

## **1.00 Clase 6**

### **Métodos y clases de Java**

### **Cómo escribir una clase**

- **En los trabajos para casa el estudiante deberá escribir sus propias clases**
  - Ya hemos visto clases en todos los ejemplos, pero no son clases típicas
    - Contienen un sólo método, main()
  - La mayoría de las clases no tienen un método main ()
- **Para crear un programa, deberá escribir varias clases, una de las cuales ha de tener un método main ()**

## Clase Point

```
public class SimplePoint {
    private double x, y;           // Data members
    public SimplePoint() {        // Constructor
        x= 0.0;
        y= 0.0; }
    // Métodos
    public double getX() { return x;}
    public double getY() { return y;}
    public void setX(double xval) { x= xval;}
    public void setY(double yval) { y= yval;}
    public void move(double deltaX, double deltaY) {
        x += deltaX;
        y += deltaY; }
} // Fin de la clase SimplePoint

// Esto no es un programa, ya que no tiene un método main()
// pero lo pueden utilizar otras clases que sí lo tengan
```

## Clase Point, main()

```
public class SimplePoint1 {
    public static void main(String[] args) {
        SimplePoint a= new SimplePoint();
        SimplePoint b= new SimplePoint();
        double xa= a.getX();
        double ya= a.getY();
        System.out.println("a= (" + xa + " , " + ya + ")");
        a.move(-9.0, 7.5);
        System.out.println("a= (" + a.getX() +
            " , " + a.getY() + ")");
    }
}
```

## ¿Por qué hacer esto?

- **Al construir una clase con métodos públicos y datos privados, usted está realizando una interfaz, no una implementación**
  - En caso de que necesite modificar una implementación, podrá hacerlo sin tener que romper códigos que dependan de ella, siempre que la interfaz (el conjunto de métodos) permanezca invariable
  - La modificación de sistemas de coordenadas, métodos de computación, etc., es bastante habitual, como se muestra en el ejemplo. Esto permite flexibilidad a la hora de ampliar o modificar el software.

## Clase Point, coordenadas polares

```
class SimplePoint {
    private double r, theta;    // Miembros de datos
    public SimplePoint() {     // Constructor
        r= 0.0;
        theta= 0.0; }
    // Métodos (las funciones Trig utilizan radianes)
    public double getX() { return r* Math.cos(theta);}
    public double getY() { return r* Math.sin(theta);}
    public void setX(double xval) {
        double yval= r*Math.sin(theta);
        r= Math.sqrt(xval*xval + yval*yval);
        theta= Math.atan2(yval, xval); }
}
```

## Clase Point, polar, p.2

```
public void setY(double yval) {
    double xval= r*Math.cos(theta);
    r= Math.sqrt(xval*xval + yval*yval);
    theta= Math.atan2(yval, xval); }

public void move(double deltaX, double deltaY) {
    double xval= r*Math.cos(theta);
    double yval= r*Math.sin(theta);
    xval += deltaX;
    yval += deltaY;
    r= Math.sqrt(xval*xval + yval*yval);
    theta= Math.atan2(yval, xval);
}

}

// Se pueden invocar desde el mismo main() que antes y producen
// los mismos resultados (distintos de los errores de redondeo)
```

## Métodos de Java

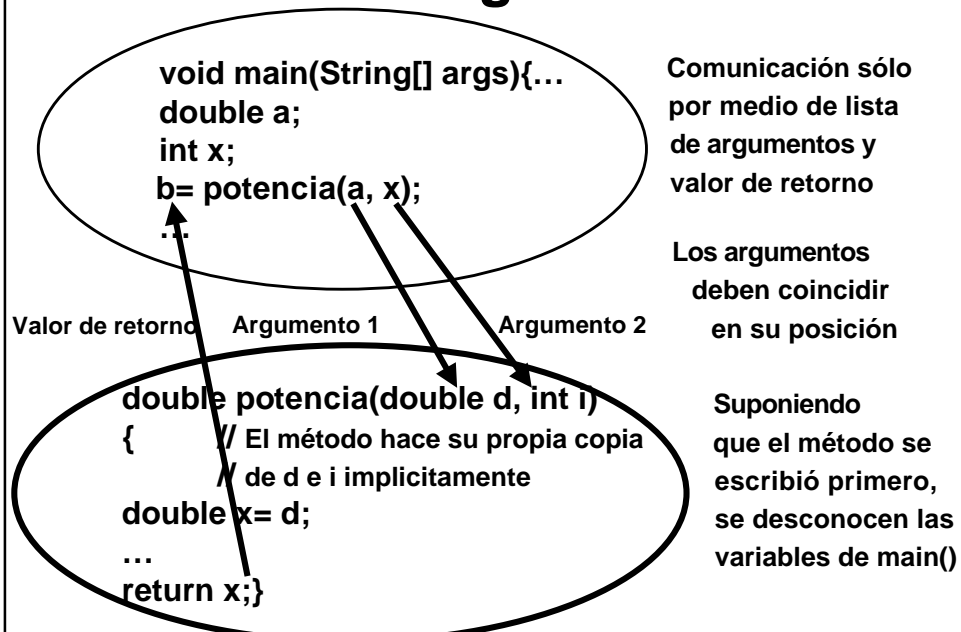
- **Los métodos son la interfaz o las comunicaciones entre las clases:**
  - Proporcionan un modo de invocar una misma operación desde distintas ubicaciones del programa evitando la repetición de códigos
  - Ocultan los detalles de la implementación al que realiza la llamada o al usuario del método
  - Las variables definidas dentro de un método no son visibles para los elementos que llaman a dicho método; disponen de un ámbito local, que se limita al interior del mismo
  - El método tampoco puede observar las variables del que lo llama. Existe una separación lógica entre ambos, lo que evita conflictos entre nombres de variables

