

INTRODUCCIÓN A PROGRAMACIÓN ORIENTADA A OBJETOS



Enumera 4 atributos y 4 acciones que comparten todos los animales en la siguiente slide.

**Por ejemplo:
atributo: peso
acción: dormir**



Nombre: Sally
Raza: Rinoceronte blanco
Peso (kg): 2300
Altura (metros): 1.8
Sexo: hembra
Hogar: África
Alimento: {fruta, arbustos, zacate}
Hambre: Mucha








Nombre: Yogi
Raza: Oso grizzly
Peso (kg): 270
Altura (metros): 2
Sexo: macho
Alimento: {pezcardo, insectos, fruta}
Hogar: Jellystone Park
Hambre: Moderada



Nombre: Rocket
Raza: Mapache
Peso (kg): 7
Altura (metros): 0.3
Sexo: hembra
Alimento: {basura, insectos}
Hogar: Monterrey
Hambre: Siempre



Nombre: Simba
Raza: León
Peso (kg): 190
Altura (metros): 1.2
Sexo: macho
Alimento: {gacela, cebrá}
Hogar: África
Hambre: Poca

- ▼  > AnimalProject [INFO2 master]
 - ▼  > src
 - ▼  > animal
 - >  Animal.java
 - >  AnimalDemo.java

Para este ejemplo, creamos un proyecto llamado **AnimalProject**.

Dentro del proyecto, creamos el paquete **animal** que contiene dos clases:

1. Animal.java
2. AnimalDemo.java

```
package animal;

public class Animal {
    public String name;
    public String race;
    public String[] foods; //array with food the animal likes
    public int hunger; //0 -> not hungry, 10 -> very hungry
}
```

Las clases de Java pueden estar compuestas por una colección de variables.

En el siguiente escenario, la clase Animal nos va a generar una plantilla para representar cualquier animal.

```
package animal;

public class AnimalDemo {

    public static void main(String[] args) {
        Animal rhino = new Animal();
        rhino.name = "Sally";
        rhino.race = "White Rhino";
        rhino.hunger = 10; //very hungry!
        rhino.foods = new String[]{"fruit", "bushes", "grass"};
    }
}
```



Ojo en la instrucción

`Animal rhino = new Animal();`

Estamos instanciando un objeto de la clase `Animal`. Es decir, vamos a generar una variable que tenga toda la información que definimos en la clase `Animal.java`

```
package animal;

public class AnimalDemo {

    public static void main(String[] args) {
        Animal rhino = new Animal();
        rhino.name = "Sally";
        rhino.race = "White Rhino";
        rhino.hunger = 10; //very hungry!
        rhino.foods = new String[]{"fruit", "bushes", "grass"};

        Animal bear = new Animal();
        bear.name = "Yogi";
        bear.race = "Grizzly";
        bear.hunger = 5; //moderate
        bear.foods = new String[] {"fish", "berries"};

    }

}
```



Respirar

Correr

Dormir

Comer

Métodos de instancia

Las clases pueden implementar **comportamientos** mediante la ejecución de métodos.

Estos métodos utilizan las variables del objeto para representar el estado del objeto.

Simulemos un animal comiendo de la siguiente manera:

Método eat

Un animal va a comer cuando tiene hambre.

Un animal sólo va a comer cosas que le gusten.

Cuando un animal se alimenta, su hambre disminuye.

```

package animal;

public class Animal {
    public String name;
    public String race;
    public String[] foods; //array with food the animal likes
    public int hunger;      //1 -> not hungry, 10 -> very hungry

    public void eat(String inputFood) {
        if (hunger <= 0) {
            System.out.println("I'm full!");
            return;
        }

        //check if Animal eats inputFood
        for(String food: foods) {
            if (food.equals(inputFood)) {
                //When animal eats, hunger is decreased
                hunger--;
                System.out.println("Delicious! I love " + inputFood);
                return;
            }
        }
        System.out.println("I don't like " + inputFood);
    }
}

