Polimorfismo en Java -Interface- (Parte II), con ejemplos

El proyecto de este post lo puedes descargar pulsando [AQUI](https://github.com/jarroba/PolimorfismoInterfaceJarroba).

Post original <https://jarroba.com/polimorfismo-en-java-interface-parte-ii-con-ejemplos/>

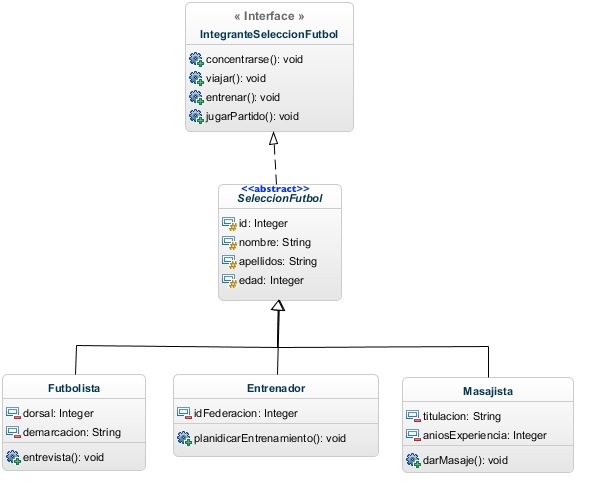
En esta entrada vamos a continuar hablando de polimorfismo y en concreto vamos a explicar las "***Interface***" y como siempre lo vamos ha hacer con un ejemplo. Para entender lo que vamos a contar en esta entrada es imprescindible que sepais que es la herencia y el polimorfismo, por tanto sino teneis muy claros estos conceptos recomendamos que mireis los siguientes tutoriales en los que hablamos sobre ello:

* ***Herencia ->***  [Herencia en Java, con ejemplos](https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/)
* ***Polimorfismo ->***  [Polimorfismo en Java (Parte I), con ejemplos](https://jarroba.com/polimorfismo-en-java-parte-i-con-ejemplos/)

El concepto de ***Interface*** lleva un paso más alla el concepto de una clase abstracta en la que vimos que **una clase abstracta es una clase que no se puede instanciar (crear un objeto de esa clase) pero si se pueden definir atributos e implementar métodos en ella para que sus clases hijas los puedan utilizar**. Pues bien **una Interface es una clase abstracta pura en la que todos sus métodos son abstractos y por tanto no se pueden implementar en la clase Interface**. Mucho podreis pensar para que vale una clase abstracta pura en la que no se permiten implementar métodos y que encima en las clases hijas de esa interface tengan que tener "si o si" implementados estos métodos; pues bien, **las Interfaces sirven para establecer la forma que debe de tener una clase**. Un ejemplo que hay en Java sobre Interace es la [Interface Map](https://jarroba.com/map-en-java-con-ejemplos/). En la entrada "[Map en Java, con ejemplos](https://jarroba.com/map-en-java-con-ejemplos/" \t "_blank)", vimos que habia diferentes tipos de Map; en concreto los *HashMap****,****TreeMap y LinkedHashMap*, lo que quiere decir que todas las clases Map deben de tener implementadas a su manera los mismo métodos como el "put()", "get()", "remove()", etc. y asi lo vimos en esta entrada en la que un HashMap inserta los datos de manera diferente a un TreeMap y al LinkedHashMap. Por tanto vemos como se ha establecido una forma común que deben de respetar todos los Maps de Java.

Por otro lado se ha de decir que una Interface no se pueden definir atributos salvo que estos sean estaticos o constantes; es decir, "static" o "final".

Siguiendo con los ejemplos que hemos estado poniendo en las entradas de la [Herencia](https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/) y el [Polimorfismo](https://jarroba.com/polimorfismo-en-java-parte-i-con-ejemplos/), vamos a ver un ejemplo en el que simularemos el comportamiento que tendrían los diferentes integrantes de la selección española de fútbol; tanto los Futbolistas como el cuerpo técnico (Entrenadores, Masajistas, etc…). Para este ejemplo nos vamos a basar en el siguiente diagrama de clases:

[](https://jarroba.com/wp-content/uploads/2014/04/Interface_java_jarroba.jpg)

