Práctica: Clases y relaciones

Dr. José Luis Quiroz Fabián

1. Relaciones entre clases

En la programación orientada a objetos (POO), las relaciones entre clases son fundamentales para modelar cómo diferentes entidades interactúan o se asocian entre sí. Estas relaciones son esenciales para la creación de diseños de software robustos y mantenibles. Las principales relaciones entre clases en POO son:

- **Asociación**: Esta relación implica que una clase conoce a otra y puede interactuar con ella. Las asociaciones pueden ser unidireccionales (solo una clase conoce a la otra) o bidireccionales (ambas clases se conocen mutuamente).
- **Agregación**: Es un tipo especial de asociación que representa una relación "tiene-un.º "parte-de.entre dos clases, donde una clase es un contenedor y la otra es un contenido. Sin embargo, el contenido puede existir independientemente del contenedor.
- **Composición**: Es una forma más fuerte de agregación donde la clase contenida no puede existir independientemente de la clase contenedora. Cuando la clase contenedora es destruida, también lo son sus contenidos. Por ejemplo, una clase Motor puede ser parte de una clase Automóvil, y sin el automóvil, el motor no tiene un contexto significativo.
- **Herencia**: Esta relación se da cuando una clase (subclase) hereda características (atributos y métodos) de otra clase (superclase). La herencia permite la reutilización de código y la creación de jerarquías de clases. Por ejemplo, puedes tener una superclase Vehículo con subclases Automóvil y Bicicleta, donde Automóvil y Bicicleta heredan propiedades y comportamientos comunes de Vehículo.
- **Dependencia**: Se produce cuando una clase depende de otra para su funcionamiento, pero esta relación es menos fuerte que la asociación. En la dependencia, un cambio en una clase puede afectar a la otra. Por ejemplo, si una clase Calculadora utiliza un objeto Operacion temporalmente dentro de un método, entonces Calculadora depende de Operacion.
- **Realización**: Se da cuando una clase implementa una interfaz. La clase que implementa la interfaz se compromete a implementar todos los métodos definidos en la interfaz. Por ejemplo, una clase VehículoEléctrico podría implementar una interfaz Recargable.

1.1. Asociación

La Figura 1 muestra un ejemplo de una relación de asociación entre las clases Automovil y TarjetaCirculacion. La linea dado que no tiene sentido (flecha) se dice que representa una relación bidireccional. Además, los números al extremos de la linea representa la multiplicidad o cardinalidad de la relación, es decir, cuántos objetos de un tipo están relacionados con cuántos del otro tipo. La implementación de esta relación en Java se muestra en el Código 1.

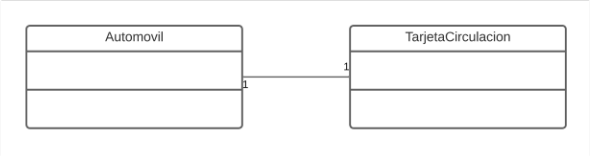

Figura 1: Relación de asociación.

```java
public class TarjetaCirculacion {

    Automovil automovil;


}

public class Automovil {

    TarjetaCirculacion tarjetaCirculacion;

}
```

Código 1: Implementación de la relación de asociación en Java

1.2. Agregación

La Figura 2 muestra un ejemplo de una relación de agregación entre las clases Automovil y Motor. El rombo vacío representa la relación del contenedor (Automovil) y su contenido (Motor). La implementación de esta relación en Java se muestra en el Código 2. Es importante resaltar en este ejemplo que si se destruye el Automovil el Motor no se destruye (se puede reutilizar).

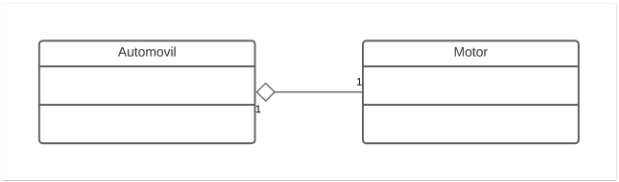

Figura 2: Relación de agregación.

```java
public class Motor {
    // Atributos y métodos del motor
}
```

```
4
5  public class Automovil {
6      Motor motor;
7
8      Automovil(Motor motor) {
9          this.motor = motor;
10     }
11 }
12
13 // Uso
14 Motor motor = new Motor();
15 Automovil coche = new Automovil(motor);
```

Código 2: Implementación de la relación de agregación en Java

1.3. Composición

La Figura 3 muestra un ejemplo de una relación de composición entre las clases Automovil y Motor. El rombo lleno representa la relación del contenedor (Automovil) y su contenido (Motor). La implementación de esta relación en Java se muestra en el Código 2. Es importante resaltar en este ejemplo que si se destruye el Automovil el Motor también (no se puede reutilizar).

