{PROG} dit UPM

Programación
Tema 4: Métodos

{PROG} dit UPM

Contenidos

- 1. El concepto de método
- 2. Definición: cabecera
- 3. Definición: cuerpo
- 4. Uso de métodos
- 5. Métodos útiles

{PROG} dit UPM

Métodos

- Un método es un bloque de código con una misión:
  - Puede utilizar unos argumentos de entrada para producir un resultado
  - Puede manipular atributos de un objeto (setters/getters)
- Un conjunto de métodos define una abstracción que nos facilita resolver **nuestro problema** usando nuestros propios conceptos

Definición de métodos

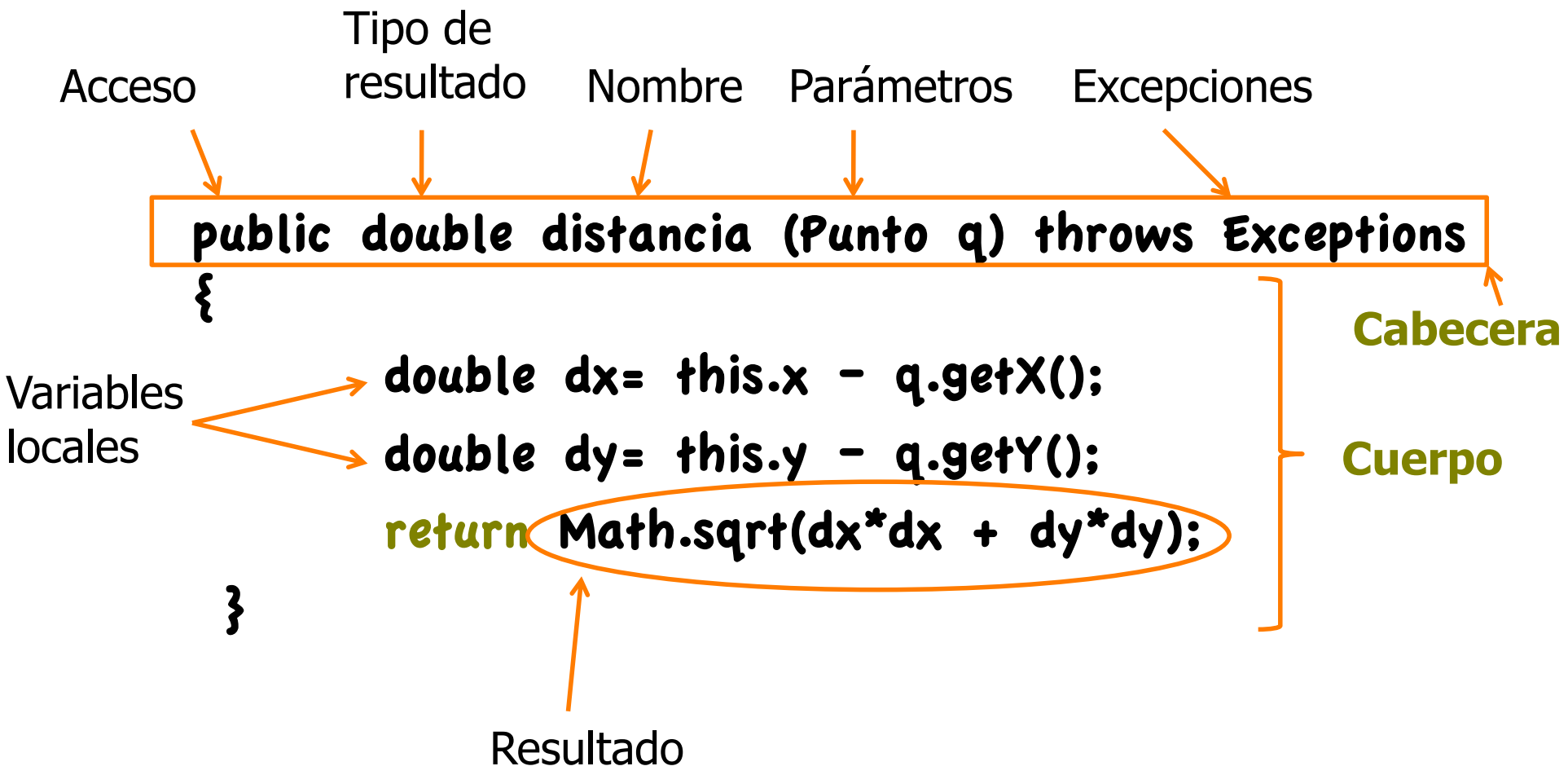
- Cabecera

  acceso tipo nombre (parametros ...) excepciones

- Cuerpo
  - bloque
  - secuencia de instrucciones
  - declaración de **variables automáticas**
  - los **parámetros se comportan como variables**

- Colocación
  - Dentro de las clases, el orden de los métodos no importa
  - **NO** dentro de otro método ni fuera de una clase

Anatomía


Acceso
Tipo de resultado
Nombre
Parámetros
Excepciones
public double distancia (Punto q) throws Exceptions
{
double dx= this.x - q.getX();
double dy= this.y - q.getY();
return Math.sqrt(dx*dx + dy*dy);
}
Cabecera
Cuerpo
Variables locales
Resultado

Signatura

- La signatura de un método está compuesta por
  - **<clase, nombre, <tiposDeLosParámetros> >**
- No forman parte ni el tipo de retorno ni las excepciones (si hay)
- **No puede haber 2 métodos con igual signatura**
  - Java no sabría a cual nos referimos
  - Si hay varias signaturas, decimos que hay *sobrecarga*

<Punto, distancia, <Punto> >

<Punto, distancia, <double, double>

{PROG} dit UPM

Accesibilidad

- **private**
  - sólo es accesible desde el texto de la propia clase
    - `this.metodo ( parametros )`
    - `metodo ( parametros )`
- no se dice nada
  - es accesible por cualquier clase del mismo paquete
    - `objeto.metodo ( parametros )`
