¿Por qué la herencia múltiple no se admite en Java?

Formulada hace 2 años y 4 meses Activa hace 1 mes Vista 16k veces

16

Quisiera saber ¿cuál es el **problema** con manejar *herencia* multiple en java? es decir estaba trabajando en algo cuando llegue al punto en el que necesitaba *heredar* varias clases abstractas con `extends` en Java pero me doy cuenta de que no esta permitido o al menos intente algo como `extends CLaseA,ClaseB` he buscado un rato y no he encontrado nada que me explique el por que.

11

entonces **¿alguien puede ayudarme a saber por que?**, ademas ya que no es posible hacer esto **¿que otras alternativas tengo?**

java

editada el 21 dic. 19 a las 7:28
Aprendiz
13k 7 15 39

formulada el 21 sep. 17 a las 21:58
theboshy
1,183 1 8 22

3 respuestas

25

Respuesta corta

Podemos decir que existen básicamente dos tipos de herencias múltiples: *herencia múltiple de implementación de Clases* y *herencia múltiple de implementación de Tipos*.

Los creadores de Java prohibieron de forma explícita la primera. De modo que cualquier intento

- Evitar la ambigüedad de código
- Dar más simplicidad a Java, con respecto a C++ y otros lenguajes
- Consideraron que la herencia múltiple causaba más problemas que los que resolvía

Todo esto será explicado con más detalle en la respuesta larga.

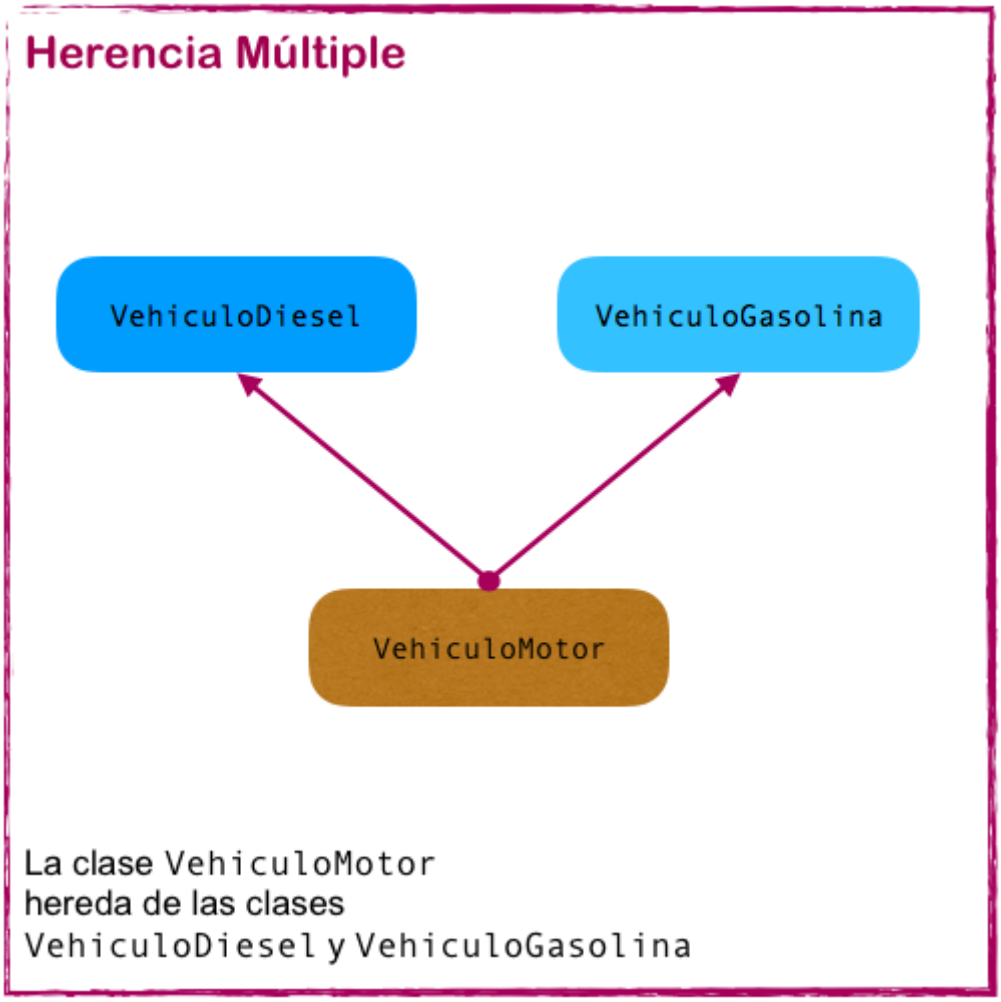
Respuesta larga

Todos sabemos lo que significa *heredar*. En pocas palabras es recibir algo de alguien como nuestro y poder disfrutar de ello.

