



---

# Ingeniería en Sistemas Computacionales

## Fundamentos de Programación

Clave AED-1285

SATCA 2-3-5

---

### Unidad 5. Modularidad

Dra. María Lucía Barrón *Estrada*

# **Fechas de Examen/Evaluación**

## **Agosto-Diciembre 2018**

~~**Unidad 1- 21 septiembre**~~

~~**Unidad 2- 22 octubre**~~

~~**Unidad 3- 23 noviembre**~~

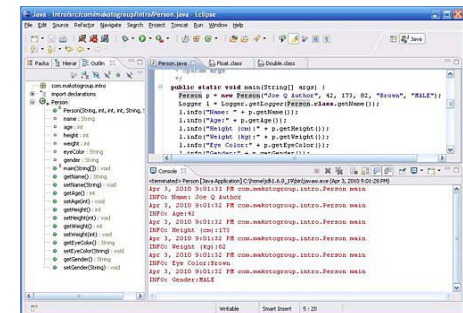
**Unidad 4- 7 diciembre**

**Unidad 5- 7 diciembre**



# Evaluación Unidad 5

- 5-6 DICIEMBRE
  - Revisión de proyectos
- **Unidad 5- 7 DICIEMBRE**
  - Portafolio individual
  - Examen teórico/práctico



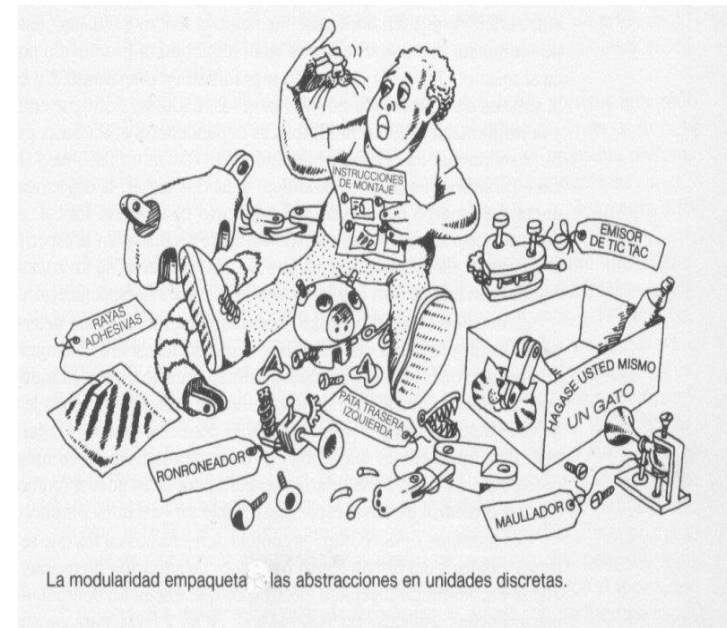
# 5. Modularidad

Conoce y aplica la modularidad en el desarrollo de programas para la optimización de los mismos y reutilización de código.

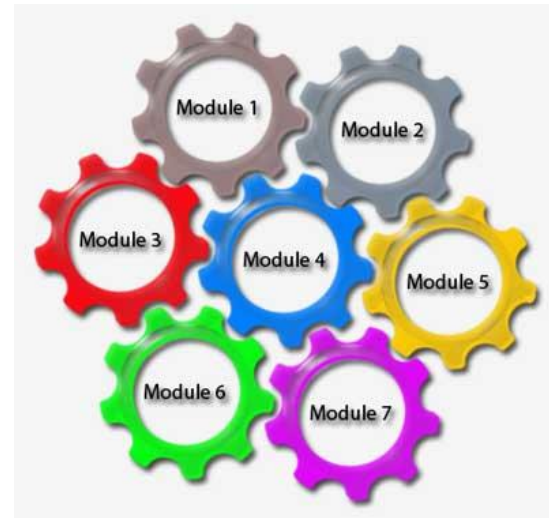
5.1 Declaración y uso de módulos.