- **public**
  - es accesible por cualquiera
    - `objeto.metodo ( parametros )`
- aplicable a atributos y métodos

{PROG} dit UPM

Parámetros

- Parámetro **formal**: el nombre asignado a los parámetros al definir el método
  - `public void mueve (double nuevox, double nuevoy) { ... }`
- Parámetro **real**: valores con los que se llama al método
  - `p.mueve (3.15, Math.sqrt(22.11*x)+9);`
- **Deben ser compatibles en posición, número y tipo**

Parámetros

- Se crean variables nuevas internas al método **cada vez que se llama**, cuyo valor es **una copia del valor** que se pasa como parámetro
  - Si son de tipos primitivos: por valor
    - ***se hace una copia del valor***
  - Si son de tipos referenciados: por referencia (nombreDelObjeto)
    - **Se hace una copia de la referencia** (se comparte el objeto referenciado, pero no se duplica el objeto)

Paso del valor

- tipos primitivos
  - enteros, reales, boolean, caracteres
  - se hace una copia
  - **las modificaciones no se notan fuera**

```
void m (int x) {
   x*= 2;
}

int n= 5;
System.out.println(n);
```

n=5

Paso del valor

- tipos primitivos
  - enteros, reales, boolean, caracteres
- se hace una copia
- **las modificaciones no se notan fuera**

```
void m (int x) {
   x*= 2;
}
```

x=5

x=10

```
int n= 5;
System.out.println(n);
m(n);
```

n=5

Paso del valor

- tipos primitivos
  - enteros, reales, boolean, caracteres
- se hace una copia
- **las modificaciones no se notan fuera**

```
void m (int x) {
  x*= 2;
}
```

```
int n= 5;
System.out.println(n);
m(n);
System.out.println(n);
```

n=5

Paso de la referencia

- Objetos creados por el programador
  - tras un new (...)
  - y también los array
- Se **comparte** el objeto
- Si el método llamado modifica el objeto, éste **queda modificado**
  - efecto colateral
- **Es a la vez cómodo...**
  y peligroso

