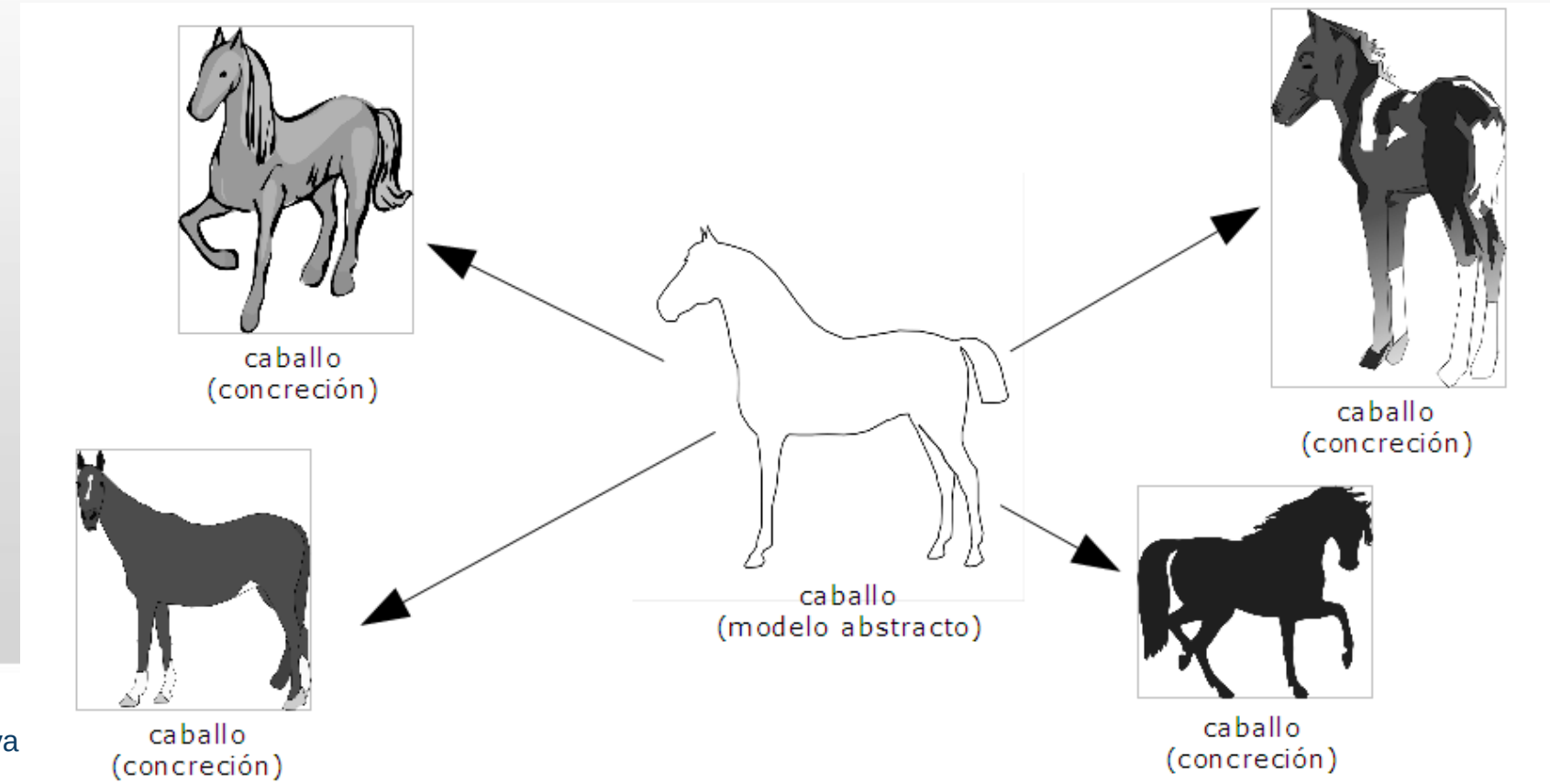
Clases

- Abstracción de alto nivel, modela el mundo real (y el mundo informatico)
- Se puede ver como un molde, que nos sirve para crear objetos, o sea un objeto es una instancia de una clase, también podemos pensar en ellos como tipos de datos.
- Conjunto de:
  - Atributos: Variables.
  - Métodos: Funciones para manipular esas variables.