En POO *heredar* significaría que un objeto recibe (hereda) algo de otro objeto para poder usarlo.

La palabra para expresar esto en Java es `extends`. Cuando una clase extiende más de una clase, esto se denomina herencia múltiple.

Por ejemplo, la clase `VehiculoMotor` extiende de las clases `VehiculoDiesel` y `VehiculoGasolina`.



Ahora bien, podemos hablar de dos tipos de herencia múltiple, según la documentación de Java: *herencia múltiple de implementación* (de clases) y *herencia múltiple de implementación de tipos* (con interfaces).

1. Herencia múltiple de implementación (de Clases)

La *herencia múltiple de implementación* es la capacidad de heredar definiciones de métodos de varias clases.

Este tipo de herencia **no es posible en Java**.

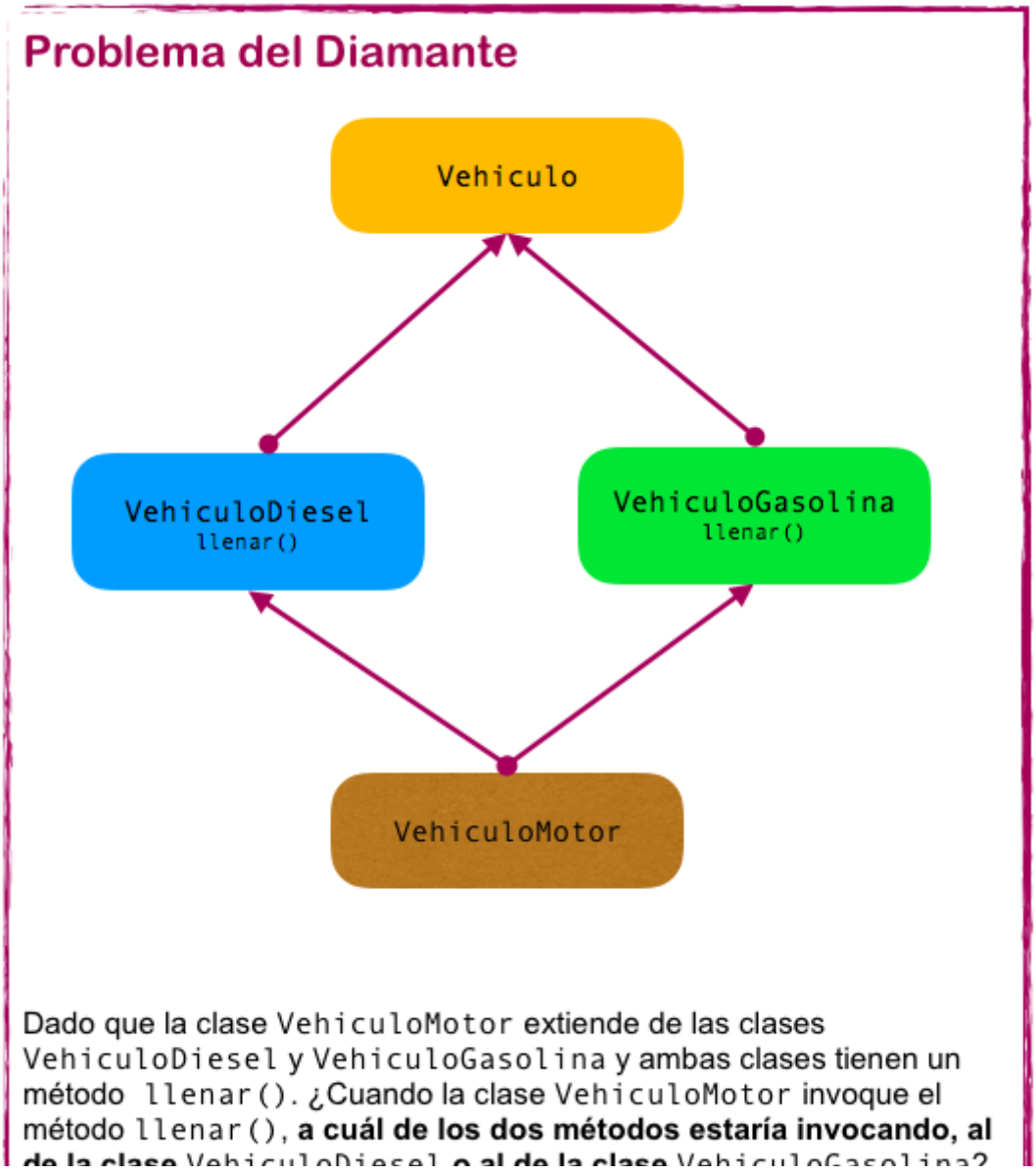
¿Por qué Java no admite herencia múltiple?

