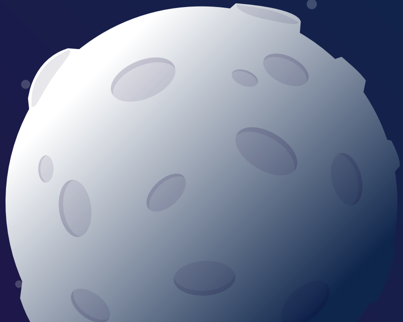
Introduction to object oriented programming



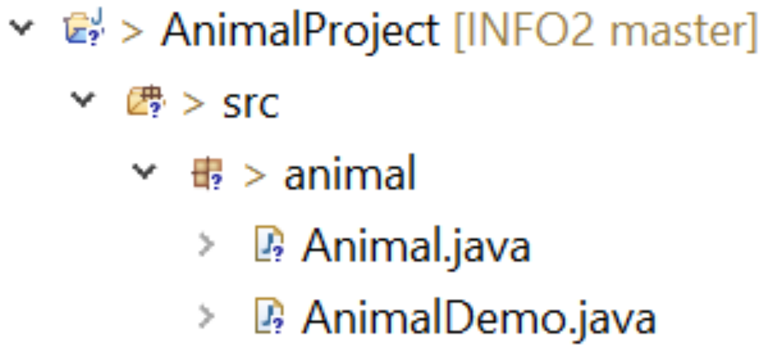
List 4 attributes and 4 actions that all the animals from the next slide share.
For example:
attribute: weight
action: sleep

Name: Sally
Race: White Rhino
Weight (kg): 2300
Height (meters): 1.8
Sex: Female
Home: Africa
Foods: {fruits, bushes, grass}
Hunger: 10

Name: Yogi
Race: Grizzly bear
Weight (kg): 270
Height (meters): 2
Sex: Male
Home: Jellystone Park
Foods: {fish, fruit, insects}
Hambre: 6

Name: Rocket
Race: Racoon
Weight (kg): 7
Height (meters): 0.3
Sex: Male
Home: Monterrey
Foods : {trash, insects}
Hambre: 8

Name: Simba
Race: Lion
Weight (kg): 190
Height (meters): 1.2
Sex: Male
Hogar: South Africa
Foods: {gazelle, zebra}
Hambre: 2



AnimalProject [INFO2 master]
src
animal
Animal.java
AnimalDemo.java

Lets start by creating a project called **AnimalProject.**

Inside the project, we will create a new package **animal** containing two classes:

1. Animal.java
2. AnimalDemo.java

```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
}
```

Java classes can be made up of a list of attributes.

In the following example, we will use the Animal class to create a template that can represent any 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"};
  }

}
```

Pay special attention to the following instruction:

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

We are instantiating an object of the Animal class. In other words, we are creating a variable that holds everything that we defined in the Animal.java class.

```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"};

  }

}
```

breathe

run

sleep

eat

Instance methods

Classes can implement behaviors through the execution of methods.

This methods use the attributes of the object to represent the state of it. Lets simulate an animal eating on the following example:

eat() method

1. An animal will only eat when it is hungry.
2. An animal will only eat things that it likes.
3. When an animal eats, its hunger will decrease.

```java
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);
  }
}
```

Through the eat method, any object from the Animal class can implement the action of eating.

```java
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.

```java
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.

```java
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:

```java
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!

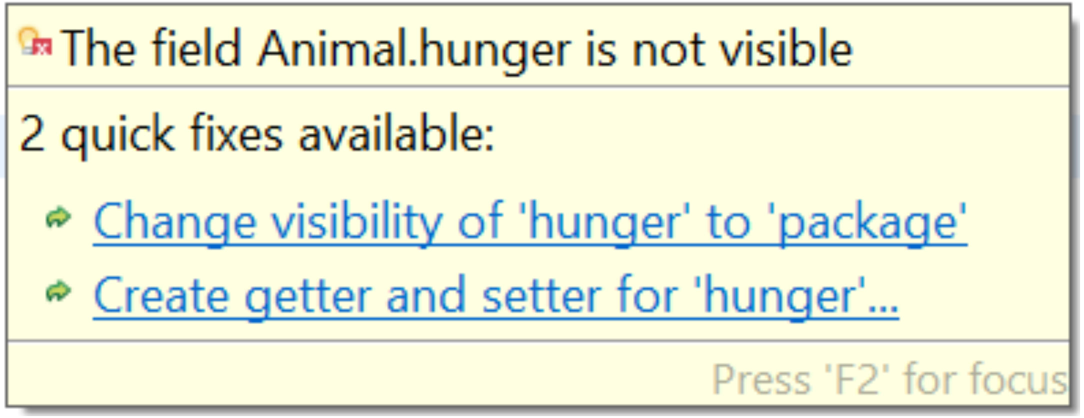
```java
package animal;

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