

## Curso de Java a distancia

### Clase 26: Colecciones: HashSet, TreeSet y LinkedHashS

La diferencia fundamental entre las clases HashSet, TreeSet, LinkedHashSet con respecto a las listas ArrayList y LinkedList es que no puede haber elementos repetidos en las colecciones que implementan la interfaz Set.

A su vez se han creado estas tres clases que tienen pequeñas diferencias entre una y otras:

- HashSet: El conjunto de datos no se almacena en un orden específico, si bien se garantiza que no hay duplicados.
- TreeSet: Los elementos del conjunto se almacenan de menor a mayor.
- LinkedHashSet: Los elementos del conjunto se encuentran en el orden que se insertan, similar a una lista pero sin dejar ingresar valores repetido.

El siguiente programa muestra la sintaxis para crear objetos de estas clases y los métodos principales que disponen:

#### Programa:

```
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class PruebaSet {

    public static void main(String[] args) {
        Set<Integer> conjunto1 = new HashSet<Integer>();
        conjunto1.add(20);
        conjunto1.add(10);
        conjunto1.add(1);
        conjunto1.add(5);
        // El valor 20 no se inserta en el conjunto ya que se encuentra repetido
        conjunto1.add(20);
        // La impresión no asegura un orden específico
        for (int elemento : conjunto1)
            System.out.print(elemento + " - ");
        System.out.println();

        Set<Integer> conjunto2 = new TreeSet<Integer>();
        conjunto2.add(20);
```

```

conjunto2.add(10);
conjunto2.add(1);
conjunto2.add(5);
// El valor 20 no se inserta en el conjunto ya que se encuentra repetido
conjunto2.add(20);
// Los elementos se muestran de menor a mayor
for (int elemento : conjunto2)
    System.out.print(elemento + " - ");
System.out.println();