5.2 Pase de parámetros o argumentos.

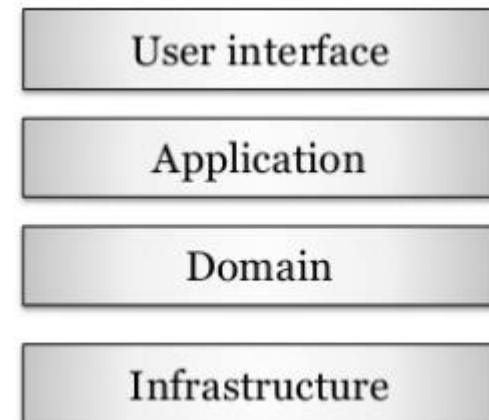
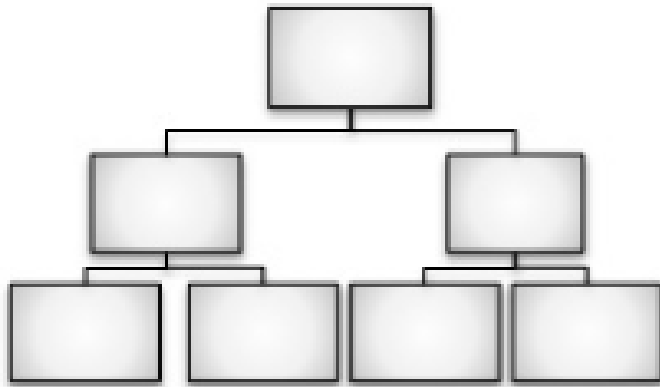
5.3 Implementación.



