	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	155/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía práctica de estudio 12: Encapsulamiento (2da parte)

---





---

***Elaborado por:***

M.C. M. Angélica Nakayama C.  
Ing. Jorge A. Solano Gálvez

***Autorizado por:***

M.C. Alejandro Velázquez Mena

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	156/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Guía práctica de estudio 12: Encapsulamiento (2da parte)

### Objetivo:

Aplicar el concepto encapsulamiento para proteger la información dentro de una organización de clases.

### Actividades:


- Agrupar clases en paquetes.
- Importar clases de diversos paquetes.

### Introducción

Las clases de las bibliotecas estándar del lenguaje están organizadas en **jerarquías de paquetes**. Esta organización en jerarquías ayuda a que las personas encuentren clases particulares que requieren utilizar de manera fácil.

Está bien que varias clases tengan el mismo nombre si están en paquetes distintos. Así, encapsular grupos pequeños de clases en paquetes individuales según su funcionalidad permite reusar el nombre de una clase dada en diversos contextos.

**NOTA:** En esta guía se tomará como caso de estudio el lenguaje de programación JAVA, sin embargo, queda a criterio del profesor el uso de éste u otro lenguaje orientado a objetos.

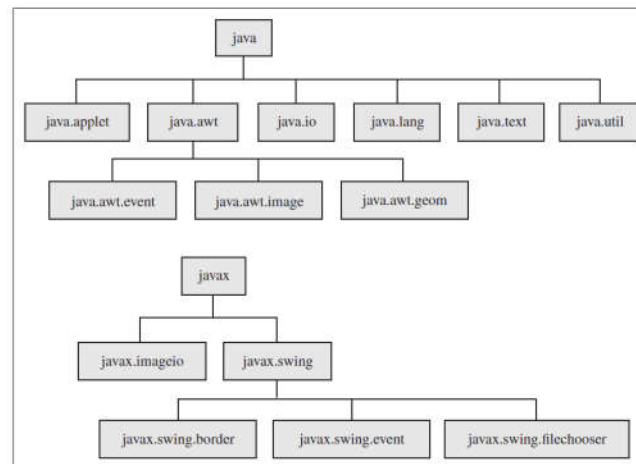
	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	157/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Paquetes

Un **paquete** o **package** es una agrupación de clases. La API de Java cuenta con muchos **paquetes** que contienen clases agrupadas bajo un mismo propósito. Dado que la biblioteca de Java contiene miles de clases, es necesaria alguna estructura en la organización de la biblioteca para facilitar el trabajo con este enorme número de clases.

Java utiliza **paquetes** para acomodar las clases de la biblioteca en grupos que permanecen juntos. Las clases del API están organizadas en **jerarquías de paquetes**. Esta organización en jerarquías ayuda a encontrar clases particulares de forma eficiente.


A continuación, se muestra parte de la **jerarquía de paquetes** de la API de Java.



Los **paquetes** se utilizan con las finalidades siguientes:

- Para agrupar clases relacionadas.
- Para evitar conflictos de nombres. En caso de conflicto de nombres entre clases el compilador obliga a diferenciarlos usando su nombre calificado.
- Para ayudar en el control de la accesibilidad de clases y miembros.

El nombre completo o nombre calificado (**Fully Qualified Name**) de una clase debe ser único y está formado por el nombre de la clase precedido por los nombres de los subpaquetes en donde se encuentra hasta llegar al paquete principal, separados por puntos.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	158/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo:

*java.util.Random* es el **Fully Qualified Name** de la clase *Random* que se encuentra en el paquete *java.util*

## Importar paquetes

Las clases de Java que se almacenan en la biblioteca de clases, no están disponibles automáticamente para su uso. Para poder disponer de alguna de estas clases, se debe indicar en el código que se va usar una clase de la biblioteca usando su **fully qualified name**.

Ejemplo:

```
public class PruebaPaquetes {
    public static void main(String[] args) {
        java.util.Random rnd = new java.util.Random();
        System.out.println(rnd.nextInt(100));
    }
}
```

Para hacer que las clases en un paquete particular estén disponibles para el programa que se está escribiendo, es necesario **importar** ese paquete, esto permite abreviar los nombres de las clases, variables y métodos, evitando el tener que escribir continuamente el nombre completo de la clase.


La sentencia **import** tiene la forma general:

*import fullyQualifiedName;*

Estas sentencias deben ir antes de la declaración de la clase. Ejemplo:

```
import java.util.Random;

public class PruebaPaquetes {
    public static void main(String[] args) {
        Random rnd = new Random();
        System.out.println(rnd.nextInt(100));
    }
}
```

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	159/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Java también permite importar paquetes completos con sentencias de la forma

***import** nombreDePaquete.\*;*

Por ejemplo, la siguiente sentencia importaría todas las clases del paquete *java.util*:

***import** java.util.\*;*

El **importar** un paquete no hace que se carguen todas las clases del paquete, sino que sólo se cargarán las clases **public** del paquete.

