Clases y objetos

Introducción

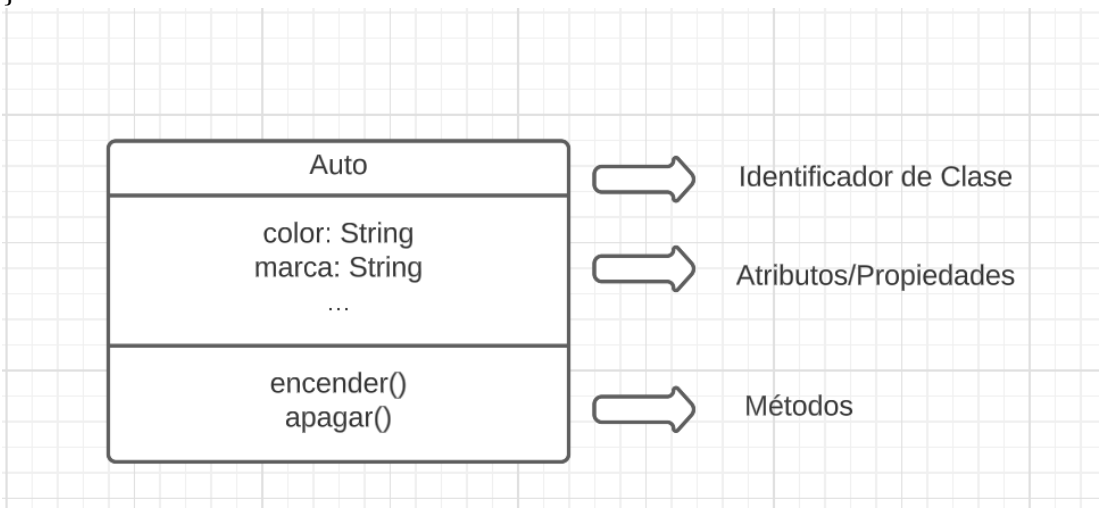
Una clase es una plantilla para la creación de objetos (instancias de una clase). Una clase define las propiedades y comportamientos de un objeto. Por ejemplo, una clase "Auto" podría tener propiedades como "color" y "marca", y métodos como "encender" y "apagar". Los objetos son instancias de una clase y tienen su propia copia de las propiedades de la clase. Por ejemplo, podrías crear dos objetos de la clase "Auto", uno de color rojo y otro de color azul.

Clases

Una clase es una plantilla para la creación de objetos (instancias de una clase). Una clase define las propiedades y comportamientos de un objeto mediante variables de instancia y métodos. Una clase también puede tener constructores, que son métodos especiales que se utilizan para crear objetos de una clase.

Para crear una clase en Java, se utiliza la palabra clave "class" seguida del nombre de la clase. Dentro de las llaves { } se definen las propiedades y comportamientos de la clase.
La declaración de una clase sigue la siguiente sintaxis:

```
[modificadores] class IdentificadorClase {
   // Declaraciones de atributos y metodos
   ...
}
```



Ejemplo de una clase en Java:

```
class Auto {
  String color;
```

```
  String marca;

  void encender() {
    System.out.println("El auto ha sido encendido.");
  }

  void apagar() {
    System.out.println("El auto ha sido apagado.");
  }
}
```

En este ejemplo, la clase "Auto" tiene dos propiedades, "color" y "marca", y dos métodos, "encender" y "apagar".

Convención de los programadores en Java: los identificadores de las clases deberían ser simples, descriptivos y sustantivos y, en el caso de nombres compuestos, con la primera letra de cada uno en mayúsculas. Es conveniente utilizar las palabras completas y evitar los acrónimos, a menos que la abreviatura sea mucho más utilizada que la forma no abreviada como en URL o HTML.

En Java, las propiedades de una clase se llaman atributos, y los comportamientos de una clase se llaman métodos.

Atributos: Son las variables de instancia de una clase que representan las características de un objeto. Los atributos pueden ser de cualquier tipo de datos (numéricos, cadenas, etc.) y pueden ser declarados como públicos o privados. Si se declara como privado solo podrá ser accedido por los métodos de la clase.