# Introducción

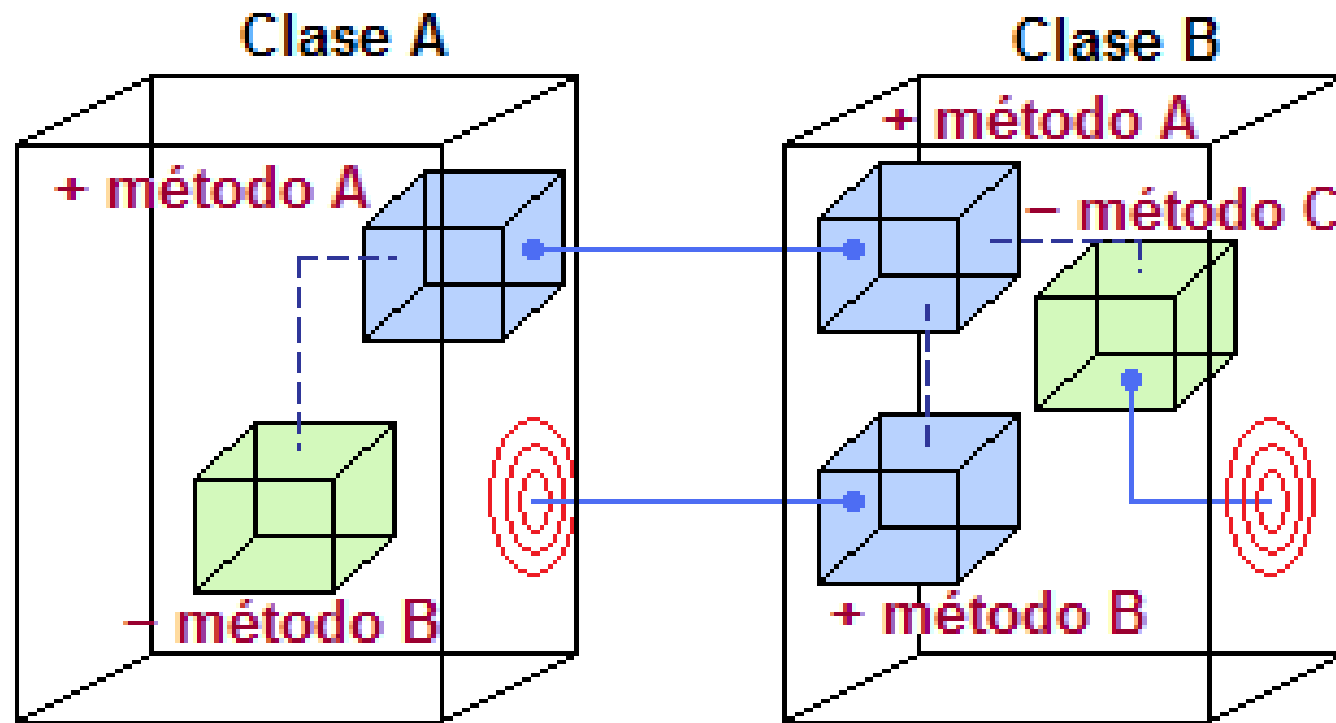


- **Módulo:** representan una unidad de software separada que define límites lógicos entre los componentes y mejora el mantenimiento.



# Definiciones de Modularidad

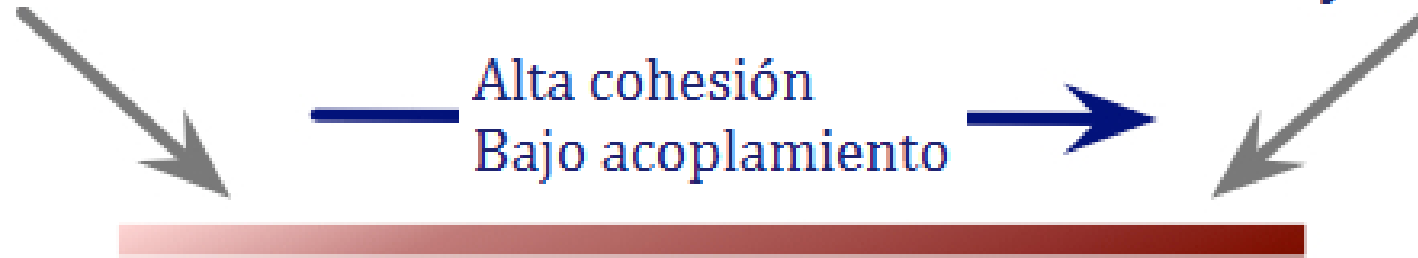
1. Es el grado en que los componentes de un sistema pueden ser separados y combinados.
2. Es la capacidad de descomponerse y reorganizarse en módulos.
3. Es la característica por la cual un programa esta compuesto de porciones que se conocen como módulos.
4. Es la propiedad que permite **subdividir una aplicación en partes más pequeñas** (llamadas módulos), cada una de las cuales **debe ser tan independiente** como sea posible de la aplicación en sí y de las restantes partes.



- Cada módulo tiene sus entradas y sus salidas
- Cada módulo hace algo diferente
- La salida de un módulo puede ser usada como entrada en otro módulo

