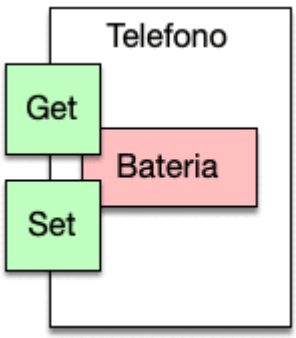
Tabla de Contenidos

- Java Encapsulamiento y reutilización
- ¿El porque de la encapsulación?
  - Java Delegación

Java Encapsulamiento y reutilización

¿Java Encapsulamiento ? . Cuando uno comienza a programar en Java el concepto de encapsulamiento es de los primeros que aparece y hace referencia a limitar el acceso a las variables de nuestras clases Java de tal forma que podamos tener un mayor control sobre ellas. Normalmente utilizando métodos set/get . La gente se queda contenta con esta respuesta y no le da más vueltas.

Eso si siempre le quedan a uno dudas de porque hay que usar el Eclipse para generar continuamente métodos set/get cuando probablemente con variables publicas lo arreglaríamos todo de una forma mucho más directa .



En programación muchas veces me he encontrado con que la “Fe” es una variable muy a

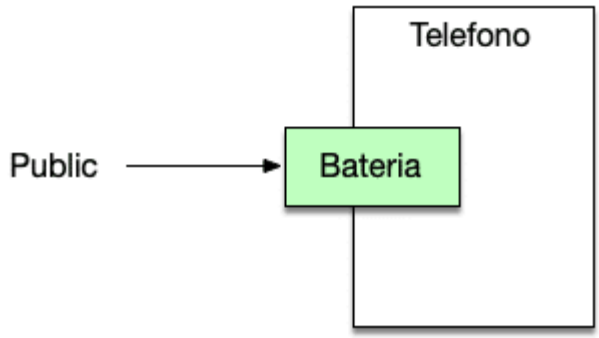
tener en cuenta y que elimina análisis. Vamos a construir un ejemplo que nos ayude a entender porque el uso de la encapsulación nos puede ayudar a mejorar la reutilización de nuestro código y su uso es más que recomendable.

Supongamos que disponemos de una clase Java que se denomina Teléfono Esta clase es sencilla y solo tiene la marca y la capacidad de batería de nuestro Teléfono.

```
package com.arquitecturajava;

public class Telefono {

	private String marca;
	private int capacidad;
	public String getMarca() {
		return marca;
	}
	public void setMarca(String marca) {
		this.marca = marca;
	}
	public int getCapacidad() {
		return capacidad;
	}
	public void setCapacidad(int capacidad) {
		this.capacidad = capacidad;
	}
	public Telefono(String marca, int capacidad) {
		super();
		this.marca = marca;
		this.capacidad = capacidad;
	}
	public int duracionBateria() {
```

```
                if (capacidad<3000) {
                        return 16;
                }else {
                        return 24;
                }
        }
}
```

Disponemos de dos propiedades con sus métodos get/set y un constructor. ¿Realmente es necesario usar los métodos get/set? .



¿Sería suficiente con declarar los métodos públicos ? .Vamos a verlo

```
package com.arquitecturajava.ejemplo2;

public class Telefono {

        public String marca;
        public int capacidad;

        public Telefono(String marca, int capacidad) {
                super();
                this.marca = marca;
                this.capacidad = capacidad;
```

```
        }

        public int duracionBateria() {

                if (capacidad < 3000) {
                        return 16;
                } else {

                        return 24;
                }
        }
}
```

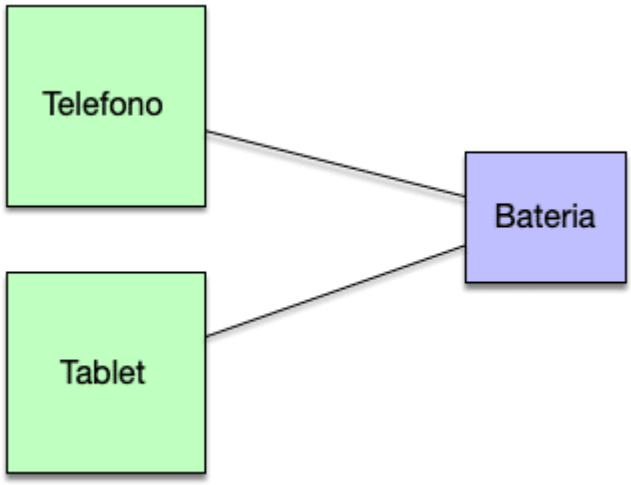
¿El porque de la encapsulación?

