

## EJERCICIOS CLASES, OBJETOS Y HERENCIA

### PROBLEMA 1 – Modificadores de acceso

Comprobaremos que estas reglas de los modificadores de acceso se cumplen:

- **private**: únicamente la clase puede acceder a la propiedad o método.
- **package private** (valor por defecto si no se indica ninguno): solo las clases en el mismo paquete pueden acceder a la propiedad o método.
- **protected**: las clases del mismo paquete y que heredan de la clase pueden acceder a la propiedad o método.
- **public**: la propiedad o método es accesible desde cualquier método de otra clase.

Modificador de acceso	Misma clase o anidada	Clase en el mismo paquete	Clase que hereda en otro paquete	Clase que no hereda en otro paquete
private	Sí	No	No	No
default (package private)	Sí	Sí	No	No
protected	Sí	Sí	Sí	No
public	Sí	Sí	Sí	Sí

a) Crea la clase vehículo, que contendrá los siguientes atributos:

- Color
- Caballos
- Marca
- Modelo

Hazlos todos “public” para que sean accesibles desde fuera de la clase y del package. Agrega un constructor como el de la figura:

```
package intro00;

public class Vehiculo {

    public String color;
    public int caballos;
    public String marca;
    public String modelo;

    public Vehiculo() {
        this.marca="";
        this.modelo="";
    }
}
```

b) Crea la clase Main **dentro del mismo paquete** que utilice la clase Vehículo realizada en el apartado anterior. Asigna un valor a cada atributo de la clase vehículo.

```
package intro00;

public class Main {

    public static void main(String[] args) {

        Vehiculo v1 = new Vehiculo();
        //Llama al constructor por defecto implicito
        v1.Color = "amarillo";
        v1.Caballos = 180;
        v1.Marca = "Nissan";
        v1.Modelo = "Primera";
    }
}
```

c) Crea la clase Coche **que hereda** de la clase Vehiculo **pero en otro paquete**. Para manipular los datos de Vehiculo, crea un constructor en Coche de la siguiente manera:

```
package intro002;

import intro00.Vehiculo;

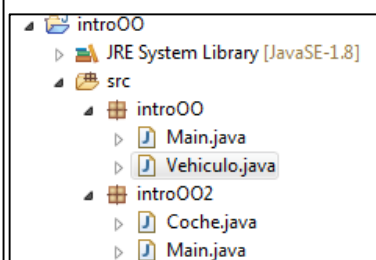
public class Coche extends Vehiculo {
    public Coche() {
        this.marca="";
        this.modelo="";
    }
}
```

d) Copia el fichero Main del apartado b) **en otro package**, el mismo donde se ha creado coche:

```
package intro002;

import intro00.Vehiculo;

public class Main {
    public static void main(String[] args) {
        Vehiculo v1 = new Vehiculo();
        //Llama al constructor por defecto implicito
        v1.Color = "amarillo";
        v1.Caballos = 180;
        v1.Marca = "Nissan";
        v1.Modelo = "Primera";
    }
}
```



**NOTA:** De esta manera hemos simulado 3 situaciones de acceso diferentes:

- Clase en package propio
- subclase en otro package
- clase en otro package.

Como el modificador de los atributos de la clase Vehiculo es el public, tenemos acceso total desde cualquier sitio, y por lo tanto no debe de haber ningún error en el código en este momento

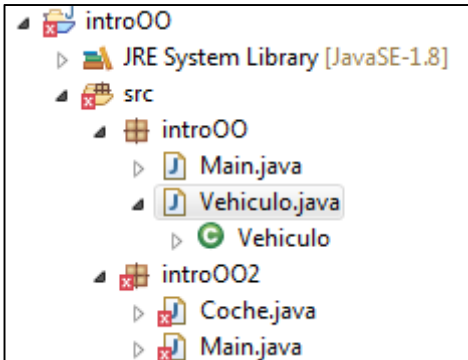
e) Quitamos el modificador public de los atributos de Vehiculo, dejando el modificador default (por defecto). ¿En qué ficheros ocurren errores? ¿Es coherente con lo que dice la teoría?

```
package intro00;

public class Vehiculo {

    String color;
    int caballos;
    String marca;
    String modelo;

    public Vehiculo() {
        this.marca="";
        this.modelo="";
    }
}
```



