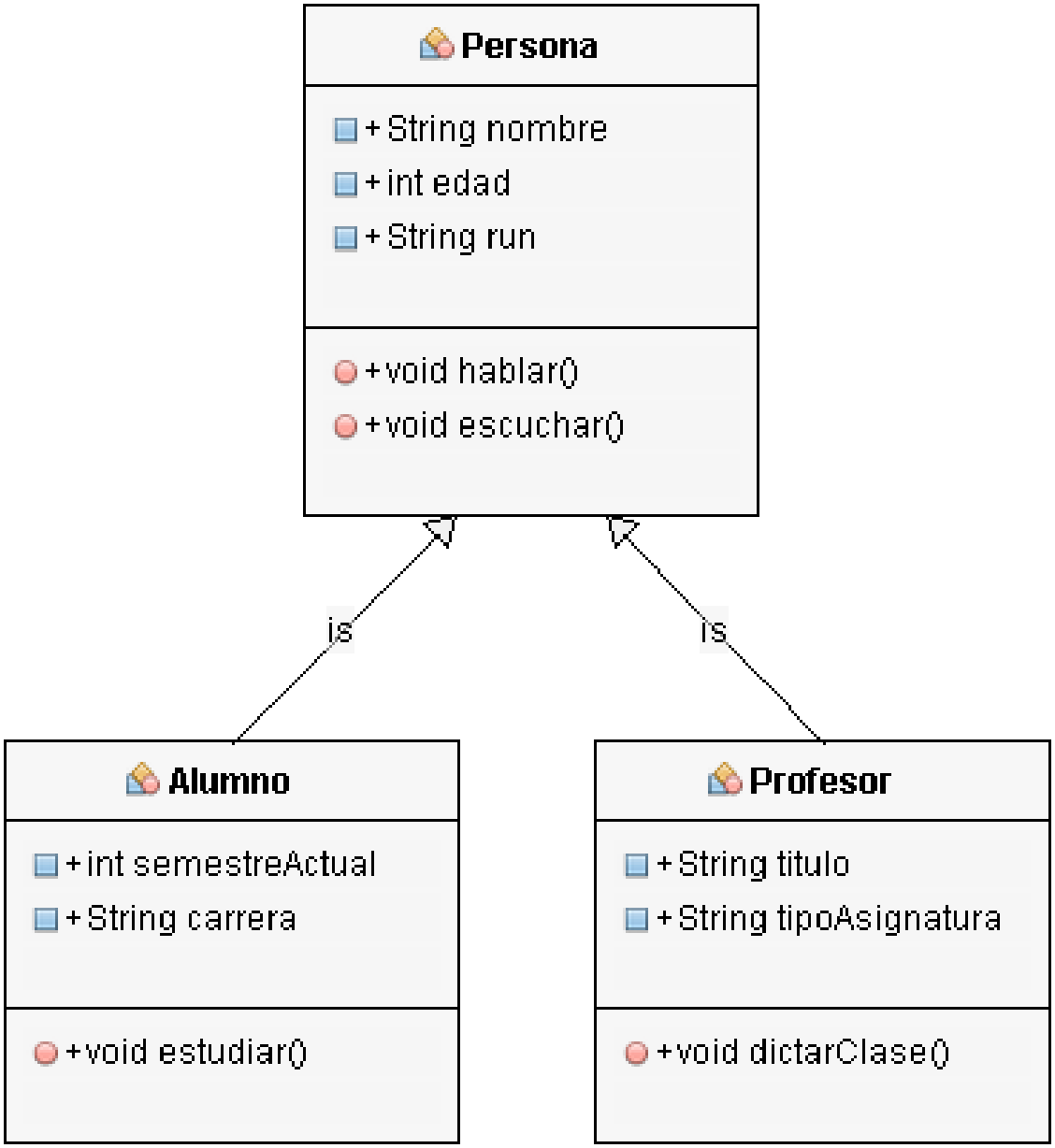
1. Evita el duplicado de código y favorece la reutilización de el mismo (las subclases utilizan el código de la clase padre/superclase).
2. Facilita el mantenimiento de aplicaciones a través de la extensión o modificación en los comportamiento anteriormente existente en las clases padres.

UNIVERSIDAD DEL BÍO-BÍO

¿Cuál sería una herencia posible?