Es más que evidente que la clase queda simplificada y su funcionamiento es idéntico. ¿Entonces por qué tanta insistencia en usar los métodos get/set? . Imaginemonos que nosotros ahora disponemos de otra clase Tablet . Esta clase también dispondrá de una batería

```
package com.arquitecturajava;

public class Tablet {

        private String marca;
        private int capacidad;
        public String getMarca() {
                return marca;
        }
        public void setMarca(String marca) {
                this.marca = marca;
        }
```

```
	public int getCapacidad() {
		return capacidad;
	}
	public void setCapacidad(int capacidad) {
		this.capacidad = capacidad;
	}
	public Tablet(String marca, int capacidad) {
		super();
		this.marca = marca;
		this.capacidad = capacidad;
	}
	public int duracionBateria() {
		if (capacidad<3000) {
			return 16;
		}else {
			return 24;
		}
	}
}
```

Nos podemos dar cuenta que el código es muy similar y que nos vendría bien refactorizar el código de tal forma que existiera el concepto de batería.




```
package com.arquitecturajava.ejemplo3;

public class Bateria {

	private int capacidad;

	public int getCapacidad() {
		return capacidad;
	}

	public void setCapacidad(int capacidad) {
		this.capacidad = capacidad;
	}

	public Bateria(int capacidad) {
		super();
		this.capacidad = capacidad;
	}

	public int duracionBateria() {
```

```
                if (capacidad < 3000) {
                        return 16;
                } else {

                        return 24;
                }
        }
}
```

Java Delegación

De esta forma podríamos reutilizar la bateria en nuestras clases utilizando el concepto de delegación (una clase delega en otra)

```
package com.arquitecturajava.ejemplo3;

public class Telefono {

        private String marca;
        private Bateria bateria;
        public String getMarca() {
                return marca;
        }
        public void setMarca(String marca) {
                this.marca = marca;
        }
        public int getCapacidad() {
                return bateria.getCapacidad();
        }
        public void setCapacidad(int capacidad) {
                bateria.setCapacidad(capacidad);
        }
```

```
        public Telefono(String marca, int capacidad) {
                super();
                this.marca = marca;
                this.bateria= new Bateria(capacidad);
        }
        public int duracionBateria() {
                return bateria.duracionBateria();
        }
}
```

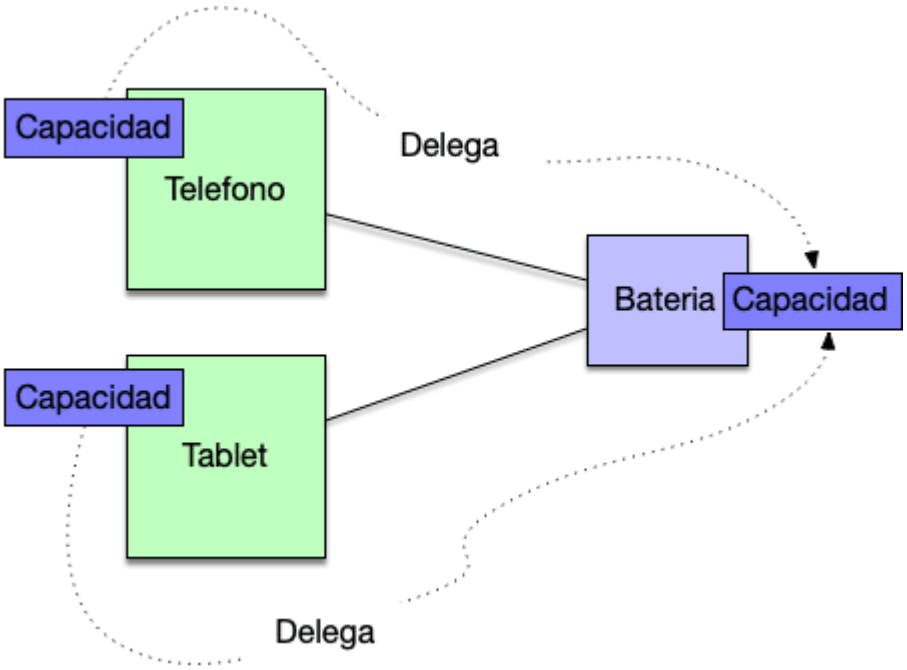
```
package com.arquitecturajava.ejemplo3;

public class Tablet {

        private String marca;

        private Bateria bateria;
        public String getMarca() {
                return marca;
        }
        public void setMarca(String marca) {
                this.marca = marca;
        }
        public int getCapacidad() {
                return bateria.getCapacidad();
        }
        public void setCapacidad(int capacidad) {
                bateria.setCapacidad(capacidad);
        }
        public Tablet(String marca, int capacidad) {
                super();
                this.marca = marca;
```

```
                this.bateria= new Bateria(capacidad);
        }
        public int duracionBateria() {
                return bateria.duracionBateria();
        }
}
```

Cursos Asociados

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

Estamos apoyándonos con el principio DRY al crear el concepto de batería y eso solo lo podemos conseguir usando el concepto de la encapsulación ya que nuestro nuevo código encapsula el concepto de batería mediante delegación.



El uso de la encapsulación nos permite construir estructuras más complejas relacionándolas

e incrementando la flexibilidad y la capacidad de reutilización de nuestro código. Algo que el simple uso de las propiedades publicas no permite.

- Java y Herencia
- Java Interfaces y simplicidad
- Java Composite Pattern y recursividad/
- Eclipse Set/Get