**Solo las clases en el mismo paquete pueden acceder a la propiedad o método.**

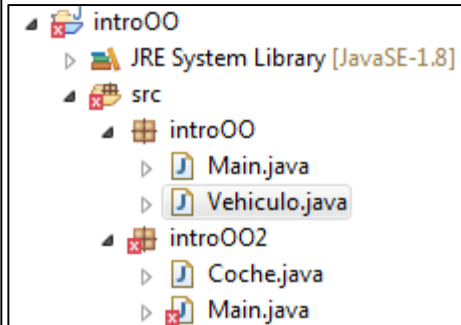
f) Ponemos el modificador protected a los atributos de Vehiculo ¿En qué ficheros ocurren errores? ¿Es coherente con lo que dice la teoría?

```
package intro00;

public class Vehiculo {

    protected String color;
    protected int caballos;
    protected String marca;
    protected String modelo;

    public Vehiculo() {
        this.marca="";
        this.modelo="";
    }
}
```



**Las clases del mismo paquete y que heredan de la clase pueden acceder a la propiedad o método**

g) Ponemos ahora el modificador private a los atributos de Vehiculo ¿En qué ficheros ocurren errores? ¿Es coherente con lo que dice la teoría?

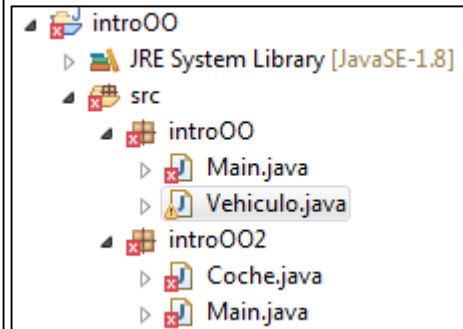
```
package intro00;

public class Vehiculo {

    private String color;
    private int caballos;
    private String marca;
    private String modelo;

    public Vehiculo() {
        this.marca="";
        this.modelo="";
    }

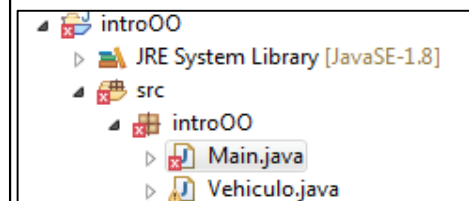
}
```



## PROBLEMA 2 – Getters y Setters

a) Borrarnos el package extra y las clases contenidas en el. Dejamos los atributos de la clase Vehiculo con el modificador private. ¿Cómo queda el Main?

```
5 public static void main(String[] args) {
6
7     Vehiculo v1 = new Vehiculo();
8     //Llama al constructor por defecto implicito
9     v1.color = "amarillo";
10    v1.caballos = 180;
11    v1.marca = "Nissan";
12    v1.modelo = "Primera";
13 }
14 }
```



b) La propiedad de la encapsulación de la POO indica que los atributos de un objeto se deben de ocultar para que los otros objetos no puedan acceder a ellos directamente. Por tanto la única forma de arreglar esta situación es creando las funciones Getters y Setters en la clase Vehiculo para que sus atributos privados puedan ser accedidos. Hacer click botón derecho/Source/Generate Gettes and Setters:

Quick Fix	Ctrl+1	Generate Element Comment	Alt+Shift+C
Source	Alt+Shift+S	Correct Indentation	Ctrl+I
Refactor	Alt+Shift+T	Format	Ctrl+Shift+F
Local History		Format Element	
References		Add Import	Ctrl+Shift+M
Declarations		Organize Imports	Ctrl+Shift+O
Add to Snippets...		Sort Members...	
Coverage As		Clean Up...	
Run As		Generate Hibernate/JPA annotations...	
Debug As		Override/Implement Methods...	
Profile As		Generate Getters and Setters...	
		Generate Delegate Methods...	

```

public class Vehiculo {
    private String color;
    private int caballos;
    private String marca;
    private String modelo;

    public Vehiculo() {
        this.marca="";
        this.modelo="";
    }

    public String getColor() {
        return color;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public int getCaballos() {
        return caballos;
    }
    public void setCaballos(int caballos) {
        this.caballos = caballos;
    }
}

```

c) Arregla el problema en Main, corrigiendo los errores pero manteniendo la funcionalidad:

```

public static void main(String[] args) {

    Vehiculo v1 = new Vehiculo();
    //Llama al constructor por defecto implicito
    v1.setColor("amarillo");
    v1.setCaballos(180);
    v1.setMarca("Nissan");
    v1.setModelo("Primera");
}