Ejemplo de un atributo en una clase:

```
class Auto {
  String color;
  String marca;
  //...
}
```

Métodos: Son las funciones de una clase que representan las acciones que un objeto puede realizar. Los métodos pueden recibir parámetros y devolver valores, y también pueden ser declarados como públicos o privados.

Ejemplo de un método en una clase:

```
class Auto {
  //...
  void encender() {
    System.out.println("El auto ha sido encendido.");
  }
```

```
    //...
}
```
En el ejemplo anterior la clase Auto tiene dos atributos color y marca y un método encender.

Tipos

En Java, existen varios tipos de clases, algunos de los cuales son:

Clases normales: Son las clases regulares que se utilizan para definir objetos y su comportamiento. Por ejemplo, una clase "Person" que tiene atributos como nombre, edad y dirección, y métodos como getName(), setAge() y so on.

Clases abstractas: Son las clases que no se pueden instanciar directamente, pero pueden ser utilizadas como base para otras clases. Contienen métodos abstractos, que deben ser implementados por las clases que heredan de ella.

Interfaces: Son un conjunto de métodos sin implementación que deben ser implementados por las clases que las implementan. Una clase puede implementar varias interfaces.

Clases finales: Son las clases que no se pueden heredar. Es decir, no se pueden crear subclases de una clase final.

Clases anidadas: Son clases que están dentro de otras clases. Pueden ser estáticas o no estáticas.

Clases de enums: Son un tipo especial de clase que representa un conjunto finito de valores.

Clases anónimas: Son clases que no tienen nombre y se utilizan para crear objetos de una sola vez.

Objetos

Un objeto es una instancia de una clase. Los objetos tienen su propia copia de las propiedades de la clase, y pueden acceder a los métodos de la clase. Los objetos se crean mediante la palabra clave "new" seguida del nombre de la clase y un conjunto de paréntesis.

Ejemplo de creación de un objeto en Java:

```
Auto miAuto = new Auto();
```
En este ejemplo, se ha creado un objeto "miAuto" de la clase "Auto". El objeto "miAuto" tiene su propia copia de las propiedades "color" y "marca" y puede acceder a los métodos "encender" y "apagar".

Una vez creado el objeto, se pueden acceder y modificar sus propiedades y llamar a sus métodos mediante el operador "." (punto).

Ejemplo:

```
miAuto.color = "Rojo";
miAuto.marca = "Toyota";
miAuto.encender();
```

En este ejemplo se está accediendo a las propiedades del objeto miAuto y se está llamando al método encender.

El anterior código puede compilarse:
$>javac Auto.java
generando el archivo de bytecodes Auto.class. Este archivo no es directamente ejecutable por el intérprete, ya que el código fuente no incluye ningún método principal (main). Para poder probar el código anterior, puede construirse otro archivo con el código fuente que se muestra a continuación:

```
/**
 * Ejemplo de uso de la clase Auto
 */
public class PruebaAuto {
   public static void main (String []args ) {
      Auto auto;          // Crea una referencia de la clase Auto
      auto = new Auto();   // Crea el objeto de la clase Auto
      auto.color= “Azul”; //setea la propiedad color a Azul
      auto.encender();       // Llamada al método encender
      System.out.println("Color = " + auto.color); // Imprime por pantalla el color de auto
   }
}
```

Referencia Null

En Java, "null" es un valor especial que se utiliza para indicar que una variable de referencia no está apuntando a ningún objeto. Por ejemplo, si se declara una variable de referencia de una clase y no se le asigna ningún objeto, su valor será "null".

Ejemplo:

```
Auto miAuto; //declaración de una variable de referencia de la clase Auto
System.out.println(miAuto); //imprime "null"
```

En este ejemplo, la variable "miAuto" se declara como una referencia a un objeto de la clase "Auto", pero no se le asigna ningún objeto. Por lo tanto, su valor es "null".

Intentar utilizar una variable con valor null puede causar una excepción llamada NullPointerException, es importante siempre verificar antes de utilizar una referencia si esta es null o no.

