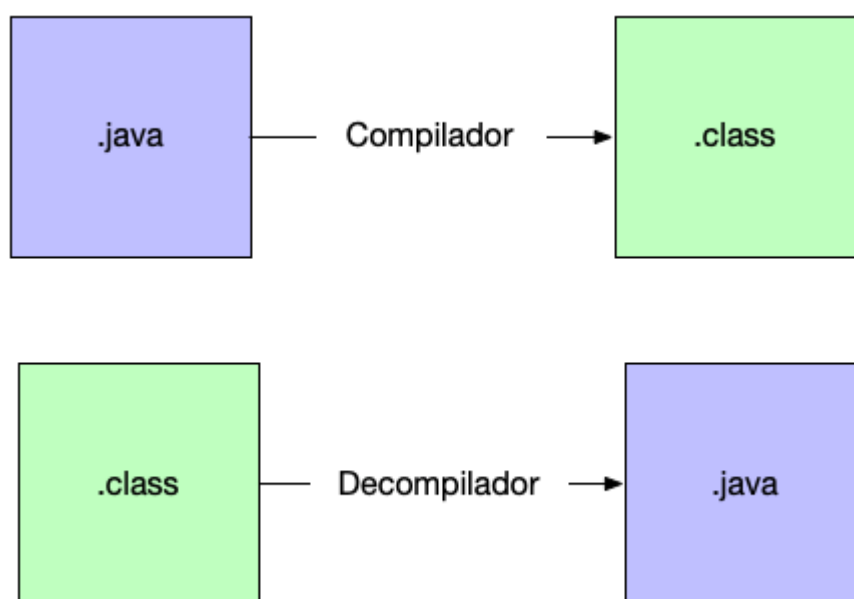


El concepto de Java Decompiler es un concepto importante . Eclipse y otras herramientas soportan decompiladores que permiten hacer la operación inversa de la compilación . Es decir en vez de pasar del código fuente al binario de Java se trata de pasar del binario a los fuentes. Son operaciones que no siempre se necesitan pero puede pasar que en algún momento tengamos algunas librerías o clases que no tengamos el código fuente y deseemos obtenerlo.



Un decompilador facilita sobremanera esto ya que nos permite pasar el fichero .class y obtener el fichero .java. En mi caso lo uso de vez en cuando para cerciorarme de que código me generan algunas librerías que modifican el ciclo de vida de compilación como puede **ser el caso de Lombok** . Vamos a ver un ejemplo sencillo con la clase Product

```
package es.arquitecturajava.proyecto1.models;
```

```
import java.util.ArrayList;
```

```
import java.util.List;
```

```
import jakarta.persistence.Entity;
```

```
import jakarta.persistence.GeneratedValue;
```

```
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotBlank;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Entity
@Table(name="products")
@Getter
@Setter
@ToString
@EqualsAndHashCode(of = "id")
@RequiredArgsConstructor
@NoArgsConstructor
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @NonNull
    private int id;
    @NonNull
    @NotBlank
```