```

### PROBLEMA 3 – Constructores

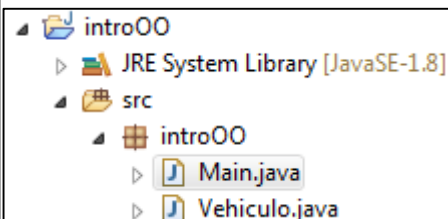
a) Elimina el constructor creado anteriormente en Vehiculo. ¿Ocurre algún error? ¿Por qué?

```

public class Vehiculo {
    private String color;
    private int caballos;
    private String marca;
    private String modelo;

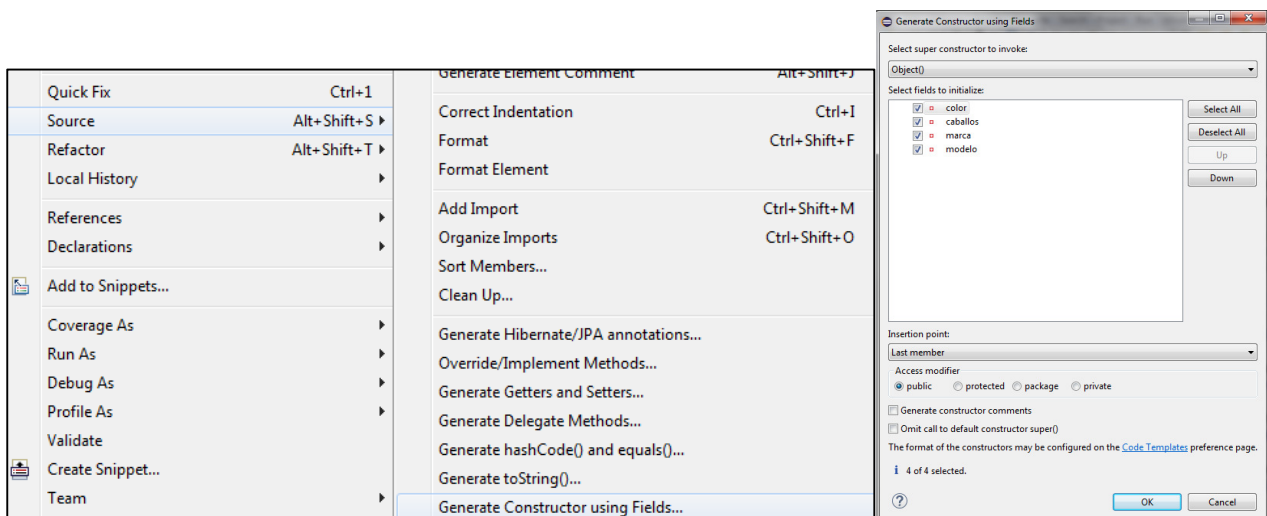
    //public Vehiculo() {
    //    this.marca="";
    //    this.modelo="";
    //}
}

```



Si no hay definido ningún constructor, el sistema interpreta de forma implícita que hay uno por defecto (es decir sin parámetros).

b) Crea un constructor full, es decir, con todos los parámetros de la clase en Vehiculo.  
¿Ocurre algún error? ¿Por qué?



```
public class Vehiculo {  
    private String color;  
    private int caballos;  
    private String marca;  
    private String modelo;  
  
    //public Vehiculo() {  
    //    this.marca="";  
    //    this.modelo="";  
    //}  
  
    public Vehiculo(String color, int caballos,  
        String marca, String modelo) {  
        this.color = color;  
        this.caballos = caballos;  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
}
```

```
5 public static void main(String[] args) {  
6  
7     Vehiculo v1 = new Vehiculo();  
8     //Llama al constructor por defecto implicito  
9     v1.setColor("amarillo");  
10    v1.setCaballos(180);  
11    v1.setMarca("Nissan");  
12    v1.setModelo("Primera");  
13 }
```

Si se define un constructor diferente al por defecto, el sistema te obliga a definir de forma explícita el constructor por defecto.

