everis
an NTT DATA Company

04 GENERICIDAD Y JAVA IO

everis
an NTT DATA Company

DEFAULT INTERFACE METHODS

PROBLEMA

¿Qué pasa si tenemos una interfaz de la que extienden muchas clases y queremos añadir un nuevo método?

DEFAULT INTERFACE METHODS

SOLUCIÓN

Los métodos DEFAULT son una nueva característica de Java 8 que permite al desarrollador añadir nuevos métodos a las interfaces sin romper las implementaciones existentes de esa interfaz.

Nos da flexibilidad para definer la implementación que se usará por default en el caso de que una clase no tenga esta última.

Los métodos default pueden ser añadidos a una interfaz sin afectar a las clases que la implementan, ya que incluye una implementación por defecto.

Al ser ya definido en la interfaz no es necesario que se reimplemente en las implementaciones, aunque si se puede hacer el override si conviene,

DEFAULT INTERFACE METHODS

EJEMPLO

```
public interface Iterable<T> {
    public void forEach(Consumer<? super T> consumer);
}
```

Ahora con default:

```
public interface Iterable<T> {
public default void forEach(Consumer<? super T> consumer) {
    for (T t : this) {
        consumer.accept(t);
    }
}
}
```

DEFAULT INTERFACE METHODS

PROBLEMA DE AMBIGÜEDAD EN HERENCIA MULTIPLE

Dado que una clase java puede implementar varias interfaces,y puede ser que esa interfaz tenga un default method con el mismo nombre, los metodos heredados pueden tener conflictos.

```
public interface InterfaceA {
    default void defaultMethod(){ System.out.println("Interface A default method");  }
}
public interface InterfaceB {   default void defaultMethod(){  System.out.println("Interface B default method");   }
}
public class Impl implements InterfaceA, InterfaceB  {} ERROR DE COMPILACION
```

Solucion→Implementar el default method:

```
public class Impl implements InterfaceA, InterfaceB {
        public void defaultMethod(){
        }
}
```

REFLECTION

DEFINICION

Mediante la Java Reflection API el programador puede inspeccionar y manipular clases e interfaces (así como sus métodos y campos) en tiempo de ejecución, sin conocer a priori (en tiempo de compilación) los tipos y/o nombres de las clases específicas con las que está trabajando.

El concepto de **Reflection** simplemente nos quiere decir que **el programa tiene la capacidad de observar y modificar su estructura de manera dinámica**. De ahí que se refleje su mismo código: podemos elegir una clase que hemos escrito previamente en un IDE y después de compilada, mientras el programa se ejecuta, poder modificarla.

Articulo de consulta: https://jarroba.com/reflection-en-java/

GENERICIDAD

- Facilidad de un lenguaje de programación para definir clases parametrizadas con tipos de datos.
- Resultan de utilidad para la implementación de tipos de datos contenedores como las colecciones.
- La genericidad sólo tiene sentido en lenguajes con comprobación estática de tipos, como Java.
- La genericidad permite escribir código reutilizable.
- La genericidad en el lenguje Java fue introducida a partir de la versión 5

GENERICIDAD

- Una clase genérica es una clase que en su declaración utiliza un tipo variable (parámetro), que será establecido cuando sea utilizada.
- Al parámetro de la clase genérica se le proporciona un nombre (T, K, V, etc.) que permite utilizarlo como tipo de datos en el código de la clase.
- Sobre las variables cuyo tipo sea el parámetro (T, K, V, etc.) sólo es posible aplicar métodos de la clase Object: dado que representan “cualquier dato” sólo podemos aplicar operaciones disponibles en todos los tipos de datos del lenguaje Java.

GENERICIDAD

- Las clases genéricas no pueden ser parametrizadas a tipos primitivos.
- Para resolver este problema el lenguaje define clases envoltorio de los tipos primitivos:

  Integer, Float, Double, Character, Boolean, etc.
- El compilador transforma automáticamente tipos primitivos en clases envoltorio y viceversa:

  autoboxing.
- Las reglas del polimorfismo se mantienen entre clases genéricas. Sin embargo, en una asignación polimórfica no está permitido que tengan distintos parámetros.
- Es una limitación en el paso de parámetros.

JAVA INPUT/OUTPUT

El package de java.io contiene las clases que necesitamos para ejecutar las entradas y salidas en JAVA.

Esta basado en streams de datos desde una entrada o hacia una salida.