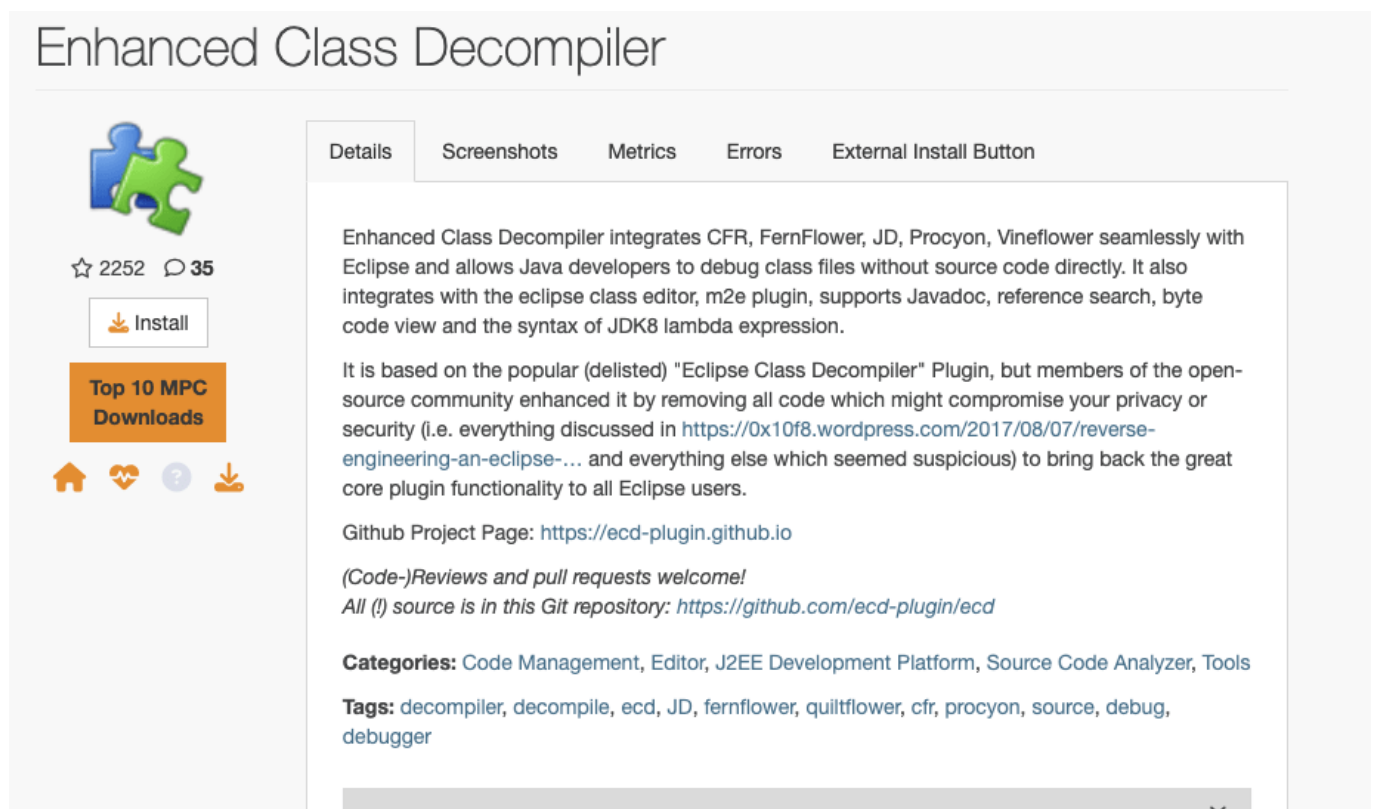
```

private String description;
@OneToMany(mappedBy="product", orphanRemoval = true)
@ToString.Exclude
private List<Price> prices= new ArrayList<Price>() ;
@ToString.Exclude
@ManyToMany (mappedBy="products")
private List<Store> stores= new ArrayList<Store>();
public Product(String description) {
    super();
    this.description = description;
}
public Product(int id) {
    super();
    this.id = id;
}
public void addPrice(Price price) {
    this.prices.add(price);
    price.setProduct(this);
}
public void removePrice(Price price) {
    this.prices.remove(price);
    price.setProduct(null);
}
public void addStore(Store store) {
    stores.add(store);
}
public void removeStore(Store store) {
    stores.remove(store);
    store.removeProduct(this);
}
}

```

Java y Anotaciones

En este caso estamos ante una clase relativamente sencilla que se relaciona con otras pero en la que hemos usado Lombok para simplificar el manejo y automatizar la creación de muchos métodos. Por lo tanto cuando esta clase se compile añadirá métodos adicionales para poderlos ver tendremos que acceder al fichero .class que se encuentra en la carpeta target de Maven y usar por ejemplo Enhanced Class Decompiler



The screenshot shows the Eclipse Marketplace page for the 'Enhanced Class Decompiler' plugin. The page has a header with the title 'Enhanced Class Decompiler' and a navigation bar with tabs: 'Details', 'Screenshots', 'Metrics', 'Errors', and 'External Install Button'. The 'Details' tab is selected. On the left sidebar, there is a puzzle icon, a star rating of 2252, a comment count of 35, an 'Install' button, a 'Top 10 MPC Downloads' badge, and a row of icons including a home, heart, question mark, and download. The main content area contains the following text:

Enhanced Class Decompiler integrates CFR, FernFlower, JD, Procyon, Vineflower seamlessly with Eclipse and allows Java developers to debug class files without source code directly. It also integrates with the eclipse class editor, m2e plugin, supports Javadoc, reference search, byte code view and the syntax of JDK8 lambda expression.

