PROGRAMACIÓN UD-8

HERENCIA

ÍNDICE

[SUBCLASE Y SUPERCLASE 3](#_Toc126222958)

[EJECUCIÓN 3](#_Toc126222959)

[MODIFICADORES DE ACCESO PARA HERENCIA 4](#_Toc126222960)

[REDEFINICIÓN DE MIEMBROS HEREDADOS 5](#_Toc126222961)

[super Y super() 5](#_Toc126222962)

[SELECCIÓN DINÁMICA DE MÉTODOS 5](#_Toc126222963)

[OBJECT 6](#_Toc126222964)

[CLASES ABSTRACTAS 6](#_Toc126222965)

# SUBCLASE Y SUPERCLASE

Las clases nos han permitido definir los atributos y métodos de un objeto del mundo real. Con la herencia, conseguimos que esos atributos y métodos puedan ser heredados por sus hijos y así poder reutilizar código común a varias clases.

La clase de la que se hereda es la clase padre o superclase y la clase que hereda se denomina su clase o clase hija.

Si nos fijamos en el API de Java veremos que la mayoría de las clases no se definen desde cero, sino que heredan de otras clases. Todas las clases acaban heredando de la clase Object.

Al crear un objeto de una subclase dispondremos de todos los métodos y atributos de la clase padre y de los definidos en la propia clase.

Si tenemos una clase de Persona con atributos nombre, edad y estatura, y una clase Empleado con atributo salario, el Empleado dispondrá de los atributos nombre, edad, estatura y salario.

Una subclase puede ser superclase de otra.

Diagrama

Descripción generada automáticamente

# EJECUCIÓN

Para definir la herencia, usamos la palabra extends tras el nombre de la clase e indicamos cuál es el nombre de la superclase:

Texto

Descripción generada automáticamente con confianza baja

Veamos un ejemplo con Empleado que hereda de Persona:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

# MODIFICADORES DE ACCESO PARA HERENCIA

Existe un nuevo modificador de visibilidad llamado protected, es similar a la visibilidad por defecto, con la diferencia de que sean siempre visibles para las clases que hereden.

Diagrama

Descripción generada automáticamente

Por tanto, con este nuevo modificador de acceso, el alcance de visibilidad de las clases es el siguiente:

Tabla

Descripción generada automáticamente

# REDEFINICIÓN DE MIEMBROS HEREDADOS

Puede ser interesante modificar tanto atributos (ocultación) como métodos (*overriding*) heredados. En caso de que se modifique un atributo, el atributo del padre se oculta y si es un método se sustituye.

Diagrama

Descripción generada automáticamente

# super Y super()

Igual que existe el indicativo this para referirnos a la propia clase, con super nos referimos a la clase padre. Se puede llamar desde el constructor de la subclase, llamar al constructor de la clase padre con super(). En este caso tiene que ser la primera instrucción del constructor de la subclase, y puede llevar parámetros. Es necesario que exista un constructor en la superclase con los parámetros indicados en la llamada a super().

# SELECCIÓN DINÁMICA DE MÉTODOS

Una variable de una clase padre puede referenciar a una de la clase hija, al revés no es posible. esto funciona igual que en la vida real, un informático siempre va a ser persona, por tanto, la persona puede referenciar a un informático. Por otra parte, podemos intentar referenciar desde informático a una persona que sea por ejemplo panadero, y eso no es correcto.

Diagrama, Texto

Descripción generada automáticamente

# OBJECT

La clase object de la clase java.lang es la clase superior en la jerarquía de objetos, o sea todos los datos los objetos heredan de esta clase, salvo los tengan una herencia explícita.

Existe una serie de métodos asociados a esta clase que se podrán utilizar o sobrecargar en cualquier clase:

* toString(): devuelve una cadena con la información del objeto, [paquete.clase@refAObjeto](mailto:paquete.clase@refAObjeto). esta información no suele ser relevante, por lo que se suele sobrecargar el método.
* equals(): para dos objetos y devuelve un booleano, si son iguales- entre objetos no se puede comparar con el operador ==, porque comparas solamente donde apuntan y no el contenido. en este método el objeto con el que se compara se pasa como Object y hay que hacer un cash con la clase a comprar. luego se comparan los atributos que se consideren oportunos (DNI de una persona).
* getClass(): devuelve un String con la clase del objeto. El formato que devuelve es “class paquete.class”.

si se desea obtener solamente el nombre de la clase se puede hacer invocando al método getName() de la clase String. También podemos invocar al método getSuperclass() que devuelve la clase de padre.

Diagrama

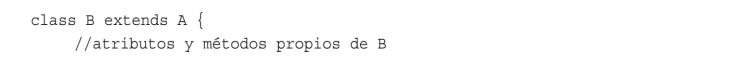
Descripción generada automáticamente

# CLASES ABSTRACTAS

Es el que en una clase no podamos implementar un método por falta de datos concretos, pero que sepamos que dicho método se va a implementar en las clases que hereden a esta clase, donde tendrán los atributos necesarios. esto lo podemos definir implementando métodos abstractos “abstract tipo nombre([parámetros])”.

Si una clase tiene algún método abstracto, esta se debe definir también como abstracta y no será instanciable (no se pueden crear objetos de esta clase). Si una clase hereda de una clase abstracta y alguna de las clases abstractas heredables sin definir, esta nueva clase será abstracta también. se puede definir una referencia a una clase abstracta, que será igualada a un objeto de sus hijos y, por tanto, podrá ejecutar métodos diferentes en función de la clase del objeto asociado.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente Texto

Descripción generada automáticamente Imagen que contiene Patrón de fondo

Descripción generada automáticamente  