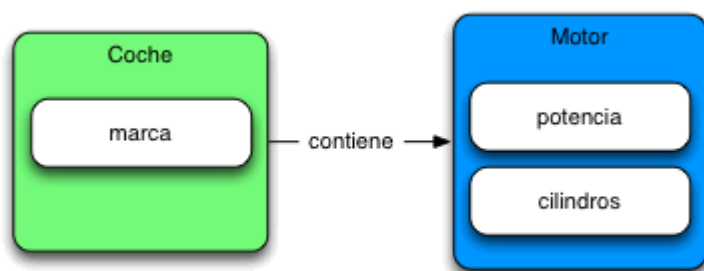


Cuando programamos en Java en muchas ocasiones nos encontramos con la necesidad de usar el concepto de delegación . Este concepto es muy habitual cuando tenemos estructuras de clase de composición. Es decir un objeto A contiene un Objeto B. Por ejemplo supongamos que tenemos el diagrama de clases de Coche y Motor.



Vamos a ver su código Java :

```
package com.arquitecturajava;
```

```
public class Coche {
```

```
    private String marca;
    private Motor motor;
```

```
    public Motor getMotor() {
        return motor;
    }
```

```
    public void setMotor(Motor motor) {
        this.motor = motor;
    }
```

```
    public String getMarca() {
        return marca;
    }
```

```
public void setMarca(String marca) {  
    this.marca = marca;  
}
```

```
package com.arquitecturajava;
```

```
public class Motor {
```

```
    private int potencia;  
    private int cilindros;
```

```
    public int getPotencia() {  
        return potencia;  
    }
```

```
    public void setPotencia(int potencia) {  
        this.potencia = potencia;  
    }
```

```
    public int getCilindros() {  
        return cilindros;  
    }
```

```
    public void setCilindros(int cilindros) {  
        this.cilindros = cilindros;  
    }  
}
```

```
package com.arquitecturajava;
```

```

public class Principal {

    public static void main(String[] args) {
        Coche c = new Coche();
        c.setMarca("Toyota");

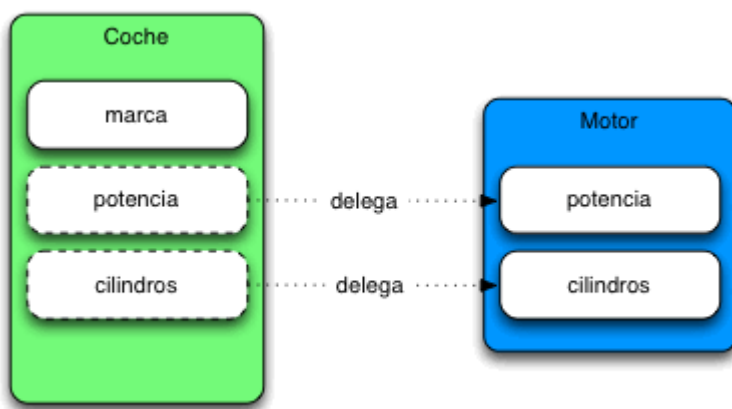
        Motor m = new Motor();
        m.setCilindros(6);
        m.setPotencia(100);
        c.setMotor(m);

        System.out.println(c.getMotor().getPotencia());
    }
}