It is based on the popular (delisted) "Eclipse Class Decompiler" Plugin, but members of the open-source community enhanced it by removing all code which might compromise your privacy or security (i.e. everything discussed in <https://0x10f8.wordpress.com/2017/08/07/reverse-engineering-an-eclipse-...> and everything else which seemed suspicious) to bring back the great core plugin functionality to all Eclipse users.

Github Project Page: <https://ecd-plugin.github.io>

(Code-)Reviews and pull requests welcome!
All (!) source is in this Git repository: <https://github.com/eclipse/eclipse>

Categories: Code Management, Editor, J2EE Development Platform, Source Code Analyzer, Tools

Tags: decompiler, decompile, ecd, JD, fernflower, quiltflower, cfr, procyon, source, debug, debugger

Este plugin nos permite ver la clase decompilada en este caso si le solicitamos que muestre la informacion obtendremos algo como:

```
package es.arquitecturajava.proyecto1.models;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
```

```
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotBlank;
import java.util.ArrayList;
import java.util.List;
import lombok.Generated;
import lombok.NonNull;

@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private @NonNull int id;
    private @NotBlank @NonNull String description;
    @OneToMany(mappedBy = "product", orphanRemoval = true)
    private List<Price> prices = new ArrayList();
    @ManyToMany(mappedBy = "products")
    private List<Store> stores = new ArrayList();

    public Product(String description) {
        this.description = description;
    }

    public Product(int id) {
        this.id = id;
    }

    public void addPrice(Price price) {
```

```
        this.prices.add(price);
        price.setProduct(this);
    }

    public void removePrice(Price price) {
        this.prices.remove(price);
        price.setProduct((Product) null);
    }

    public void addStore(Store store) {
        this.stores.add(store);
    }

    public void removeStore(Store store) {
        this.stores.remove(store);
        store.removeProduct(this);
    }

    @Generated
    public @NonNull int getId() {
        return this.id;
    }

    @Generated
    public @NonNull String getDescription() {
        return this.description;
    }

    @Generated
    public List<Price> getPrices() {
        return this.prices;
    }
}
```

```

    }

    @Generated
    public List<Store> getStores() {
        return this.stores;
    }

    @Generated
    public void setId(@NonNull int id) {
        this.id = id;
    }

    @Generated
    public void setDescription(@NonNull String description) {
        if (description == null) {
            throw new NullPointerException("description is
marked non-null but is null");
        } else {
            this.description = description;
        }
    }

    @Generated
    public void setPrices(List<Price> prices) {
        this.prices = prices;
    }

    @Generated
    public void setStores(List<Store> stores) {
        this.stores = stores;
    }

```

```
@Generated
public String toString() {
    return "Product(id=" + this.getId() + ", description="
+ this.getDescription() + ")";
}
```

```
@Generated
public boolean equals(Object o) {
    if (o == this) {
        return true;
    } else if (!(o instanceof Product)) {
        return false;
    } else {
        Product other = (Product) o;
        if (!other.canEqual(this)) {
            return false;
        } else {
            return this.getId() == other.getId();
        }
    }
}
```

```
@Generated
protected boolean canEqual(Object other) {
    return other instanceof Product;
}
```

```
@Generated
public int hashCode() {
    int PRIME = true;
    int result = 1;
```



```

        result = result * 59 + this.getId();
        return result;
    }

    @Generated
    public Product(@NonNull int id, @NonNull String description) {
        if (description == null) {
            throw new NullPointerException("description is
marked non-null but is null");
        } else {
            this.id = id;
            this.description = description;
        }
    }

    @Generated
    public Product() {
    }
}

```

Esto nos permitirá entender mejor como funciona el código que se ha generado a través de anotaciones. Tener un decompilador a mano siempre es algo muy práctico.

- [Java Stream if else y Optionals](#)
- [Curso Experto Arquitectura y Productividad](#)
- [El modelo vista controlador y sus responsabilidades](#)
- [Angular ngIf, opciones y componentes](#)
- [Kotlin Ranges y sentencia de Control](#)