La misma documentación (enlace de más arriba), explica algunos motivos:

- Conflictos de nombres y ambigüedad. Cuando los compiladores de los lenguajes de programación que soportan este tipo de superclases de sucesión múltiple contienen métodos con el mismo nombre, a veces no pueden determinar a qué miembro o método acceder o invocar.
- Además, un programador puede involuntariamente introducir un conflicto de nombres añadiendo un nuevo método a una superclase.

Un problema típico que se produciría en la herencia múltiple de implementación sería el llamado *problema del diamante*.

Veamos una imagen:



Este problema se produciría si las clases `VehiculoDiesel` y `VehiculoGasolina` tuviesen métodos con el mismo nombre. Imaginemos que ambas tienen un método `llenar()`. Hasta ahí todo perfecto. Pero, dado que la clase `VehiculoMotor` extiende de estas dos clases. Si hacemos esto:

```
VehiculoMotor camion = new VehiculoMotor();
camion.llenar();
```

¿De qué vamos a llenar nuestro `camion`, de Diesel o de Gasolina? Es una pregunta delicada, ¿no?. Podríamos estropear el camión.

Ese es el principal motivo por el que la herencia múltiple de implementación no es posible en Java.

2. Herencia múltiple de implementación con tipos

A partir de Java 8, gracias a la introducción de los métodos predeterminados (default methods) se puede usar una forma de herencia múltiple de implementación, usando interfaces. Es decir, una clase puede implementar (`implements`) más de una interfaz, que puede contener métodos predeterminados que tienen el mismo nombre. El compilador Java proporciona algunas reglas para determinar qué método predeterminado utiliza una clase en particular.

El lenguaje de programación Java admite múltiples herencias de tipo, que es la capacidad de una clase para implementar más de una interfaz. Un objeto puede tener varios tipos: el tipo de su propia clase y los tipos de todas las interfaces que implementa la clase. Esto significa que si se declara que una variable es el tipo de una interfaz, entonces su valor puede referenciar cualquier objeto que se instancia desde cualquier clase que implemente la interfaz. Esto se discute en la sección Utilización de una interfaz como tipo de la documentación de Java.

Al igual que con la herencia múltiple de implementación, una clase puede heredar diferentes implementaciones de un método definido (como predeterminado o estático) en las interfaces que se extiende. En este caso, el compilador o el usuario deben decidir cuál utilizar.

De este modo se resuelve el *problema del diamante* mencionado más arriba y que era el principal motivo por el cual en el lenguaje Java se impidió la herencia múltiple entre clases.

Tomando el mismo ejemplo de la imagen, podemos hacer que la clase `VehiculoMotor` implemente dos interfaces `LlenableGasolina` y `LlenableDiesel`, cada una de ellas con un método por defecto `llenar()`.

Interfaz `LlenableDiesel`

```
interface LlenableDiesel {

    public default void llenar() {
        System.out.println("Llenar con Diesel");
    }
}
```

Interfaz `LlenableGasolina`

```
interface LlenableGasolina {

    public default void llenar() {
```

Teniendo las dos interfaces así, podemos crear una clase `VehiculoMotor` en la cual el programador puede sobre-escribir el método `llenar()` indicando dentro de él **cuál** de los dos métodos será llamado, usando para ello el método por defecto de cualquiera de las interfaces que implementa la clase. Así, si ha de ser llamado el método correspondiente a los vehículos diesel:

```
@Override
public void llenar() {
    LlenableDiesel.super.llenar();
}
```

En cambio, para llamar al método `llenar()` de los vehículos de gasolina:

```
@Override
public void llenar() {
    LlenableGasolina.super.llenar();
}
```

La salida de este código:

Clase `VehiculoMotor`

```
public class VehiculoMotor implements LlenableDiesel, LlenableGasolina {

    public static void main(String args[]) {

        VehiculoMotor camion = new VehiculoMotor();
        camion.llenar();

    }

    @Override
    public void llenar() {
        LlenableDiesel.super.llenar();
    }

}
```

Sería entonces esta:

```
Llenar con Diesel
```

- Fuente: Does Java support Multiple inheritance?

Más detalles

El asunto fue tratado hace años (en el 2002) en JavaWorld.