```

```

Set<Integer> conjunto3 = new LinkedHashSet<Integer>();
conjunto3.add(20);
conjunto3.add(10);
conjunto3.add(1);
conjunto3.add(5);
// El valor 20 no se inserta en el conjunto ya que se encuentra repetido
conjunto3.add(20);
// Los elementos se muestran en el orden que se insertaron
for (int elemento : conjunto3)
    System.out.print(elemento + " - ");
System.out.println();

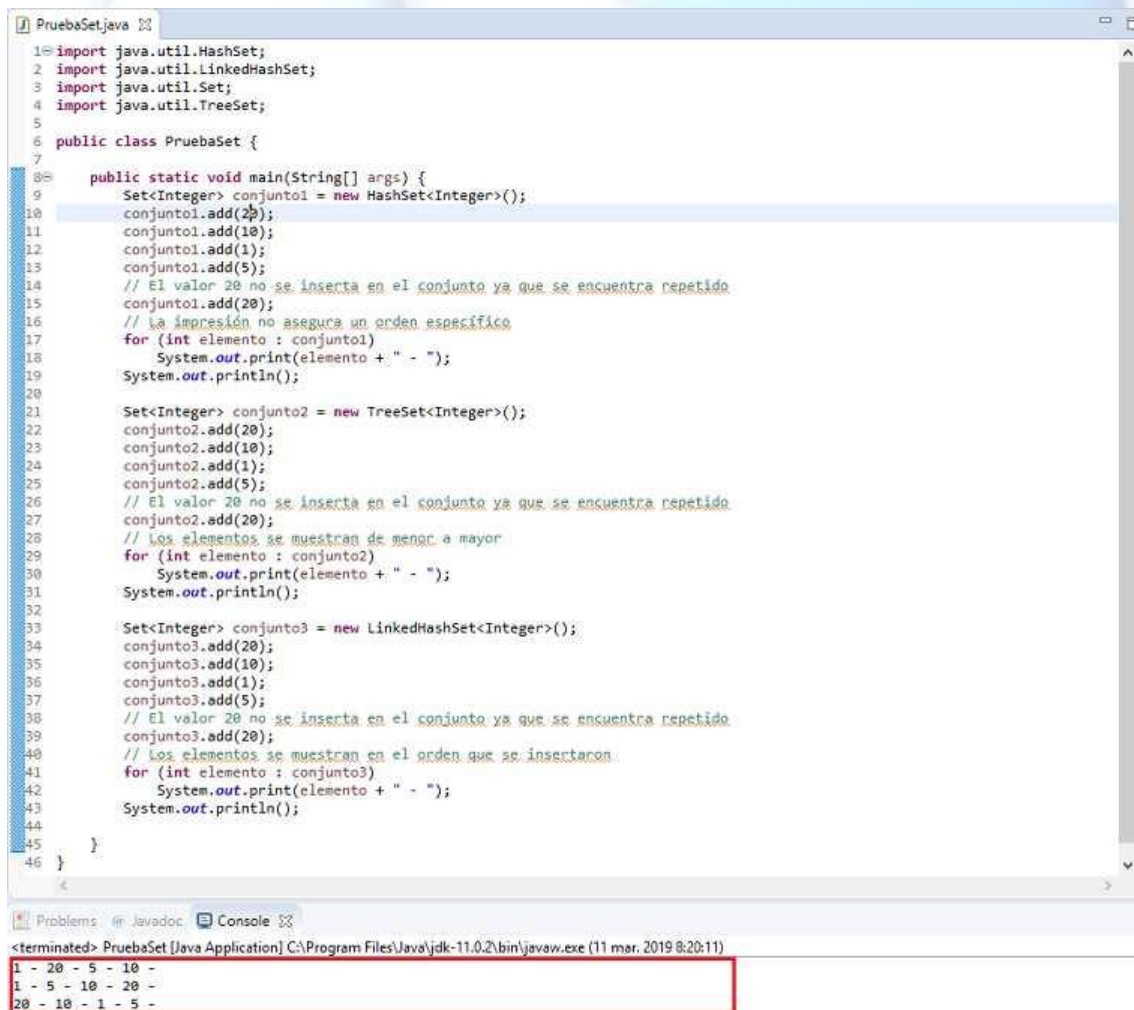
```

```

    }
}

```

El resultado de ejecutar el programa es similar a:



```

1 import java.util.HashSet;
2 import java.util.LinkedHashSet;
3 import java.util.Set;
4 import java.util.TreeSet;
5
6 public class PruebaSet {
7
8     public static void main(String[] args) {
9         Set<Integer> conjunto1 = new HashSet<Integer>();
10        conjunto1.add(20);
11        conjunto1.add(10);
12        conjunto1.add(1);
13        conjunto1.add(5);
14        // El valor 20 no se inserta en el conjunto ya que se encuentra repetido
15        conjunto1.add(20);
16        // La impresión no asegura un orden específico
17        for (int elemento : conjunto1)
18            System.out.print(elemento + " - ");
19        System.out.println();
20
21        Set<Integer> conjunto2 = new TreeSet<Integer>();
22        conjunto2.add(20);
23        conjunto2.add(10);
24        conjunto2.add(1);
25        conjunto2.add(5);
26        // El valor 20 no se inserta en el conjunto ya que se encuentra repetido
27        conjunto2.add(20);
28        // Los elementos se muestran de menor a mayor
29        for (int elemento : conjunto2)
30            System.out.print(elemento + " - ");
31        System.out.println();
32
33        Set<Integer> conjunto3 = new LinkedHashSet<Integer>();
34        conjunto3.add(20);
35        conjunto3.add(10);
36        conjunto3.add(1);
37        conjunto3.add(5);
38        // El valor 20 no se inserta en el conjunto ya que se encuentra repetido
39        conjunto3.add(20);
40        // Los elementos se muestran en el orden que se insertaron
41        for (int elemento : conjunto3)
42            System.out.print(elemento + " - ");
43        System.out.println();
44    }
45 }
46

