



Tema 6: Programación Orientada a Objetos



1. Introducción

■ Programación orientada a procedimientos:

- Datos
- Subprogramas

Programación orientada a objetos

Datos

+

Subprogramas que los manejan

=

OBJETO



Programación Orientada a Objetos

Las características más importantes de la programación orientada a objetos son las siguientes:

Encapsulamiento

Polimorfismo

Herencia



Encapsulamiento

- # Propiedad que tienen los objetos de ocultar sus atributos y/o métodos, a otras partes del programa u otros objetos.
- # Es la propiedad que nos va a permitir establecer la interfaz que se va a ver del objeto, no vemos como se ha programado el objeto, sino las propiedades y métodos declarados públicos.



Polimorfismo

- # El polimorfismo permite que
 - # objetos distintos puedan tener métodos con el mismo nombre
 - # un mismo objeto puede tener nombres de métodos idénticos pero con distintos parámetros.

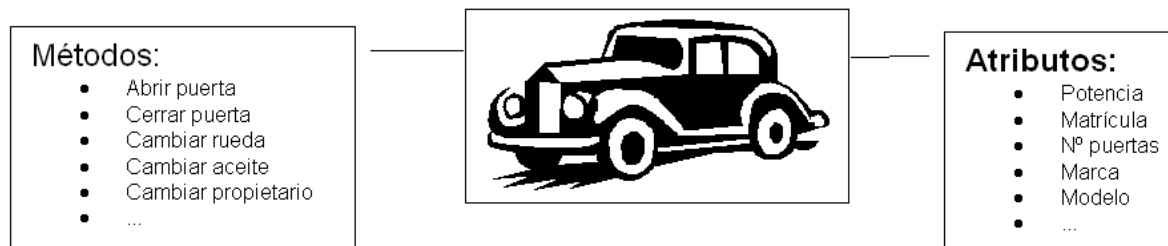


Herencia

- Permite definir nuevas clases a partir de otras ya existentes. La nueva clase hereda de su antecesora sus ATRIBUTOS y MÉTODOS y puede definir nuevos o incluso puede redefinir los ya existentes.
- Esta propiedad hace posible la reutilización del código, al aprovechar el código de clases ya existentes para crear nuevas clases.

2. Concepto de objeto

- Un objeto es una persona, animal o cosa que se distingue de otros objetos por:
 - Tener unas determinadas “propiedades” (Atributos)
 - Se pueden realizar distintas operaciones con/sobre ese objeto (métodos).





Clases y objetos

- **Clase:** define las características y métodos generales de los objetos que la integran. Es como una plantilla que lo define.
- **Objeto:** es un miembro o instancia de esa clase con unas características particulares.

ANALOGÍA		
OBJETO	↔	VARIABLE
CLASE	↔	TIPO

Definición de una clase en JAVA

[Acceso] class *nom_clase* {

Nombre que se da a la clase para poder crear objetos de la misma

//Atributos

Propiedades, que permiten diferenciar un objeto de otro

//Métodos

Operaciones que se pueden efectuar con los datos y que son utilizables por todos los objetos pertenecientes a la clase

}



Interfaz de una clase

- Define la parte de la clase que será accesible desde el exterior, tanto en los atributos como en los métodos



Interfaz de la clase

- El ámbito definirá la visibilidad o no de los atributos o métodos dentro de una clase:
 - **Public:** Ninguna restricción de acceso
 - **Private:** Accesible únicamente en la clase que contiene la declaración
 - **Protected:** Accesible en la clase que contiene la declaración y en las heredadas.



Declaración de atributos de la clase en JAVA

- Se declaran igual que cualquier dato:

public private protected	<i>tipo_dato nombre;</i>
---	--------------------------



Declaración de métodos de una clase

- Igual que la declaración de un método o subprograma (procedimiento o función), con el acceso (public, private, protected) delante