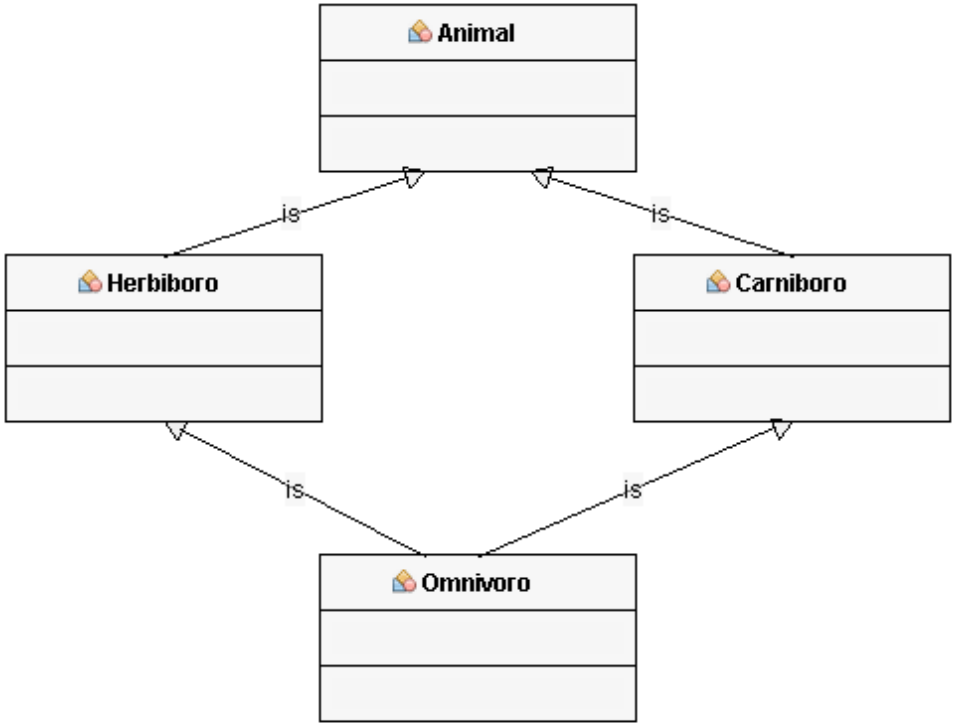
Para una correcta herencia, es importante generalizar el concepto base (Alumno/Profesor) e identificar lo común entre ellos, ya sea características(Atributos) o acciones(Métodos).


Persona
+ String nombre
+ int edad
+ String run
+ void hablar()
+ void escuchar()
is
is
Alumno
+ int semestreActual
+ String carrera
+ void estudiar()
Profesor
+ String titulo
+ String tipoAsignatura
+ void dictarClase()

Herencia Múltiple

Herencia múltiple hace referencia a la característica de los lenguajes de programación orientada a objetos en la que una clase puede heredar comportamientos y características de más de una superclase. Esto contrasta con la herencia simple, donde una clase sólo puede heredar de una superclase.

En lenguajes de programación como C++, es posible aplicar dicha característica, pero en el caso de Java no es posible su aplicación directamente sino por medio de Interfaces.



UNIVERSIDAD DEL BÍO-BÍO

Herencia en Java

Para efectuar la herencia entre clases, en la declaración de la clase hijo se ocupa la palabra clave **extends** seguido del nombre de la clase que será su padre.

```
public class Alumno extends Persona {

    public int semestreActual;
    public String carrera;

    public void estudiar() {
    }

}
```

UNIVERSIDAD DEL BÍO-BÍO

Constructores de la clase Padre/Súper clase

1. En Java los constructores no se heredan.
2. La primera sentencia del constructor de la clase hija **siempre** es una llamada al constructor de la clase padre.
3. La llamada al constructor del padre puede ser:

   Implícita: Se llama al constructor por defecto, es equivalente a poner como primera sentencia **super();**

   Si no existe el constructor por defecto en la clase padre dará un error en tiempo de compilación.

   Explícita(Se ingresa los datos solicitados por el constructor Padre):

   super(); super(a, b); super(a, b, c); ……

Para tener un correcto funcionamiento, el constructor de la clase Alumno primero deberá llamar al constructor de la clase Padre, cargando los datos solicitados. En caso contrario, dará error en tiempo de compilación.

```java
public class Persona {

    public String nombre;
    public int edad;
    public String run;

    public Persona(String nombre, int edad, String run)
    {
        this.nombre = nombre;
        this.edad = edad;
        this.run = run;
    }
}
```

```java
public class Alumno extends Persona {

    public int semestreActual;
    public String carrera;

    public Alumno() {
        super("Juan", 20, "00.000.000-0");
    }

    public void estudiar() {
    }

}
```