c) Crea un constructor por defecto en Vehiculo.

```
public class Vehiculo {  
    private String color;  
    private int caballos;  
    private String marca;  
    private String modelo;  
  
    public Vehiculo() {  
        this.color = "";  
        this.caballos = 0;  
        this.marca="";  
        this.modelo="";  
    }  
  
    public Vehiculo(String color, int caballos,  
        String marca, String modelo) {  
        this.color = color;  
        this.caballos = caballos;  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
}
```

d) Crea otro constructor full de 4 parámetros en la clase Vehiculo ¿Cuántos constructores diferentes se pueden crear en esta clase?

```
public Vehiculo() {  
    this.color = "";  
    this.caballos = 0;  
    this.marca="";  
    this.modelo="";  
}  
  
public Vehiculo(String color, int caballos,  
    String marca, String modelo) {  
    this.color = color;  
    this.caballos = caballos;  
    this.marca = marca;  
    this.modelo = modelo;  
}  
  
public Vehiculo(String color, String marca,  
    int caballos,String modelo) {  
    this.color = color;  
    this.caballos = caballos;  
    this.marca = marca;  
    this.modelo = modelo;  
}
```

**Se pueden definir infinitos constructores en una clase, lo único necesario es que se diferencien en el número y/o en el tipo de los parámetros.**

## PROBLEMA 4 – Herencia

a) Crea una clase Coche, que heredará de Vehículo, y además contendrá los siguientes atributos propios:

- numPuertas.
- capacidadMaletero.

```
public class Coche extends Vehiculo {  
    //Atributos propios de coche  
    private int numPuertas;  
    private int capacidadMaletero;  
}
```

¿Cuántos atributos tiene ahora la clase Coche?

**Al heredar de Vehiculo, la clase Coche hereda tanto sus métodos como sus atributos.**

b) Crea los Getters y Setters correspondientes en la clase Coche

```
public class Coche extends Vehiculo {  
    //Atributos propios de coche  
    private int numPuertas;  
    private int capacidadMaletero;  
  
    public int getNumPuertas() {  
        return numPuertas;  
    }  
    public void setNumPuertas(int numPuertas) {  
        this.numPuertas = numPuertas;  
    }  
    public int getCapacidadMaletero() {  
        return capacidadMaletero;  
    }  
    public void setCapacidadMaletero(int capacidadMaletero) {  
        this.capacidadMaletero = capacidadMaletero;  
    }  
}
```

c) Crea un constructor por defecto en la clase Coche. En ellos se debe de inicializar todos los atributos de la clase (los heredados y los suyos propios), pero sin usar ningún método/constructor de la clase base Vehiculo. ¿Ocurre algún error?



```

5 public class Coche extends Vehiculo {
6     //Atributos propios de coche
7     private int numPuertas;
8     private int capacidadMaletero;
9
10    public Coche() {
11        this.color = "";
12        this.caballos = 0;
13        this.marca="";
14        this.modelo="";
15        this.numPuertas = 0;
16        this.capacidadMaletero = 0;
17    }

```

**El modificador de acceso correcto para que los atributos de la clase base Vehiculo sean accesibles desde la clase hijo Coche debe ser protected.**

d) Soluciona el error del apartado anterior.

```

public class Vehiculo {
    protected String color;
    protected int caballos;
    protected String marca;
    protected String modelo;

    public Vehiculo() {
        this.color = "";
        this.caballos = 0;
        this.marca="";
        this.modelo="";
    }

```

e) Modifica el constructor por defecto de la clase Coche, pero en este caso usando constructores de la clase base Vehiculo.

```

public class Coche extends Vehiculo {
    //Atributos propios de coche
    private int numPuertas;
    private int capacidadMaletero;

    public Coche() {
        super();
        this.numPuertas = 0;
        this.capacidadMaletero = 0;
    }

```

**La primera instrucción en cualquier constructor de una subclase debe ser una llamada al constructor del padre**

e) Crea un constructor full de parámetros de la clase Coche. La idea básica es que los parámetros del padre los inicialice el constructor del padre y los parámetros del hijo los inicialice el hijo.

```
public Coche() {
    super();
    this.numPuertas = 0;
    this.capacidadMaletero = 0;
}
public Coche(String color, int caballos, String marca, String modelo,
              int numPuertas, int capacidadMaletero) {
    super(color, caballos, marca, modelo);
    this.numPuertas = numPuertas;
    this.capacidadMaletero = capacidadMaletero;
}
```

f) Instancia dos coches usando métodos diferentes, de manera que al final ambos coches tengan el mismo estado (el mismo valor de sus atributos).

```
public static void main(String[] args) {

    Coche c1 = new Coche(); //Llama al constructor por defecto
    c1.setColor("amarillo");
    c1.setCaballos(180);
    c1.setMarca("Nissan");
    c1.setModelo("Primera");
    c1.setNumPuertas(5);
    c1.setCapacidadMaletero(200);

    Coche c2= new Coche("amarillo",180, "Nissan","Primera",5,200);

}
```