Entre los motivos de por qué Java no permite la herencia múltiple se dice:

> Las razones para omitir la herencia múltiple del lenguaje Java provienen principalmente de la meta *"simple, orientada a objetos y familiar"*. Como lenguaje simple, los creadores de Java querían un lenguaje que la mayoría de los desarrolladores pudieran captar sin un entrenamiento extensivo. Para ello, trabajaron para hacer que el lenguaje sea similar a C ++

> En opinión de los diseñadores, la herencia múltiple causa más problemas y confusión de lo que resuelve... La extensa experiencia C ++ de los diseñadores les enseñó que la herencia múltiple era un dolor de cabeza que no valía la pena.

Y dicen que se optó como solución usar múltiples interfaces:

> En cambio, los diseñadores de Java optaron por permitir la herencia de múltiples interfaces a través del uso de interfaces, una idea tomada de los protocolos de Objective C. La herencia de múltiples interfaces permite que un objeto herede muchas firmas de métodos diferentes con la advertencia de que el objeto heredador debe implementar esos métodos heredados.
>
> La herencia de múltiples interfaces permite que un objeto herede métodos y se comporte de forma polimórfica en esos métodos.

En ese artículo hay enlaces a otros artículos y discusiones interesantes sobre el tema, por si se quiere profundizar.

editada el 9 oct. 17 a las 21:36

respondida el 21 sep. 17 a las 22:18

A. Cedano
63.5k 11 79 166

Gracias @A. Cedano veía esto bastante amplio abarcando mucho tema pero su respuesta me ayudo a comprender mas concreta mente el por que. – theboshy el 21 sep. 17 a las 22:24

1 @theboshy la herencia múltiple sí se admite en Java pero bajo ciertas restricciones. – user227 el 22 sep. 17 a las 6:02

13

Para agregar a la respuesta de A.Cedano, hay dos consideraciones adicionales:

1. Java soporta la herencia múltiple a nivel de interfaces. Ejemplo:

```
public interface Calentable {
    void aumentarTemperatura(double incremento);
    void disminuirTemperatura(double decremento);
}
public interface Redimensionable {
    void crecer(Dimension incremento);
    void decrecer(Dimension decremento);
}
//al heredar de 2 o más interfaces
//esta interfaz permite tener todos los métodos de
//todas las interfaces que hereda
public interface Termodinamico extends Calentable, Redimensionable {
}

//una clase puede implementar la interfaz Termodinamico
//en consecuencia, está implementando las demás interfaces
public class Metal implements Termodinamico {
    @Override
    public void aumentarTemperatura(double incremento) {
        /* implementación */
    }
    @Override
    public void crecer(Dimension incremento) {
        /* implementación */
    }
```

```java
//una clase puede ser diseñada para implementar solo una interfaz
public class Agua implements Calentable {
    @Override
    public void aumentarTemperatura(double incremento) {
        /* implementación */
    }
}
```

2. Desde Java 8, se extiende el poder de las interfaces al tener métodos por defecto, es decir, que una interfaz puede proveer la implementación de sus métodos. Esto significa que una interfaz/clase, al extender/implementar más de una interfaz, ahora podría *caer* en el problema del diamante. El compilador soluciona estos casos exigiendo a la clase concreta que provea una implementación para el método conflictivo. Ejemplo:

```java
public interface Afectuoso {
    default String saludar(String nombre) {
        return "¡Saludos " + nombre + "! Venga ese abrazo";
    }
}

public interface Cordial {
    default String saludar(String nombre) {
        return "Buenas " + nombre;
    }
}

//ambas interfaces tienen un método String saludar(String nombre)
//y ambas proveen una implementación por defecto
//la clase se ve forzada a implementar dicho método
public class Persona implements Afectuoso, Cordial {
    @Override
    public String saludar(String nombre) {
        //se puede elegir la implementación de una interfaz
        //por ejemplo, escogemos que sea Afectuoso
        return Afectuoso.super.saludar(nombre);
    }

    public static void main(String[] args) {
        Persona persona = new Persona();
        System.out.println(persona.saludar("Luiggi"));
    }
}
```

Salida:

```
¡Saludos Luiggi! Venga ese abrazo
```

respondida el 22 sep. 17 a las 5:55

user227

2

James Gosling lo descartó al crear java. Observó que la herencia múltiple no añadía grandes ventajas en C++, y qué ésta podía ser implementada de mejor manera con el correcto uso de las interfaces.

respondida el 21 sep. 17 a las 22:11

5 ¿Tienes una fuente para esta afirmación? Deberías añadirla para completar tu respuesta. – Alvaro Montoro ♦ el 22 sep. 17 a las 4:47