```

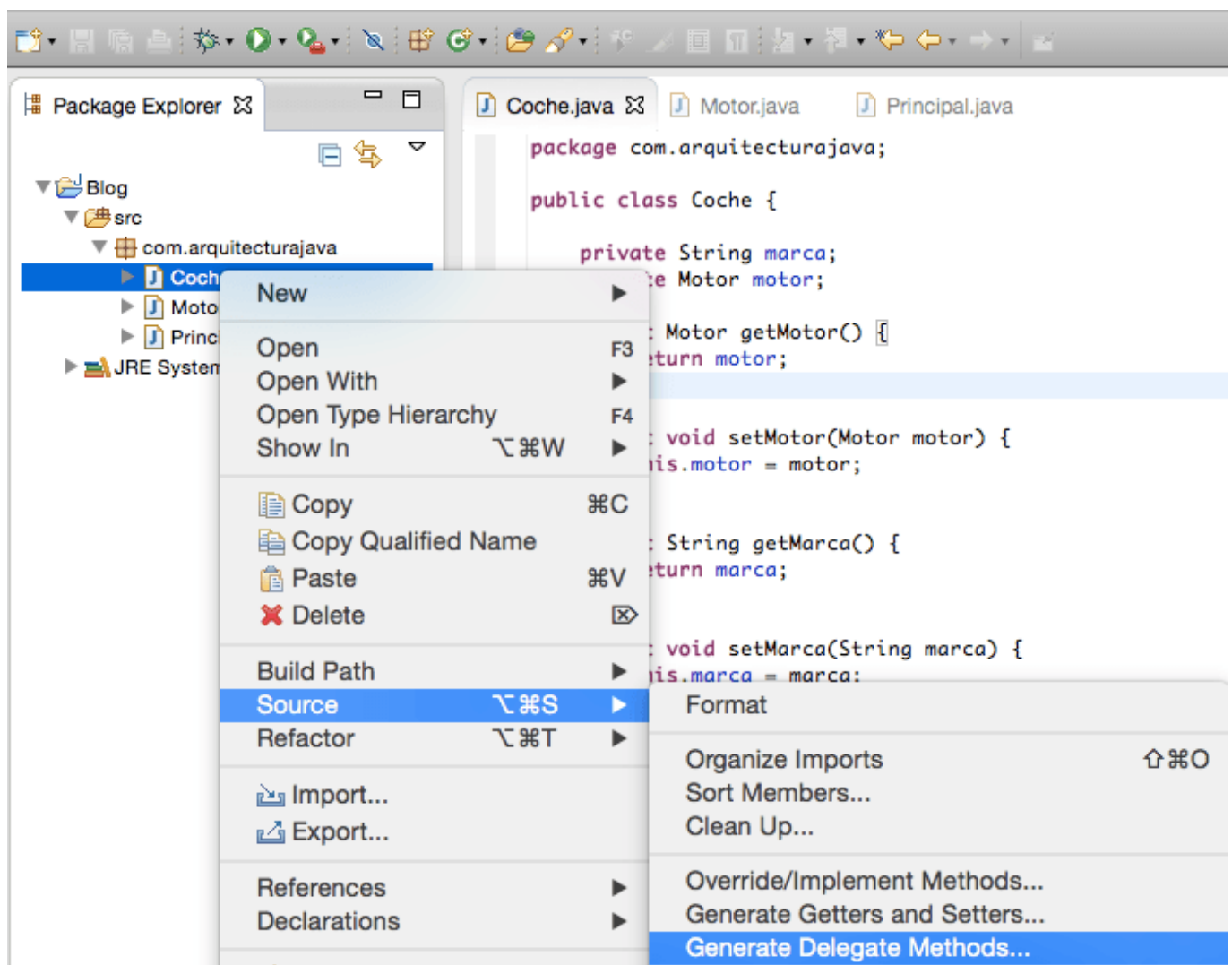
## El concepto de Delegación

Como podemos ver acceder en nuestro código a la potencia es un poco enrevesado. Podemos apoyarnos en el concepto de delegación y generar nuevos métodos a nivel de la clase Coche para que “delegen” en los de motor y todo sea más sencillo.

