1. Qué es una Clase

Una clase es un modelo o prototipo definido por el usuario a partir del cual se crean los objetos. Representa el conjunto de propiedades o métodos que son comunes a todos los objetos de un tipo. En general, las declaraciones de clase pueden incluir estos componentes, en orden:

- **Modificadores**: una clase puede ser pública o tener acceso predeterminado (default). (Veremos otros más adelante)
- **Nombre de clase**: el nombre debe comenzar con una letra (en mayúscula por convención).
- **Superclase** (si corresponde): el nombre del elemento primario de la clase (superclase), si lo hay, precedido por la palabra clave **extends**. Una clase solo puede extender (subclase) a uno de los padres.
- **Interfaces** (si corresponde): una lista de interfaces separadas por comas implementadas por la clase, si las hay, precedidas por la palabra clave **implements**. Una clase puede implementar más de una interfaz.
- **Cuerpo**: El cuerpo de la clase rodeado de llaves: {}.

Los constructores se utilizan para inicializar nuevos objetos. Los campos son variables que proporcionan el estado de la clase y sus objetos, y los métodos se utilizan para implementar el comportamiento de la clase y sus objetos.

Hay varios tipos de clases que se utilizan en aplicaciones en tiempo real como clases anidadas, clases anónimas, expresiones lambda. (posteriormente aprenderá cada una de ellas en este blog.)

Aprende más

Clases en Java

IR AL ARTÍCULO

2. Qué es un Objeto

Un objeto es una unidad básica de Programación Orientada a Objetos y representa las entidades de la vida real. Un programa típico de Java crea muchos objetos, que como usted sabe, interactúan al invocar métodos. Un objeto consiste en:

- **Estado**: está representado por atributos de un objeto. También refleja las propiedades de un objeto.
- **Comportamiento**: se representa mediante métodos de un objeto. También refleja la respuesta de un objeto con otros objetos.
- **Identidad**: le da un nombre único a un objeto y permite que un objeto interactúe con otros objetos.

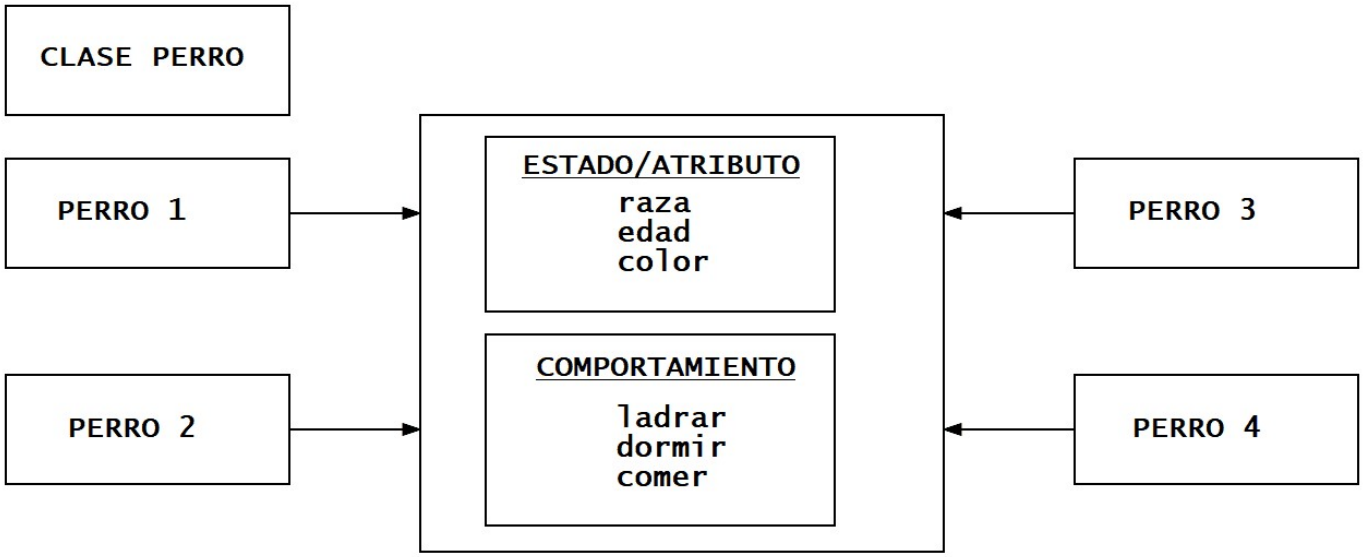
Ejemplo de un objeto: Perro

```
Identidad: nombre del perro

Estado/atributo: Color, Edad, Raza

Comportamiento: Dormir, comer, ladrar
```

Los objetos corresponden a cosas que se encuentran en el mundo real. Por ejemplo, un programa de gráficos puede tener objetos tales como «círculo», «cuadrado», «menú». Un sistema de compra en línea podría tener objetos como «carrito de compras», «cliente» y «producto».

3. Declaración de objetos

Cuando declaramos objetos hablamos en términos de instanciar una clase. Cuando se crea un objeto de una clase, se dice que **la clase está instanciada**. Todas las instancias comparten los atributos y el comportamiento de la clase. Pero los valores de esos atributos, es decir, el estado son únicos para cada objeto. Una sola clase puede tener cualquier cantidad de instancias.



Declaración de objetos

A medida que declaramos variables como (*tipo nombre;*). Esto notifica al compilador que utilizaremos el *nombre* para referirnos a los datos cuyo tipo es *tipo*. Con una variable primitiva, esta declaración también reserva la cantidad adecuada de memoria para la variable. Entonces, para la variable de referencia, el tipo debe ser estrictamente un nombre de clase concreto. En general, no podemos crear objetos de una clase abstracta o una interfaz.

```
Perro clifford;
```

Si declaramos una variable de referencia (clifford) como la de arriba, su valor será indeterminado (nulo) hasta que realmente se cree y se le asigne un objeto. Simplemente declarar una variable de referencia no crea un objeto.