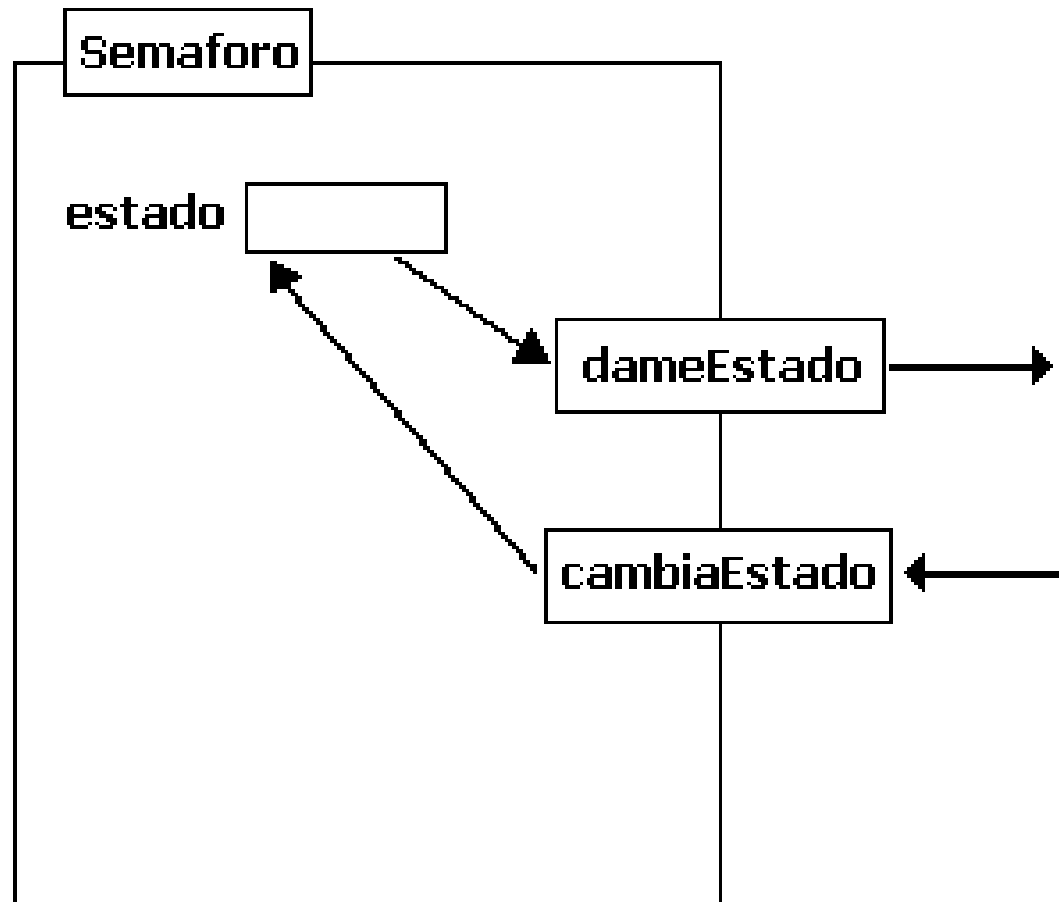
```
[Acceso] tipoDato nombreMetodo ([parámetros]){  
    [return dato;]  
}
```



Un ejemplo: la clase semáforo

```
public class Semaforo {  
  
    private String estado = "Rojo";  
  
    public String dameEstado() {  
        return estado;  
    }  
  
    public void cambiaEstado() {  
        if (this.estado.equals("Rojo")) {  
            estado = "Verde";  
        } else {  
            if (this.estado.equals("Amarillo")) {  
                estado = "Rojo";  
            } else {  
                estado = "Amarillo";  
            }  
        }  
    }  
}
```

Representación gráfica





Crear instancias de la clase Semáforo

- Consiste en crear objetos de la clase semáforo de forma que evolucionen de forma independiente.
- Cada semáforo se llamará de una manera, estará en un estado en un momento determinado que se podrá mostrar o cambiar de forma independiente al resto de los semáforos



Crear objetos de una clase en JAVA

- Dos formas:

- Definir y crear el objeto con una única instrucción

NombreClase nom_obj = new nom_clase();

- Definir el objeto y luego crearlo

NombreClase nom_obj;
nom_obj = new NombreClase();



Acceso a un objeto

- Solo se podrán acceder desde la aplicación a aquellas propiedades y métodos que sean **“public”**.
- La forma de acceder es a través del nombre del objeto:

NombreObjeto.metodo()
NombreObjeto.propiedad



Encapsulamiento

- Es la propiedad que tienen los objetos de ocultar sus atributos y/o métodos, a otras partes del programa u otros objetos.
- En Java se hace poniendo delante de la definición del atributo o método el ámbito del mismo (private, public).
- En nuestro ejemplo anterior, no es visible el atributo ***estado*** desde fuera al ser privado, aunque puede ser consultado o cambiado a través de los métodos públicos ***dameEstado*** y ***cambiaEstado***.