Accediendo a métodos y atributos de la clase Padre

```java
public class Persona {

    protected String nombre;
    protected int edad;
    protected String run;

    public void hablar()
    {
        System.out.println("Hola");
    }

}
```

```java
public class Alumno extends Persona {

    public int semestreActual;
    public String carrera;

    public void presentacion() {

        super.hablar();

        System.out.println("Mi nombre es:" + super.nombre + "\n"
                + "Y tengo " + super.edad + " años.");
    }

}
```

Para poder acceder a los métodos y atributos de la clase Padre se deberá hacer uso de la palabra clave **super.**

Herencia Implícita en Java

En Java, cualquier objeto que se instancie siempre Heredará por defecto de la clase Object.
De esta forma se explica la razón del porqué al momento de instanciar un objeto aparecen una serie de métodos asociados a nuestra clase que jamás se han declarados.

```
public class Proceso {

}

public static void main(String[] args) {

    Proceso proceso = new Proceso();

    proceso.
}
```

```
equals(Object o) boolean
getClass()          Class<?>
hashCode()              int
notify()                void
notifyAll()             void
toString()            String
wait()                  void
wait(long l)            void
wait(long l, int i)     void
```

UNIVERSIDAD DEL BÍO-BÍO

Sobre-escritura

En cualquier lenguaje de programación orientado a objetos, la sobre-escritura es una característica que permite a una subclase o clase secundaria proporcionar una implementación específica de un método que ya proporciona su clase Padre. Cuando un método en una subclase tiene el mismo nombre, los mismos parámetros o firma y el mismo tipo de retorno (o subtipo), se dice que el método en la subclase sobre-escribe el método en la superclase, invalidándolo.

```java
public class Persona {

    protected String nombre;
    protected int edad;
    protected String run;

    public void hablar() {
        System.out.println("Hola");
    }

}

public class Alumno extends Persona {

    public int semestreActual;
    public String carrera;

    @Override
    public void hablar() {
        System.out.println("Hola, mi nombre es:" + super.nombre);
    }

}
```

Observación 1: La clase Alumno invalida el método hablar del Padre y lo rediseña.
Observación 2: Cada método que se encuentra sobre-escrito, se agrega la palabra **@Overrride** sobre el.
Observación 3: Aunque el método del Padre se encuentre invalidado, se podrá llamar **solamente desde la clase hijo** ocupando la palabra clave super.
Ejemplo:
super.hablar();
Llamará al método del Padre ignorando la sobre-escritura de la clase Hijo.

UNIVERSIDAD DEL BÍO-BÍO

Reglas en la sobre-escritura

1. La lista de argumentos debe ser exactamente la misma que la del método sobre-escrito.
2. El tipo de retorno debe ser el mismo o un subtipo del tipo de retorno declarado en el método original sobre-escrito en la superclase.
3. El nivel de acceso no puede ser más restrictivo que el nivel de acceso del método sobre-escrito.

   Ejemplo: si el método de la superclase se declara público, el método de sobre-escritura de la subclase no puede ser privado ni protegido.
4. Un método declarado final no puede ser sobre-escrito.

UNIVERSIDAD DEL BÍO-BÍO

Reglas en la sobre-escritura

1. Los métodos de instancia solo se pueden sobre-escribir si la subclase los hereda.
2. Un método declarado estático no se puede sobre-escribir, pero se puede volver a declarar.
3. Si un método no puede ser heredado, entonces no puede ser sobre-escrito.
4. Una subclase dentro del mismo paquete que la superclase puede sobre-escribir cualquier método de superclase que no se declare privado o final.
5. Una subclase en un paquete diferente solo puede sobre-escribir los métodos no finales declarados públicos o protegidos.
6. Los constructores no pueden ser anulados.

UNIVERSIDAD DEL BÍO-BÍO

Sobre-escritura del método toString de la Clase Object

El método toString devuelve una representación String del objeto. En general, toString devuelve una cadena que "representa textualmente" este objeto. El resultado debe ser una representación concisa pero informativa que sea fácil de leer para una persona. **Se recomienda que todas las subclases sobre-escriban este método**. toString para la clase Object devuelve una cadena que consiste en el nombre de la clase más el carácter de signo "@" y la representación hexadecimal sin signo del código hash de la instancia. En otras palabras, este método devuelve una cadena igual al valor de:

getClass().getName() + '@' + Integer.toHexString(hashCode())

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

    Persona persona = new Persona("Fran", 25);
    System.out.println(persona);

}
```

```
run:
estudio.Persona@1db9742
BUILD SUCCESSFUL (total time: 0 seconds)
```

```java
public class Persona {

    public String nombre;
    public int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    @Override
    public String toString() {
        return "Nombre: " + nombre + " Edad: " + edad;
    }