**Difícil de  
mantener  
y extender**

**Fácil de  
mantener  
y extender**



## ***Modularidad***

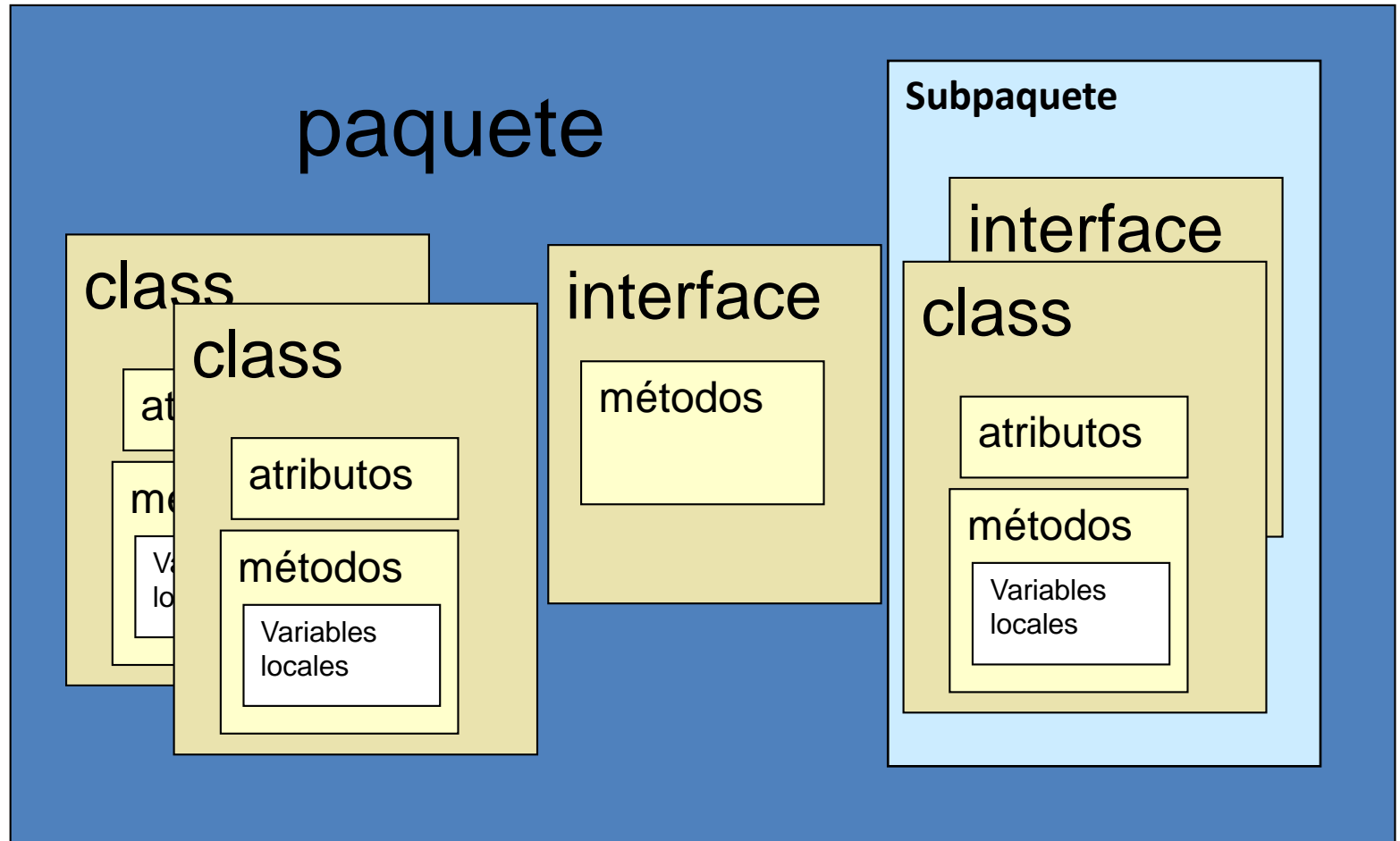
- Ventajas
  - Se pueden asignar a equipos diferentes para su desarrollo
  - Pueden ser reutilizados
  - Se pueden desarrollar en paralelo
  - Promueve la abstracción:
    - Esconde detalles de implementación
    - Provee interfaces para saber QUÉ hace no CÓMO lo hace



