Francisco Sitjar

Constructor copia

```java
public class Taller {

    //Declaracion de atributos
    private int aceite;//En litros
    private int ruedas;
    private static int contadorCambiosTotales;
    private static int contadorCambiosParciales;

    public Taller(int aceite, int ruedas) {
        this.aceite = aceite*5;
        this.ruedas = ruedas;

    }

    public Taller(Taller n){
        this.aceite = n.aceite;
        this.ruedas = n.ruedas;

    }
```

Método con void cuando no tengo que devolver nada:

```java
//Set
public void setIsbn(String isbn){
    this.isbn = isbn;
}
```

(método set para cambiar valor de algún atributo)
Con variable cuando hay return:

```java
//Get
public String getIsbn(){
    return isbn;
}
```

(método get para ver valor de algún atributo, también se puede trabajar con el)
Modificadores de acceso

	La misma clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
public	X	X	X	X
protected	X	X	X	
default	X	X		
private	X			

Sobrecarga de método;

```
public class Matematicas
{
    public double suma(double x, double y)
    {
        return x+y;
    }
    public double suma(double x, double y, double z)
    {
        return x+y+z;
    }
    public double suma(int numero1, int numero2)
    {
        return numero1 + numero2;
    }
}
```

Método Static para métodos que no requieren objetos porque no modifican atributos ni interactúan con estos.

Métodos recursivos se llama al mismo método:

```
public int factorial(int n)
{
int facto= n<=1 ? 1: n*factorial(n-1);
return facto;
```

para llamar al método se hace “factorial(n)”

String

substring, beginIndex(incluyendo a este mismo), endIndex(no añade este mismo)

```
int num = Integer.parseInt(s: dni.substring(beginIndex: 0,endIndex: 8))%23;
```

Para pasar de String a otro tipo de variable:
-Conversión de variables:
Integer.parseInt convierte a int
Short.parseShort convierte a short
Byte.parseByte convierte a byte
Long.parseLong convierte a long
Float.parseFloat convierte a float
Double.parseDouble convierte a double
Boolean.parseBoolean convierte a boolean
Para variables tipo char = charAT(pos), donde pos es la posición del carácter a pasar

Para **comparar Strings** con el método “string.equal”

```
public boolean esEspanol(){
    return isbn.substring(beginIndex: 3, endIndex:5).equals(anObject:"84") ;
}
```

String.format para “simular” un printf

```
e + " litros.\nINGRESOS: Total: " + String.format(format: "%,.2f", args:total) + " €";
```

otros métodos de String:
-**startsWith(subcadena)** y **endsWith(subcadena)**: para comprobar si una cadena comienza o finaliza con una subcadena determinada.
-**trim()**: elimina los espacios en blanco de una cadena que tenga por delante o por detrás. No elimina los espacios intermedios.
-**toUpperCase()** y **toLowerCase()**: me permite cambiar todos los caracteres por mayúsculas o minúsculas.
-**indexOf(cadenaABuscar)**: permite buscar una cadena dentro de otra.
-**indexOf(cadenaABuscar,posicion)**: igual que la anterior, pero desde una posición determinada.
-**replace(cadenaABuscar, cadenaSustituta)**: permite reemplazar una cadena por otra.
-**lastIndexOf (String cad)**: Retorna la posición de la última ocurrencia de la cadena dada como parámetro.
-**lastIndexOf (String cad,int ini)**: Retorna la posición de la última ocurrencia de la cadena dada como parámetro buscando en retroceso a partir de la posición dada como parámetro.

Teoría

Encapsulado
Permite que todo lo referente a un objeto esté separado del resto.

Abstracción
Enseña la característica de alguna entidad al mundo exterior ignorando la complejidad interior. Ejemplo: usar la clase Scanner y sus métodos, ignorando como funciona internamente.

Polimorfismo
Cambias el uso de algún método de una clase padre creando otro método con el mismo nombre en la clase hijo así cuando se llame al método el programa ya sabe que debe priorizar el método sobreescrito en el hijo.

Modularidad
La descomposición de una clase, donde todo lo interno de una clase tiene alta cohesión y baja acoplamiento.
-**Alta cohesión:** cada clase puede describirse con un solo nombre y cada método realiza una única tarea donde se puede describir su función con una frase.

-**Bajo acoplamiento:** las clases son lo más independientes posibles entre sí, además tienen una parte pública pequeña y bien definida donde no se requiere conocer su contenido para su correcto uso.

Todo esto facilita la comprensión del código así como su mantenimiento para localizar errores, la reutilización del programa es más probable y probar las clases suele ser más fácil.