```

Mediante el método eat, cualquier objeto de la clase Animal puede simular el comportamiento de alimentarse.

```
Animal bear = new Animal();
bear.name = "Yogi";
bear.race = "Grizzly";
bear.hunger = 3; //moderate
bear.foods = new String[] {"fish", "berries"};

//We call method eat with the bear object
bear.eat("bushes"); //hunger = 3
bear.eat("fish"); //hunger = 2
bear.eat("berries"); //hunger = 1
bear.eat("fruit"); //hunger = 1
bear.eat("fish"); //hunger = 0
bear.eat("fish"); //Bear is not hungry anymore
```

Output

```
I don't like bushes
Delicious! I love fish
Delicious! I love berries
I don't like fruit
Delicious! I love fish
I'm full!
```

Clases y Métodos

Una clase en Java está compuesta por dos elementos:

- Atributos (datos)
- Métodos (acciones)

Método Constructor

Cuando ejecutamos la siguiente línea de código, estamos invocando al método constructor de la clase Animal.

```
Animal rhino = new Animal();
```

El constructor es un método especial que sirve para instanciar un objeto.

Un constructor no tiene valor de retorno, y su nombre únicamente tiene el nombre de la clase.

```

package animal;
public class Animal {

    public String name;
    public String race;
    public String[] foods; //array with food the animal likes
    public int hunger;      //1 -> not hungry, 10 -> very hungry

    //Constructor
    public Animal(String name, String race, String[] foods, int hunger) {
        this.name = name;
        this.race = race;
        this.foods = foods;
        this.hunger = hunger;
    }

    //...
    //rest of the code
    //...
}

```

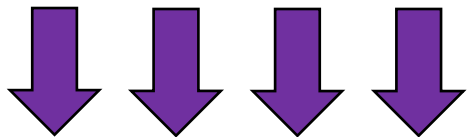
OJO con el nombre de la variable name.
 Cuando utilizamos `this.name` nos referimos a la variable de instancia name definida como parte de la clase.

La variable `name` hace referencia al parámetro de entrada del método constructor.

```
package animal;

public class AnimalDemo {

    public static void main(String[] args) {
        Animal bear = new Animal();
        bear.name = "Yogi";
        bear.race = "Grizzly";
        bear.hunger = 3; //moderate
        bear.foods = new String[] {"fish", "berries"};
    }
}
```



**El código anterior se
convierte en:**

```
package animal;

public class AnimalDemo {

    public static void main(String[] args) {

        Animal bear = new Animal("Yogi", "Grizzly", new String[] {"fish", "berries"}, 3);

    }
}
```


¿Cómo aseguramos la congruencia de un objeto?

Las variables de cualquier objeto deben ser mantener una congruencia interna para que nuestra clase funcione de la manera esperada.

```
Animal bear = new Animal("Yogi",  
                           "Grizzly",  
                           new String[] {"fish", "berries"},  
                           3);  
  
bear.eat("bushes");  
bear.hunger--;
```

La variable **hunger** sólo debería actualizarse cuando el objeto utiliza el método `eat()`!!!

¿Cómo podemos lograr esto?

Modificadores de Acceso

Podemos utilizar un modificador de acceso. Si nosotros cambiamos la variable `hunger` de pública a privada, su contenido no podrá ser modificado desde la clase `AnimalDemo.java`!


```
package animal;

public class Animal {
    public String name;
    public String race;
    public String[] foods; //array with food the animal likes
    private int hunger;    //1 -> not hungry, 10 -> very hungry

    //...
    //rest of the code
    //...
}
```

```
Animal bear = new Animal("Yogi",  
                           "Grizzly",  
                           new String[] {"fish", "berries"},  
                           3);  
  
bear.eat("bushes");  
bear.hunger--;
```

Error durante la compilación

 The field Animal.hunger is not visible

2 quick fixes available:

- ➡ [Change visibility of 'hunger' to 'package'](#)
- ➡ [Create getter and setter for 'hunger'...](#)

Press 'F2' for focus