U9 – DESARROLLO DE CLASES.

En la unidad anterior se ha expuesto cómo definir los atributos de las clases para establecer su visibilidad. Aquí se estudiará cómo se puede controlar:

- La visibilidad de las clases y de sus miembros
- La sobrecarga de métodos
- Cómo definir métodos estáticos y sus capacidades.
- Delimitar el ámbito de atributos / variables.

Además, se introducirá el concepto de librerías.

9.1. ÁMBITO DE ATRIBUTOS Y VARIABLES.

9.1.1. ATRIBUTOS:

Los atributos se declaran dentro de una clase, dependiendo de qué modificadores se utilicen en la declaración, **los métodos tendrán diferente visibilidad**:

- PUBLIC. Accesible desde cualquier lugar en el que sea accesible la clase.
- PRIVATE. Sólo accesible para la propia clase.
- PROTECTED. Accesible por las subclases y clases del mismo paquete.
- - (friendly). Si no se especifica nada el atributo es accesible para las clases del mismo paquete.

NOTA: En todo caso se **recomienda que los atributos sean privados**.

9.1.2. VARIABLES:

Las variables de declaran dentro del ámbito de los métodos.

OJO: El ámbito de toda variable declarada entre llaves comprende desde donde está declarada hasta la llave de cierre.

9.2. SOBRECARGA DE MÉTODOS.

La sobrecarga consiste en la definición de **dos o más métodos con el mismo nombre, pero con diferente número y/o tipo de argumentos de entrada**.

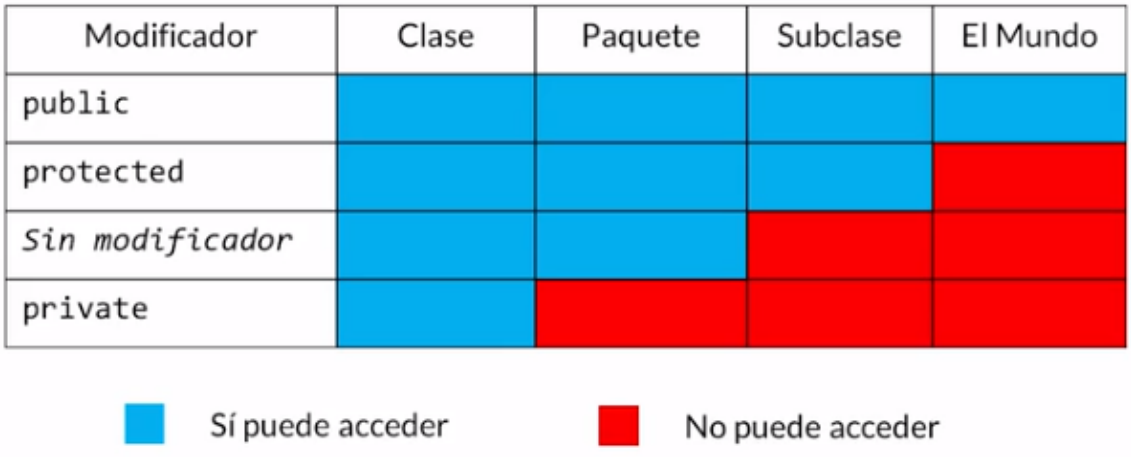
1. int suma(int, int)
2. double suma(double, double)
3. int suma(long, long)

9.3. MODIFICARES DE CLASE. VISIBILIDAD.

Indica el tipo de acceso y visibilidad de la clase. Los modificadores existentes:

- **abstract**. Son clases con al menos un método abstracto. No se instancian, sino que se utilizan como clase base para la herencia -> Se verá en HERENCIA.
- **final**. Se declara como la clase que termina una cadena de herencias, impidiendo que se pueda heredar de una clase final -> Se verá en HERENCIA.
- **public**. Son accesibles desde otras clases dentro del mismo paquete y fuera del paquete para ser utilizado debe importarse (utilizar "import").
- **- (ningún modificador).** Sino es especifica ningún modificador, cualquier objeto dentro del mismo paquete puede utilizarlo.

VISIBILIDAD CLASES – MÉTODOS – ATRIBUTOS.



9.4. UTILIZACIÓN DE MÉTODOS ESTÁTICOS.

Como se ha visto hasta ahora los métodos se invocan a través del objeto que se instancia de la clase para poder aplicar/utilizar dicho método.