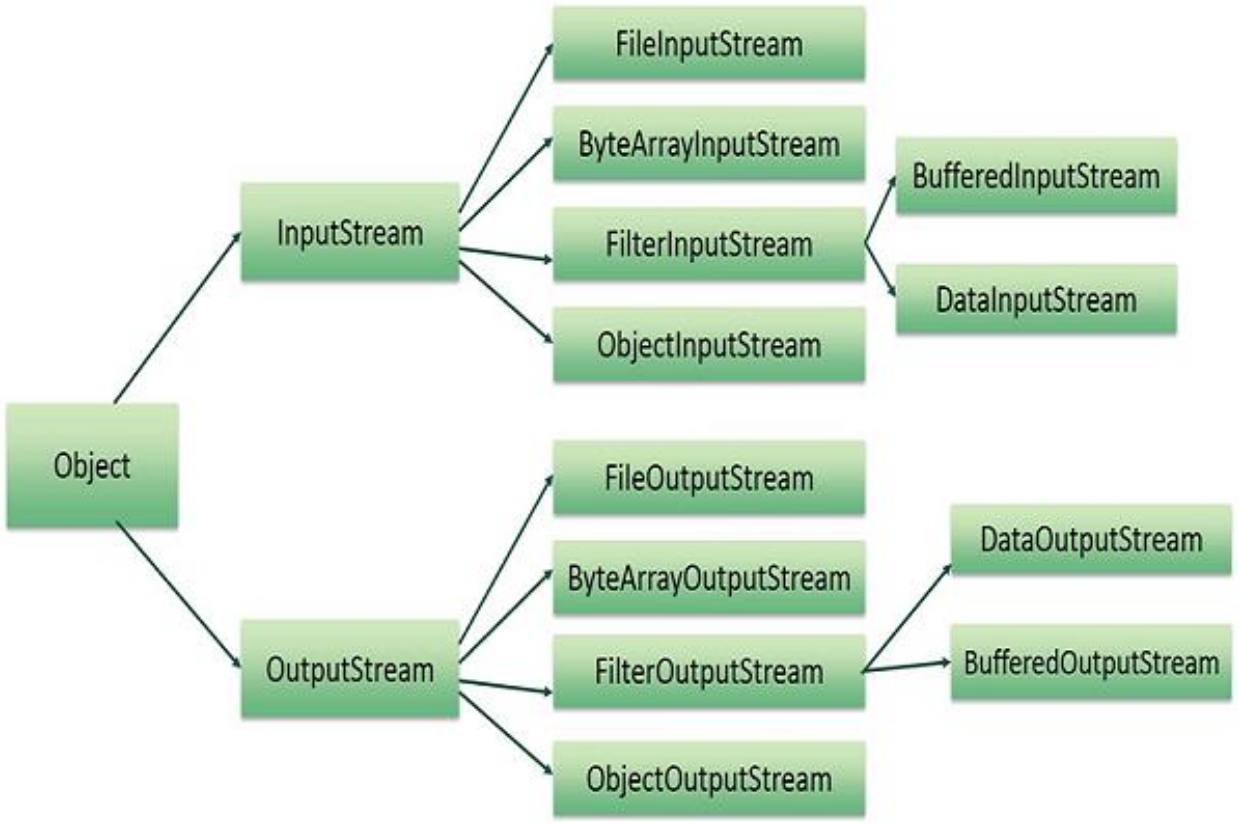
Un stream en java soporta muchos tipos como pimitivos,object,etc..

Un stream puede ser definifo como una secuencia de datos y tenemos dos tipos:

- **InPutStream** − Lee datos de una fuente
- **OutPutStream** − Escribe datos a otra fuente.

everis
an NTT DATA Company

JAVA I/O



docs.oracle.com/javase/tutorial/essential/io/

ANOTACIONES

Las anotaciones son una especie de comentarios o metadatos que puedes introducir en tu código, pero que no son en realidad parte del programa.

Puedes elegir que sean procesadas durante la compilación o bien durante la ejecución, a través del API de Reflection.

Hay algunas anotaciones ya predefinidas en el lenguaje y que resultan útiles a la hora de compilar:

@Override - informa al compilador que el método al que anota está sobreescribiendo un método de la superclase. Si por algún motivo no sobreescribimos bien el método, el compilador genera un error indicando dónde está el fallo.

@Deprecated - un elemento marcado con esta anotación indica que está en desuso, bien porque es peligroso, bien porque hay otra alternativa mejor. Un elemento así debería ser también comentado utilizando la etiqueta @deprecated(con minúscula). El compilador genera un aviso cuando se utiliza un elemento marcado @Deprecated.

@SuppressWarnings - indica al compilador que elimine dos tipos de advertencias que generaría de otro modo:

ANOTACIONES

EJEMPLO

@Target : Esta anotación sirve para delimitar en que partes de nuestro código podemos utilizar la anotación.

@Retention : Esta anotación sirve para asegurarnos que la información que nos provee la anotacion disponible en tiempo de ejecución .

```
@Target(value={ElementType.FIELD} )
@Retention(RetentionPolicy.RUNTIME)
public @interface MiAnotacion {

 String campo();
 String tipo();

}
```