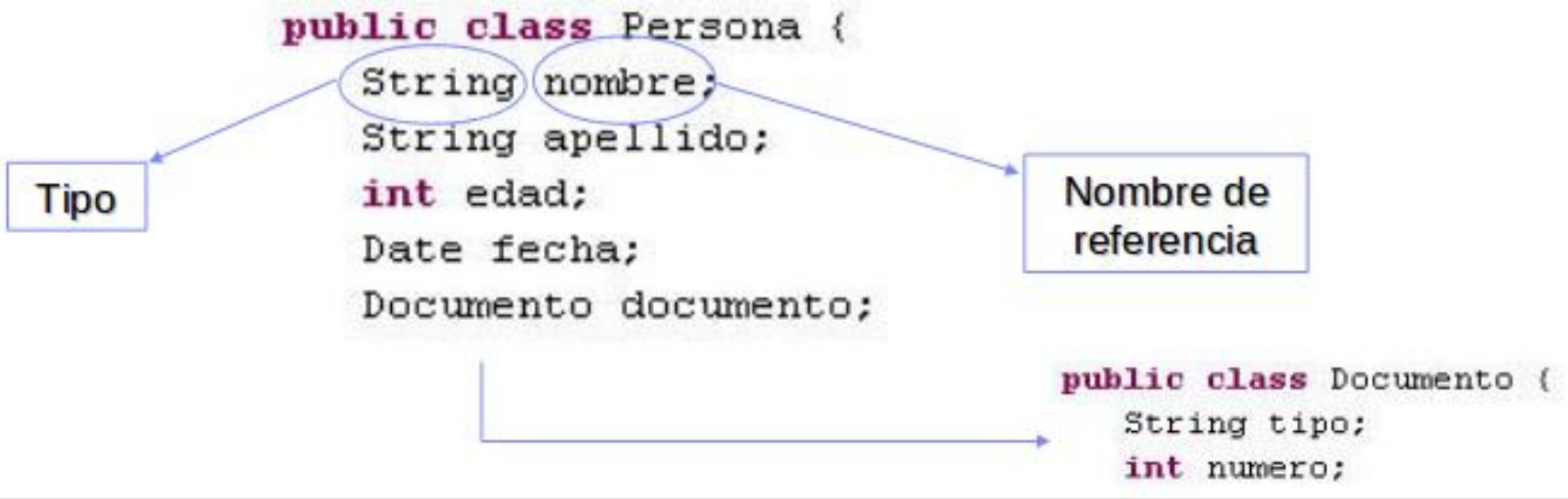
Objetos

- Los Objetos son ejemplares de una clase.
- Cuando creamos un ejemplar (instancia) tenemos que especificar la clase a partir de la cual se creara.
- Esta acción de crear un objeto a partir de una clase se llama instanciar.



Atributos

- Las propiedades o atributos son las características de los objetos.
- Cuando se define una propiedad se define su tipo y su nombre
- Son variables donde se almacenan los datos relacionados al objeto.



Métodos

- Representan el comportamiento del objeto. Es decir, lo que el objeto puede hacer.
- Pueden recibir parámetros o no.
- Puede ser que el método cambie el estado del objeto o que solo informe el estado de alguna propiedad

```
public class Persona {
    String nombre;
    String apellido;
    int edad;
    Date fecha;
    Documento documento;

    public String dameTuNombre(){
        return nombre + ", " + apellido;
    }
```