## PROBLEMA 5 – Clase Abstracta

a) Declara la clase Vehículo “abstract”.

```
public abstract class Vehiculo {
    protected String color;
    protected int caballos;
    protected String marca;
    protected String modelo;

    public Vehiculo() {
        this.color = "";
        this.caballos = 0;
        this.marca="";
        this.modelo="";
    }
}
```

b) En el main instancia una variable de tipo Vehiculo ¿Qué ocurre?

```
5 public static void main(String[] args) {  
6  
7     Vehiculo v1 = new Vehiculo();  
8     Cannot instantiate the type Vehiculo = new Vehiculo();  
9  
10 }
```

**Una clase abstracta no se puede instanciar, solo se usa para definir una jerarquía de clases y para que otras clases deriven de ella.**

c) Comenta los dos objetos de tipo Vehiculo creados anteriormente para que no hayan errores:

```
public static void main(String[] args) {  
  
    //Vehiculo v1 = new Vehiculo();  
    //Vehiculo v2 = new Vehiculo();  
  
}
```

d) Elimina la clausula abstract de la clase Vehiculo, pero ahora crea un método abstract por ejemplo arrancar(). ¿Qué ocurre? ¿Surgen nuevos problemas?

```
public class Vehiculo {  
    protected String color;  
    protected int caballos;  
    protected String marca;  
    protected String modelo;  
  
    abstract public void arrancar();  
}
```

Remove 'abstract' modifier  
Make type 'Vehiculo' abstract

**Una clase que tenga un método abstract, convierte la clase en abstract**

e) Vuelve a definir la clase Vehiculo abstract ¿Qué ocurre ahora?

Surge un problema en la clase hijo Coche:

```
public class Coche extends Vehiculo {  
    //Atribut  
    private i  
    private i
```

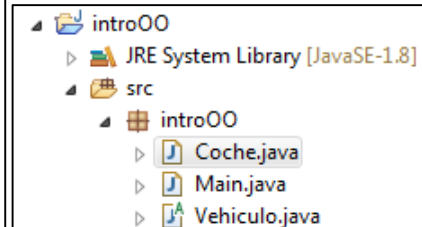
Add unimplemented methods  
Make type 'Coche' abstract  
Rename in file (Ctrl+2, R)  
Rename in workspace (Alt+Shift+R)

Una clase abstract con métodos abstract obliga a que sean redefinidos por las clases hijo o subclases de esa clase

f) Redefine el método arrancar en la subclase Coche:

```
public class Coche extends Vehiculo {
    //Atributos propios de coche
    private int numPuertas;
    private int capacidadMaletero;

    @Override
    public void arrancar() {
        // TODO Auto-generated method stub
    }
}
```



## PROBLEMA 6 – Variables final y static

a) Redefine los atributos propios de Coche, añadiendo la etiqueta final a uno y static al otro, siendo ahora públicos ambos.

```
public class Coche extends Vehiculo {
    //Atributos propios de coche
    public final int numPuertas;
    public static int capacidadMaletero;
}
```

b) ¿Qué errores ocurren en la clase Coche?

```
16 public int getNumPuertas() {
17     return numPuertas;
18 }
19 public void setNumPuertas(int numPuertas) {
20     this.numPuertas = numPuertas;
21 }
22 public int main() {
23     return 0;
24 }
```

The final field Coche.numPuertas cannot be assigned  
1 quick fix available:  
Remove 'final' modifier of 'numPuertas'

c) Crea un objeto Coche en el main e intenta modificar su atributo final.

```
5 public static void main(String[] args) {
6
7     Coche c1 = new Coche();
8     c1.numPuertas=6;
9
10 }
11
```