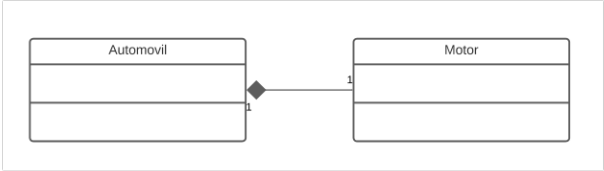

Figura 3: Relación de composición.

```
1  public class Motor {
2      // Atributos y métodos del motor
3  }
4
5  public class Automovil {
6      Motor motor;
7
8      Automovil() {
9          this.motor = new Motor();
10     }
11 }
12
13 // Uso
14 Automovil coche = new Automovil();
```

Código 3: Implementación de la relación de composición en Java

1.4. Herencia

La Figura 4 muestra un ejemplo de una relación de herencia entre las clases TransporteTerrestre y Automovil. El flecha representa la relación de la clase padre (TransporteTerrestre) y su clase hija (Automovil). La implementación de esta relación en Java se muestra en el Código 4. .

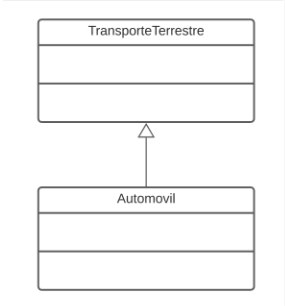

Figura 4: Relación de herencia.

```java
public class TransporteTerrestre{

}
public class Automovil  extends TransporteTerrestre{

}
// Uso
Automovil coche = new Automovil();
```

Código 4: Implementación de la relación de composición en Java

1.5. Dependencia

La Figura 5 muestra un ejemplo de una relación de dependencia entre las clase Automovil y la clase Conductor. La implementación de esta relación en Java se muestra en el Código 5. .

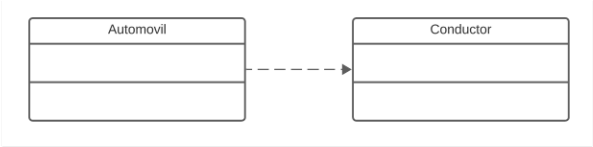

Figura 5: Relación de dependencia.

```java
public class Conductor{

}
public class Automovil {
    public void asignarConductor(Conductor conductor) {
        // Lógica para asignar el conductor al automóvil
    }
}
// Uso
Automovil coche = new Automovil();
```

Código 5: Implementación de la relación de dependencia en Java

1.6. Realización

La Figura 6 muestra un ejemplo de una relación de realización entre las clase Vehiculo y la clase Automovil. La implementación de esta relación en Java se muestra en el Código 6.

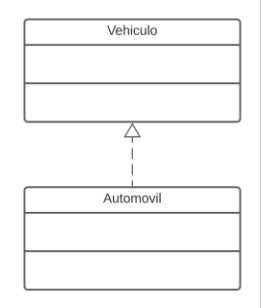

Figura 6: Relación de realización.

```java
public interface Vehiculo {
    void arrancar();
    void detener();
    void acelerar();
}
public class Automovil implements Vehiculo {

    @Override
    public void arrancar() {
        System.out.println("El automóvil está arrancando.");
    }

    @Override
    public void detener() {
        System.out.println("El automóvil se ha detenido.");
    }

    @Override
    public void acelerar() {
        System.out.println("El automóvil está acelerando.");
    }

    // Otros métodos y atributos específicos de Automovil
}
// Uso
Automovil coche = new Automovil();
```

Código 6: Implementación de la relación de realización en Java

2. Ejemplo de interfaz Comparable

Una interfaz muy útil es **Comparable**. Ésta interfaz permite establecer una relación de orden, es decir, comparar objetos y saber si uno es menor que otro, es mayor o son iguales. Usando la interfaz **Comparable** uno está obligado a implementar el método **compareTo**. Este método es el usado para comparar dos

objetos y decidir cuando uno es mayor que otro. Devuelve 1 si el objeto actual es mayor que el que pasamos como parámetro. Devuelve 0 si en la comparación se produce una relación de igualdad y devuelve -1 si es menor.

En el siguiente ejemplo se muestra como utilizar la interfaz Comparable haciendo un Libro comparable. Los libros se comparan por medio de su ISBN. Aprovechando que los libros son comparables, se ordena un arreglo donde se almacenan los libros.

```
1  import java.util.GregorianCalendar;
2
3  public class Libro implements Comparable<Libro>{
4    String autor;
5    String titulo;
6    GregorianCalendar fpublicacion;
7      String isbn;
8
9    public Libro() {
10     super();
11
12   }
13   public Libro(String autor, String titulo, GregorianCalendar
       fpublicacion) {
14     super();
15     this.autor = autor;
16     this.titulo = titulo;
17     this.fpublicacion = fpublicacion;
18   }
19   public Libro(String autor, String titulo, GregorianCalendar
       fpublicacion, String isbn) {
20     super();
21     this.autor = autor;
22     this.titulo = titulo;
23     this.fpublicacion = fpublicacion;
24         this.isbn = isbn;
25   }
26   public String getAutor() {
27     return autor;
28   }
29   public void setAutor(String autor) {
30     this.autor = autor;
31   }
32   public String getTitulo() {
33     return titulo;
34   }
35   public void setTitulo(String titulo) {
36     this.titulo = titulo;
37   }
38   public GregorianCalendar getFpublicacion() {
39     return fpublicacion;
40   }
41   public void setFpublicacion(GregorianCalendar fpublicacion) {
42     this.fpublicacion = fpublicacion;
43   }
44   @Override
45   public String toString() {
46     return "Libro [\n"
47         + "Autor= " + autor+"\n"
48         + "Titulo= " + titulo+"\n"
49                 + "ISBN= " + isbn+"\n"
50         + "Fecha= " + fpublicacion.get(GregorianCalendar.DAY_OF_MONTH)+
       "/"+
```

```
51          (fpublicacion.get(GregorianCalendar.MONTH)+1)+"/"+fpublicacion.
        get(GregorianCalendar.YEAR)+
52          "\n]";
53    }
54    public String getIsbn() {
55      return isbn;
56    }
57    public void setIsbn(String isbn) {
58      this.isbn = isbn;
59    }
60      @Override
61      public int compareTo(Libro o) {
62
63          return this.isbn.compareTo(o.getIsbn());
64      }
65
66  }
```