```

terminated> PruebaSet [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (11 mar. 2019 9:20:11)

```

1 - 20 - 5 - 10 -
1 - 5 - 10 - 20 -
20 - 10 - 1 - 5 -

```

## Métodos más comunes

Los métodos más comunes que tienen estas clases son:

- size: Retorna la cantidad de elementos del conjunto.
- clear: Elimina todos los elementos.
- remove: Elimina el elemento si existe en el conjunto:

```
lista1.remove(20);
```

- isEmpty: Nos informa si la lista está vacía.
- contains: Le pasamos como parámetro el dato a buscar en el conjunto:

```
if (conjunto1.contains(20))  
...
```

Más datos podemos conseguir visitando la documentación oficial de las clases HashSet, TreeSet y LinkedHashSet.

## Problema

Generar una lista de 10 valores enteros comprendidos entre 1 y 100. Validar que no se repitan, para esto utilizar la ayuda de una de las colecciones de conjuntos visto en este concepto.

### Programa:

```
import java.util.Set;  
import java.util.TreeSet;  
  
public class Lista10Valores {  
    public static void main(String[] args) {  
        Set<Integer> conjunto1 = new TreeSet<Integer>();  
        while (conjunto1.size() < 10) {  
            int aleatorio = (int) (Math.random() * 100) + 1;  
            conjunto1.add(aleatorio);  
        }  
        System.out.println(conjunto1);  
    }  
}
```

Dentro de un while mientras el objeto 'conjunto1' tenga menos de 10 elementos, procedemos a generar otro valor aleatorio y lo agregamos al conjunto, como sabemos si el valor ya existe en el conjunto1 luego el método 'add' no lo agrega:

```
while (conjunto1.size() < 10) {  
    int aleatorio = (int) (Math.random() * 100) + 1;  
    conjunto1.add(aleatorio);  
}
```

Podemos recorrer el conjunto para imprimirlo mediante un for o inclusive utilizar el método 'println':

```
System.out.println(conjunto1);
```

# Colecciones: HashMap, TreeMap y LinkedHashMap

Estas clases: HashMap, TreeMap y LinkedHashMap nos permite almacenar elementos asociando a cada clave un valor.

Para cada clave tenemos un valor asociado. Podemos después buscar fácilmente un valor para una determinada clave. Las claves en el diccionario no pueden repetirse.

Algunos ejemplos donde podríamos usar un Mapa:

- Guardar en la clave las extensiones de archivos y en el valor los nombres de archivos que lo pueden abrir
- En una agenda podemos guardar como 'clave' la fecha y hora y las actividades en el 'valor'.

## Problema

Almacenar un diccionario las palabras en castellano como 'clave' y las traducciones de las mismas en el 'valor'. Probar los métodos más significativos de la clase HashMap.

## Programa:

```
import java.util.HashMap;
import java.util.Map;

public class PruebaHashMap {

    public static void main(String[] args) {
        Map<String, String> mapa1 = new HashMap<String, String>();
        mapa1.put("rojo", "red");
        mapa1.put("verde", "green");
        mapa1.put("azul", "blue");
        mapa1.put("blanco", "white");
        System.out.println("Listado completo de valores");
        for (String valor : mapa1.values())
            System.out.print(valor + "-");
        System.out.println();
        System.out.println("Listado completo de claves");
        for (String clave : mapa1.keySet())
            System.out.print(clave + "-");
        System.out.println();
        System.out.println("La traducción de 'rojo' es:" + mapa1.get("rojo"));
        if (mapa1.containsKey("negro"))
            System.out.println("No existe la clave 'negro'");
        System.out.println("La traducción de 'marron' es:" + mapa1.getOrDefault("marrón", "No existe la clave marrón"));
        mapa1.remove("rojo");
        System.out.println(mapa1);
    }
}
```

La interfaz Map deben implementar estas clases, luego si creamos un objeto de la clase HashMap debemos hacerlo con la siguiente sintaxis:

```
Map<String, String> mapa1 = new HashMap<String, String>();
```

La clase HashMap utiliza datos genéricos tanto para la clave como para el valor, en este ejemplo la clave y el valor son datos de tipo String.