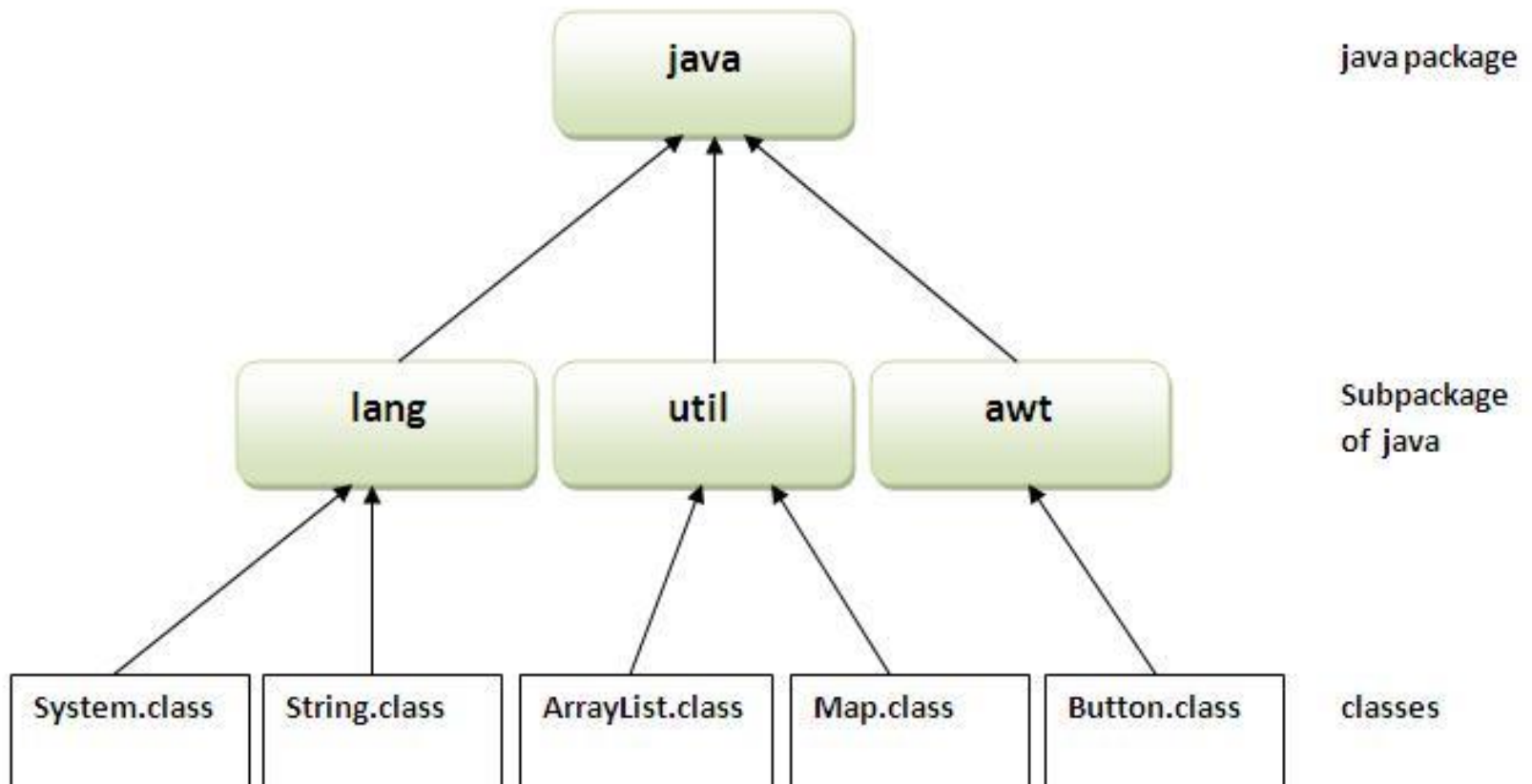
# Dos puntos de vista

- **Diseño de software:** modularidad se refiere a la partición lógica del software que permite que sistemas complejos sean manejables con el propósito de implementación y mantenimiento.
- **Programación modular:** la modularidad se refiere a la compartición e interrelación de las partes de un sistema (paquete de software).

# package ( Paquetes )



# Java Package



# 5.1 Declaración y uso

- Java tiene dos unidades de compilación:
  - Clase (class)
  - Interfaz (interface)
- Una **clase** contiene en su cuerpo el código para la manipulación de objetos de esa clase.
  - **Constructores** para crear objetos,
  - **Atributos** (variables) para almacenar el estado del objeto y de la clase
  - **Métodos** que implementan el comportamiento de la clase y sus objetos.