}

public static void main(String[] args) {

    Persona persona = new Persona("Fran", 25);
    System.out.println(persona);

}
```

```
run:
Nombre: Fran Edad: 25
BUILD SUCCESSFUL (total time: 0 seconds)
```

UNIVERSIDAD DEL BÍO-BÍO

Sobre-escritura del método equals de la Clase Object

El método equals para la clase Object implementa la relación de equivalencia más discriminatoria posible en los objetos; es decir, para cualquier valor de referencia x y y que no sea nulo, este método devuelve verdadero si y solo si x e y se refieren al mismo objeto (x == y tiene el valor verdadero).

En general es necesario sobre-escritura dicho comportamiento para que las comparaciones se hagan en base a los valores de los atributos y no a las referencias de instancias.

Observación: Las clases de colección de objetos utilizan regularmente en sus métodos el equals de las instancias que deben gestionar.

Ejemplo: contains de la Clase ArrayList.
Al realizar la sobre-escritura de equals, ya dejará de comparar referencias entre instancias sino por el nuevo comportamiento que le hemos asignado.

```java
public class Persona {

    public String nombre;
    public int edad;

    @Override
    public boolean equals(Object obj) {
        Persona p = (Persona) obj;

        return this.nombre.equals(p.nombre) && this.edad == p.edad;
    }

}
```

UNIVERSIDAD DEL BÍO-BÍO

Utilizando instanceof

instanceof es un operador binario utilizado para probar si un objeto es de un tipo dado. **El resultado de la operación es verdadero o falso** . También se conoce como operador de comparación de tipos porque compara la instancia con el tipo.

Observación: Antes de lanzar un objeto desconocido, siempre se debe usar la instancia de verificación. Hacer esto ayuda a evitar **ClassCastException(Convertir un objeto a algo que no es)** en tiempo de ejecución.

La sintaxis básica del operador instanceof es:

(objeto) instanceof (tipo de dato)

instanceof – Ejemplo de uso

```
public class Persona {

    public String nombre;
    public int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    @Override
    public boolean equals(Object obj) {

        if (obj instanceof Persona)
        {
            Persona aux = (Persona) obj;
            return this.nombre.equals(aux.nombre) && this.edad == aux.edad;
        } else
            return false;

    }

}
```

Observación:
instanceof siempre retornará falso si el objeto que se compara es **NULO(Sin importa que sean del mismo tipo)**.

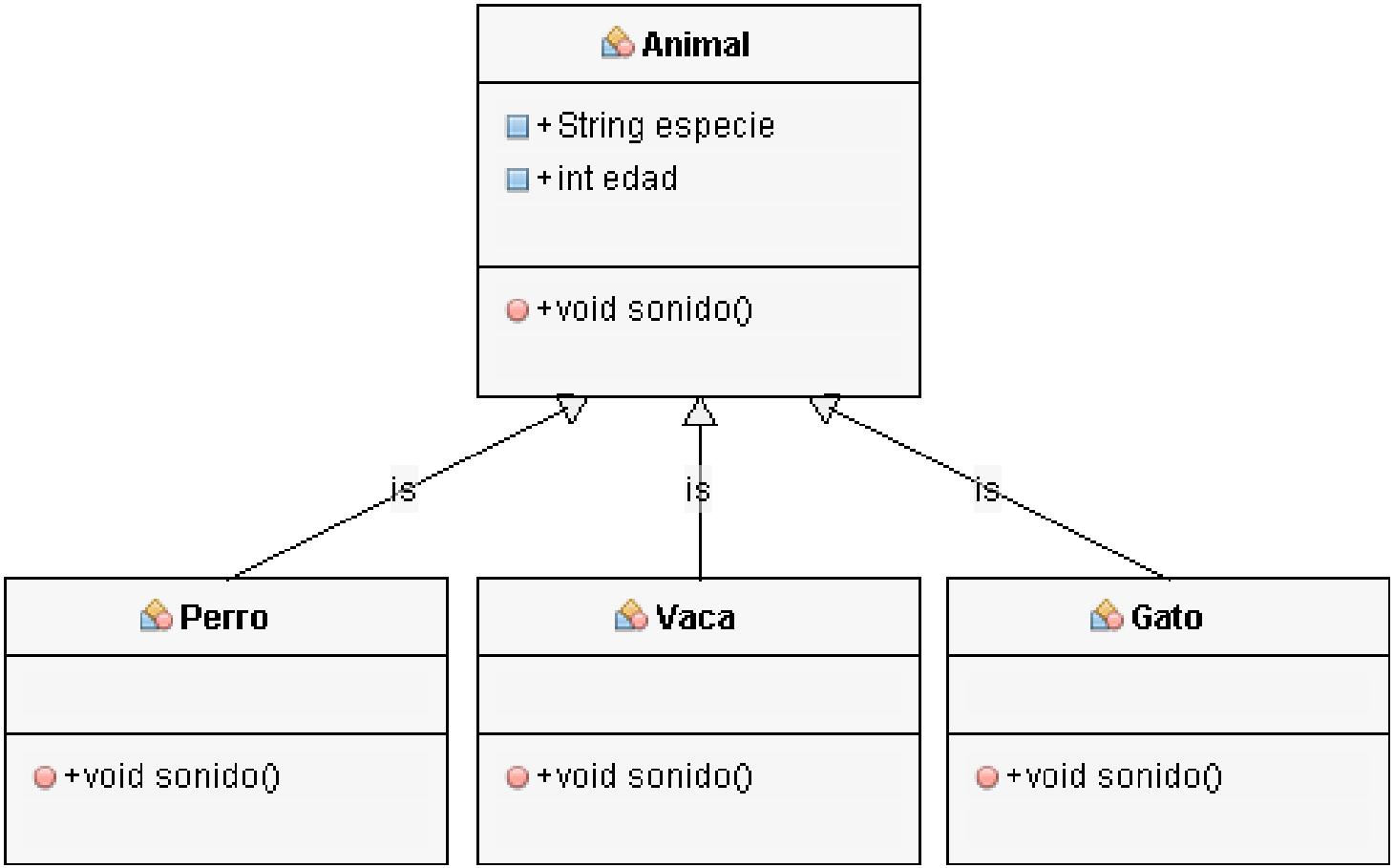
UNIVERSIDAD DEL BÍO-BÍO

Polimorfismo

La definición de polimorfismo se refiere a un principio en biología en el que un organismo o especie puede tener muchas formas o etapas diferentes. Este principio también puede aplicarse a la programación orientada a objetos y lenguajes como Java. Las subclases de una clase pueden definir sus propios comportamientos únicos y, sin embargo, compartir algunas de las mismas funciones de la clase principal.

UNIVERSIDAD DEL BÍO-BÍO

Teniendo el siguiente ejemplo