Mediante el método put agregamos un elemento a la colección de tipo HashMap:

```
mapa1.put("rojo", "red");
mapa1.put("verde", "green");
mapa1.put("azul", "blue");
mapa1.put("blanco", "white");
```

Para imprimir todos los valores del mapa lo recorreremos mediante un for:  
for (String valor : mapa1.values()) System.out.print(valor + "-");  
De forma similar si queremos recorrer todas las claves del mapa:

```
for (String clave : mapa1.keySet())
    System.out.print(clave + "-");
```

Para recuperar un valor para una determinada clave llamamos al método 'get' y le pasamos la clave a buscar, si dicha clave no existe en el mapa se nos retorna el valor 'null':

```
System.out.println("La traducción de 'rojo' es:" + mapa1.get("rojo"));
```

Si queremos verificar si una determinada clave existe en el mapa lo hacemos mediante el método 'containsKey':

```
if (mapa1.containsKey("negro"))
    System.out.println("No existe la clave 'negro'");
```

Una variante del método 'get' es 'getOrDefault' que nos retorna el segundo parámetro si no encuentra la clave en el mapa:

```
System.out.println("La traducción de 'marron' es:" +
    mapa1.getOrDefault("marrón", "No existe la clave marrón"));
```

Para eliminar un elemento de la colección debemos hacer uso del método 'remove', pasamos una clave del mapa:

```
mapa1.remove("rojo");
```

Para imprimir el mapa completo en la Consola podemos hacer uso del método 'println':

```
System.out.println(mapa1);
```

Hemos utilizado la clase HashMap para resolver el problema. La clase TreeMap es idéntica a HashMap con la salvedad que mantiene ordenado los datos por la clave.

Finalmente la clase LinkedHashMap mantiene ordenado los elementos del mapa según el orden de inserción.

## Creación de paquetes (package)

En programas pequeños o cuando estamos aprendiendo a programar en Java es común definir la clase o conjunto de clases del proyecto en el paquete por defecto (default package), es así como lo hemos hecho en este tutorial.

Cuando disponemos las clases en el paquete por defecto no pueden ser reutilizadas en otros proyectos.

Los paquetes son un mecanismo de Java que nos permite agrupar un conjunto de clases relacionadas con un nombre de paquete, luego dicho paquete puede compilarse y ser reutilizado en otros proyectos que desarrollemos nosotros u otras empresas.



Para definir una clase dentro de un determinado paquete debemos agregar en la primer línea de nuestro código fuente la palabra clave 'package' y seguidamente el nombre de paquete.

Por convención se propone que todo paquete comience por la URL de la empresa (en forma invertida) que lo desarrolla y seguidamente otros nombres que nos den una idea de que clases agrupa dicho paquete.

Para ver los pasos desde Eclipse crearemos dos paquetes.