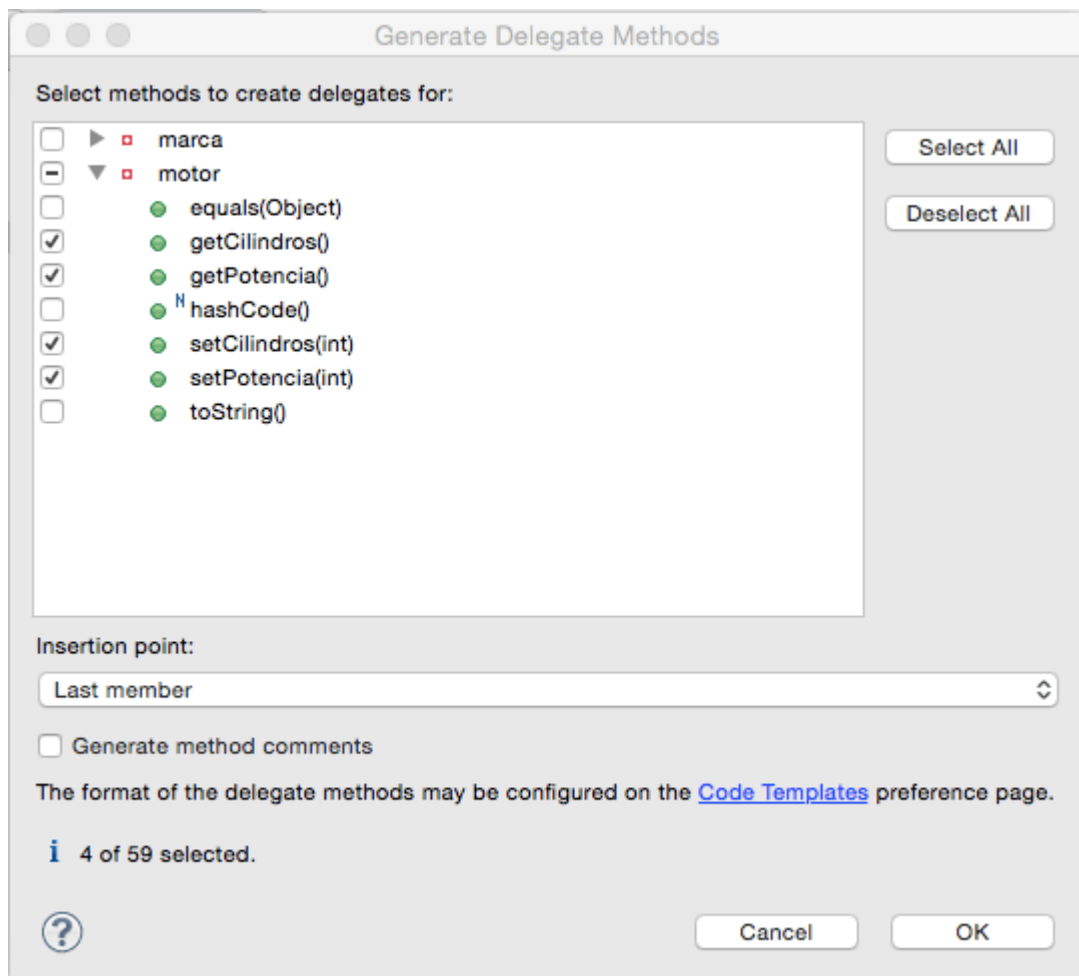


# Eclipse y Refactorings

Para construir estos nuevos métodos nos vamos a apoyar en el eclipse y en sus capacidades de refactoring. Nos posicionamos en la clase Coche y pulsamos boton derecho Source ->Generate Delegate Methods.



Una vez seleccionada esta opción Eclipse nos permitirá seleccionar que métodos queremos generar como delegados. Marcamos los de potencia y cilindros que pertenecen al Motor.



Eclipse automáticamente nos generará los nuevos métodos en la clase Coche:

```
public int getPotencia() {  
    return motor.getPotencia();  
}
```

```
public void setPotencia(int potencia) {  
    motor.setPotencia(potencia);  
}
```

```
public int getCilindros() {  
    return motor.getCilindros();  
}
```

```
}
```

```
public void setCilindros(int cilindros) {  
    motor.setCilindros(cilindros);  
}
```

Ahora podremos asignarlos directamente de una forma mucho más cómoda:

```
public class Principal {  
    public static void main(String[] args) {  
        Coche c = new Coche();  
        c.setMarca("Toyota");  
        Motor m = new Motor();  
        m.setCilindros(6);  
        m.setPotencia(100);  
        c.setMotor(m);  
  
        System.out.println(c.getPotencia());  
    }  
}
```

Hemos implementado el concepto de delegación de una forma automática. La delegación también la podríamos haber implementado a nivel de constructor si nos fuera necesario . Ese concepto permite simplificar la forma en la que trabajamos con las clases en estructuras de composición complejas.

- [Clases y Objetos Java con sus responsabilidades](#)
- [Kotlin Delegación buenas prácticas.](#)
- [Diseño Flexibilidad y Sencillez](#)
- [Thymeleaf , un motor de plantillas](#)