En cambio, **los métodos estáticos son métodos de clase y se invocan a través de la clase directamente, no a través del objeto.**

Es decir, para utilizar un método estático no es necesario instanciar un objeto para utilizar dicho método estático dentro de la clase. Accediendo a través de:

```
NombreClase.método().
```

9.5. PASO DE PARÁMETROS. PASOR POR VALOR Y PASO POR REFERENCIA.

Dentro de los lenguajes de programación, el paso de parámetros a una función se puede realizar dos formas:

1. **Paso de parámetros por valor.** En el paso de parámetros por valor, al invocar el método, se hace una copia de la variable utilizada como parámetro y a continuación se ejecuta el método utilizando la copia, así las modificaciones dentro de la función se realizan sobre la copia. Al finalizar la función se retorna dicha copia o esta es borrada / destruye.

2. **Paso de parámetros por referencia.** En el paso de parámetros por referencia no se realiza una copia de la variable, sino que la función recibe una referencia o apuntador (una dirección de memoria donde se encuentra la variable – puntero). Por lo que el método actúa y modifica directamente la variable original dentro del método.

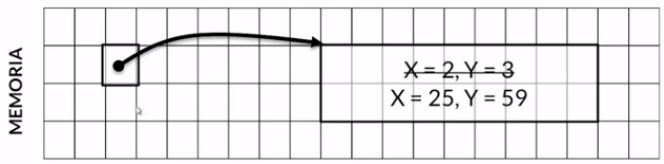
JAVA: Sólo admite paso por valor. Pero ojo, con al pasar como parámetros objetos. ¿Qué sucede? http://lineadecodigo.com/java/parametros-por-referencia-en-java/

Ok, todo el paso de parámetros en Java es por valor. Pero, es necesario diferenciar:

- **Paso de tipos primitivos:** Se hace por valor.

```
public class PasoPorValor {
    public static void main(String[] args) {
        int x = 3;
        //invocamos el argumento y le pasamos x
        pasoPorValor(x);
        //imprimimos x y vemos si el parámetro ha cambiado
        System.out.println("Después de invocar pasoPorValor, x = " + x);
    }
    // cambiamos el valor en el método
    public static void pasoPorValor(int p) {
        p = 10;
    }
}
```

- **Paso de objetos.** También se hace por valor, pero como hemos visto un objeto es una referencia a un espacio de memoria, entonces estamos pasando/copiando el valor de dicha referencia.



Por lo tanto, aunque el paso es por valor, el método puede modificar el objeto directamente dentro de dicho método.

9.6. DESTRUCCIÓN DE OBJETOS Y LIBERACIÓN DE MEMORIA.

Los objetos tienen un tiempo de vida y consumen recursos del ordenador mientras están en uso. ¿Qué hacer cuando un objeto no se va a volver a utilizar? Necesario liberar memoria.

En el lenguaje de programación Java, la recolección y liberación de memoria es responsabilidad de un thread (proceso especial que está continuamente funcionando), llamado automatic garbage collector (recolector automático de basura), que monitoriza continuamente los objetos referenciados y los que no.

Los objetos que ya no están referenciados son excluidos de la memoria asignada al programa, liberándola para que pueda ser reutilizada por el mismo u otro programa que esté en ejecución en el computador en ese momento.

9.7. LIBRERÍAS Y PAQUETES DE CLASES. UTILIZACIÓN Y CREACIÓN.

9.7.1. PAQUETE DE CLASES.

Java permite agrupar las clases en paquetes, en la cual se puede agrupar también otros ficheros asociado al proyecto.

Cada clase se inicia con la declaración del paquete en donde se encuentra, partiendo del proyecto raíz del mismo.

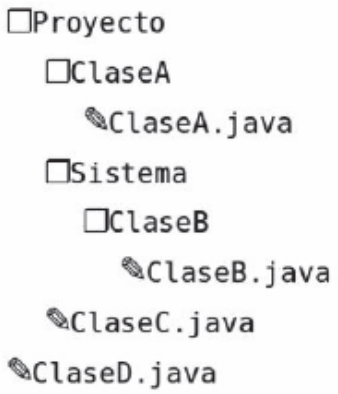
```
package nombrePaquete;
```

```
package proyecto;
class ClaseA
{}

package proyecto.sistema.ClaseB;
public class ClaseB
{}

package proyecto.ClaseC;
class ClaseC
{}

class ClaseD
{}
```



9.7.2. LIBRERÍAS.

Las librerías suministran un código ya implementado que permiten a los desarrolladores su reutilización.

Una librería está conformada con una serie de paquetes, todos con un propósito común.

Estas librerías podrán ser utilizadas por el propio desarrolladores o terceros.