# Cuerpo de una Clase

- El cuerpo de la clase esta delimitado por { }
- Declaración de **Miembros**
  - **Atributos (declaración e inicialización)**
    - Atributos de instancia
    - Atributos de clase
  - **Métodos (declaración e implementación)**
    - Métodos de instancia
    - Métodos de clase
- Declaración e implementación de **Constructores**
- Bloque de inicialización

# Concepto de método

- Un método es una parte de código que se encarga de implementar parte del comportamiento de un objeto.
- Un método tiene dos partes:
  - Encabezado
  - Cuerpo
- Un método se invoca (ejecuta) usando un objeto.  
`objeto.método(parámetros)`
- Dentro del cuerpo de un método se pueden declarar variables locales.

# Declaración de métodos

*Modificadores* **void** Identificador (*parámetros*) ;

*Modificadores* **void** Identificador (*parámetros*) Cuerpo

*Modificadores* tipo Identificador (*parámetros*) ;

*Modificadores* tipo Identificador (*parámetros*) Cuerpo

*Modificadores*

**public**

**private**

**protected**

**package**

**static**

**final**

**native**

**synchronized**

**abstract**

*parámetros*

**tipo identificador , ...tipo identificador**

*Cuerpo*

**{ estatutos }**

```

public class Punto {
    private int x=0;
    private int y=0;
    public static int totalPuntos =0;
    Punto (int _x, int _y){
        x=_x;
        y=_y;
        totalPuntos++;
    }
    public void setx(int _x){
        x= _x;
    }
    public String toString (){
        return "("+x+"," +y+")";
    }
}

```

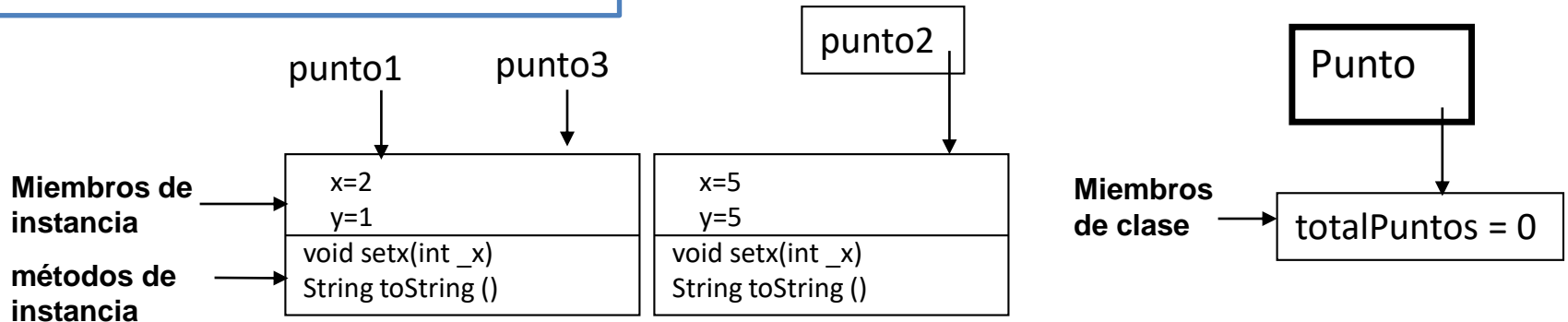
# Ejemplo

```

public class UsaPunto {
    public static void main(String[] args){
        Punto punto1= new Punto(2,1);
        Punto punto2 = new Punto(5,5);
        Punto punto3 = punto1;

        Punto[] puntos = {punto1,punto2};
        System.out.println(Arrays.toString(puntos));
        System.out.println("Puntos creados "+ Punto.totalPuntos);
    }
}

```






# Ejercicio de laboratorio

- Escribe las clases Punto y UsaPunto
- Compila ambas clases y corrige los errores
- Ejecuta la clase UsaPunto
- Escribe un reporte con:
  - Los errores de compilación que recibiste
  - Los resultados de la ejecución de UsaPunto
  - Modifica el programa para
    - Declarar un arreglo de Punto
    - Almacena 10 puntos solicitando los datos (x, y) por teclado
    - Imprime todos los puntos almacenados en el arreglo
    - Agrega los métodos get a la clase Punto
    - Agrega el método setY a la clase Punto
    - Agrega un método para mover un punto a una nueva posición x,y
    - Agrega un método que calcule la distancia entre dos puntos

## 5.2 Paso de parámetros o argumentos.

- Los objetos pueden realizar operaciones para:
  - Manipular o inspeccionar sus variables
  - Ejecutar sus métodos.
- Los objetos se comunican con otros objetos a través de mensajes.
- Un mensaje es una señal que recibe un objeto para ejecutar alguno de sus métodos.

`objeto.método()`  Llamada a ejecución SIN parámetros

`objeto.método(val1, val2, ... valn)`  Llamada a ejecución  
CON la lista de valores  
de entrada

# Pase de parámetros

- Java usa el modelo de **pase de parámetros por Valor**, esto significa que el valor del argumento actual se copia a la variable del argumento formal.

```
public class Punto {  
    ...  
    public void setx(int _x){  
        x= _x;  
    }  
}
```

Argumento Formal

```
public class UsaPunto {  
    public static void main(String[] ar){  
        Punto punto1= new Punto(2,1);  
        punto1.setx(8);  
    }  
}
```

Argumento Actual

# Ejercicios

**Escribe un método para cada uno de los siguientes:**

- reciba como parámetro un entero y regrese como resultado **true** si el parámetro es par y **false** si es impar.
- reciba dos valores enteros y regrese como resultado su suma.
- reciba como parámetro un arreglo de valores enteros y regrese como resultado su promedio.
- reciba como parámetro Un String y un char, y regrese como resultado cuantas veces aparece el char en el String.
- reciba como parámetro un valor entero y regrese como resultado el numero de Fibonacci correspondiente.
- reciba como parámetro dos valores (estatura y peso de una persona y regrese su Índice de Masa Corporal (IMC).
- reciba como parámetro un arreglo de valores enteros y regrese como resultado cuantos son positivos.

## 5.3 Implementación.

- Un problema se divide en partes pequeñas para solucionarlo.
- Todas las partes pueden estar almacenadas en carpetas llamadas **package (paquete)**
- Las clases e interfaces de un paquete se pueden compartir con otros programas pero es necesario importarlas para poder usarlas

# Definición y creación de paquetes / librería.

- Un paquete (package) es una colección de archivos .class relacionados, los cuales se encuentran en un subdirectorio.
- Un paquete es una biblioteca (library) de clases y a su vez es un subdirectorio.

**Paquete = Directorio**

(nombre del paquete debe ser igual al nombre del directorio)

**Clase = Archivo**

(nombre de la clase debe ser igual al nombre del archivo)