Código 7: Clase Libro en Java con la interfaz Comparable.

```
1  import java.util.Arrays;
2  import java.util.GregorianCalendar;
3
4  public class App {
5
6    public static void main(String[] args) {
7
8          Libro[] libros = new Libro[2];
9
10     Libro l1 = new Libro();
11
12     l1.setAutor("Erl");
13     l1.setTitulo("Service Oriented Arc...");
14     GregorianCalendar fecha1 = new GregorianCalendar(2018,11,11);
15     l1.setFpublicacion(fecha1);
16         l1.setIsbn("978");
17     System.out.println(l1);
18
19
20     GregorianCalendar fecha2 = new GregorianCalendar(2017,7,7);
21     Libro l2 = new Libro("Joyanes","Big Data",fecha2,"842-3-16-148410-0
       ");
22     System.out.println(l2);
23
24         System.out.println("Resultados de compareTo:");
25         System.out.println(l1.compareTo(l2));
26         System.out.println(l1.compareTo(l1));
27         System.out.println(l2.compareTo(l1));
28
29         libros[0]=l1;
30         libros[1]=l2;
31
32         System.out.println("Impresion del arreglo:");
33         System.out.println(Arrays.toString(libros));
34
35
36         Arrays.sort(libros);
37
38
39         System.out.println("Impresion del arreglo ordenado:");
40         System.out.println(Arrays.toString(libros));
41
```

```
42
43
44
45
46
47
48     }
49
50 }
```

Código 8: Clase principal App para ordenar libros.

3. Ejemplos relaciones entre clases

La Figura 7 muestra la relación entre una clases para un sistema de una biblioteca.

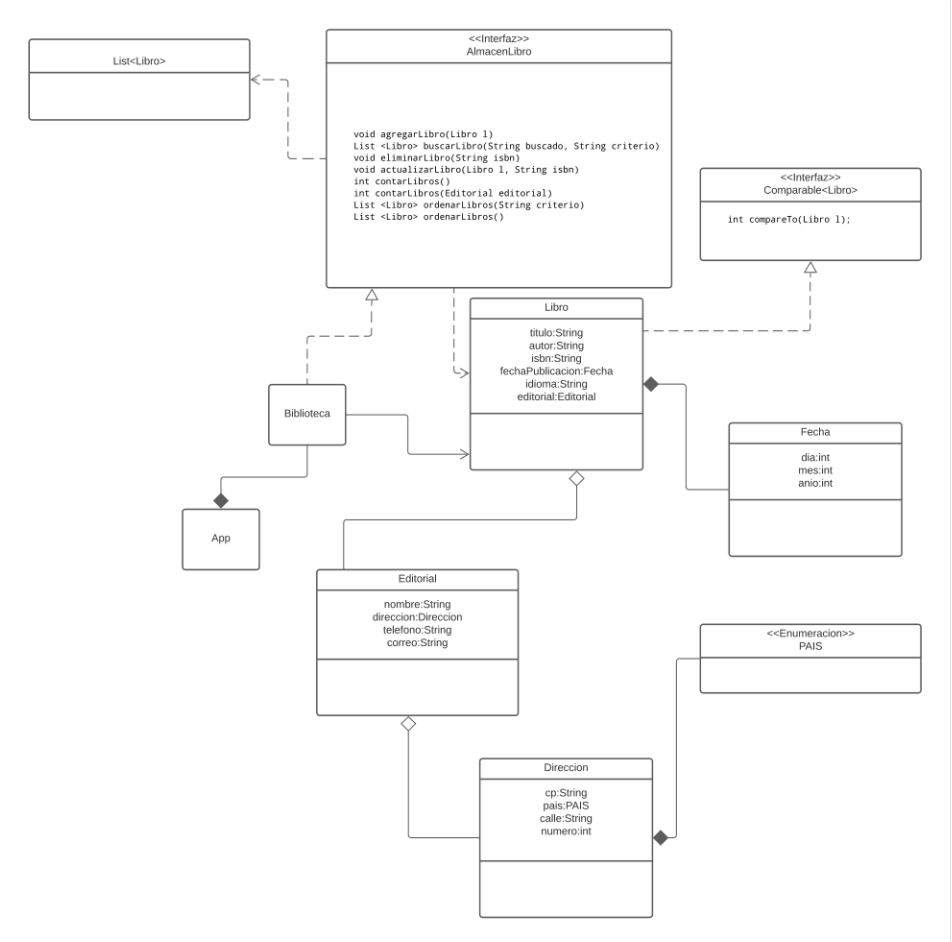

Figura 7: Relación de clases para una biblioteca.