9.7.2.1 UTILIZACIÓN DE LIBRERÍAS.

Para utilizar una librería se utiliza la palabra clave “import”, la cuál establece la ruta del paquete o clase de la que se quiere hacer uso.

LIBRERÍAS MÁS UTILIZADAS:

- **import java.math.*** : Librería especializada en operaciones matemáticas.
- **import java.util.*** : se importan todas las utilidades suministradas en la librería util de Java.
- **import java.util.Random**. Uso de una parte de la librería “util" para obtener números aleatorios.

9.7.2.2. DESCARGAR Y UTILIZAR LIBRERÍA EXTERNA: JODA TIME

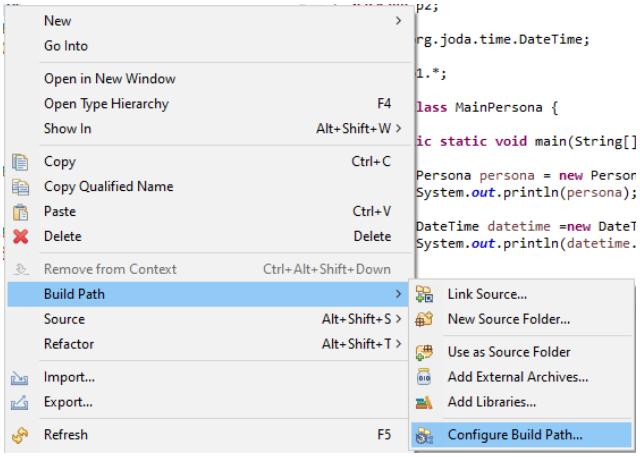
REPOSITORIO MUY COMPLETO: https://mvnrepository.com/

LIBRERÍA JODA-TME: https://mvnrepository.com/artifact/joda-time/joda-time

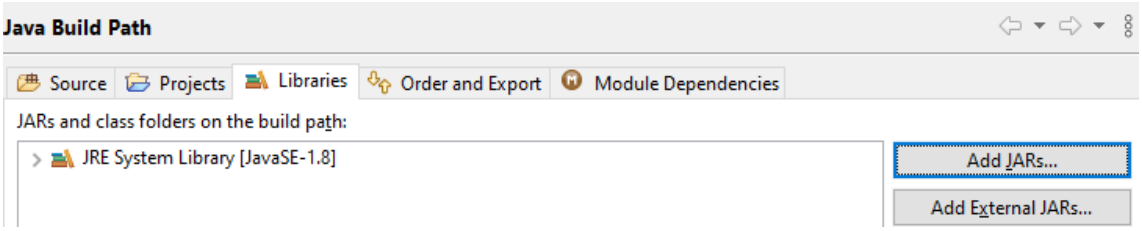
PAGINA OFICIAL JODA-TIME: https://www.joda.org/joda-time/

PASOS:

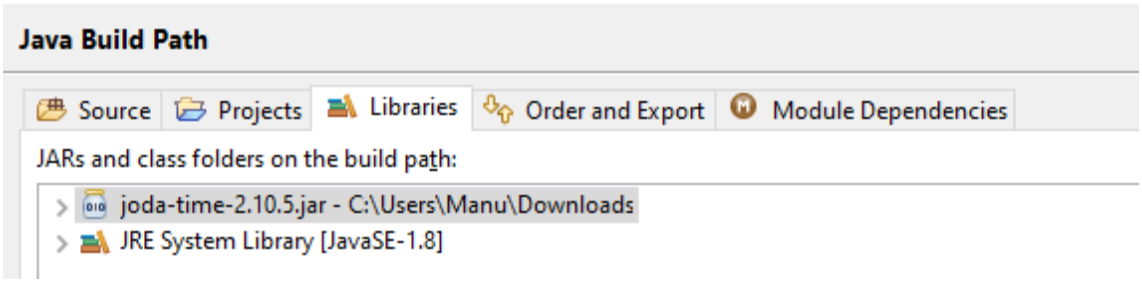
1. Descargar .jar desde https://mvnrepository.com/artifact/joda-time/joda-time/2.10.5
2. Importar en el proyecto de Eclipse:
    a. Click derecho sobre el proyecto > Build Path > Configure Build Path



b. Libraries > Add External JARs…



c. Select joda-time-2.10.5.jar > Abrir



3. Crear un objeto de la clase DateTime situado en el paquete: org.joda.time

```
package p2;

import org.joda.time.DateTime;

public class MainPersona {

    public static void main(String[] args) {

        DateTime datetime =new DateTime();
        System.out.println(datetime.getDayOfWeek());

    }
}
```