

Orientación a Objetos

Herencia

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

22 de agosto de 2024



Herencia simple

- 1 Herencia simple
- 2 Herencia múltiple
- 3 Ejemplo

Temas

- 1 Herencia simple
 - Relación de generalización/especialización
 - Herencia en Java
 - Sobreescritura

Herencia simple

Definición (Generalización/especialización)

- Se dice que una clase *generaliza* a otras cuando abstrae las características comunes a todas ellas y las concentra en su propia definición.
- Se dice que una clase es una *especialización* cuando agrega características o funcionalidad particular a la definición de una clase más general.
- A esta relación también se le suele etiquetar como *a es un tipo de b*.

Ejemplo: piezas de ajedrez

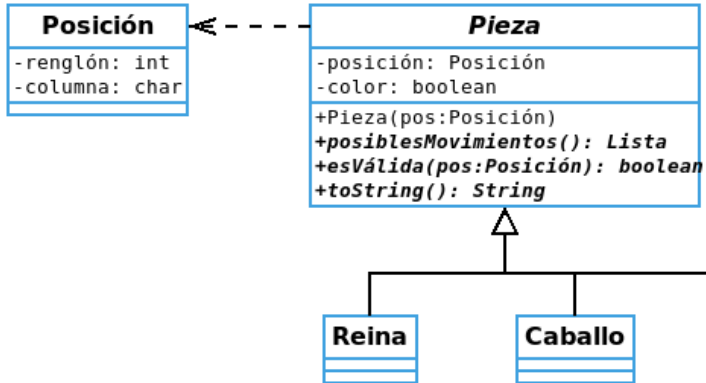


Figura: En el juego de ajedrez todas las piezas tienen información y métodos comunes, pero cada tipo de pieza ejecuta esos métodos de forma particular.

Temas

1 Herencia simple

- Relación de generalización/especialización
- Herencia en Java
- Sobreescritura

Herencia en Java

- Java permite crear relaciones de especialización por medio del un mecanismo de *herencia simple*, donde una clase *hereda de* o *extiende* una sola **clase padre**.
- La clase padre puede a su vez haber extendido a otra clase.
- El ancestro común a todas las clases en Java es la clase `Object`.
- Toda clase **hija** *hereda* todos los atributos y métodos de su clase padre. Esto ocurre de forma recursiva, obteniendo así también los elementos de todos sus ancestros.
- Una clase hija no podrá ver los elementos privados (`private`) de sus ancestros, si necesita accederlos tendrá que ser de forma indirecta, pero puede acceder a los protegidos (`protected`) como si fueran propios.
- Se dice que un objeto es del tipo (`instanceof`) de la clase de la cual fue instanciado, pero también de cualquiera de sus ancestros.

Accesos en Java

En este contexto, conviene introducir, en detalle, qué visibilidad tienen en Java los elementos calificados con los distintos tipos de accesos.

Modificador	Visibilidad			
	Clase	Paquete	Subclase	El mundo
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
∅	✓	✓	✗	✗
private	✓	✗	✗	✗

Ancestros en la documentación

OVERVIEW MODULE PACKAGE **CLASS** USE TREE DEPRECATED INDEX HELP

ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module javafx.graphics
Package javafx.scene.paint
Class Color

java.lang.Object
 javafx.scene.paint.Paint
 javafx.scene.paint.Color

All Implemented Interfaces:
 Interpolatable<Color>

```
public final class Color
extends Paint
implements Interpolatable<Color>
```

Figura: Documentación de Java. Debajo del nombre de la clase se tienen referencias a la documentación de todos sus ancestros. <https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/paint/Color.html>

Declaración

Sintaxis (Declaración de un clase que hereda de otra)

```
<clase hija> ::= <acceso> <modificador> class <nombre> extends <nombre> {  
    <implementación>  
}  
<modificador> ::= abstract | final | ∅
```

Código: Círculo especializa a Figura

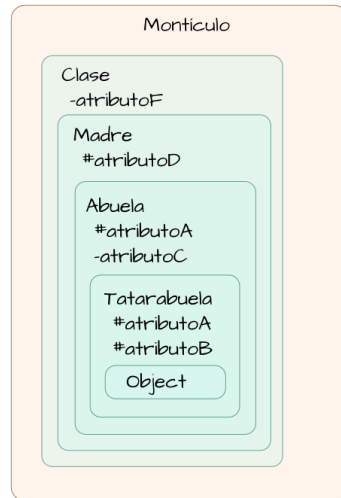
```
1 public class Círculo extends Figura {  
2     ...  
3 }
```

Visualización de un objeto en el montículo

```

1 public class Tatarabuela {
2     protected Object atributoA;
3     protected Object atributoB;
4 }
5 public class Abuela extends Tatarabuela {
6     protected Object atributoA;
7     private Object atributoC;
8 }
9 public class Madre extends Abuela {
10     protected Object atributoD;
11 }
12 public class Clase extends Madre {
13     private Object atributoF;
14 }

```



Tipos de heredabilidad

Las clases se clasifican según su funcionalidad en la jerarquía de herencia.