Ejemplo:

```
if(miAuto != null){
  miAuto.encender();
}
```
En este ejemplo se esta verificando que la variable miAuto no sea null antes de llamar al método encender. Si fuera null no se llamaría al método y se evitaría la excepción.

Referencias compartidas por varios objetos

En Java, varios objetos pueden compartir una misma referencia. Esto significa que varios objetos pueden apuntar al mismo objeto en memoria.

Ejemplo:

```
Auto miAuto = new Auto();
Auto otroAuto = miAuto;
```
En este ejemplo, se crea un objeto "miAuto" de la clase "Auto" y se crea una segunda variable "otroAuto" que apunta al mismo objeto en memoria al que apunta "miAuto". Ambos objetos "miAuto" y "otroAuto" comparten la misma referencia y por lo tanto, cualquier cambio realizado a través de uno de ellos, será reflejado en el otro objeto.

Ejemplo:

```
miAuto.color = "Rojo";
System.out.println(otroAuto.color); //imprime "Rojo"
```

En este ejemplo, se cambia el valor de la propiedad color del objeto miAuto y luego se imprime el valor de la propiedad color del objeto otroAuto, como ambos objetos comparten la misma referencia, el valor impreso es "Rojo"

Ten en cuenta que si deseas que los objetos tengan valores diferentes, debes crear nuevos objetos.
Ejemplo:

```
Auto miAuto = new Auto();
miAuto.color = "Rojo";
Auto otroAuto = new Auto();
otroAuto.color = "Azul";
```

En este ejemplo, se crean dos objetos diferentes y se les asigna valores diferentes a sus atributos.

Ciclo de Vida de un objeto

El ciclo de vida de un objeto en Java se refiere al periodo de tiempo durante el cual un objeto existe y es accesible en la aplicación. El ciclo de vida de un objeto comienza cuando se crea y termina cuando es eliminado de la memoria.

El ciclo de vida de un objeto en Java se divide en cuatro fases:

Creación: El objeto es creado en la memoria mediante la palabra clave "new" o mediante una referencia a un objeto existente.

Inicialización: El objeto recibe valores iniciales para sus atributos y se ejecutan los métodos de inicialización.

Uso: El objeto es utilizado en la aplicación, se accede y modifica sus atributos, y se llaman a sus métodos.

Destrucción: El objeto es eliminado de la memoria cuando ya no es necesario. Esto puede ocurrir de forma automática mediante el Garbage Collector o mediante una llamada explícita al método System.gc() o al método finalize().

Ten en cuenta que el Garbage Collector es un sistema automático en Java que se encarga de liberar la memoria de los objetos que ya no son necesarios. No se puede garantizar exactamente cuando se liberará la memoria de un objeto, pero se asegura que se liberará en algún momento.

Atributos

Los atributos son las variables de instancia de una clase que representan las características de un objeto. Los atributos pueden ser de cualquier tipo de datos (numéricos, cadenas, objetos, etc.) y pueden ser declarados como públicos, privados o protegidos.

La accesibilidad de un atributo se determina por su modificador de acceso, que puede ser "public", "private" o "protected". Si un atributo es declarado como "public", puede ser accedido desde cualquier parte del programa. Si es "private", solo puede ser accedido desde la clase que lo contiene. Si es "protected", puede ser accedido desde la clase que lo contiene y sus subclases.

Ejemplo de un atributo en una clase:

```
class Auto {
  private String color;
  public String marca;
  //...
}
```

En este ejemplo, la clase "Auto" tiene dos atributos, "color" y "marca", el atributo color tiene el modificador de acceso private, lo que significa que solo puede ser accedido desde la clase Auto,

mientras que el atributo marca tiene el modificador de acceso public, lo que significa que puede ser accedido desde cualquier parte del programa.

Además de los modificadores de acceso, los atributos también pueden tener modificadores estáticos, final y volatile.

Métodos

Los métodos son funciones de una clase que representan las acciones que un objeto puede realizar. Los métodos pueden recibir parámetros y devolver valores, y también pueden ser declarados como públicos, privados o protegidos.