Ejemplo de uso de la clase Semáforo

Crear una pequeña aplicación que cree tres semáforos (s1, s2 y s3) y que muestre un menú:

1. Mostrar estado: muestra el estado actual de todos los semáforos
2. Cambiar estado: pide el nº de semáforo a cambiar el estado y lo cambia
3. Salir

La aplicación acabará al seleccionar la opción Salir del menú



Constructores

- Son métodos especiales que inicializan los objetos al ser creados, dándole valores a sus atributos. Ese método se ejecuta al hacer el new
- Su sintaxis es como un método normal:
 - Sin especificación de acceso
 - No devuelven ningún valor
 - Su nombre coincide con el de la clase
- ***Suelen estar sobrecargados*** para dar más posibilidades de inicialización a los objetos.



Tipos enumerados

Son conjuntos de valores constantes para los que no existe un tipo predefinido.

Sintaxis:

enum *nombreTipo* {*CTE1*,...,*CTEN*}

Ejemplo:

enum diaSem {LUNES,MARTES,MIERCOLES,JUEVES,VIERNES,SABADO,DOMINGO}



Definición y uso

```
enum diaSem {LUNES,MARTES,MIERCOLES,JUEVES,VIERNES,SABADO,DOMINGO}
```

```
diaSem d;
```

```
d=diaSem.MIERCOLES;
```

```
if (d==diasem.VIERNES){
```

```
    System.out.println("Por fin es viernes");
```

```
}
```



La clase Semaforo con tipo enumerado

- En el caso del semáforo, el estado puede ser ROJO, AMARILLO o VERDE.
- Vamos a rehacer la clase utilizando un tipo enumerado.
- Esto no afectará a la aplicación, puesto que no vamos a modificar la interfaz.

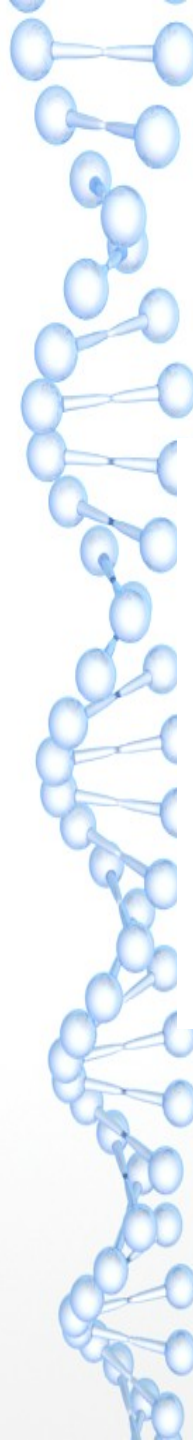


Definir el tipo enumerado

Los tipos enumerados sirven para restringir el contenido de una variable a una serie de valores predefinidos, ayudando a reducir los errores en nuestro código.

Se pueden definir fuera o dentro de una clase. Su sintaxis es:

```
[public | private] enum nombreTipoEnumerado { ELEMENTO1, ELEMENTO2, ..., ELEMENTOn };
```



```
public enum ColorSem {  
    ROJO,  
    AMARILLO,  
    VERDE  
}
```

```
public class Semaforo2 {  
    private ColorSem estado ;  
  
    public Semaforo2() {  
        estado=ColorSem.ROJO;  
    }  
  
    public ColorSem getEstado() {  
        return estado;  
    }  
  
    public void cambiaEstado() {  
        if (estado==ColorSem.ROJO) {  
            estado=ColorSem.VERDE;  
        } else {  
            if (estado==ColorSem.AMARILLO) {  
                this.estado=ColorSem.ROJO;  
            } else {  
                this.estado=ColorSem.AMARILLO;  
            }  
        }  
    }  
}
```