Lo primero que vemos es la clase "IntegranteSeleccionFutbol" que es una Interface en la que tiene definido cuatro métodos (es decir una clase abstracta pura). Como se ha dicho en esta clase solo se definen los métodos pero no se implementan; así que a nivel de código la clase quedaria de la siguiente manera:

public interface IntegranteSeleccionFutbol {

void concentrarse();

void viajar();

void entrenar();

void jugarPartido();

}

Lo siguiente que vemos en el diagrama de clases es la clase***"SeleccionFutbol" que es una clase abstracta que utiliza (implements) la Interface "IntegranteSeleccionFutbol"***. Al ser esta una clase abstracta no se puede instanciar y por tanto en ella no es necesario implementar los métodos de la "Interfece"; pero si será obligatorio implementarlo en sus clases hijas (Futbolista, Entrenador y Masajista). Para que veamos bien el uso de las interfaces, vamos ha hacer primero una "chapuzilla" para entender su utilización. Lo que vamos ha hacer es "*no implementar los métodos de la interface en la clase abstracta SeleccionFutbol*" con el fin de que veamos que en las clases hijas nos van a exigir que implementemos los métodos de la interface. Para ello la clase abstracta quedaría de la siguiente forma (sin ningún método implementado):

public abstract class SeleccionFutbol implements IntegranteSeleccionFutbol {

protected int id;

protected String nombre;

protected String apellidos;

protected int edad;

public SeleccionFutbol() {

}

public SeleccionFutbol(int id, String nombre, String apellidos, int edad) {

this.id = id;

this.nombre = nombre;

this.apellidos = apellidos;

this.edad = edad;

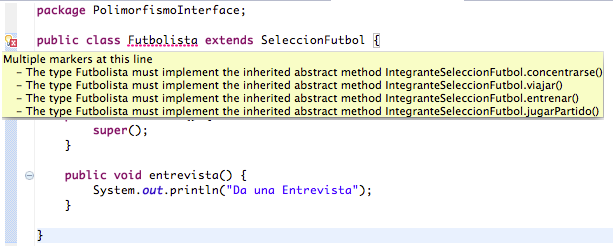
}

*// getter y setter*

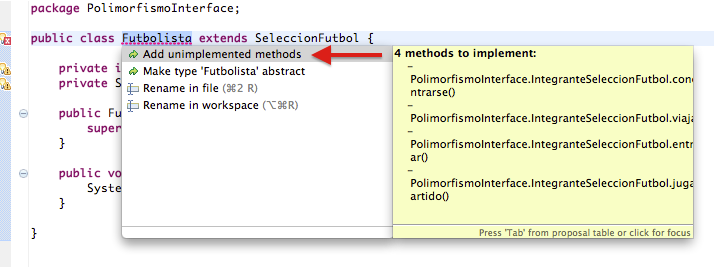
}

En este punto vemos una nueva palabra reservada como es la palabra "***implements***" que quiere decir que la clase "SeleccionFutbol" debe ser una clase que adopte la forma que tiene la Interface "IntegranteSeleccionFutbol"; es decir, que debe tener implementados los métodos de la Interface. En este caso al ser una clase abstracta las podemos implementar ahi (si queremos) o en las clases hijas (obligatoriamente sino las implementamos en la clase padre), por eso el ejemplo que ponemos es para ver como el compilador nos exigirá que esos métodos estén implementados en las clases hijas.

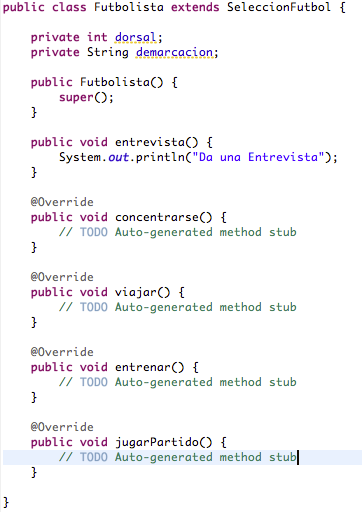
Por tanto sino implementamos los métodos en la clase padre ("SeleccionFutbol") y nos vamos a cualquiera de las clases hijas (por ejemplo la de futbolista) vemos como nos exigen que implementemos esos métodos:

[](https://jarroba.com/wp-content/uploads/2014/04/Polimorfismo_Interface_jarroba_2.png)