Al igual que los atributos, la accesibilidad de un método se determina por su modificador de acceso. Si un método es declarado como "public", puede ser llamado desde cualquier parte del programa. Si es "private", solo puede ser llamado desde la clase que lo contiene. Si es "protected", puede ser llamado desde la clase que lo contiene y sus subclases.

Ejemplo de un método en una clase:

```
class Auto {
  //...
  public void encender() {
    System.out.println("El auto ha sido encendido.");
  }
  private void apagar() {
    System.out.println("El auto ha sido apagado.");
  }
  //...
}
```

En este ejemplo, la clase "Auto" tiene dos métodos, "encender" y "apagar", el método encender tiene el modificador de acceso public, lo que significa que puede ser llamado desde cualquier parte del programa, mientras que el método apagar tiene el modificador de acceso private, lo que significa que solo puede ser llamado desde la clase Auto.

Además de los modificadores de acceso, los métodos también pueden tener modificadores estáticos, final y abstractos.

Los métodos son fundamentales para la programación orientada a objetos ya que permiten la encapsulación de la lógica y la abstracción de la funcionalidad de un objeto.

Los métodos también pueden tener modificadores adicionales, como "static", "final" o "abstract". Un método declarado como "static" pertenece a la clase en lugar de a un objeto individual y puede ser llamado directamente desde la clase sin crear una instancia. Un método declarado como "final"

no puede ser sobreescrito por una subclase y un método declarado como "abstract" debe ser implementado por las subclases.

Existen diferentes tipos de métodos, algunos de los cuales son:

Métodos de acceso: Son aquellos que se utilizan para acceder y modificar los atributos de una clase. Por ejemplo, los métodos "get" y "set" son comunes en las clases para obtener y establecer los valores de los atributos.

Métodos constructores: Son aquellos que son llamados automáticamente cuando se crea un objeto de una clase. El constructor es un método especial que tiene el mismo nombre que la clase y no tiene tipo de retorno. El constructor se utiliza para inicializar los atributos del objeto.

Métodos estáticos: Son aquellos que no necesitan una instancia de la clase para ser llamados, se pueden llamar directamente a través del nombre de la clase. Por ejemplo, un método estático puede ser utilizado para realizar un cálculo y devolver un resultado sin crear un objeto de la clase.

Métodos sobreescritos: Son aquellos que tienen el mismo nombre y la misma firma (nombre y tipo de los parámetros) que un método de una clase padre, pero su implementación es diferente en una clase hija.

Métodos abstractos: Son aquellos que no tienen una implementación específica y deben ser implementados por una clase hija. Los métodos abstractos son declarados con la palabra clave "abstract" y no pueden ser llamados directamente.

Método Constructor

Los métodos constructores en Java son aquellos que son llamados automáticamente cuando se crea un objeto de una clase. El constructor es un método especial que tiene el mismo nombre que la clase y no tiene tipo de retorno.

Sintaxis de un constructor:

Copy code

```
class Auto{
    Auto() {
        // cuerpo del constructor
    }
}
```

Un constructor también puede tener parámetros, esto permite pasar valores iniciales para los atributos del objeto al momento de su creación.

Ejemplo:

```
class Auto {
  private String color;
  private String marca;

  // constructor
  public Auto(String color, String marca) {
    this.color = color;
    this.marca = marca;
  }
  //...
}
```

Si una clase no tiene ningún constructor definido, Java proporcionará uno por defecto (sin parámetros) automáticamente.

En este ejemplo, la clase Auto tiene dos atributos privados "color" y "marca" y un constructor público que recibe dos parámetros. El constructor asigna los valores recibidos a los atributos de la clase.

Para crear un objeto de la clase Auto se utiliza la siguiente sintaxis

```
Auto miAuto = new Auto("Rojo", "Toyota");
```

Al crear el objeto se llama al constructor y se inicializan los atributos color y marca con los valores "Rojo" y "Toyota" respectivamente.

Métodos Accesores (Getters/Setters)

Son métodos que se utilizan para obtener o establecer el valor de una propiedad o atributo de una clase. Estos métodos son también conocidos como getters y setters.
Los métodos getter se utilizan para obtener el valor de una propiedad, mientras que los métodos setter se utilizan para establecer el valor de una propiedad. Estos métodos son comúnmente utilizados en la programación orientada a objetos para controlar el acceso a los atributos de una clase.