La Figura 8 muestra la relación entre una clases para un sistema de una

agencia de automóviles.

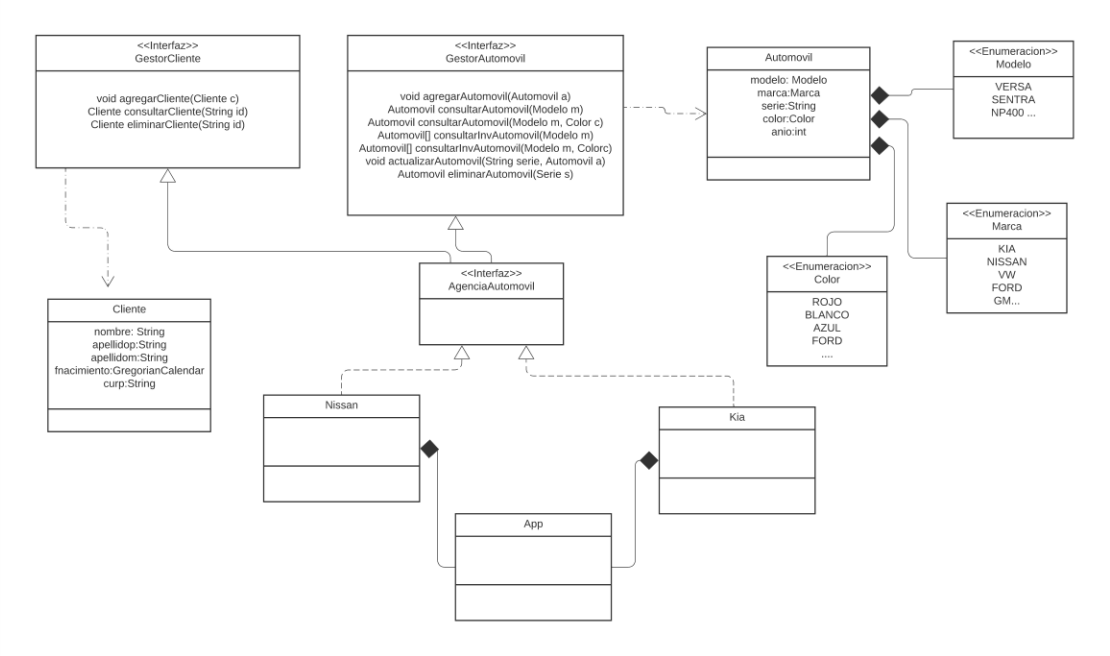

Figura 8: Relación de clases para una agencia de automóviles.