4. Inicializando un objeto

El nuevo operador instancia una clase asignando memoria para un nuevo objeto y devolviendo una referencia a esa memoria. El nuevo operador también invoca el constructor de clase.

Ejemplo:

```
// Declaración de clase


public class Perro
{
    // Variables de instancia
    String nombre;
    String raza;
    int edad;
    String color;


    // Declaración del constructor de clase
```

```
public Perro(String nombre, String raza,
               int edad, String color)
{
    this.nombre = nombre;
    this.raza= raza;
    this.edad = edad;
    this.color = color;
}

// método 1
public String getNombre()
{
    return nombre;
}

// método 2
public String getRaza()
{
    return raza;
}

// método 3
public int getEdad()
{
    return edad;
}

// método 4
public String getColor()
{
    return color;
}

@Override
public String toString()
{
    return("Hola mi nombre es "+ this.getNombre()+
           ".\nMi raza, edad y color son: " +
           this.getRaza()+"," + this.getEdad()+
           ","+ this.getColor());
```

```
    }


    public static void main(String[] args)
    {
        Perro clifford = new Perro("clifford","pitbull", 5, "blanco");
        System.out.println(clifford.toString());
    }
}
```

Salida:

```
Hola mi nombre es clifford.
Mi raza, edad y color son: pitbull,5,blanco
```

Esta clase contiene un solo constructor. Podemos reconocer un constructor porque **su declaración usa el mismo nombre que la clase** y **no tiene un tipo de retorno (return).** El compilador de Java diferencia los constructores en función del número y el tipo de los argumentos. El constructor en la clase *Perro* toma cuatro argumentos. La siguiente declaración proporciona a «clifford», «pitbull», 5, «blanco» como valores para esos argumentos:

```
Perro clifford = new Perro("clifford","pitbull", 5, "blanco");
```

✕ **Nota**: **Todas las clases tienen al menos un constructor**. Si una clase no declara explícitamente ninguna, el compilador de Java proporciona automáticamente un constructor sin argumentos, también llamado «constructor default». Este constructor predeterminado llama al constructor sin argumentos del padre de la clase (ya que contiene solo una declaración, es decir *super ();*), o el constructor de la clase *Object* si la clase no tiene otro padre (ya que la clase Object es padre de todas las clases ya sea directa o indirectamente)

5. Maneras de crear el objeto de una clase

Hay cuatro formas de crear objetos en java. En pocas palabras, solo hay una forma (mediante el uso de la palabra clave `new`), y el resto internamente utiliza una nueva palabra clave.

- **Usar palabra clave (keyword) new**: es la forma más común y general de crear objetos en Java.

Ejemplo:

```
// creando un objeto de clase Test
Test t = new Test ();
```

- **Usando el método Class.forName (String className)**: hay una clase predefinida en el paquete java.lang con el nombre Class. El método forName (String className) devuelve el objeto Class asociado con con el nombre de la clase. Tenemos que dar el nombre completo para una clase. Al llamar al método **new Instance()** en este objeto Class, se devuelve una nueva instancia de la clase con el nombre de cadena proporcionado.

```
// crear un objeto de public class Test
// considerar la clase Test presente en el paquete com.p1
```

```
Test obj = (Test)Class.forName("com.p1.Test").newInstance();
```

- **Usando el método clone()**: el método clone() está presente en la clase Object. Crea y devuelve una copia del objeto.

```
// creando el objeto de la clase Test
Test t1 = new Test ();
// creación del clon del objeto anterior
Test t2 = (Test)t1.clone();
```

- **Deserialización**: Deserialización es la técnica de leer un objeto desde el estado guardado en un archivo. Hablaremos más adelante sobre Serialización/Des-serialización en Java.

```
FileInputStream file = new FileInputStream(filename);
ObjectInputStream in = new ObjectInputStream(file);
Object obj = in.readObject();
```

6. Crear objetos múltiples solo por un tipo (Una buena práctica)

En tiempo real, necesitamos diferentes objetos de una clase en diferentes métodos. Crear una serie de referencias para almacenarlas no es una buena práctica y, por lo tanto, declaramos una variable de referencia estática y la usamos siempre que sea necesario. En este caso, el desperdicio de memoria es menor. Los objetos que ya no se referencia serán destruidos por el recolector de basura de Java. Ejemplo:

```
Test test = new Test();
test = new Test();
```

En el sistema de herencia, usamos una variable de referencia de clase primaria para almacenar un objeto de subclase. En este caso, podemos cambiar a diferentes objetos de subclase usando la misma variable referenciada.

```
class Animal {}


class Dog extends Animal {}
class Cat extends Animal {}


public class Test
{
    // usando el objeto Dog
    Animal obj = new Dog();


    // usando el objeto Cat
    obj = new Cat();
```

```
}
```

7. Objetos anónimos

Los objetos anónimos son los objetos que se instancian pero no se almacenan en una variable de referencia.

- Se usan para llamadas de método inmediato.
- Serán destruidos después de llamar al método.
- Son ampliamente utilizados en diferentes bibliotecas. Por ejemplo, en las bibliotecas AWT se utilizan para realizar alguna acción al capturar un evento (por ejemplo, presionar una tecla).

En el siguiente ejemplo, cuando una tecla/botón (referido por btn) es presionada, simplemente estamos creando un objeto anónimo de la clase EventHandler solo para llamar al método handle.

```
btn.setOnAction(new EventHandler()
{
    public void handle(ActionEvent event)
    {
        System.out.println("Hola Mundo!");
    }
});
```