## Problema

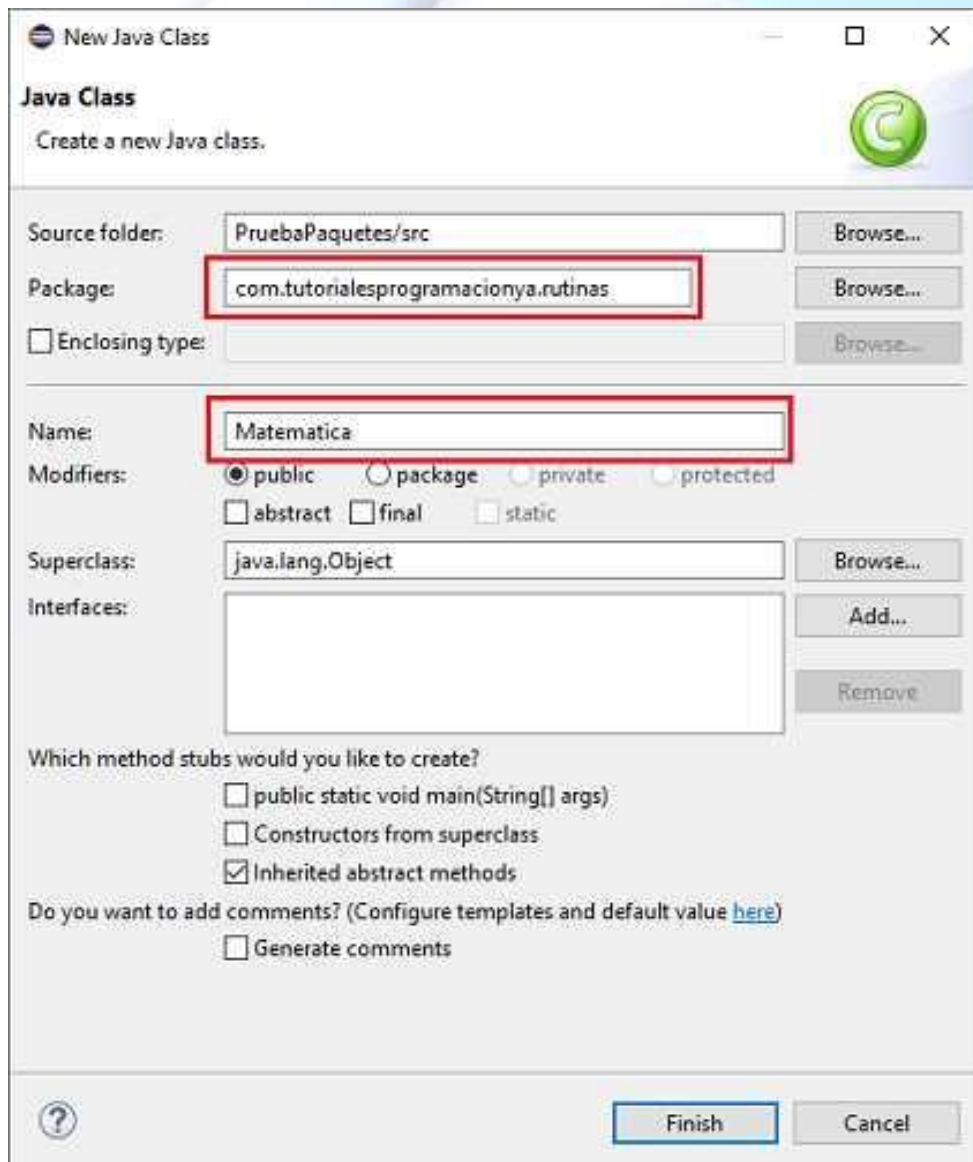
Crear dos clase llamadas 'Matematica' y 'Cadena', definir una serie de métodos estáticos en cada una de ellas.

Disponer las dos clases en el paquete: `com.tutorialesprogramacionya.rutinas`

Crear un segundo paquete llamado '`com.tutorialesprogramacionya.calculadora`' en el mismo proyecto y acceder a las clases del primer paquete.

Veamos los pasos para resolver el problema:

- Creamos un proyecto llamado 'PruebaPaquetes' en Eclipse.
- Creamos la clase 'Matematica' en el paquete: `com.tutorialesprogramacionya.rutinas`



- El mismo paso hacemos para crear la clase 'Cadena' en el paquete: com.tutorialesprogramacionya.rutinas