Vemos en la clase futbolista (en el IDE de Eclipse) que nos da un error y como solución nos dice que tenemos que implementar los métodos "concentrarse()", "viajar()", etc. Si en Eclipse seleccionamos una de las opciones para solucionar el error, vemos que nos da la opción de "*Añadir los métodos no implementados*" (add unimplemented methods):

[](https://jarroba.com/wp-content/uploads/2014/04/Polimorfismo_Interface_jarroba_3.png)

Si seleccionamos esa opción vemos como en Eclipse nos implementa los métodos de la Interface de forma automática:

[](https://jarroba.com/wp-content/uploads/2014/04/Polimorfismo_Interface_jarroba_4.png)

Como ya hemos visto que nos exigen implementar los métodos en la Interface en las clases hijas, ahora vamos ha hacer las cosas bien y vamos a implementar los métodos de la Interface en la clase padre ("SeleccionFutbol") para que solamente haya que implementar los métodos que queramos especializarlos en las clases hijas. Por tanto la clase "SeleccionFutbol" bien implementada quedaria de la siguiente forma:

public abstract class SeleccionFutbol implements IntegranteSeleccionFutbol {

protected int id;

protected String nombre;

protected String apellidos;

protected int edad;

*// Constructor, getter y setter*

public void concentrarse() {

System.out.println("Concentrarse (Clase Padre)");

}

public void viajar() {

System.out.println("Viajar (Clase Padre)");

}

public void entrenar() {

System.out.println("Entrenar (Clase Padre)");

}

public void jugarPartido() {

System.out.println("Asiste al Partido de Fútbol (Clase Padre)");

}

}

Y en las clases hijas solo implementaremos los métodos de la clase padre que queramos redefinir, quedando las clases hijas de la siguiente forma (recordar que la etiqueta "***@Override***" significa que ese método esta siendo redefinido)::

public class Futbolista extends SeleccionFutbol {

private int dorsal;

private String demarcacion;

*// Constructor, getter y setter*

@Override

public void entrenar() {

System.out.println("Realiza un entrenamiento (Clase Futbolista)");

}

@Override

public void jugarPartido() {

System.out.println("Juega un Partido (Clase Futbolista)");

}

public void entrevista() {

System.out.println("Da una Entrevista");

}

}

public class Entrenador extends SeleccionFutbol {

private int idFederacion;

*// Constructor, getter y setter*

@Override

public void entrenar() {

System.out.println("Dirige un entrenamiento (Clase Entrenador)");

}

@Override

public void jugarPartido() {

System.out.println("Dirige un Partido (Clase Entrenador)");

}

public void planificarEntrenamiento() {

System.out.println("Planificar un Entrenamiento");

}

}

public class Masajista extends SeleccionFutbol {

private String titulacion;

private int aniosExperiencia;

*// Constructor, getter y setter*

@Override

public void entrenar() {

System.out.println("Da asistencia en el entrenamiento (Clase Masajista)");

}

public void darMasaje() {

System.out.println("Da un Masaje");

}

}

Llegados a este punto ya tenemos implementada las misma funcionalidad que hicimos en la entrada de  [Polimorfismo en Java (Parte I), con ejemplos](https://jarroba.com/polimorfismo-en-java-parte-i-con-ejemplos/) pero diseñada e implementada de otra forma, así que vamos a poner el código que escribimos para esa entrada y el resultado de la ejecución del mismo para que podais ver el resultado:

public class Main {

*// ArrayList de objetos SeleccionFutbol. Idenpendientemente de la clase hija a la que pertenezca el objeto*

public static ArrayList integrantes = new ArrayList();

public static void main(String[] args) {

SeleccionFutbol delBosque = new Entrenador(1, "Vicente", "Del Bosque", 60, 28489);

SeleccionFutbol iniesta = new Futbolista(2, "Andres", "Iniesta", 29, 6, "Interior Derecho");

SeleccionFutbol raulMartinez = new Masajista(3, "Raúl", "Martinez", 41, "Licenciado en Fisioterapia", 18);

integrantes.add(delBosque);

integrantes.add(iniesta);

integrantes.add(raulMartinez);

*// CONCENTRACION*

System.out.println("Todos los integrantes comienzan una concentracion. (Todos ejecutan el mismo método)");

for (SeleccionFutbol integrante : integrantes) {

System.out.print(integrante.getNombre() + " " + integrante.getApellidos() + " -> ");

integrante.concentrarse();