Aquí hay un ejemplo de cómo se pueden crear métodos accesores en Java para una clase llamada "Persona":

```
class Persona {
    private String nombre;
    private int edad;

    // Método getter para el nombre
```

```
    public String getNombre() {
        return nombre;
    }

    // Método setter para el nombre
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Método getter para la edad
    public int getEdad() {
        return edad;
    }

    // Método setter para la edad
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

En este ejemplo, la clase Persona tiene dos atributos privados: "nombre" y "edad". Los métodos "getNombre()" y "getEdad()" son los métodos getter para estos atributos, mientras que los métodos "setNombre(String nombre)" y "setEdad(int edad)" son los métodos setter. Los métodos getter simplemente devuelven el valor del atributo correspondiente, mientras que los métodos setter establecen el valor del atributo correspondiente.

Puede usar estos métodos accesores desde otras clases o desde la propia clase Persona:

```
Persona persona = new Persona();
persona.setNombre("Juan");
persona.setEdad(25);
System.out.println(persona.getNombre() + " tiene " + persona.getEdad() + " años.");
```

Este código imprimiría: "Juan tiene 25 años."

Métodos Estaticos

Estos no requieren una instancia de la clase para ser llamados. En lugar de eso, se pueden llamar directamente desde la clase. Los métodos estáticos se definen con la palabra clave "static". Los métodos estáticos también pueden acceder a los atributos estáticos de la clase.

Un ejemplo de un método estático es el método "Math.sqrt()" en la clase Math de Java. Este método calcula la raíz cuadrada de un número y se puede llamar directamente desde la clase Math, sin necesidad de crear una instancia de Math.

Aquí hay un ejemplo de cómo se puede crear un método estático en Java:

```
class Calculadora {
    public static int sumar(int a, int b) {
        return a + b;
    }
}
```
En este ejemplo, la clase "Calculadora" tiene un método estático llamado "sumar", que toma dos números enteros como argumentos y devuelve su suma. El método se puede llamar directamente desde la clase "Calculadora" sin necesidad de crear una instancia de la clase.

```
int resultado = Calculadora.sumar(5, 10);
System.out.println(resultado);
```
Este código imprimiría: "15"

Es importante mencionar que los métodos y atributos estáticos son compartidos entre todas las instancias de una clase, es decir, no tienen un estado propio, por lo cual se recomienda usarlos con precaución.

Métodos Abstractos

Un método abstracto es un método que no tiene un cuerpo (es decir, no tiene sentencias de código entre las llaves {}).
En cambio, se deja a las clases que heredan de la clase abstracta que proporcionen la implementación del método. Los métodos abstractos se declaran utilizando la palabra clave "abstract" antes del tipo de retorno del método. También se puede utilizar la palabra clave "abstract" antes del nombre de la clase para declararla como clase abstracta.

Aquí hay un ejemplo de cómo se pueden utilizar métodos abstractos en Java:

```
abstract class Animal {
    //método abstracto
    public abstract void makeSound();
}

class Dog extends Animal {
    //implementación del método abstracto heredado
    public void makeSound() {
        System.out.println("Woof");
    }
}

class Cat extends Animal {
```

```
    //implementación del método abstracto heredado
    public void makeSound() {
        System.out.println("Meow");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound(); //imprime "Woof"

        Cat cat = new Cat();
        cat.makeSound(); //imprime "Meow"
    }
}
```

En este ejemplo, la clase Animal es una clase abstracta que tiene un método abstracto llamado "makeSound()". Las clases Dog y Cat heredan de la clase Animal y proporcionan implementaciones específicas del método makeSound(). Al llamar makeSound() en un objeto de Dog o Cat, se imprime "Woof" o "Meow" respectivamente.

Lectura requerida:
Sznajdleder, Pablo Augusto: El gran libro de Java a Fondo Editorial Marcombo.
Ceballos, Francisco Javier: Java 2. Curso de Programación, Editorial Ra-Ma

Lectura ampliatoria:
https://www.oracle.com/ar/java/