- ❶ Clases comunes extendibles. Son clases de las cuales se pueden crear objetos y también son o podrían ser extendidas en algún futuro por otras clases.
- ❷ Clases finales (`final`). Son clases que ya no se pueden extender.
- ❸ Clases abstractas (`abstract`). Su declaración incluye el modificador `abstract`.
 - No es posible crear objetos de este tipo.
 - Forzosamente se debe heredar de ellas y crear objetos de los tipos derivados.
 - Una clase es abstracta por los siguientes motivos:
 - Porque tiene uno o más **métodos abstractos**.
 - Porque extiende un clase abstracta y no va a implementar sus métodos abstractos.
 - Porque implementa una interfaz, pero no implementa todos los métodos indicados por esta.
 - Por que en su declaración se utilizó el modificador `abstract`, aunque no esté en ninguno de los casos anteriores.

Métodos abstractos

- Los *métodos abstractos* no están implementados en la clase, sólo se incluye su encabezado con el modificador `abstract` y la **declaración** termina con `;`.

Código: Clase con un método abstracto.

```
1 public abstract class UnaClaseAbstracta {  
2     public abstract Object unMétodo(Object parámetro);  
3 }
```

Temas

- 1 Herencia simple
 - Relación de generalización/especialización
 - Herencia en Java
 - Sobreescritura

Sobreescritura de métodos

- Si una clase **hija** declara un método con la misma **firma** que una clase **ancestra**, el código que se ejecuta es el de la clase hija. En este caso se dice que *sobreescribe* el método.
- Un método que se declare como `final` no puede ser sobreescrito.

Herencia múltiple

- 1 Herencia simple
- 2 Herencia múltiple
- 3 Ejemplo

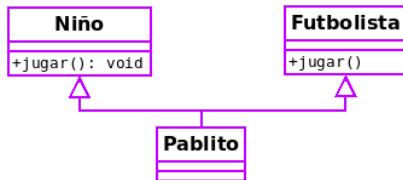
Temas

2 Herencia múltiple

- Concepto
- Interfaces

Herencia múltiple

- Se habla de *herencia múltiple* cuando una clase puede heredar atributos y métodos de varias clases.
- Este tipo de herencia existe en lenguajes como C++ y Python.
- Es difícil de implementar porque, si dos o más clases no relacionadas, de las que hereda la clase de interés, definieron métodos con las mismas firmas ¿a cuál se debe llamar?
- Por este motivo Java no permite herencia múltiple.



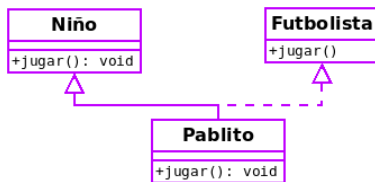
Temas

2 Herencia múltiple

- Concepto
- Interfaces

Interfaces

- Java sustituye la falta de herencia múltiple con la posibilidad de implementar varias interfaces.
- Aunque una interfaz no tiene implementaciones de métodos, sí permite garantizar que cualquier clase que la implementa tendrá un método con esa firma.



- Elegir dentro de un enunciado switch.

```
1 private void actúa(Comando c) {  
2     switch(c) {  
3         case salir:  
4             // Salir  
5         case bin:  
6             // ...  
7             break;  
8         case suma:  
9             // ...  
10            break;  
11        default:  
12            System.out.println("Comando inválido");  
13            imprimeComandos();  
14    }  
15 }
```

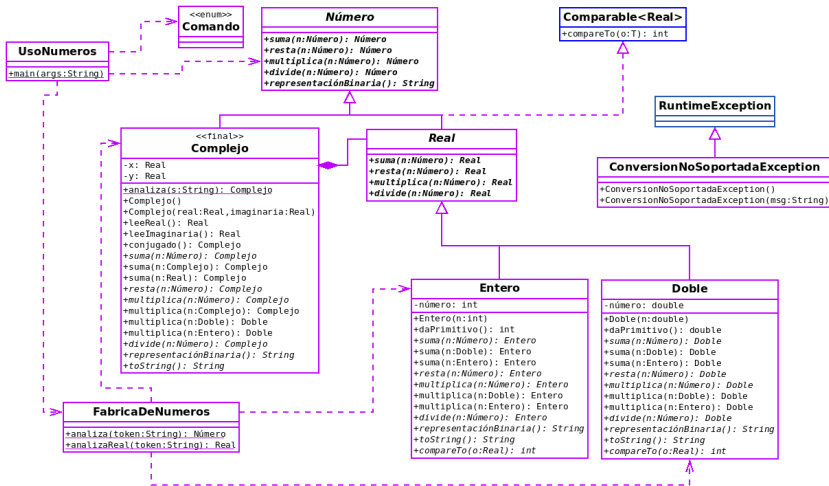
Ejemplo

- 1 Herencia simple
- 2 Herencia múltiple
- 3 Ejemplo

Temas


- 3 Ejemplo
 - Números

Números



Bibliografía I



 Gallardo, Raymond et al. (30 de nov. de 2020). *Lesson: Interfaces and Inheritance*. Ed. por Oracle. URL: <https://docs.oracle.com/javase/tutorial/java/IandI/index.html>.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual