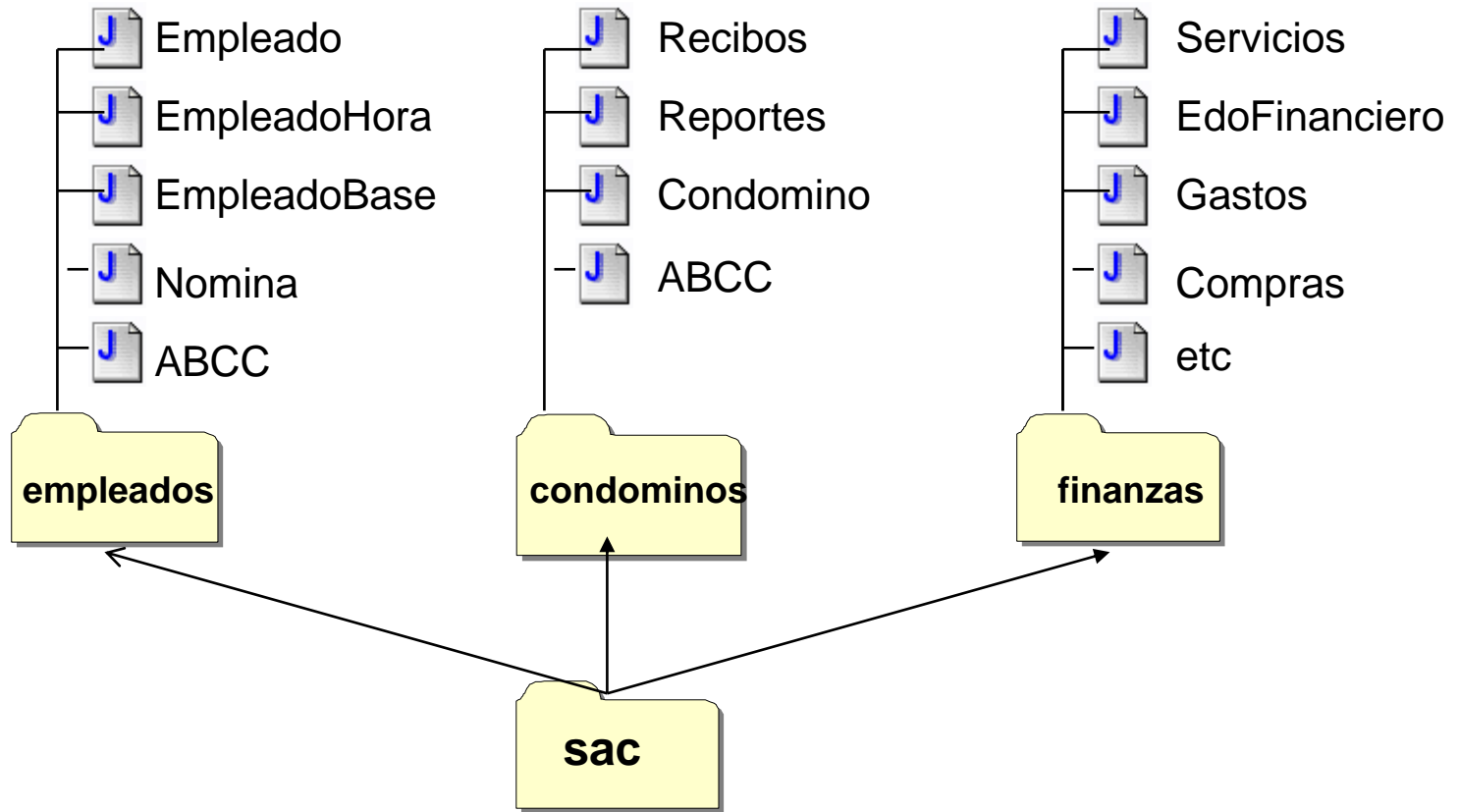
# Definición y creación de paquetes / librería

- Los paquetes se utilizan para agrupar diversas clases dándoles un nombre de grupo que pueda identificarlas.
- Los paquetes permiten asignar a las clases nombres que incluyen la ruta donde estas se localizan. (Nombres completos)

```
package java.util.zip;  
public class ZipFile {...}
```

**Nombre completo de la clase**  
**java.util.zip.ZipFile**

# Ejemplos





# Ejemplos

- Escribe una clases para diversas figuras geométricas (Triangulo, Circulo, Cuadrado)
  - Atributos
    - los necesarios para representar sus valores
  - Métodos
    - get y set para los atributos
    - area() para calcular su área
    - perimetro() para calcular su perímetro
    - toString () para imprimir el objeto

# Ejercicio

- Escribe un método que reciba como parámetro un arreglo de Triángulos y obtenga la suma de sus áreas.

# Fin de curso!!