Sobrecarga y Sobre-escritura

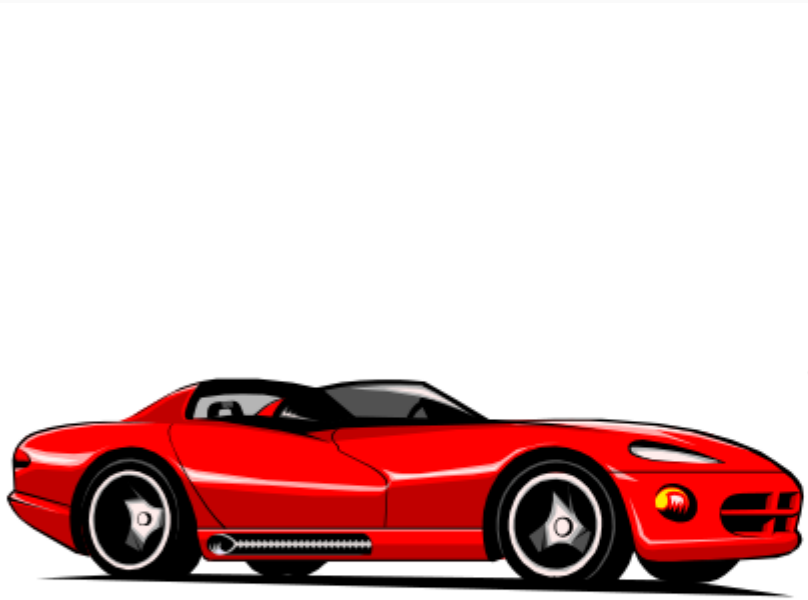
Sobrecarga: Es la acción de agregar un método casi igual a otro, pero donde variamos el numero de parámetros o los tipos de datos

Sobre-escritura: Es la acción de cambiar las definición de un método que ya hizo nuestro padre.

	Sobrecarga de Métodos	Sobreescritura de Métodos
Argumento(s)	Debe Cambiar	No debe cambiar
Tipo de Retorno	Puede Cambiar	No puede variar, exepto por los covariantes de retorno
Excepciones	Puede Cambiar	Se puede reducir o eliminar, pero no debe lanzar nuevas excepciones no identificadas
Acceso	Puede Cambiar	Puede, pero no se puede hacer más restrictivo
Invocación	Tipo de referencia determina cual valor de la sobrecarga es seleccionado, esto ocurre en tiempo de compilación	Tipo de Objeto de salida es determinado por el método seleccionado

Atributos y métodos de instancia

- Cuando nos referimos a algo de instancia, estamos hablando que es algo propio de cada objeto, o sea la propiedad color de cada auto pertenece a un solo auto, porque cada uno puede tener un color distinto, si fuera una propiedad global (static) todos los autos compartirían el mismo color.



- Atributos:
  - color
  - velocidad
  - ruedas
  - motor
- Métodos:
  - arranca()
  - frena()
  - dobla()

Visibilidad

En lo que concierne a los modificadores de visibilidad o de acceso, el objetivo principal es lograr el **encapsulamiento**, controlando o prohibiendo el acceso a los atributos o elementos de una clase.

Pueden usarse para establecer la visibilidad de: ***clases, métodos y variables.***

En java podemos mencionar 4 tipos de modificadores, éstos, como su nombre lo indica, "modifican" o determinan la "visibilidad" o el acceso a un cierto elemento, son:

public*:* Visible dentro y fuera del paquete. Los métodos y las variables, son heredados por todas las subclases o hijas de la clase. Accesibles desde cualquier lugar.

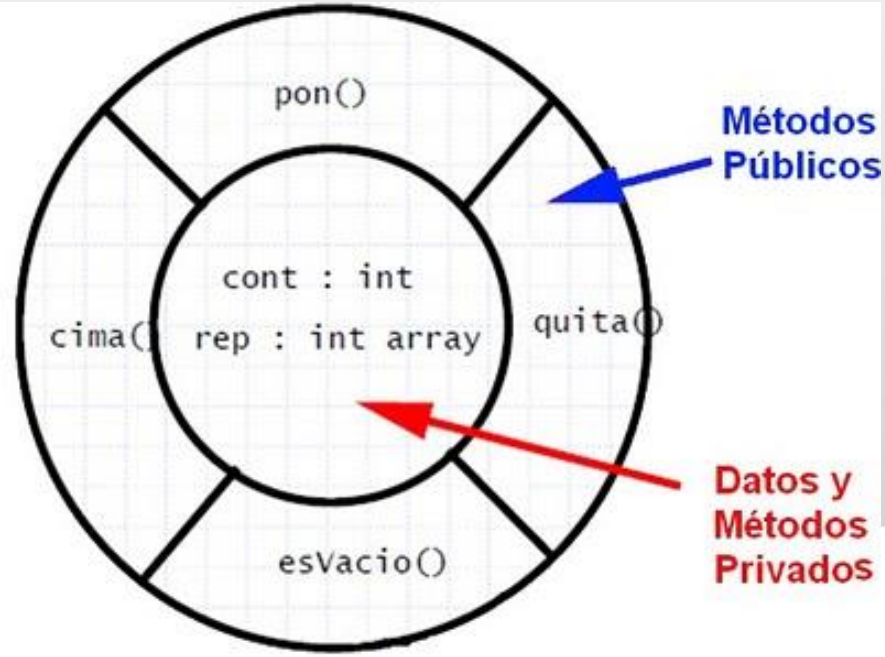
private*:* Visible sólo desde el interior de la clase. No son heredados por ninguna sub-clase. No son accesibles desde ninguna otra clases, por más que estén en el mismo paquete.

protected*:* Visible desde el interior de la clase y desde las clases heredadas (sub-clases o hijas).

friendly o default: (Sin palabra clave) visible desde cualquier clase en el mismo paquete.

Encapsulamiento

- El contenido de alguna información esta oculto (Nosotros definimos esto usando los distintos tipos de visibilidad).
  - Interfaz publica: Se puede invocar mensajes sobre una clase desde fuera de ella.
  - Implementación privada: O sea recibimos ordenes y las ejecutamos pero no necesariamente la persona que nos llama debe saber como hacemos nuestro trabajo, concepto de caja negra.
- Control de acceso para evitar uso inadecuado.

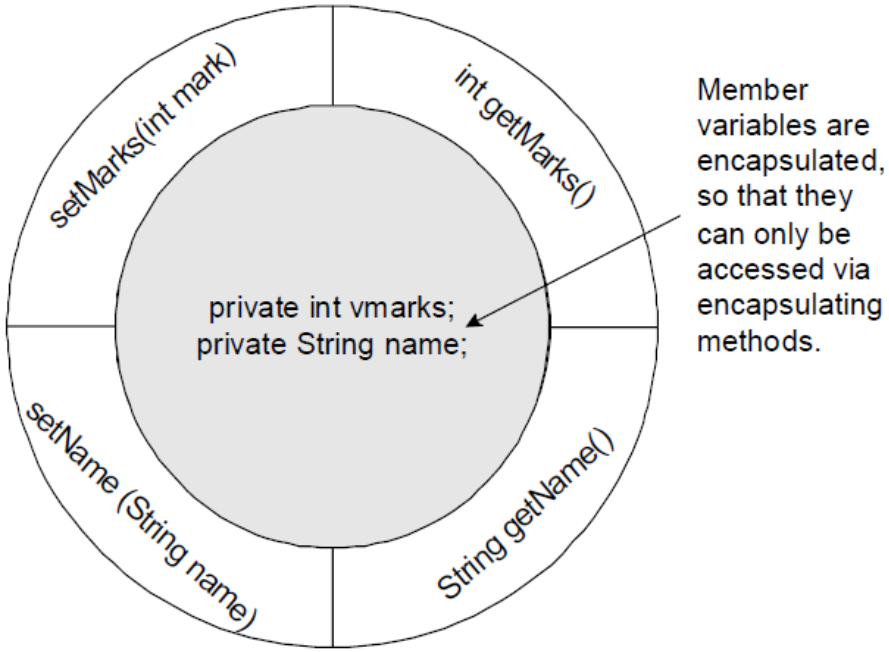


Metodos de acceso (Getters/Setters)

- Un método "getter" para "retornar el valor" (solo devolver la información del atributo para quién la solicite).
- Un método "setter" para "cargar un valor" (asignar un valor a una variable).
- En conclusión son métodos que nos permiten modificar el valor de nuestras propiedades.