The final field Coche.numPuertas cannot be assigned

Un atributo final es declarado como constante y por lo tanto hace que su valor no pueda variar durante la ejecución del programa

d) Queremos asignar un valor constante de 4 a numPuertas y que no se vuelva a modificar nunca más ¿Cómo se puede hacer tal cosa?

```
public class Coche extends Vehiculo {
    //Atributos propios de coche
    public final int numPuertas;
    public static int capacidadMaletero;

    public Coche() {
        super();
        this.numPuertas = 4;
        this.capacidadMaletero = 200;
    }

    public int getNumPuertas() {
        return numPuertas;
    }

    //public void setNumPuertas(int numPuertas) {
    //    this.numPuertas = numPuertas;
    //}
```

```
5 public static void main(String[] args) {
6
7     Coche c1 = new Coche();
8     System.out.print(c1.numPuertas);
9
10 }
11 }
```

Problems Javadoc Declaration Console Progress Debug Git

<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0\_241\bin\javaw.exe (7 feb. 2021 8:54)  
4

**Un atributo final sólo puede ser inicializado en el constructor de la clase.**

e) Para comprobar la propiedad de los atributos static, en la clase Main crea 3 objetos de tipo coche, asignando una capacidadMaletero distinta a cada uno. Por último muestra el contenido de este atributo de los 3 objetos coche.

```
5 public static void main(String[] args) {
6     Coche c1 = new Coche();
7     c1.capacidadMaletero=30;
8     Coche c2 = new Coche();
9     c2.capacidadMaletero=21;
10    Coche c3 = new Coche();
11    c3.capacidadMaletero=54;
12    System.out.println(c1.capacidadMaletero);
13    System.out.println(c2.capacidadMaletero);
14    System.out.println(c3.capacidadMaletero);
15 }
```

Problems Javadoc Declaration Console Progress Debug Git Staging

<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0\_241\bin\javaw.exe (7 feb. 2021 8:54)  
54  
54  
54

Un atributo static (denominado atributo de la clase), es compartido por todos los objetos de esa misma clase. Sólo se crea una instancia de él, siendo el mismo para todos los objetos de la misma clase

### PROBLEMA 7 – Clases Internas

a) Crea una clase Radio dentro de la clase Coche, con los siguientes atributos y métodos. Añade un atributo de tipo Radio en la clase Coche.

```
public class Coche extends Vehiculo {  
    //Atributos propios de coche  
    public final int numPuertas;  
    public static int capacidadMaletero;  
    public Radio radio;  
  
    public class Radio{  
        boolean encendido;  
        int frecuencia=100;  
        public void encender() {}  
        public void apagar() {}  
    }  
}
```

b) Añade un setter de Radio en Coche y en el constructor de Coche inicializa el atributo Radio.

```
public class Coche extends Vehiculo {  
    //Atributos propios de coche  
    public final int numPuertas;  
    public static int capacidadMaletero;  
    public Radio radio;  
  
    public class Radio{  
        boolean encendido;  
        int frecuencia=100;  
        public void encender() {}  
        public void apagar() {}  
    }  
    public Coche() {  
        super();  
        this.numPuertas = 4;  
        this.capacidadMaletero = 200;  
        this.radio = new Radio();  
    }  
    public void setRadio(Radio radio) {  
        this.radio = radio;  
    }  
}
```

c) Crea un programa que cree un coche y le asigne una Radio de forma externa.

```
public static void main(String[] args) {

    Coche micoche = new Coche();
    Coche.Radio laradio = micoche.new Radio();
    micoche.setRadio(laradio);
    laradio.encender();

}
```

**Java permite las clases internas. El hecho de que se puedan utilizar correctamente depende de su visibilidad o modificador de acceso. No obstante los casos de aplicación son mínimos y lo suyo es crear la clase dentro de un fichero independiente**

## PROBLEMA 8 – Interfaces

a) Crea un interfaz Producto que defina los siguientes métodos:

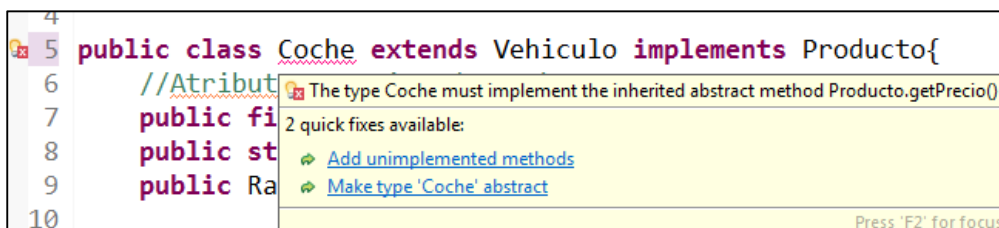
- getPrecio()
- getDescripcion()

```
public interface Producto {

    public float getPrecio();
    public String getDescripcion();

}
```

b) Haz que la clase Coche implemente la interfaz Producto. ¿Da errores?



```
4
5 public class Coche extends Vehiculo implements Producto{
6     //Atributos
7     public final int numPuertas;
8     public static int capacidadMaletero;
9     public Radio radio;
10
```

Se deben de implementar los métodos de la interfaz.