}

*// VIAJE*

System.out.println("nTodos los integrantes viajan para jugar un partido. (Todos ejecutan el mismo método)");

for (SeleccionFutbol integrante : integrantes) {

System.out.print(integrante.getNombre() + " " + integrante.getApellidos() + " -> ");

integrante.viajar();

}

*// ENTRENAMIENTO*

System.out.println("nEntrenamiento: Todos los integrantes tienen su función en un entrenamiento (Especialización)");

for (SeleccionFutbol integrante : integrantes) {

System.out.print(integrante.getNombre() + " " + integrante.getApellidos() + " -> ");

integrante.entrenar();

}

*// PARTIDO DE FUTBOL*

System.out.println("nPartido de Fútbol: Todos los integrantes tienen su función en un partido (Especialización)");

for (SeleccionFutbol integrante : integrantes) {

System.out.print(integrante.getNombre() + " " + integrante.getApellidos() + " -> ");

integrante.jugarPartido();

}

*// PLANIFICAR ENTRENAMIENTO*

System.out.println("nPlanificar Entrenamiento: Solo el entrenador tiene el método para planificar un entrenamiento:");

System.out.print(delBosque.getNombre() + " " + delBosque.getApellidos() + " -> ");

((Entrenador) delBosque).planificarEntrenamiento();

*// ENTREVISTA*

System.out.println("nEntrevista: Solo el futbolista tiene el método para dar una entrevista:");

System.out.print(iniesta.getNombre() + " " + iniesta.getApellidos() + " -> ");

((Futbolista) iniesta).entrevista();

*// MASAJE*

System.out.println("nMasaje: Solo el masajista tiene el método para dar un masaje:");

System.out.print(raulMartinez.getNombre() + " " + raulMartinez.getApellidos() + " -> ");

((Masajista) raulMartinez).darMasaje();

}

}

Como resultado a la ejecución de este programa tenemos lo siguiente:

Todos los integrantes comienzan una concentracion. (Todos ejecutan el mismo método)

Vicente Del Bosque -> Concentrarse (Clase Padre)

Andres Iniesta -> Concentrarse (Clase Padre)

Raúl Martinez -> Concentrarse (Clase Padre)

Todos los integrantes viajan para jugar un partido. (Todos ejecutan el mismo método)

Vicente Del Bosque -> Viajar (Clase Padre)

Andres Iniesta -> Viajar (Clase Padre)

Raúl Martinez -> Viajar (Clase Padre)

Entrenamiento: Todos los integrantes tienen su función en un entrenamiento (Especialización)

Vicente Del Bosque -> Dirige un entrenamiento (Clase Entrenador)

Andres Iniesta -> Realiza un entrenamiento (Clase Futbolista)

Raúl Martinez -> Da asistencia en el entrenamiento (Clase Masajista)

Partido de Fútbol: Todos los integrantes tienen su función en un partido (Especialización)

Vicente Del Bosque -> Dirige un Partido (Clase Entrenador)

Andres Iniesta -> Juega un Partido (Clase Futbolista)

Raúl Martinez -> Asiste al Partido de Fútbol (Clase Padre)

Planificar Entrenamiento: Solo el entrenador tiene el método para planificar un entrenamiento:

Vicente Del Bosque -> Planificar un Entrenamiento

Entrevista: Solo el futbolista tiene el método para dar una entrevista:

Andres Iniesta -> Da una Entrevista

Masaje: Solo el masajista tiene el método para dar un masaje:

Raúl Martinez -> Da un Masaje

CONCLUSIONES Y ACLARACIONES:

Como hemos visto el concepto de la Interface va un paso más alla en lo referente al concepto de la clase abstracta. Es una muy buena práctica de diseño la utilización de clases Interface para definir clases que tengan una misma forma, aunque en ellas realicen comportamientos distintos.

Al igual que comentamos en las conclusiones de la entrada del polimorfismo, es muy probable que los que empeceis con la POO no le veais mucho sentido al tema de las clases Interface, así que no os preocupeis si eso es asi porque estos conceptos se consolidan a base de experiencia y de ir adquiriendo más conocimientos de arquitectura y diseño del software.