```java
public class Animal {

    public String especie;
    public int edad;

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

}
```

```java
public class Gato extends Animal {

    @Override
    public void sonido() {
        System.out.println("Miau");
    }

}
```

```java
public class Vaca extends Animal {

    @Override
    public void sonido() {
        System.out.println("Muuu");
    }

}
```

```java
public class Perro extends Animal {

    @Override
    public void sonido() {
        System.out.println("Guau");
    }

}
```

UNIVERSIDAD DEL BÍO-BÍO

Polimorfismo

Teniendo la siguiente información:

1. Una vaca es un Animal
2. Un gato es un Animal
3. Un perro es un Animal

Significa que es válido decir:

1. Una vaca es tanto un animal como una Vaca
2. Una gato es tanto un animal como una Gato
3. Una perro es tanto un animal como una Perro

Por lo tanto en código es posible realizar la siguiente declaraciones:

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

    Animal animal = new Animal();
    Animal perro = new Perro();
    Animal gato = new Gato();
    Animal vaca = new Vaca();

}
```

Observar la declaración de tipo para Perro, Gato y Vaca.

UNIVERSIDAD DEL BÍO-BÍO

Polimorfismo

Observe las respuestas obtenidas al ejecutar el método sonido();

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

        Animal animal = new Animal();
        Animal perro = new Perro();
        Animal gato = new Gato();
        Animal vaca = new Vaca();

        animal.sonido();
        perro.sonido();
        gato.sonido();
        vaca.sonido();

    }

}
```

```
run:
....
Guau
Miau
Muuu
BUILD SUCCESSFUL (total time: 0 seconds)
```

UNIVERSIDAD DEL BÍO-BÍO

Polimorfismo

Como Perro, Gato y Vaca son tanto Animal como su SubClase, es posible convertirlos a su tipo específico en tiempo de ejecución.

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

    Animal animal = new Animal();
    Animal perro = new Perro();
    Animal gato = new Gato();
    Animal vaca = new Vaca();

    Perro perro_2 = (Perro) perro;
    Gato gato_2 = (Gato) gato;
    Vaca vaca_2 = (Vaca) vaca;

    animal.sonido();
    perro_2.sonido();
    gato_2.sonido();
    vaca_2.sonido();

}
```

```
run:
....
Guau
Miau
Muuu
BUILD SUCCESSFUL (total time: 0 seconds)
```