Ejercicio

Identificar si hay algo mal en este código:

```
public class RadioCasette
{
    boolean puedeGrabar = false;

    void escucharCinta()
    {
        System.out.println("Escuchándose cinta");
    }

    void grabarCinta()
    {
        System.out.println("Grabándose cinta");
    }
}

public class Test
{
    public static void main(String[] args)
    {
        r.escucharCinta();

        if(r.puedeGrabar)
            r.grabarCinta();
    }
}
```

Ejercicio (solución)

Estaba mal. No habíamos creado el objeto r.

```
public class RadioCasette
```

```
{  
    boolean puedeGrabar = false;  
  
    void escucharCinta()  
    {  
        System.out.println("Escuchándose cinta");  
    }  
  
    void grabarCinta()  
    {  
        System.out.println("Grabándose cinta");  
    }  
}
```

```
public class Test
```

```
{  
    public static void main(String[] args)  
    {  
        RadioCasette r = new RadioCasette();  
        r.escucharCinta();  
  
        if(r.puedeGrabar)  
            r.grabarCinta();  
    }  
}
```

Ejercicio

Identificar si hay algo mal en este código:

```
public class ReproductorDVD
{
    boolean puedeGrabar = false;

    void grabarDVD()
    {
        System.out.println("Grabándose");
    }
}

public class Test
{
    public static void main(String[] args)
    {
        ReproductorDVD r = new ReproductorDVD();

        if(r.puedeGrabar)
            r.grabarDVD();
    }
}
```

Ejercicio

- Estaba mal. Se estaba llamando a un método inexistente.

```
public class ReproductorDVD
{
    boolean puedeGrabar = false;

    void verDVD()
    {
        System.out.println("Viéndose");
    }

    void grabarDVD()
    {
        System.out.println("Grabándose");
    }
}

public class Test
{
    public static void main(String[] args)
    {
        ReproductorDVD r = new ReproductorDVD();

        r.verDVD();

        if(r.puedeGrabar)
            r.grabarDVD();
    }
}
```


Ejercicio

- Identificar si hay algo mal en este código, suponiendo que la clase Rectangulo existe.

```
public class Temp
{
    public static void main(String[] args)
    {
        Rectangulo miRect;
        miRect.ancho = 40;
        miRect.alto = 50;
        System.out.println("El área del rectángulo es" + miRect.area());
    }
}
```


Ejercicio (solución)

- Estaba mal. El objeto miRect no está inicializado, por tanto vale null. Con null no podemos hablarnos.

```
public class Temp
{
    public static void main(String[] args)
    {
        Rectangulo miRect = new Rectangulo();
        miRect.ancho = 40;
        miRect.alto = 50;
        System.out.println("El área del rectángulo es" + miRect.area());
    }
}
```



Polimorfismo

- El polimorfismo permite que:
 - objetos distintos puedan tener métodos con el mismo nombre.
 - o incluso un mismo objeto puede tener nombres de métodos idénticos pero con distintos parámetros (sobrecarga).

Polimorfismo

- El polimorfismo permite que:
 - objetos distintos puedan tener métodos con el mismo nombre.
 - o incluso un mismo objeto puede tener nombres de métodos idénticos pero con distintos parámetros (sobrecarga).

Sobrecarga de métodos

- Es un mecanismo que permite en una clase definir ***varios métodos con el mismo nombre.***
- ***Los parámetros*** de los métodos sobrecargados ***no pueden ser idénticos***, para que el compilador pueda identificar al método llamado
- El compilador identifica a un método por su firma, que se compone de:
 - Nombre del método
 - Número de parámetros
 - Tipo de parámetros (por orden de colocación)

Un ejemplo: la clase Persona

```
public class Persona {  
  
    private String nombre;  
    private String Apellido1;  
    private String Apellido2;  
    private int edad;  
  
    public void dameDatos(String nombre, String ap1, String ap2) {  
        this.nombre = nombre;  
        this.Apellido1 = ap1;  
        this.Apellido2 = ap2;  
    }  
  
    public void dameDatos(int edad) {  
        this.edad = edad;  
    }  
  
    public void dameDatos(String nombre, String ap1, String ap2, int edad) {  
        this.nombre = nombre;  
        this.Apellido1 = ap1;  
        this.Apellido2 = ap2;  
        this.edad = edad;  
    }  
}
```

Constructores

- Son métodos especiales que inicializan los objetos al ser creados, dándole valores a sus atributos.
- Su sintaxis es como un método normal:
 - Sin especificación de acceso
 - No devuelven ningún valor
 - Su nombre coincide con el de la clase
- ***Suelen estar sobrecargados*** para dar más posibilidades de inicialización a los objetos.