Paso la referencia

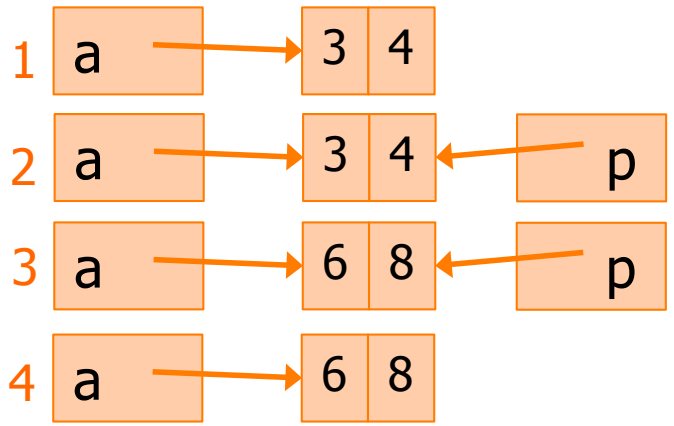
```
class Punto {
  int x, y;

  Punto (int _x, int _y) { x= _x; y= _y; }

  void escala (int s) { x*= s; y*= s; }

  void print () {
    System.out.println("x= " + x);
    System.out.println("y= " + y);
  } //fin print
2 void doble (Punto p) {
3    p.escala(2);
  } // fin doble
} // fin clase Punto

1 Punto a= new Punto(3, 4);
  a.print();
2 doble(a);
4 a.print();
```



Bloque

- Secuencia de **instrucciones** entre llaves
- Tamaño ideal inferior a una pantalla / hoja
  - Debe ser íntegro, puede tener tamaño 0 (sin cuerpo)
- Un método -> un concepto
- Se pueden declarar **variables automáticas**
- Se ejecuta hasta } o hasta encontrar un **return**
  - Puede haber return donde haga falta
- Los parámetros formales
  - **Se cargan** con los valores de los reales
  - **Se comportan** como variables
  - **Se eliminan** al acabar el bloque

return

- Los métodos pueden devolver valores
  - **void**: No devuelve nada
    - efectos tendrán que ser colaterales

```
public static void main (String[] args) { ... }
```

  - Un valor de un tipo primitivo

```
double s = Math.sqrt(2.0);
```

  - Un objeto de una clase definida

```
Punto q = p.desplaza(3, 4);
```

- sintaxis: **return expresion;**
- el compilador de Java comprueba que:
  - el tipo de la expresión coincide con el declarado en el método
  - en métodos que no devuelven nada (void) se puede usar sin expresión (return ;)
    - útil si no hay nada que ejecutar antes de llegar al final del método

Ámbitos

- Ámbito estático: **accesibilidad**
  - determina dónde puede referirse
    - a un componente de un objeto (atributo o método)
    - a una variable de un método (formal o automática)
- Ámbito dinámico: **vida**
  - determina cuándo nace y muere un atributo o una variable
  - cuando un contenedor muere, se pierde su valor y habrá que volver a inicializarlo

Atributos de una clase

- Nacen con la **construcción** del objeto
  - Objeto x= new Objeto(...);
- Viven hasta que el objeto es **inaccesible**
  - **se queda sin referencias**, referencia -> null
- Regla:
  - Todos los atributos deben ser **private**
    - Para controlar todos los accesos mediante métodos
    - Si un día queremos cambiar el valor de los atributos bastará con ajustar los métodos
    - Un método puede acceder a **TODOS** los campos de un objeto (atributos u otros métodos) porque está **DENTRO**, incluido los **private**