Al **importar** un paquete **no se importan los sub-paquetes**. Éstos deben ser importados explícitamente, pues en realidad son paquetes distintos. Por ejemplo, al importar *java.awt* no se importa *java.awt.event*.

Algunas clases se usan tan frecuentemente que casi todas las clases debieran importarlas. Estas clases se han ubicado en el paquete **java.lang** y este paquete se **importa automáticamente** dentro de cada clase. La clase *String* es un ejemplo de una clase ubicada en *java.lang*.

## Paquetes propios


El lenguaje Java permite crear sus **propios paquetes** para organizar clases definidas por el programador en jerarquías de paquetes.

Para que una clase pase a formar parte de un **paquete** hay que introducir en ella la sentencia:

***package** nombreDelPaquete;*

La cual debe ser la **primera sentencia del archivo** sin contar comentarios y líneas en blanco.

Los nombres de los **paquetes** se suelen escribir con minúsculas, para distinguirlos de las clases, que empiezan por mayúscula. El nombre de un paquete puede constar de varios nombres unidos por puntos.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	160/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Todas las clases que forman parte de un **paquete** deben estar en el mismo directorio. Los nombres compuestos de los paquetes están relacionados con la jerarquía de directorios en que se guardan las clases. Es recomendable que los nombres de las clases sean únicos en toda la biblioteca. Es el nombre del paquete lo que permite obtener esta característica. Una forma de conseguirlo es incluir el nombre del dominio.

Por ejemplo, la clase:

*mx.unam.fi.poo.MiClase.class*

Debería estar en:

*CLASSPATH\mx\unam\fi\poo\MiClase.class*

Ejemplo:

```
package mx.unam.fi.poo;


public class MiClase {
    public static void main(String[] args) {
        System.out.println("Clase empaquetada");
    }
}
```

## Compilación y ejecución de clases en paquetes

Se puede solicitar al compilador de Java que coloque en forma automática el archivo compilado .class en la ruta destino correspondiente. Para hacer lo anterior, debe invocar al compilador desde línea de comandos con la opción `-d`, como sigue:

*javac -d rutaOrigen archivoFuente*

El nombre de ruta completo del directorio que obtiene el código compilado es *rutaOrigen/rutaDelPaquete*.

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	161/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Si el directorio destino ya existe, entonces el archivo generado **.class** va a ese directorio. Si el directorio destino no existe, el compilador crea en forma automática el directorio requerido y luego inserta ahí el archivo generado **.class**. Por tanto, no es necesario crear explícitamente la estructura de directorios, se puede dejar que el compilador lo haga.

Ejemplo:

Se desea poner la clase *HolaMundo* en un paquete llamado *hola*, para tal efecto, se modifica el código fuente de la siguiente manera:

```
package hola;
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Hola Mundo");
    }
}
```

Para que se genere el archivo **HolaMundo.class** dentro del directorio *hola*, se compila con la opción **-d** y la ruta origen sería el directorio actual, es decir punto ( **.** ):


```
D:\>dir hola*
El volumen de la unidad D es DATA
El número de serie del volumen es: 9CD3-2B2F

Directorio de D:\
21/06/2016  09:43 p. m.          130 HolaMundo.java
               1 archivos          130 bytes
               0 dirs  741,121,339,392 bytes libres

D:\>javac -d . HolaMundo.java

D:\>dir hola*
El volumen de la unidad D es DATA
El número de serie del volumen es: 9CD3-2B2F

Directorio de D:\
21/06/2016  09:47 p. m.      <DIR>          hola
21/06/2016  09:43 p. m.          130 HolaMundo.java
               1 archivos          130 bytes
               1 dirs  741,121,339,392 bytes libres
```

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	162/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En este caso se generó un directorio llamado hola, dentro del cual se generó el archivo **HolaMundo.class** correspondiente a la clase compilada.

```
D:\>dir hola
El volumen de la unidad D es DATA
El número de serie del volumen es: 9CD3-2B2F

Directorio de D:\hola

21/06/2016  09:47 p. m.  <DIR>          .
21/06/2016  09:47 p. m.  <DIR>          ..
21/06/2016  09:47 p. m.                427 HolaMundo.class
                        1 archivos          427 bytes
                        2 dirs  741,121,339,392 bytes libres

D:\>
```