La clase persona

- En el ejemplo anterior:
 - Primero creábamos el objeto
 - Posteriormente llamábamos a un método para inicializarlo

```
Persona p=new Persona();  
p.dameDatos(22);  
p.dameDatos("Jorge", "Perez", "Gonzalez");
```

La clase Persona con constructores

```
public class Persona2 {  
  
    private String nombre;  
    private String Apellido1;  
    private String Apellido2;  
    private int edad;  
  
    public Persona2(String nombre, String Apellido1, String Apellido2) {  
        this.nombre = nombre;  
        this.Apellido1 = Apellido1;  
        this.Apellido2 = Apellido2;  
    }  
    public Persona2(int edad) {  
        this.edad = edad;  
    }  
    public Persona2(String nombre, String Apellido1, String Apellido2, int edad) {  
        this.nombre = nombre;  
        this.Apellido1 = Apellido1;  
        this.Apellido2 = Apellido2;  
        this.edad = edad;  
    }  
    public Persona2() {  
        nombre = "Luisa";  
        Apellido1 = "Lopez";  
        Apellido2 = "Gonzalez";  
        edad = 23;  
    }  
}
```


Otros métodos de la clase Persona

```
public String suNombre() {  
    return nombre;  
}  
  
public String suApellido1() {  
    return Apellido1;  
}  
  
public String suApellido2() {  
    return Apellido2;  
}  
  
public int suEdad() {  
    return edad;  
}
```

Utilizando los constructores de la clase Persona

```
Persona2 p = new Persona2();
System.out.println("Datos:");
System.out.println("Nombre: " + p.suNombre());
System.out.println("Apellidos: " + p.suApellido1() + " " + p.suApellido2());
System.out.println("Edad: " + p.suEdad());
Persona2 p1 = new Persona2("Juan", "Lopez", "Lopez");
System.out.println("Datos:");
System.out.println("Nombre: " + p1.suNombre());
System.out.println("Apellidos: " + p1.suApellido1() + " " + p1.suApellido2());
System.out.println("Edad: " + p1.suEdad());
Persona2 p3 = new Persona2(33);
System.out.println("Datos:");
System.out.println("Nombre: " + p3.suNombre());
System.out.println("Apellidos: " + p3.suApellido1() + " " + p3.suApellido2());
System.out.println("Edad: " + p3.suEdad());
```

El resultado...

```
run:
Datos:
Nombre: Luisa
Apellidos: Lopez Gonzalez
Edad: 23
Datos:
Nombre: Juan
Apellidos: Lopez Lopez
Edad: 0
Datos:
Nombre: null
Apellidos: null null
Edad: 33
BUILD SUCCESSFUL (total time: 1 second)
```

Propiedades y métodos de clase y de instancia

- En una clase, las propiedades y métodos pueden definirse:
 - De clase
 - De instancia

Atributos de instancia

- Los que hemos trabajado hasta ahora.
- Al crearse un objeto de la clase se crean físicamente las variables para los atributos del objeto creado.
- Cada objeto tiene sus propios valores en los atributos.
- Los atributos públicos de una clase no se visualizarán hasta no crear un objeto de esa clase.
- El acceso a los atributos se hará a través del nombre del objeto.

Atributos o propiedades de instancia

- Estos atributos se declaran con la palabra ***static*** delante.
- Estos atributos de clase existen incluso si no hay objetos de la clase.
- Se accede a ellos a través del nombre de la clase.
- No hace falta crear un objeto para poderlos utilizar.
- Los atributos de clase son compartidos por todas las instancias de la clase.