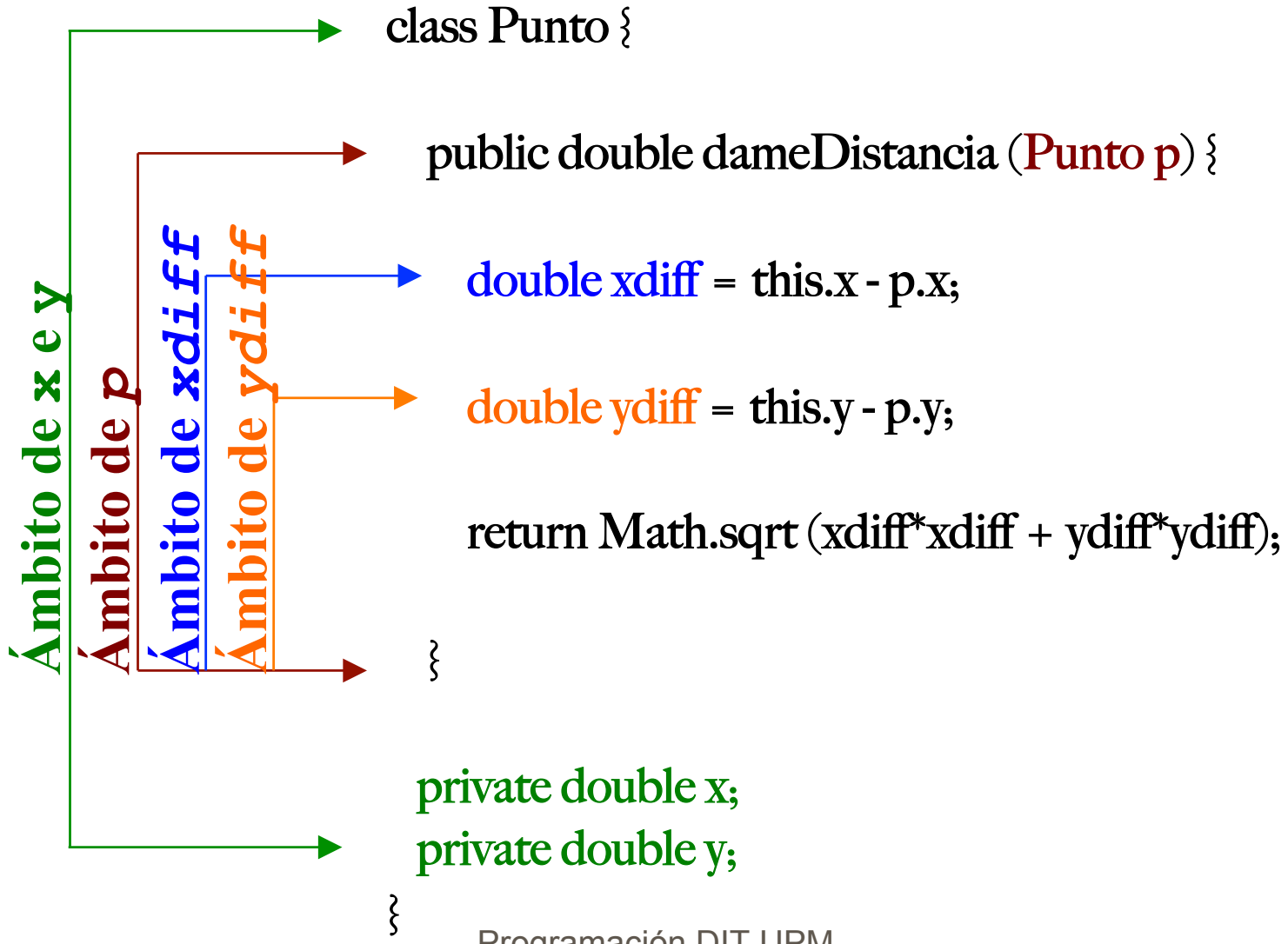
Métodos de una clase

- Nacen al invocar un método
  - Objeto.metodo(parametros);
- Viven hasta un return o }