**New Java Class**

Java Class  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

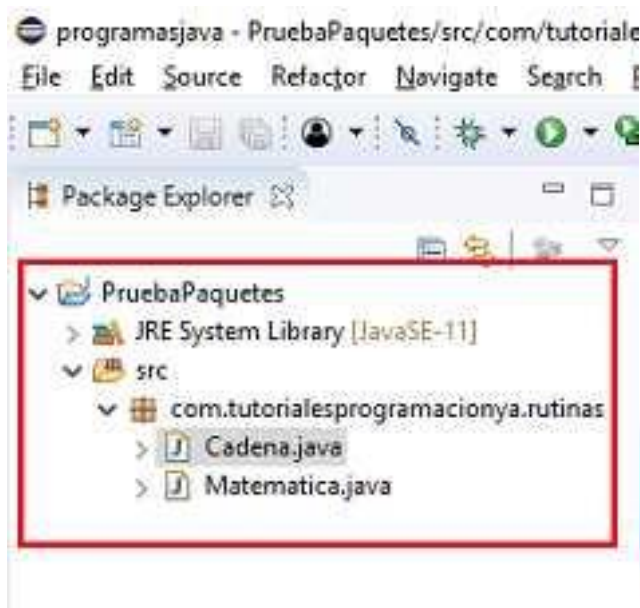
Interfaces:

Which method stubs would you like to create?

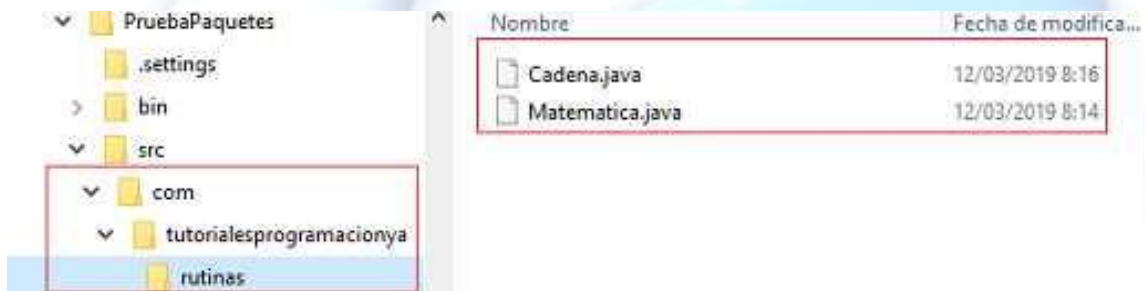
☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Si vemos el Package Explorer de Eclipse podremos ver la localización de las dos clases que hemos creados en el paquete: com.tutorialesprogramacionya.rutinas:



Si vamos al administrador de archivos del sistema operativo podemos ver como Java organiza los paquetes en directorios (los caracteres punto indican que hay un nuevo subdirectorio):



- Codificamos algunos métodos en cada clase:

```
package com.tutorialesprogramacionya.rutinas;
```

```
public class Matematica {
```

```
    public static int sumar(int x, int y) {
        return x + y;
    }
```

```
    public static int restar(int x, int y) {
        return x - y;
    }
```

```
}
```

```
package com.tutorialesprogramacionya.rutinas;
```

```
public class Cadena {
```

```
    public static String mayuscula(String cadena) {
        return cadena.toUpperCase();
    }
```

```
}
```

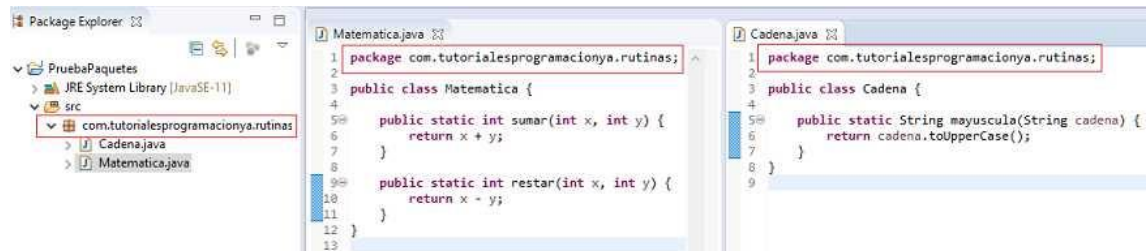
El mismo Eclipse nos codifica el nombre del paquete donde se almacena la clase:

```
package com.tutorialesprogramacionya.rutinas;
```