Un ejemplo

```
public class Persona {  
    static public int totalPersonas=0;  
    private String nombre;  
    private String Apellido1;  
    private String Apellido2;  
    private int edad;  
    public Persona() {  
        totalPersonas++;  
    }  
}
```

Utilizando el atributo de clase

```
System.out.println("N° Personas actual "+Persona.totalPersonas);
Persona p=new Persona();
p.dameDatos(22);
p.dameDatos("Jorge", "Perez", "Gonzalez");
System.out.println("Datos:");
System.out.println("Nombre: "+p.suNombre());
System.out.println("Apellidos: "+p.suApellido1()+" "+p.suApellido2());
System.out.println("Edad: "+p.suEdad());
System.out.println("N° Personas actual "+Persona.totalPersonas);
```

```
run:
N° Personas actual 0
Datos:
Nombre: Jorge
Apellidos: Perez Gonzalez
Edad: 22
N° Personas actual 1
BUILD SUCCESSFUL (total time: 3 se
```


Métodos de instancia

- Los utilizados hasta ahora.
- Los métodos públicos de la clase solo son accesibles si se ha creado un objeto de la clase.
- No pueden ser utilizados directamente utilizando el nombre de la clase.

Métodos de clase

- Pueden ser utilizados sin crear un objeto previamente.
- Estos métodos pueden referenciarse a través del nombre de la clase.
- Casos en los que se utilizan estos métodos:
 - Cuando el método proporciona una utilidad general. Ej: `Math.sqrt(x)`.
 - Cuando el método usa propiedades o métodos estáticos.

Un ejemplo: el método *main*

- Este método es un método estático, que no permite la utilización de propiedades no estáticas.

```
public class EjemploClases {  
    int numero = 8;  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args)  
    {  
        // TODO code application logic here  
        System.out.println(numero);  
    }  
}
```

non-static variable numero cannot be referenced from a static context

(Alt-Enter shows hints)

Paquetes

- Sirven para agrupar clases relacionadas.
- Cada paquete contiene un conjunto de clases, todas ellas con nombres diferentes para poder distinguirlas.
- Cuando no se especifica un nombre de paquete, esta pertenece al paquete por defecto.

Definición de paquetes

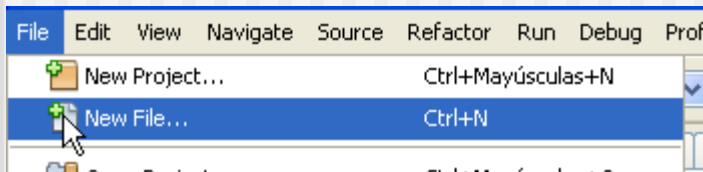
- Para indicar que una clase pertenece a un paquete, aparecerá delante de la clase la sentencia:

`package nombre_paquete;`

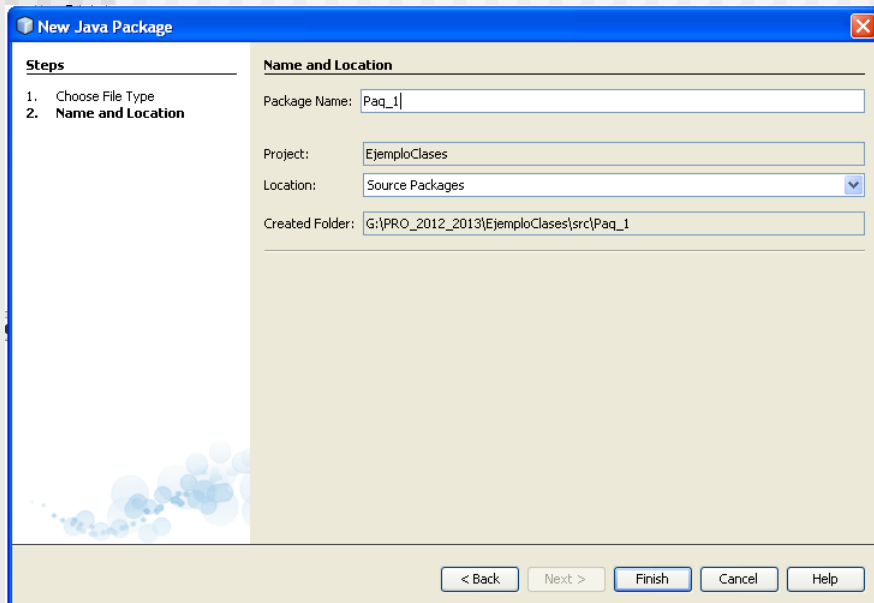
- Las clases que tengan el atributo `public`, serán accesibles desde fuera del paquete.