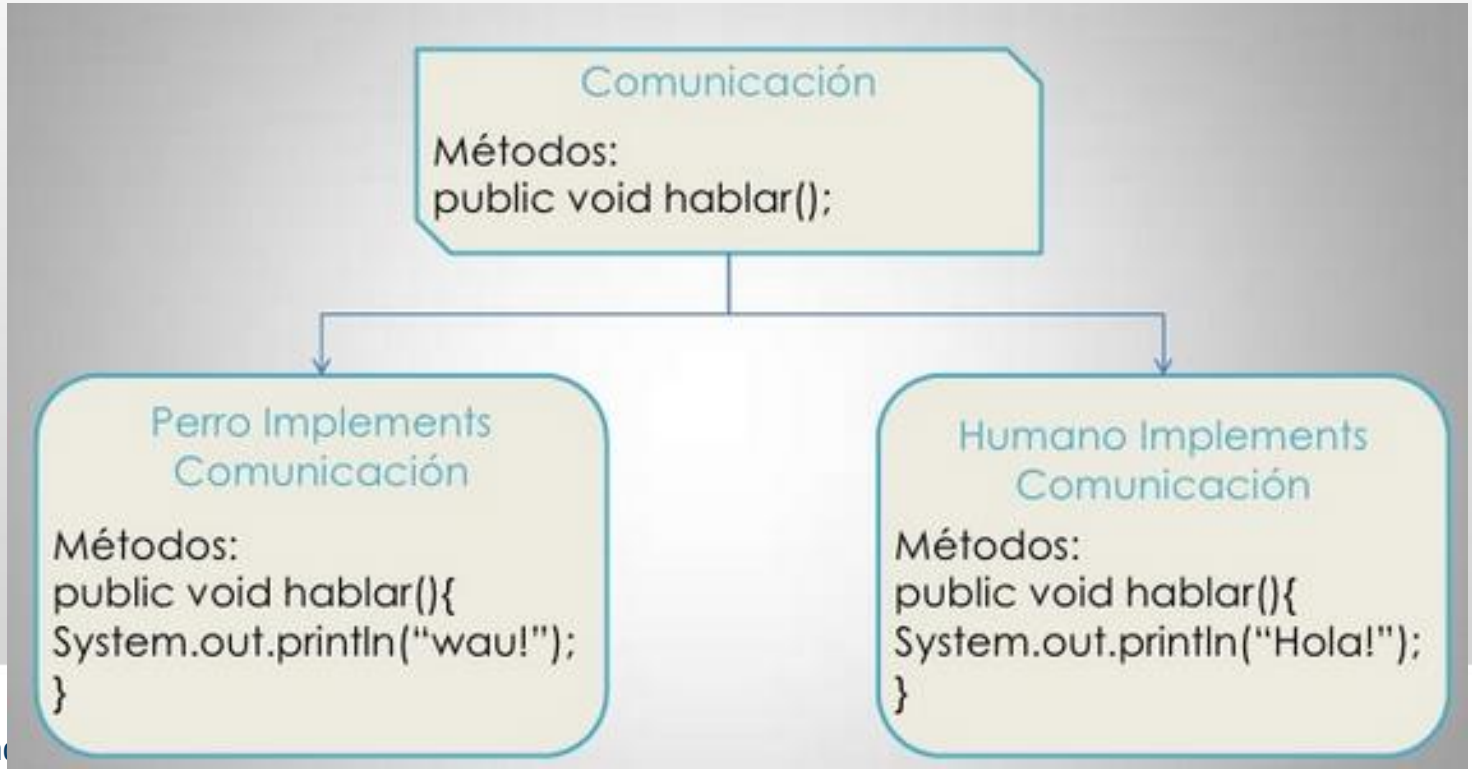
Sample code

```
Class MyMarks {
  private int vmarks =  0;
  private String name;

  public void setMarks(int mark)
                        throws MarkException {
      if(mark > 0)
         this.vmarks =  mark;
      else {
         throw new MarkException("No negative
                                  Values");
      }
  }

   public int getMarks(){
        return vmarks;
   }
   //getters and setters for attribute name goes here.
}
```



Interfaces

- Una clase con todos sus métodos abstractos.
- Puede incluir constantes que deben ser estáticos y finales.
- Sirve para establecer estándares entre clases.
- Es un contrato para las clases.