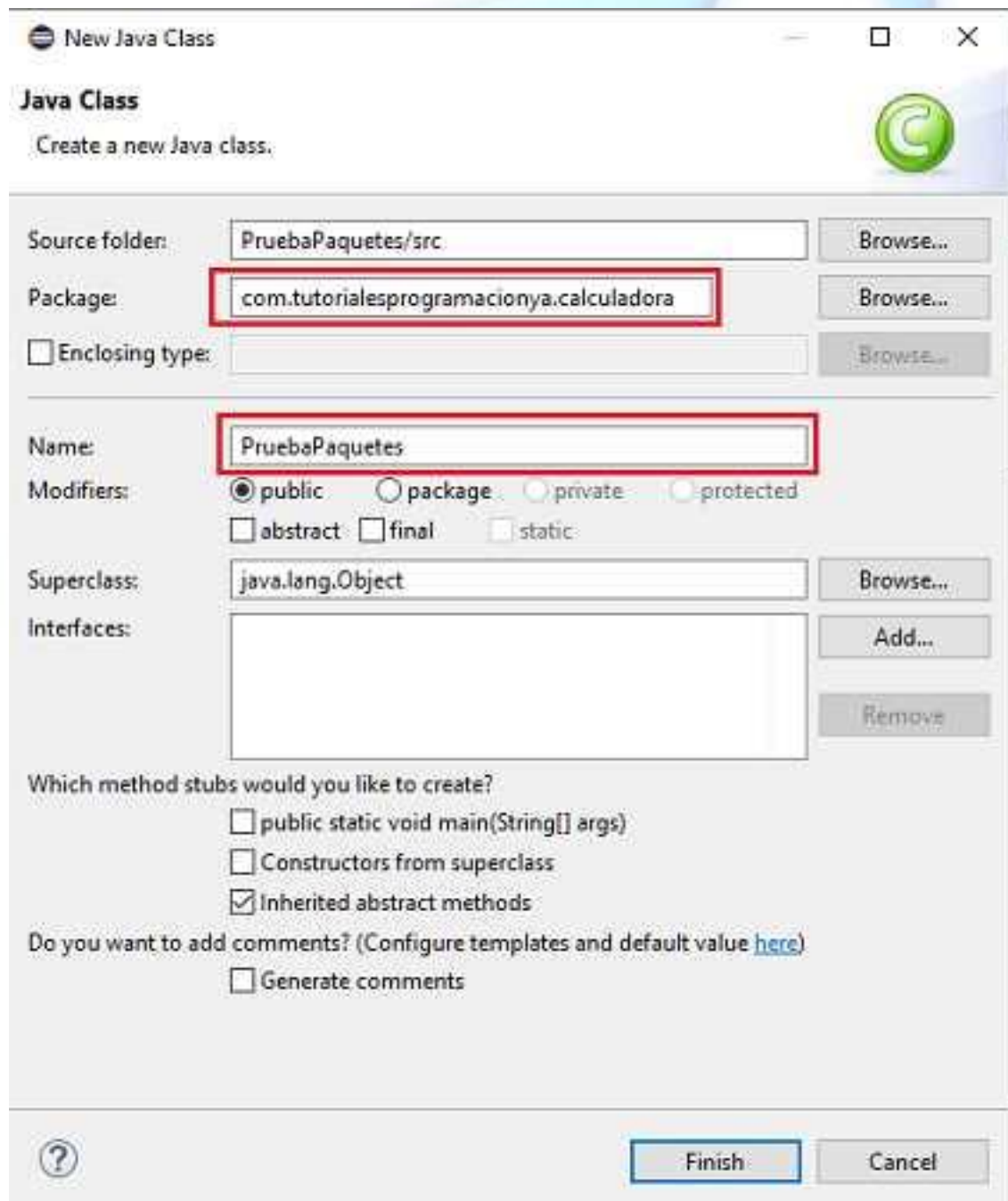


Si necesitamos hacer import se deben codificar después de la línea de definición del paquete.