4. Creación de un proyecto

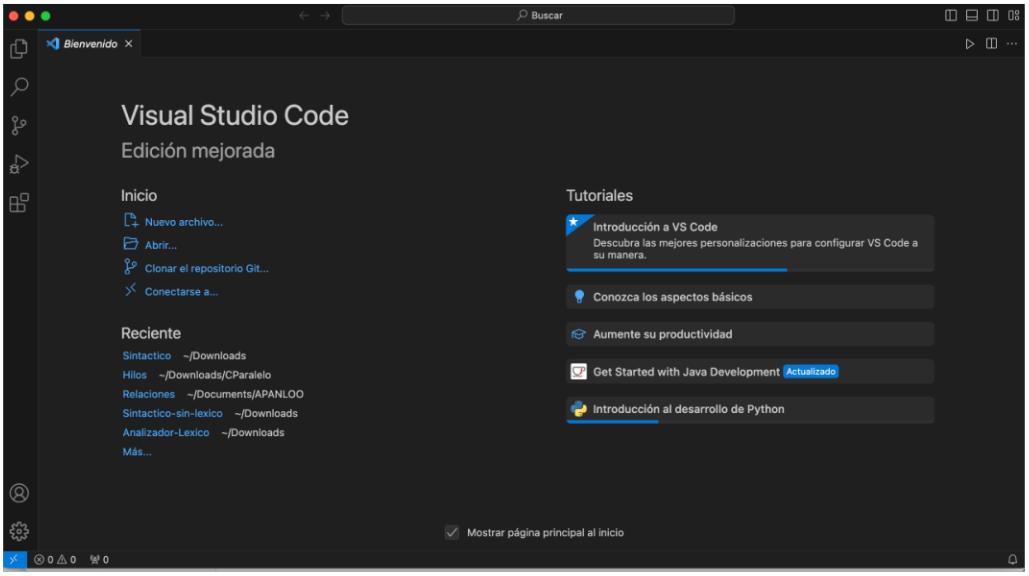

Figura 9: Interfaz de Visual Studio Code

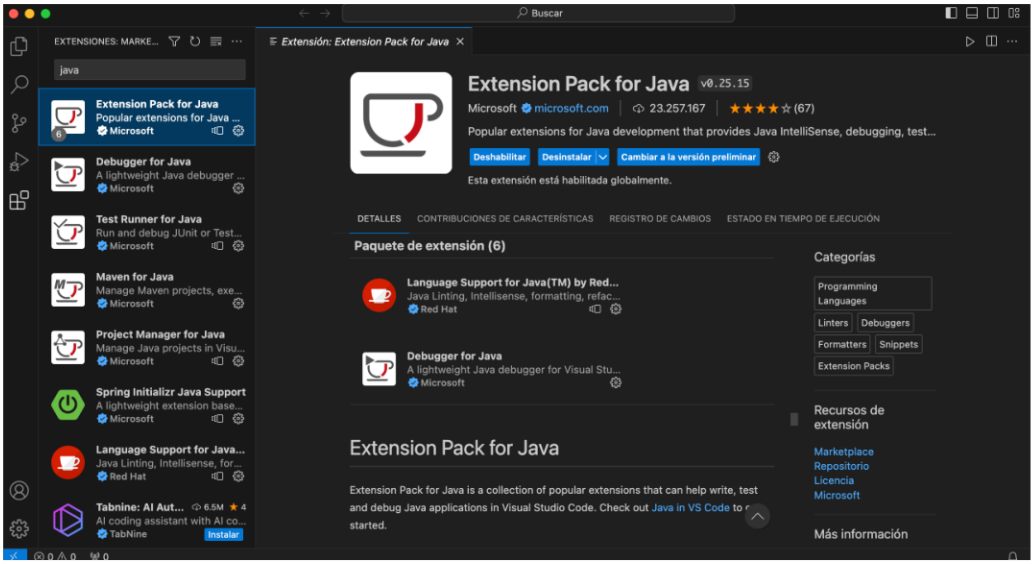

Figura 10: Instalación de extensiones en Visual Studio Code

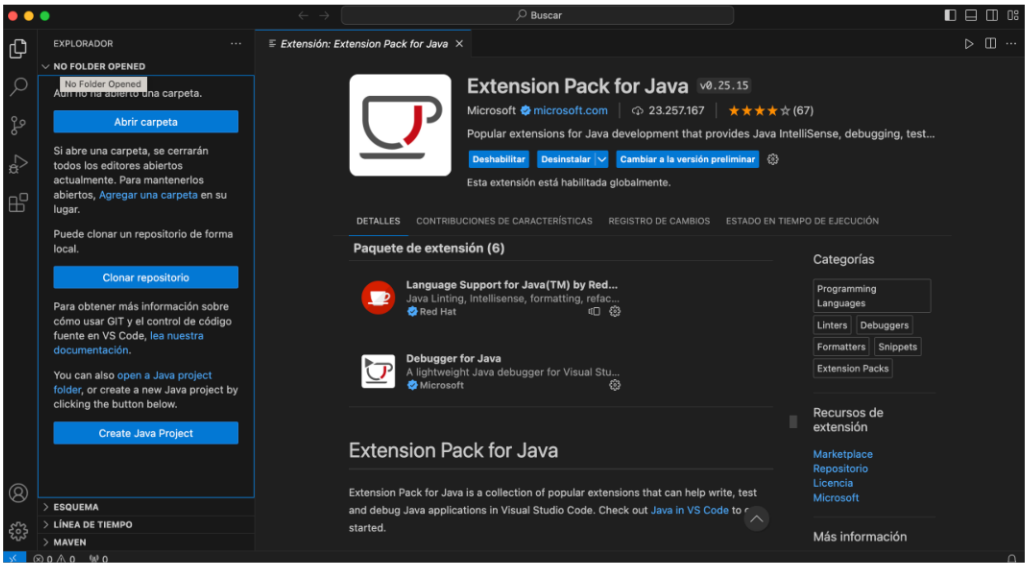

Figura 11: Creación de un proyecto Java en Visual Studio Code

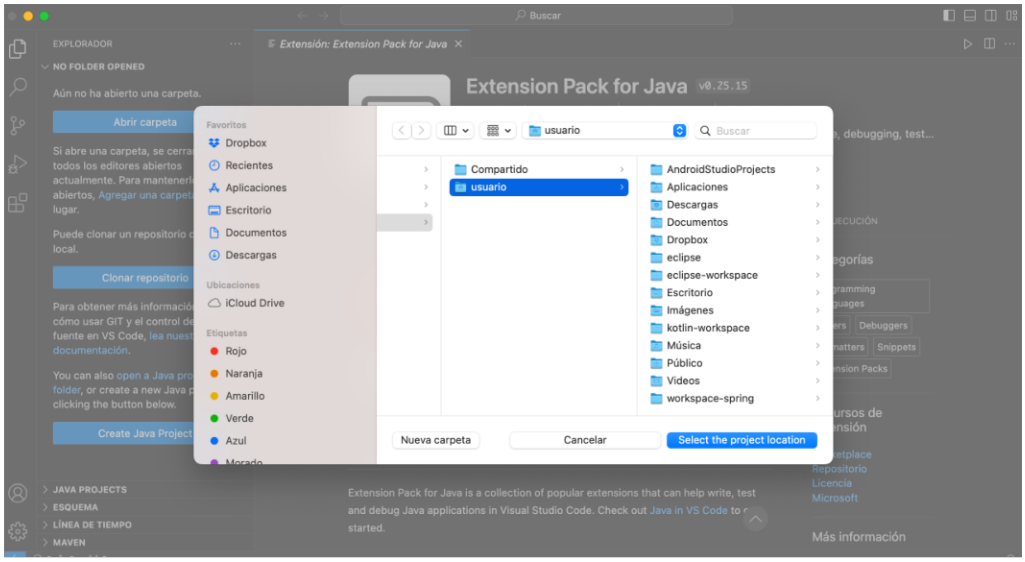

Figura 12: Selección del directorio para almacenar el proyecto en Visual Studio Code