Para poder ejecutar correctamente esta nueva clase compilada, se debe hacer usando su **Fully Qualified Name** desde la ruta origen, ya que si solo se invoca el intérprete **java** con el nombre de la clase la ejecución fallará dado que no reconoce la clase.

```
D:\>java HolaMundo
Error: no se ha encontrado o cargado la clase principal HolaMundo

D:\>java hola.HolaMundo
Hola Mundo

D:\>
```



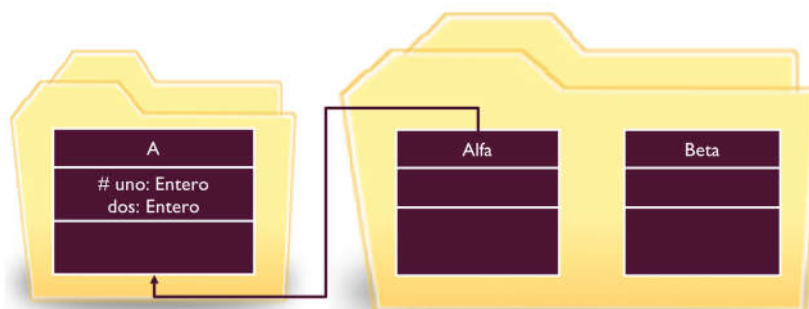
	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	163/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Encapsulamiento con paquetes


Como se mencionó en la práctica anterior, el nivel de **acceso por defecto** cuando no se especifica alguno es el de paquete, esto es, los atributos que no tienen asignado un nivel de acceso solo pueden ser accedidos por clases que se encuentran **dentro del mismo paquete**. Por otro lado, el nivel de acceso protegido (**protected**) permite que un elemento sea **accedido fuera del paquete** mientras exista una **relación de herencia** entre la clase que contiene al elemento y la clase que lo intenta acceder. A esto se le conoce como encapsulamiento en paquetes, ya que también se puede proteger la información negando el acceso a los datos según se requiera.

### Ejemplo

Dada la siguiente disposición de clases:



La clase A se encuentra dentro del paquete paquete.abecedario y tiene dos atributos, uno protegido y otro de paquete (sin modificador).

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	164/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
package paquete.abecedario;

public class A {
    protected int uno;
    int dos;
}
```


La clase Alfa se encuentra dentro del paquete paquete.abecedario y hereda de A. Por el hecho de heredar tiene acceso al atributo protegido (uno), pero no al atributo de paquete (dos).

```
package paquete.alfabeto;
import paquete.abecedario.A;

public class Alfa extends A {
    public static void main(String[] args) {
        Alfa alfa = new Alfa();
        System.out.println(alfa.uno);
        System.out.println(alfa.dos);
    }
}
```

Al compilar la clase se genera la siguiente salida:

```
paquete/alfabeto/Alfa.java:16: error: dos is not public in A; cannot be accessed from outside package
    System.out.println(alfa.dos);
                        ^
1 error
```

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	165/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


La clase Beta se encuentra dentro del paquete paquete.abecedario (igual que Alfa), pero no hereda de nadie, por lo tanto, no tiene acceso a ninguna de las variables de A, ya que solo podría acceder a variables públicas de A.

```
package paquete.alfabeto;

public class Beta {
    public static void main(String[] args) {
        Beta beta = new Beta();
        System.out.println(beta.uno);
        System.out.println(beta.dos);
    }
}
```

Al compilar la clase se genera la siguiente salida:

```
paquete/alfabeto/Beta.java:12: error: cannot find symbol
    System.out.println(beta.uno);
                        ^
    symbol:   variable uno
    location: variable beta of type Beta
paquete/alfabeto/Beta.java:13: error: cannot find symbol
    System.out.println(beta.dos);
                        ^
    symbol:   variable dos
    location: variable beta of type Beta
2 errors
```

	<b>Manual de prácticas del Laboratorio de Modelos de programación orientada a objetos</b>	Código:	MADO-21
		Versión:	02
		Página	166/166
		Sección ISO	8.3
		Fecha de emisión	14 de junio de 2018
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

## Bibliografía

*Barnes David, Kölling Michael*

***Programación Orientada a Objetos con Java.***

*Tercera Edición.*

*Madrid*

*Pearson Educación, 2007*

*Deitel Paul, Deitel Harvey.*

***Como programar en Java***

*Septima Edición.*

*México*

*Pearson Educación, 2008*

*Joyanes, Luis*

***Fundamentos de programación. Algoritmos, estructuras de datos y objetos.***

*Cuarta Edición*

*México*

*Mc Graw Hill, 2008*

*Dean John, Dean Raymond.*

***Introducción a la programación con Java***

*Primera Edición.*

*México*

*Mc Graw Hill, 2009*