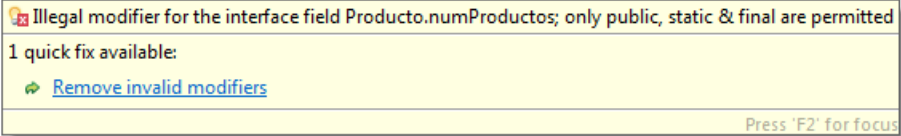
```
public class Coche extends Vehiculo implements Producto{
    //Atributos propios de coche
    public final int numPuertas;
    public static int capacidadMaletero;
    public Radio radio;

    @Override
    public float getPrecio() {
        return 0;
    }

    @Override
    public String getDescripcion() {
        return "";
    }
}
```

c) Intenta crear un atributo en la interfaz Producto:

```
public interface Producto {  
    private int numProductos;  
  
    public void  
    public void  
  
}
```



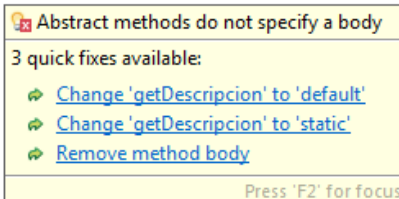
Press 'F2' for focus

```
public interface Producto {  
    public int numProductos=5;  
  
    public void getPrecio();  
    public void getDescripcion();  
  
}
```

**Una interfaz solo admite atributos públicos, estáticos y finales inicializados, es decir, básicamente constantes**

d) Intenta implementar un método en la interfaz Producto:

```
public interface Producto {  
  
    public int numProductos=5;  
  
    public float getPrecio();  
    public String getDescripcion() {  
        return "";  
    }  
  
}
```

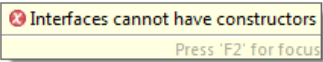


Press 'F2' for focus

**Una interfaz representa un contrato que deben de cumplir las clases que la implementan, por lo tanto sus métodos son abstractos y no tienen que especificar un cuerpo**

e) Intenta crear un constructor en la interfaz Producto:

```
public interface Producto {  
  
    public int numProductos=5;  
  
    public float getPrecio();  
    public String getDescripcion();  
  
    public Producto() {  
    }  
  
}
```



Press 'F2' for focus



## Las interficies no pueden tener constructores

f) Cual es la diferencia entre una clase abstracta y una interfaz.

A nivel de programación son parecidas, puesto que ambas definen un contrato/funcionamiento que se debe de cumplir. No obstante más concretamente las interfaces no pueden tener atributos (solo constantes) y no pueden definir ningún método ni constructor. En cambio una clase abstracta puede declarar atributos, y no todas las funciones se deben de sobrescribir en la clase hijo.

g) Cual es la utilidad real de una interfaz. ¿Es posible que un objeto herede de varias clases? ¿Y que implemente varias interfaces?

En Java no es posible la herencia múltiple, solo la simple, es decir una clase hereda solamente de su clase base. La interfaz representar otra manera de “heredar” pero sólo métodos que se deben de implementar a modo de contrato.

Un objeto puede implementar infinitas interfaces, solo agrega más comportamiento a la clase, sin datos ni atributos.

El problema de la herencia múltiple, es que puede haber ambigüedad en los datos heredados, en aquellos casos que un objeto derive de clases padre que hubieran derivado de la misma clase “abuelo”.

### PROBLEMA 9 – Casting a Interfaces o superclase

a) Crea la clase Moto que heredará de Vehículo e implementará la interfaz Producto. Debe contener los atributos enteros xx e yy.

```
public class Moto extends Vehiculo implements Producto {
    private int xx;
    private int yy;
    @Override
    public float getPrecio() {
        return 0;
    }
    @Override
    public String getDescripcion() {
        return null;
    }

    @Override
    public void arrancar() {
    }
}
```

b) En un programa de Main crea un objeto de tipo Coche y otro de tipo Moto, y guárdalos en variables tipo Vehículo. ¿Es posible?

```
public static void main(String[] args) {
    Vehiculo v1 = new Coche();
    Vehiculo v2 = new Moto();
    Coche c3 = new Coche();
    Vehiculo v3 = c3;
}
```