Hasta ahora tenemos creado el paquete `com.tutorialesprogramacionya.rutinas` y sus dos clases:



- Ahora crearemos un segundo paquete llamado `com.tutorialesprogramacionya.calculadora` donde implementaremos una clase llamada 'PruebaPaquetes' que consuma las clases del otro paquete:



Codificamos ahora la clase 'PruebaPaquetes':

```

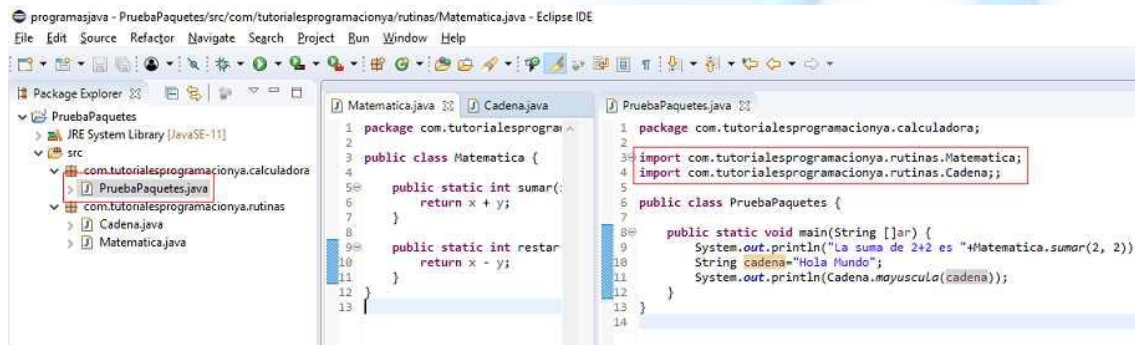
package com.tutorialesprogramacionya.calculadora;

import com.tutorialesprogramacionya.rutinas.Matematica;
import com.tutorialesprogramacionya.rutinas.Cadena;

public class PruebaPaquetes {

    public static void main(String []ar) {
        System.out.println("La suma de 2+2 es "+Matematica.sumar(2, 2));
        String cadena="Hola Mundo";
        System.out.println(Cadena.mayuscula(cadena));
    }
}

```



Para poder acceder desde el paquete 'com.tutorialesprogramacionya.calculadora' a las clases contenidas en el paquete 'com.tutorialesprogramacionya.rutinas' debemos importarlas una a una:

```

import com.tutorialesprogramacionya.rutinas.Matematica;
import com.tutorialesprogramacionya.rutinas.Cadena;

```

O si queremos importarlas a todas de una vez podemos escribir un solo import:

```

import com.tutorialesprogramacionya.rutinas.*;

```

Ahora ya podemos ejecutar la clase 'PruebaPaquetes' y ver el resultado.

Organizar las clases en paquetes es de vital importancia en proyectos grandes donde puede necesitarse codificar miles de clases. El API de Java es un buen ejemplo de como se organizan las clases en paquetes. Recordemos algunos paquetes que hemos utilizado:

- Paquete java.util donde se implementan las clases, interfaces, enumeraciones etc. para administrar colecciones vistas en los conceptos anteriores como son las clases ArrayList, LinkedList Stack etc.
- Paquete javax.swing donde se declaran entre otras las clase JFrame, JButton, JLabel etc.
- Paquete java.lang que es el único paquete de Java que no requiere efectuar el import ya que el compilador lo hace en forma automática. Este paquete agrupa las clases de uno más común como pueden ser : Object, String, System, Integer, Double etc.
- Paquete java.sql que contiene clases para el acceso a bases de datos relacionales.

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar [info@institutosanisidro.com.ar](mailto:info@institutosanisidro.com.ar)