Creación de paquetes en Netbeans

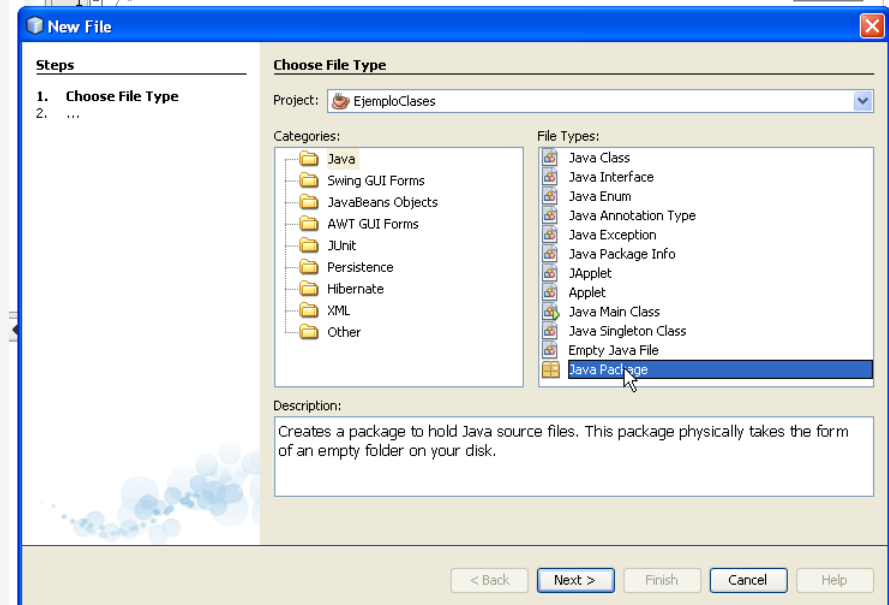
1



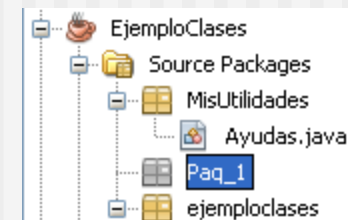
3



2

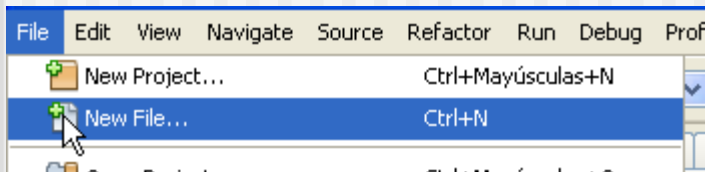


4

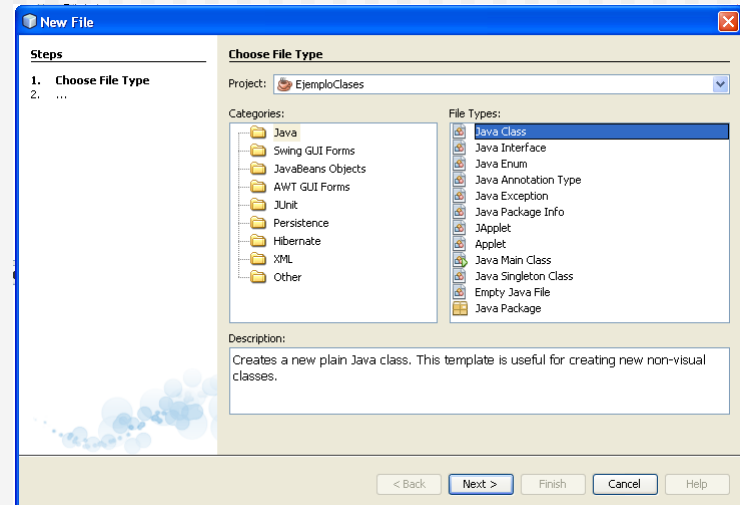


Añadiendo clases a Paq_1

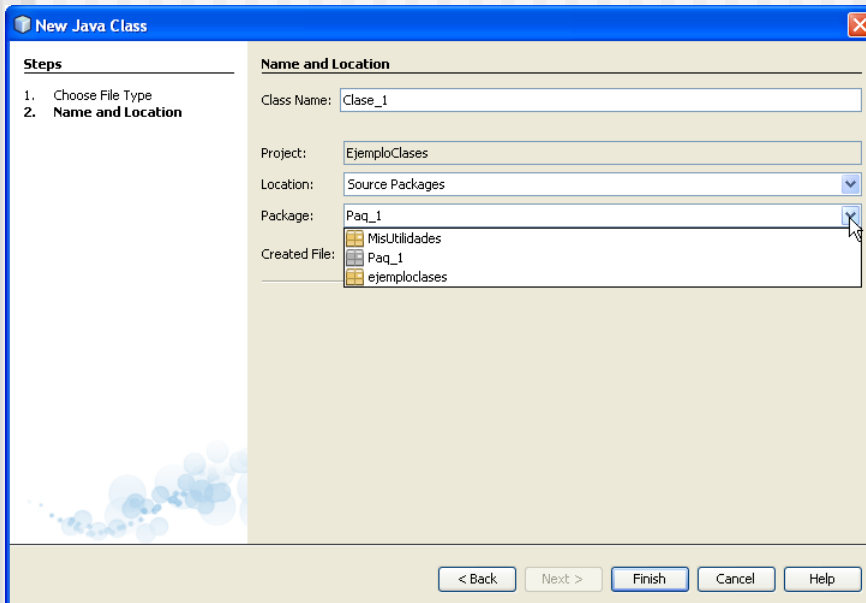
1



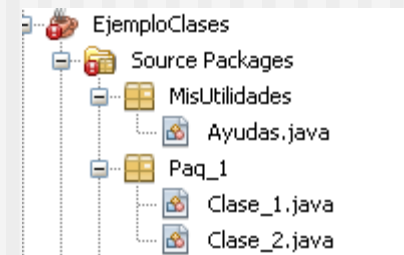
2



3



4



Las clases de Paq_1

```
package Paq_1;

/**
 *
 * @author MARÍA JESÚS
 */
public class Clase_1 {
    /** método público de clase_1 */
    public void metodo_A() {
        Escribe("metodo A de clase_1");
    }
    private void Escribe(String mensaje) {
        System.out.println(mensaje);
    }
}
```

```
package Paq_1;

/**
 *
 * @author MARÍA JESÚS
 */
public class Clase_2 {
    public void metodo_B() {
        System.out.println("método B de Clase_2");
    }
}
```


La clase Ayudas del paquete MisUtilidades

```