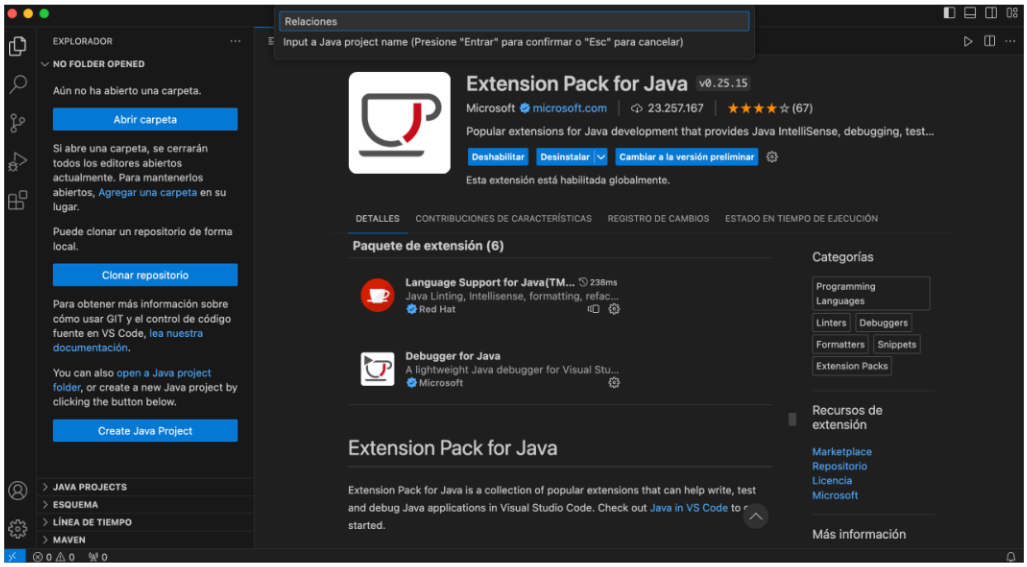

Figura 13: Definición del nombre del proyecto en Visual Studio Code

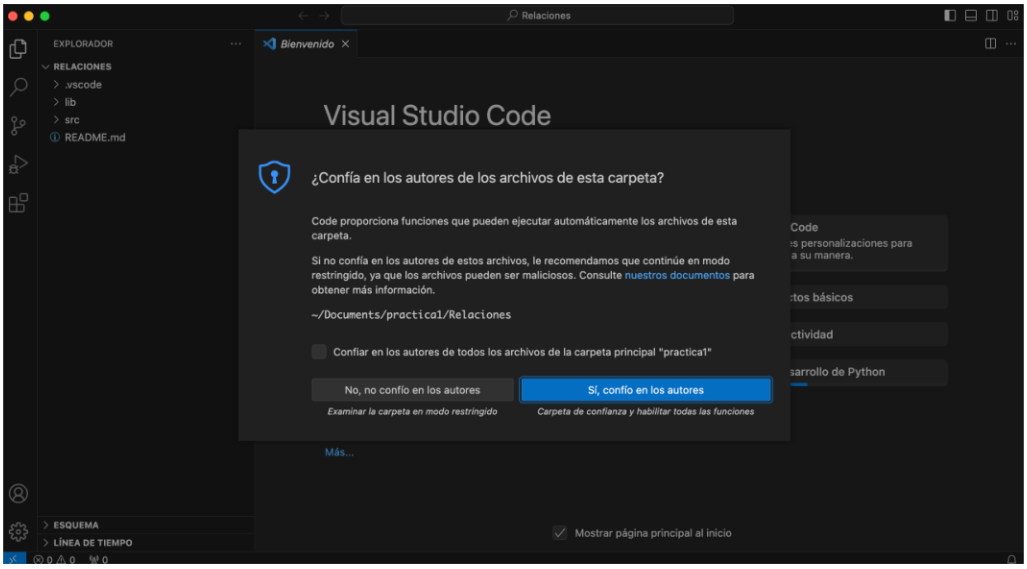

Figura 14: Creación del proyecto en Visual Studio Code

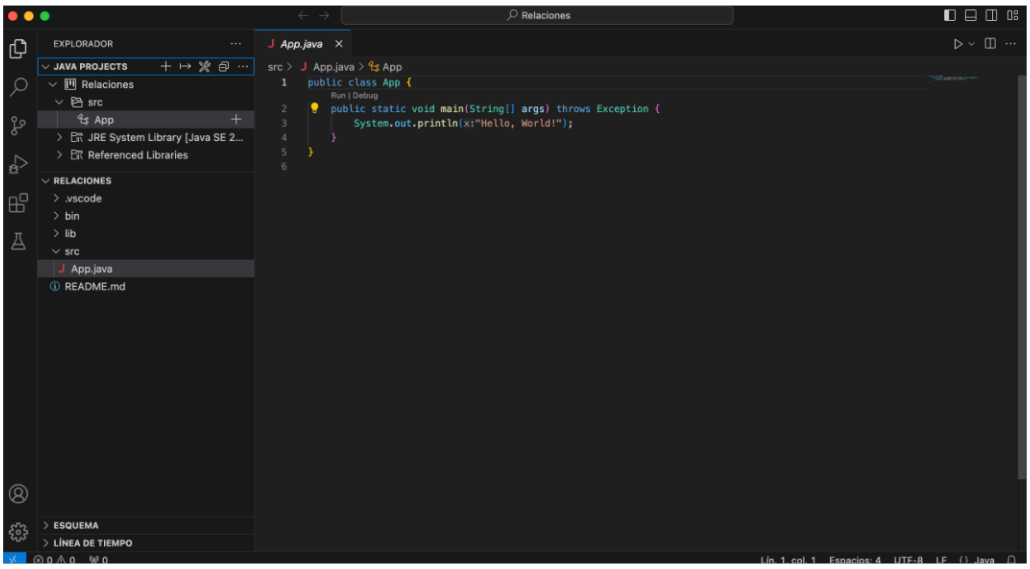

Figura 15: Estructura de un proyecto Java en Visual Studio Code

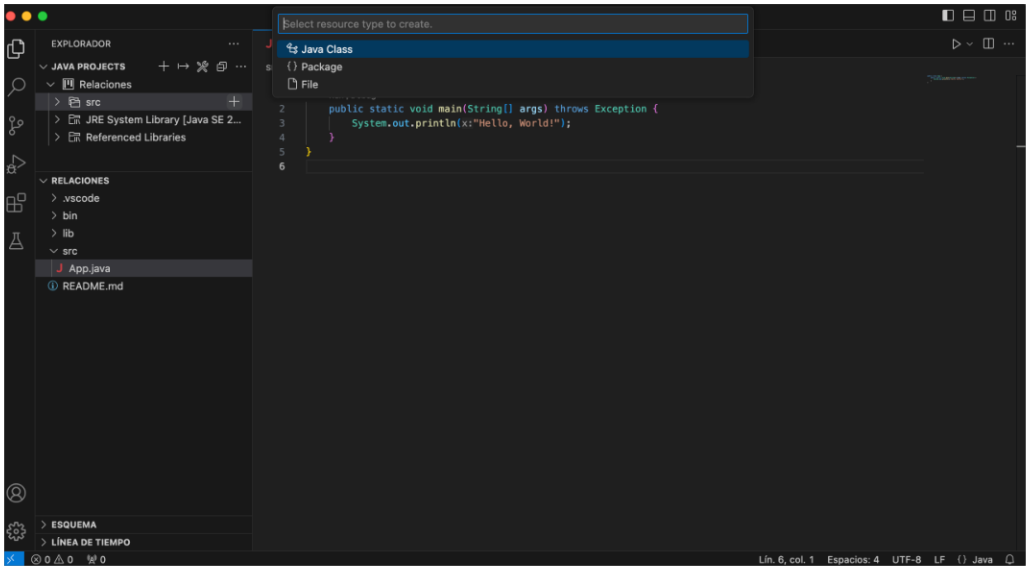

Figura 16: Creaxión de un paquete en Visual Studio Code

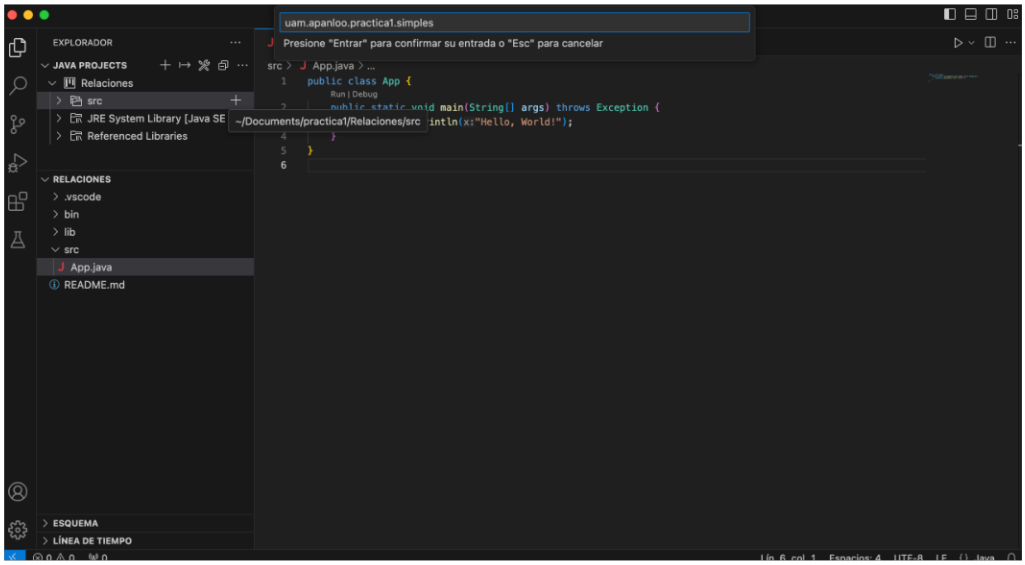

Figura 17: Nombre del paquete en el proyecto de Visual Studio Code

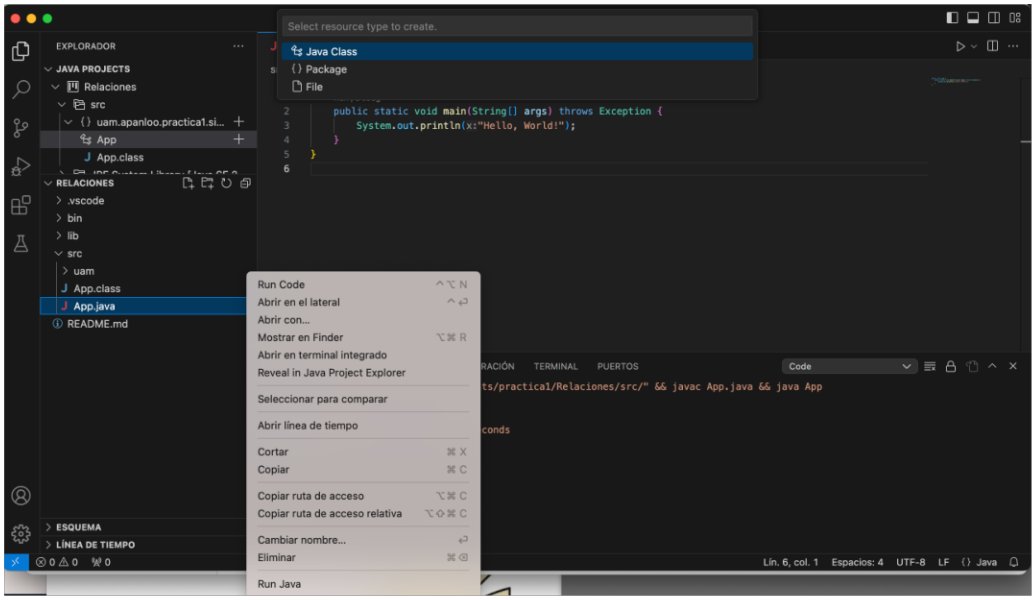

Figura 18: Ejecución de un proyecto en Visual Studio Code

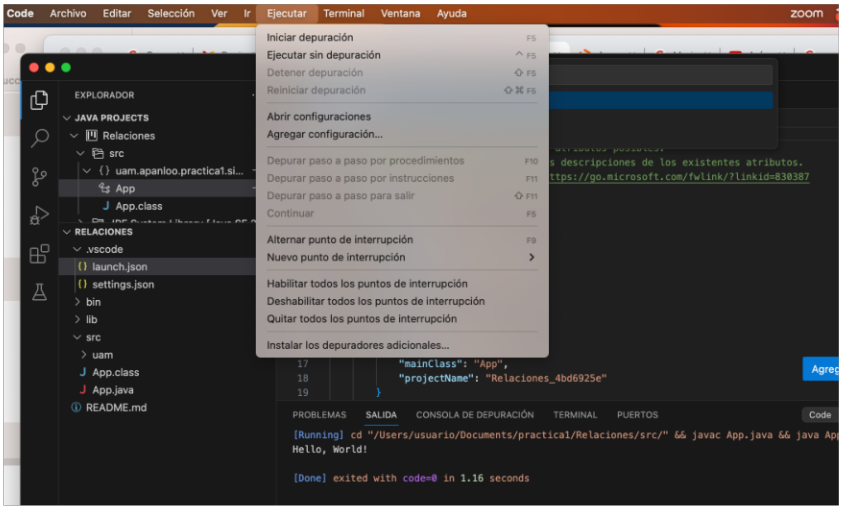

Figura 19: Configuración de ejecución de un proyecto Java en Visual Studio Code

5. Ejercicios