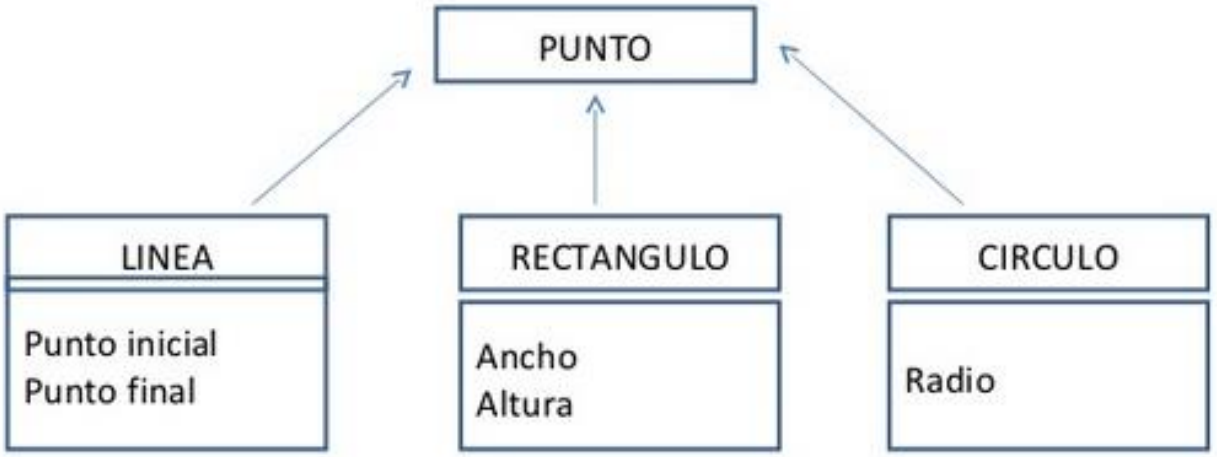
Diferencias Interfaces y Clase Abstracta

- En la interfaz todos los métodos son abstractos en una clase abstracta no necesariamente todos los métodos son abstracto.
- Las interfaces implementan, las clases se heredan (Java no permite herencia múltiple)
- Las interfaces solo puede declares constantes, las clases abstractas pueden constantes datos y no constantes.
- **Este concepto viene de la mano y el polimorfismo**

Polimorfismo

POLIMORFISMO

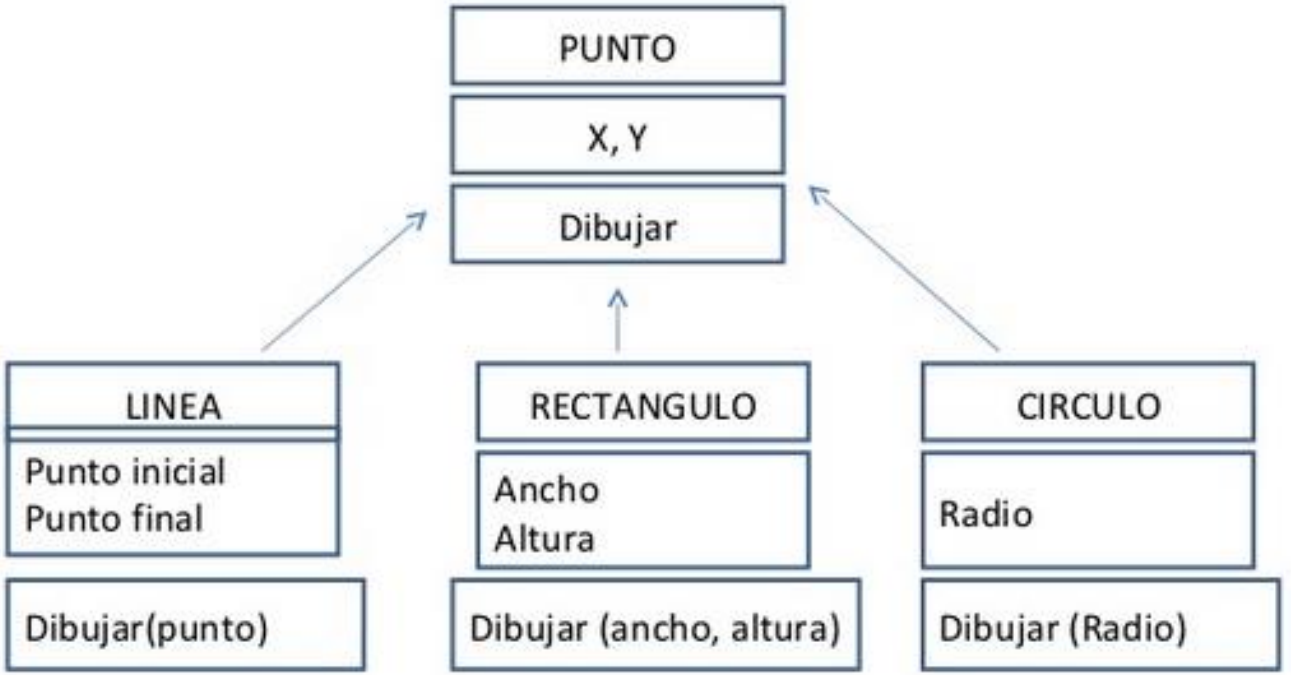
Es la capacidad que tiene los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación.