Variables automáticas

- El cuerpo de un método es un bloque
  - se pueden declarar variables (automáticas)
- Las variables del bloque **class**
  - pueden usarse aunque textualmente aparezcan después
  - "Viven" mientras "viva" el objeto
- Las variables del bloque método
  - pueden usarse sólo **después de definirse**
    - textualmente debe aparecer **primero la definición** y **luego el uso**
  - Viven sólo dentro del bloque de instrucciones

Ámbito de variables



Ámbito dinámico

- Al llamar a un método, se crea un **ámbito dinámico**
  - se generan todas las variables-parámetros
  - se generan las variables automáticas
- Al salir del método
  - **desaparecen** dichas variables
- El ámbito dinámico existe (está activo) mientras estemos ejecutando el método
  - **incluso** si se llaman nuevos métodos desde este

¿Cómo se usa un método?

- Llamada a un método
  1. en el punto del programa en el que se llama
     se calculan los valores de los parámetros reales
  2. se salta al comienzo del método
  3. se cargan los parám. formales con los parám. reales
  4. se ejecuta el bloque hasta que se alcanza **return** o }
  5. se sustituye la llamada por el valor que devuelve el método
  6. se continúa la ejecución a continuación del punto en el que se llamó el método
- Método especial `main`
  - el intérprete de Java lo llama cuando ejecutamos un programa

Perímetro de un método

- Llamada:
  - objeto.metodo (parametros)
  - clase.metodo (parametros)
  - dentro de la misma clase basta poner el nombre
    - metodo (parametros)

- Retorno:
  - Pueden devolver un valor de cualquier tipo o clase
    - se antepone el **nombre del tipo o clase** a devolver
  - o no devolver nada
    - se antepone la palabra **void**

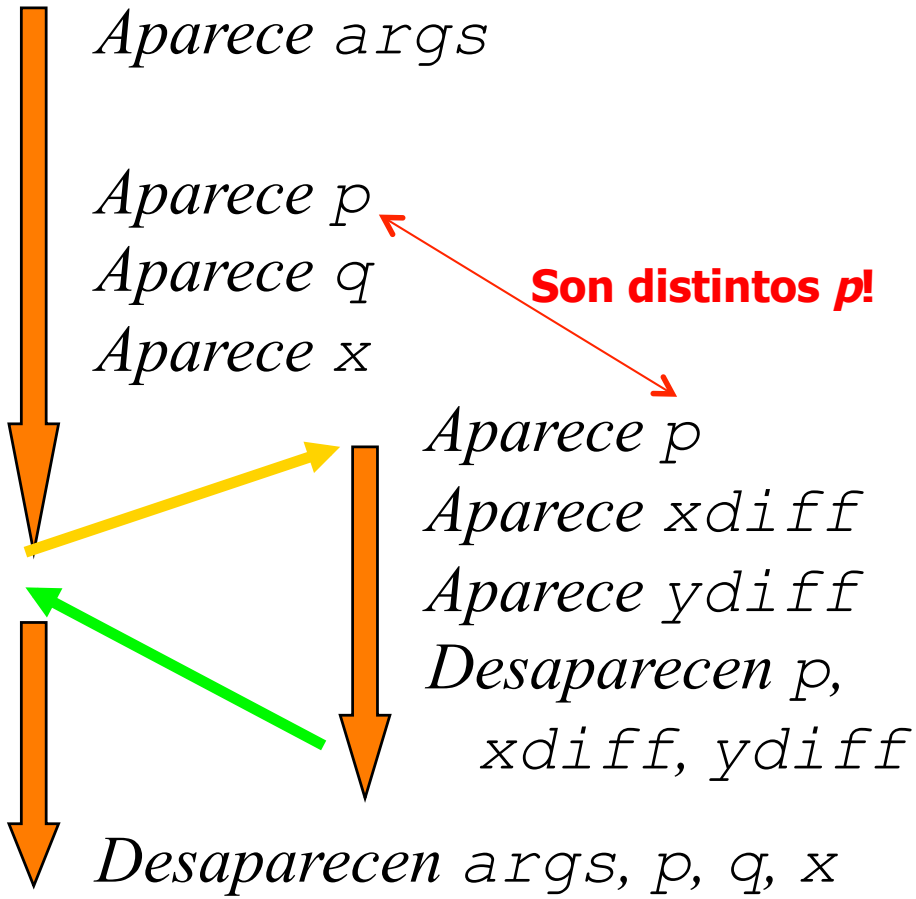
Ejemplo (V)

```
class EjemploPunto {

   ... main (String[] args) {

   Punto p = new Punto(),
   Punto q = new Punto();
   double x;
    :
   x = p.distancia(q);


  }
}
```



