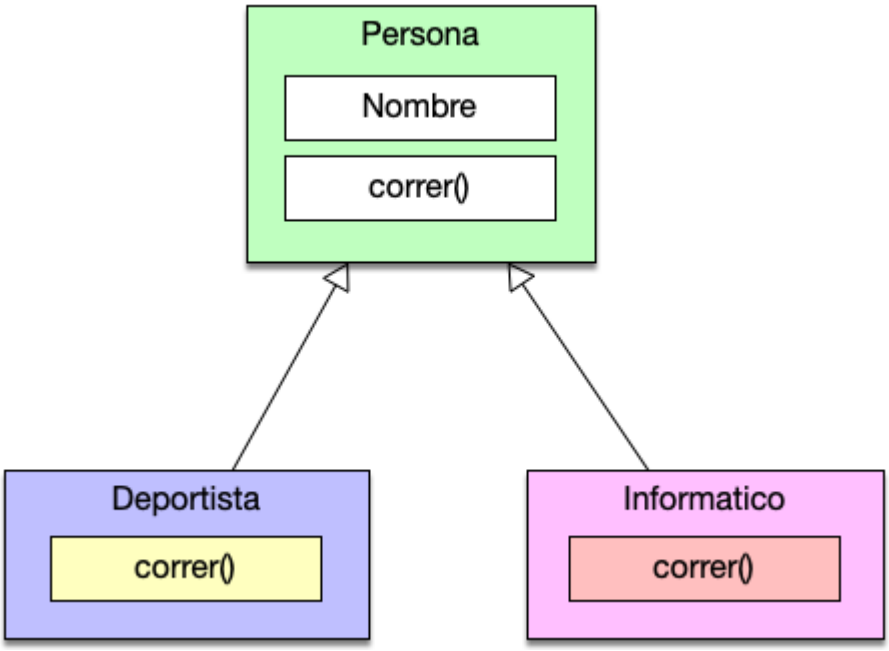
Los conceptos de Java Polimorfismo y herencia están intimamente relacionados y siempre el polimorfismo ha sido muy difícil de entender para la mayor parte de los programadores. ¿Para que sirve el Polimorfismo? .Vamos a explicarlo , partiendo de un ejemplo en el cual tenemos las siguientes clases Persona, Deportista e Informatico. Estas clases están evidentemente organizadas en una jerarquía de herencia.



El método correr de la clase Persona es un método abstracto y no tiene implementación . Por el contrario los métodos de la clases hijas tienen sobrecargado el método correr. El deportista correrá a 7 hm/hora y el informatico a 2km/h . Vamos a ver el código:

```
package com.arquitecturajava;
```

```
public abstract class Persona {

	private String nombre;

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public Persona(String nombre) {
		super();
		this.nombre = nombre;
	}
	public abstract int  correr() ;
}
```

```
package com.arquitecturajava;

public class Deportista extends Persona {

	public Deportista(String nombre) {
		super(nombre);
		// TODO Auto-generated constructor stub
	}

	@Override
	public int correr() {
		// TODO Auto-generated method stub
		return 7;
```

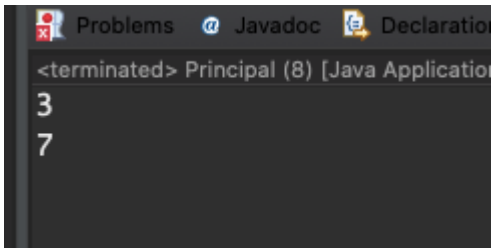
```
	}

}
package com.arquitecturajava;

public class Ingeniero extends Persona{

	public Ingeniero(String nombre) {
		super(nombre);
		// TODO Auto-generated constructor stub
	}

	@Override
	public int correr() {
		// TODO Auto-generated method stub
		return 3;
	}

}
```

Si construimos un programa principal veremos cómo el método correr se ejecuta de forma polimórfica en cada una de las clases . La clase persona lo tiene como método abstracto y cada clase hija lo sobre escribe

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {
		Ingeniero i= new Ingeniero("pedro");
		Deportista d= new Deportista("gema");
		System.out.println(i.correr());
```

```
                System.out.println(d.correr());
        }

}
```

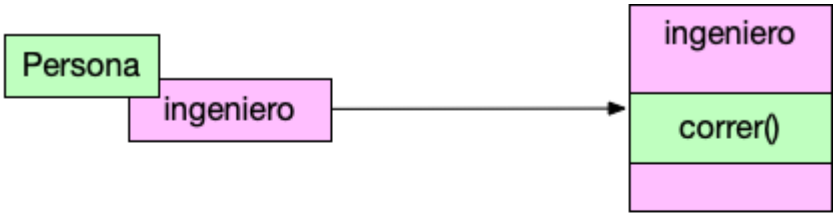
El resultado se puede ver en la consola:

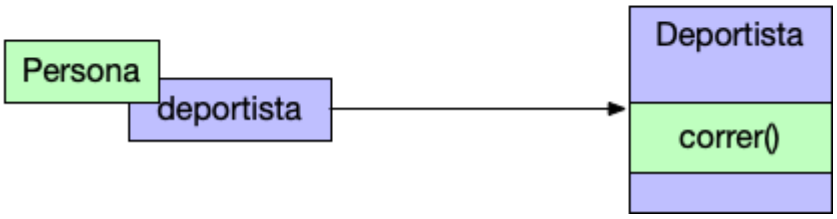


Ahora bien este mismo código se puede escribir de la siguiente manera:

```
package com.arquitecturajava;

public class Principal2 {

        public static void main(String[] args) {
                Persona i= new Ingeniero("pedro");
                Persona d= new Deportista("gema");
                System.out.println(i.correr());
                System.out.println(d.correr());
        }

}
```

Cuando la gente empieza a programar en Java , este código puede sonarle raro ya que estamos apuntando con una variable de tipo Persona a un Ingeniero y a un Deportista . Esto no es problemático ya que ambas son personas y ambas sobreescriben el método correr





Java Polimorfismo y simplificación

Muchas veces a las programadores junior les resulta difícil entender porque esta solución es más flexible que la anterior y el uso de Polimorfismo en Java genera código más sencillo . La respuesta a esto viene si por ejemplo tenemos una lista de Deportistas e Ingenieros y queremos obtener la media de velocidad de todos ellos.

```
import java.util.Arrays;
import java.util.List;
import java.util.OptionalDouble;

public class Principal3 {

        public static void main(String[] args) {

                Persona i1 = new Ingeniero("pedro");
                Persona d1 = new Deportista("gema");
                Persona i2 = new Ingeniero("angel");
                Persona i3 = new Ingeniero("antonio");
                Persona i4 = new Ingeniero("maria");
                Persona d5 = new Deportista("david");
```

```
                List<Persona> lista = Arrays.asList(i1, d1, i2, i3,
i4, i4, d5);
                OptionalDouble resultado =
calcularMediaVelocidad(lista);
                if (resultado.isPresent()) {

                        System.out.println(resultado.getAsDouble());
                }
        }

        public static OptionalDouble
calcularMediaVelocidad(List<Persona> lista) {

                return
lista.stream().mapToDouble(Persona::correr).average();
        }
}
```

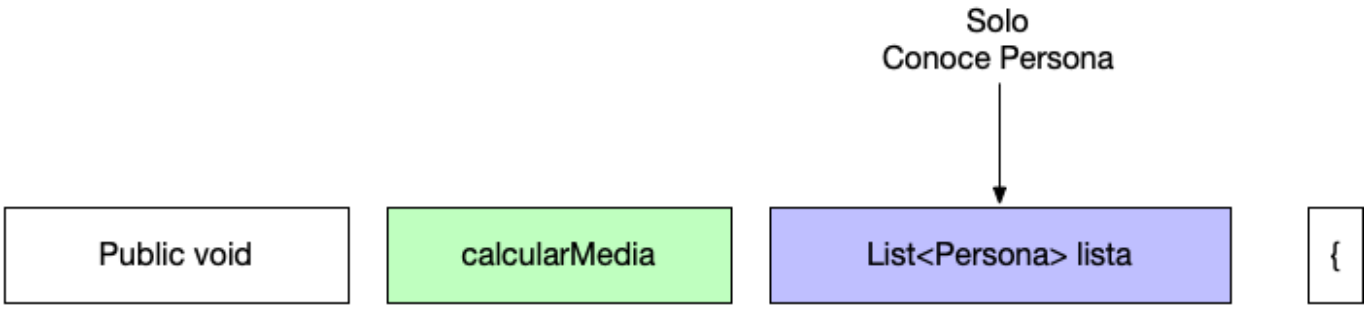
En este código acabamos de generar un método calcularMedia que se encarga de calcularnos la velocidad media de cada las personas que están en la lista . Eso sí hará un calculo polimórfico ya que si nos fijamos cada tipo de Persona devolvera el valor de velocidad que le corresponde.

Oferta Cursos 70% Descuento

- Programación Orientada a Objeto
- Java APIs
- Desarrollo Web Java
- Java 8 Stream y Lambdas

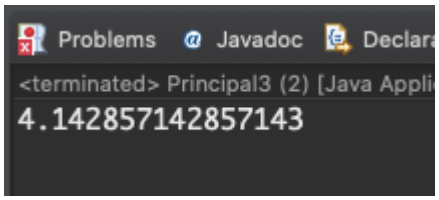
La clave se encuentra en darnos cuenta que el polimorfismo nos ayuda a simplificar el código y reducir el numero de clases que los programadores deben conocer ya que en este caso el desarrollador encargado del método que calcula la media no le hace falta conocer

todos los tipos de clases existentes es suficiente con conocer la clase padre Persona.



}

El resultado de calcular la media con programación funcional se mostrará en la consola:



Conclusiones

El concepto de Java Polimorfismo nos ayuda a la hora de generar flexibilidad en el código pero sobre todo también a la hora de simplificar el número de conceptos que un programador debe manejar.

Otros artículos relacionados

- Java Herencia
- Java Herencia y limitaciones
- Interface y el concepto de simplicidad
- Curso Programación orientada a objeto
- Java Herencia ejemplos