Polimorfismo de Sobrecarga

POLIMORFISMO DE SOBRECARGA

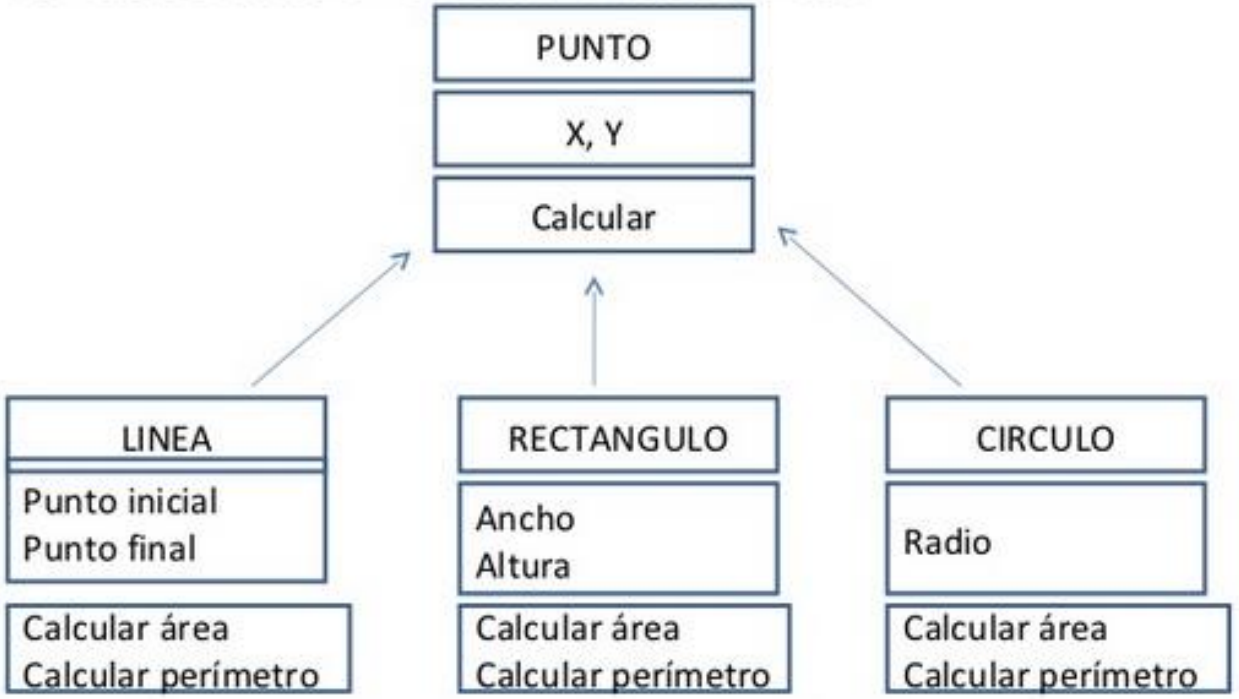
- OCURRE CUANDO LAS FUNCIONES DEL MISMO NOMBRE EXISTEN, CON FUNCIONALIDAD SIMILAR.