Implemente todos los ejercicios en un proyecto en Java donde se organicen por paquetes.

- Implementar todos los ejemplos del laboratorio.

- Implemente en Java el sistema para la biblioteca. Complete las clases con los atributos y métodos que considere pertinente.
- Implemente en Java el sistema para la agencia de automóviles. Complete las clases con los atributos y métodos que considere pertinente.
- Se desea implementar un sistema para una **Tienda**. La Tienda tiene un *almacén* de productos. Cada **Producto** tiene al menos los atributos:

  - **serie**: identificador único del producto
  - **marca**: marca del producto
  - **nombre**: nombre del producto

  Los comportamientos que se han identificado que tiene toda Tienda son:

  - Agrega un nuevo producto al almacén.
  - Obtener todos los productos del almacen.
  - Obtener los productos de un determinado nombre.
  - Obtener un producto por serie.
  - Buscar si hay productos de una determinada nombre o marca.
  - Vender un producto por nombre o serie.

  Se cuenta con Oxxo y BBB que son tiendas y cada tienda tiene al menos los productos Leche y Galleta.

  Realizar el diseño (diagrama de clases) para el problema a resolver (una hoja). Realizar su implementación en Java. Realizar la documentación de su solución (máximo una hoja). El diagrama de clases, la implementación y su documentación deben ser compartidas en una carpeta en drive. La liga del drive debe indicarse en el moodle. Después de subir sus archivos no se pueden modificar sin previa autorización del profesor.