**Cualquier subclase puede castearse/representarse a través de su clase base. Es posible combinar en un objeto de tipo superclase, objetos de las diferentes subclases heredadas.**

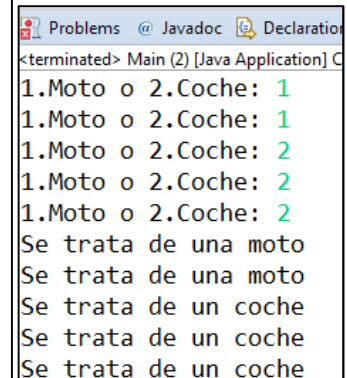
c) Intenta guardar un coche y una moto en un objeto de tipo Producto. ¿Es posible?

```
public static void main(String[] args) {  
    Producto p1 = new Coche();  
    Producto p2 = new Moto();  
    Coche c3 = new Coche();  
    Producto p3 = c3;  
}
```

**Java permite combinar en un objeto Interfaz, objetos de las clases que implementan dicho interfaz.**

d) Haz un programa que genere un array de 5 Vehículos. El usuario podrá escoger si desea un Coche o una Moto, y deberá rellenar los datos necesarios para crear el objeto. Al final, muestra por pantalla el contenido del array mediante la función instanceof.

```
public static void main(String[] args) {  
    ArrayList<Vehiculo> lista = new ArrayList<Vehiculo>();  
    Scanner sc = new Scanner(System.in);  
    for (int i=0; i<5;i++) {  
        System.out.print("1.Moto o 2.Coche: ");  
        int opcion = sc.nextInt();  
        if (opcion==1) {  
            Moto mimoto = new Moto();  
            lista.add(mimoto);  
        }  
        else {  
            Coche micoche = new Coche();  
            lista.add(micoche);  
        }  
    }  
    for (Vehiculo v:lista) {  
        if (v instanceof Moto)  
            System.out.println("Se trata de una moto");  
        else  
            System.out.println("Se trata de un coche");  
    }  
}
```



```
Problems @ Javadoc Declaration  
<terminated> Main (2) [Java Application] C  
1.Moto o 2.Coche: 1  
1.Moto o 2.Coche: 1  
1.Moto o 2.Coche: 2  
1.Moto o 2.Coche: 2  
1.Moto o 2.Coche: 2  
Se trata de una moto  
Se trata de una moto  
Se trata de un coche  
Se trata de un coche  
Se trata de un coche
```

e) Idem que el ejercicio anterior pero ahora sin utilizar la función instanceof, activando la función toString de las subclases Moto y Coche:

```
public class Moto extends Vehiculo implements Producto {  
    private int xx;  
    private int yy;  
  
    @Override  
    public String toString() {  
        return "Moto [xx=" + xx + ", yy=" + yy + "];"  
    }  
}
```

```

public class Coche extends Vehiculo implements Producto{
    //Atributos propios de coche
    public final int numPuertas;
    public static int capacidadMaletero;
    public Radio radio;

    @Override
    public String toString() {
        return "Coche [numPuertas=" + numPuertas + ", radio=" + radio + "];"
    }
}

```

```

public static void main(String[] args) {
    ArrayList<Vehiculo> lista = new ArrayList<Vehiculo>();
    Scanner sc = new Scanner(System.in);
    for (int i=0; i<5;i++) {
        System.out.print("1.Moto o 2.Coche: ");
        int opcion = sc.nextInt();
        if (opcion==1) {
            Moto mimoto = new Moto();
            lista.add(mimoto);
        }
        else {
            Coche micoche = new Coche();
            lista.add(micoche);
        }
    }
    for (Vehiculo v:lista) {
        System.out.println(v);
    }
}

```

```

<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (7 feb. 2021 13:47:
1.Moto o 2.Coche: 1
1.Moto o 2.Coche: 2
1.Moto o 2.Coche: 1
1.Moto o 2.Coche: 2
1.Moto o 2.Coche: 1
Moto [xx=0, yy=0]
Coche [numPuertas=4, radio=intro00.Coche$Radio@1b6d3586]
Moto [xx=0, yy=0]
Coche [numPuertas=4, radio=intro00.Coche$Radio@4554617c]
Moto [xx=0, yy=0]

```