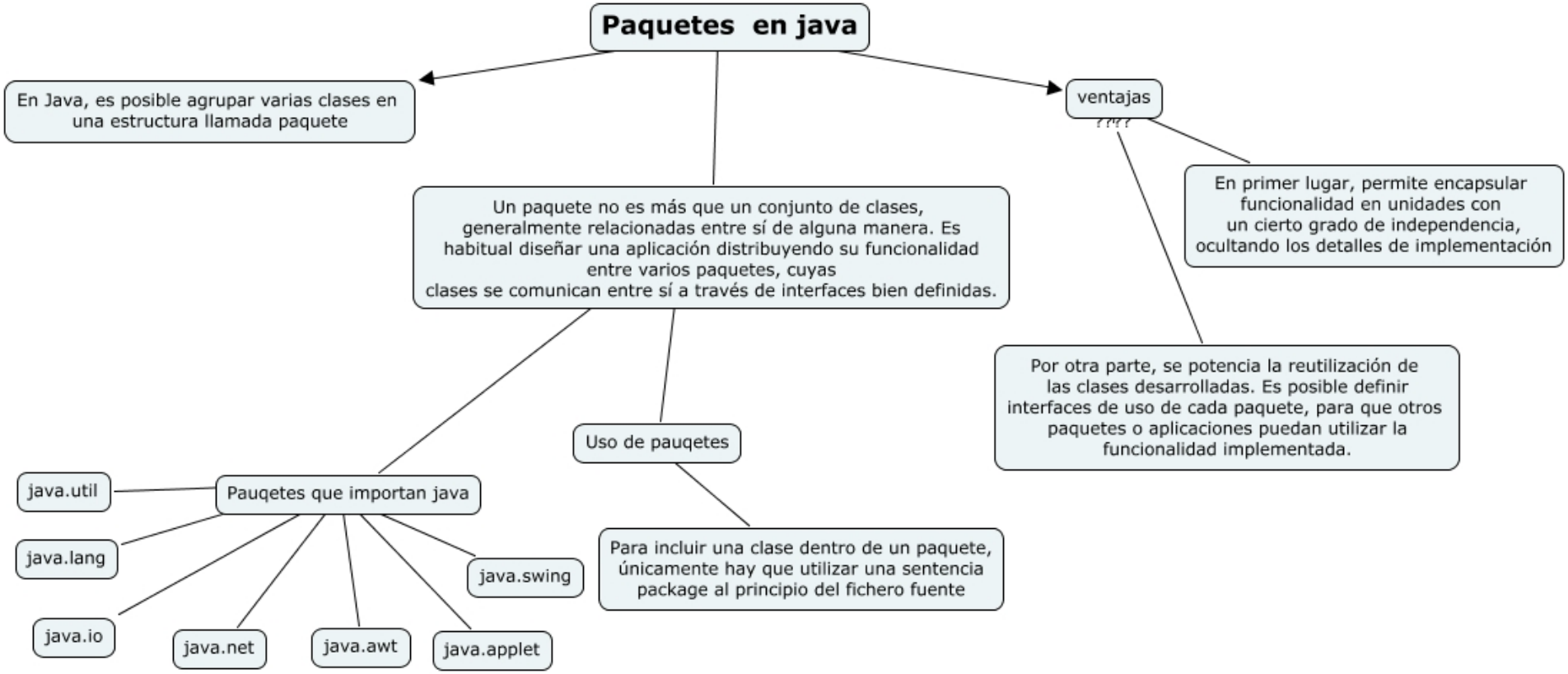
Polimorfismo de Sobre-escritura

POLIMORFISMO DE SOBRE ESCRITURA

- Se refiere a la redefinición de los métodos de la clase base en las subclases.



Paquetes


Paquetes en java
En Java, es posible agrupar varias clases en una estructura llamada paquete
ventajas
Un paquete no es más que un conjunto de clases, generalmente relacionadas entre sí de alguna manera. Es habitual diseñar una aplicación distribuyendo su funcionalidad entre varios paquetes, cuyas clases se comunican entre sí a través de interfaces bien definidas.
En primer lugar, permite encapsular funcionalidad en unidades con un cierto grado de independencia, ocultando los detalles de implementación
Por otra parte, se potencia la reutilización de las clases desarrolladas. Es posible definir interfaces de uso de cada paquete, para que otros paquetes o aplicaciones puedan utilizar la funcionalidad implementada.
Uso de pauqetes
Pauqetes que importan java
java.util
java.lang
java.io
java.net
java.awt
java.applet
java.swing
Para incluir una clase dentro de un paquete, únicamente hay que utilizar una sentencia package al principio del fichero fuente