package MisUtilidades;

/**
 *
 * @author MARÍA JESÚS
 */
public class Ayudas {
    static public void Escribir(String mensaje){
        System.out.println(mensaje);
    }
}
```

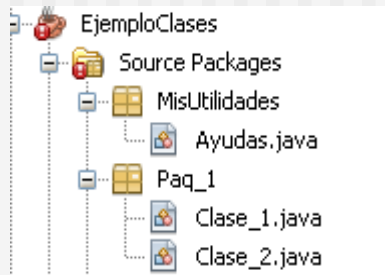
Utilización de clases de otro paquete

- **Opción 1:** Importar el paquete total o parcialmente, añadiendo la sentencia import

parcial

```
import Paq_1.Clase_1;  
import MisUtilidades.*;
```

total



- **Opción 2:** Indicar la ruta para utilizarlos:
Paquete.Clase.metodo

```
Paq_1.Clase_2 c2=new Paq_1.Clase_2();  
c2.metodo_B();
```

```
MisUtilidades.Ayudas.Escribir("Hiola");  
Clase_1 c=new Clase_1();  
c.metodo_A();
```

Un ejemplo: máquina expendedora

- Se quiere diseñar una aplicación que permita la creación de una máquina expendedora sencilla, que suministre agua, fanta y cocacola.
- El diseño permitirá crear máquinas con cualquier n^o de productos.