¿Cuándo definir un método?

- Método = concepto distinto y detallado
  - debe hacer una sola cosa
- Aumento de la legibilidad
  - su nombre debe indicar lo que hace (verbo)
  - añadir comentarios
- No debe ser muy largo
  - que quepa en una pantalla
- No debe tener muchos parámetros

¿Cómo definir un método?

- Escribir su cabecera completa
  - Darle nombre **informativo**
  - Indicar el tipo de retorno
  - Indicar los parámetros (los menos posibles)
- Escribir el cuerpo
  - Primeras sentencias para **comprobar** los parámetros
  - Operaciones
  - Definir variables locales si hace falta
  - Al final, indicar el valor de retorno con **return**
- Atención a los atributos, parámetros y variables
  - **Ámbitos lo menores posible**

Método constructor

- Para crear un objeto necesitamos construirlo
  - Pedir memoria a la JVM, darle un patrón a la memoria e inicializar los campos del objeto
  - Para crear un objeto, se construye con ***new***
  - Se crean e inicializan los atributos,...
- Es un método especial, con el **mismo nombre que la clase**, que no devuelve ningún valor

Método constructor

- Tipos de constructores:
  - **por defecto**, añadido por el compilador si no hay otro constructor
    - inicializa todos los atributos al **valor por defecto**
      (0, 0.0, false, \u0000, null)
  - definido por el usuario, **sin parámetros**
  - definido por el usuario, **con parámetros**

Método constructor

- El programador puede proporcionar un constructor específico
- Método con el nombre de la clase y sin tipo de devolución

```
private double [] posicion;
public Punto (double x, double y) {
     this.posicion = new double [2];
     this.posicion[0] = x;
     this.posicion[1] = y;
}
```

Método constructor

- Constructores definidos por el usuario
  - **Sustituye** al proporcionado por defecto
    - Pero la construcción por defecto sigue teniendo lugar en cuanto a petición de memoria
  - pueden llamar al constructor por defecto:
    primera instrucción

    `this ()`
- Razones para implementar un constructor:
  - se necesitan parámetros para el estado inicial,
  - construcción costosa, deben crearse correctamente,
  - el constructor no sea público

Accesores-modificadores

- **Accesores-get**
  - **Devuelven el valor de algún atributo**
  - No tienen parámetros
  - Tienen valores de retorno
  - Necesarios en general porque los atributos son privados

```
public double getX () {
   return this.x;
}
```

- **Modificadores-set**
  - **No devuelven ningún valor**
  - Tienen parámetros
  - No suelen tener valores de retorno
  - No se usan si un atributo NO debe ser modificado tras su creación

```
public void setX (double valorX) {
   this.x = valorX;
}
```

Otros métodos útiles

- Operaciones
  - Realizan **operaciones** con valores de atributos
  - Pueden tener parámetros o no llevarlos
  - Pueden tener, o no, valores de retorno

```
public String toString () {
  return “(“ + this.x + “,” + this.y + “)“;
}

public boolean equals (Punto otro) {
  return ((this.x == otro.x) && (this.y == otro.y));
}
```