## Invocación de un método

```
public class Power1 {
    public static void main(String[] args) {
        double a= 10.0, b;
        int x= 3;
        System.out.println("Main: a= " + a + ", x= " + x);
        b= potencia(a, x);
        System.out.println("Main: Resultado b= " + b);
    }

    public static double potencia(double d, int i){
        System.out.println("Potencia: d=" + d + ", i= " + i);
        double x= d;        //x diferente a la x de main
        for (int j= 1; j < i; j++)
            x *= d;          // x = x * d;
        return x; }
}
```

## Paso de argumentos



## Llamadas por valor

- **Java sólo acepta llamadas por valor cuando al pasar argumentos a un método:**
  - El método realiza una copia local de cada argumento
  - y a continuación opera en esa copia local
  - Un método no puede cambiar el valor de un argumento (variable) del método que lo llama
  - Un método sólo puede devolver datos al que lo llama:
    - Mediante el valor de retorno, o
    - modificando los datos de los objetos que le pasan (lo veremos en profundidad más adelante)
  - El valor de retorno puede ser de tipo *void*, es decir, no devuelve nada. El método `main()` tiene un valor de retorno *void*
  - El otro mecanismo, soportado por C++ y otros lenguajes, es de llamada por referencia, lo que hace posible que el método cambie los argumentos.

## Ejemplo de llamada por valor

```
public class CallByValue {
    public static void main(String[] args) {
        int i= 3;
        double d= 77.0;
        System.out.println("Main 1: i= " + i + ", d= " + d);
        triple(i, d); // No devuelve ningún valor
        System.out.println("Main 2: i= " + i + ", d= " + d);
        //Parte secundaria del ejemplo: conversión de argumentos
        triple(i, i); // Ok-Java convierte int en double
        System.out.println("Main 3: i= " + i);
    }
    public static void triple(int ii, double dd) {
        System.out.println("Triple 1: ii= " +ii+ ", dd= " +dd);
        ii *= 3; // ii= ii*3;
        dd *= 3.0;
        System.out.println("Triple 2: ii= " +ii+ ", dd= " +dd);
    }
}
```

## Llamada por valor con objeto

```
public class CallObjExample {
    public static void main(String[] args) {
        Demo d= new Demo(3);
        int i= 3;
        System.out.println("Main1: i= " + i + ", d.a= " + d.a);
        triple(i, d); // No devuelve ningún valor
        System.out.println("Main2: i= " + i + ", d.a= " + d.a);
    }
    public static void triple(int ii, Demo dd){
        System.out.println("T1: ii= "+ ii + ", dd.a= " + dd.a);

        dd.a *= 3;
        System.out.println("T2: ii= "+ ii + ", dd.a= " + dd.a);
    }
}

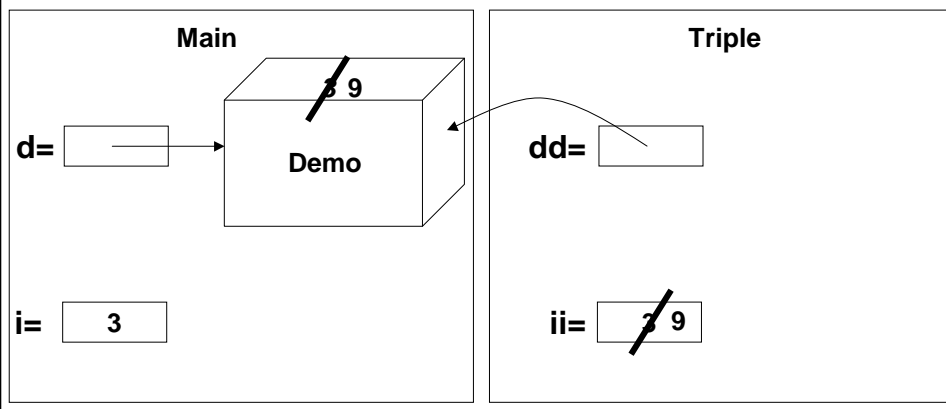
class Demo {
    public int a; // Public sólo para mostrar la llamada por valor
    public Demo(int aa) { a= aa; }
}
```

## Llamada por valor con objeto

```
Demo d= new Demo(3);
int i= 3;
triple(i, d);
```

→

```
ii *= 3;
dd.a *=3;
```



## Creación de clases

- **Una empresa que fabrica ventanas tiene 3 plantas:**
  - La planta A fabrica marcos de madera
    - Produce 200 marcos/día, a un coste de 25 \$/unidad
  - La planta B fabrica cristal
    - Produce 200 hojas de cristal/día, a un coste de 5\$/hoja
  - La planta C, vecina de B, monta las ventanas
    - Monta 200 ventanas/día, a un coste de 12\$/unidad
- **Escribamos las clases para este problema:**
  - De entre las varias posibilidades existentes le enseñaremos a utilizar la más sencilla
  - No se trata de una solución general. Sólo servirá para un producto, empleando un marco y una hoja de cristal. A veces puede resultar un tanto extraña, pero se trata de un punto de partida habitual.

## Clase Planta

- Escriba la clase correspondiente a las plantas que fabrican marcos o cristal, sin tener en cuenta el montaje de ventanas.

- Campos de datos:

---

---

---

- Constructor:

---

---

---

---

---

## Clase Plant (planta)

- **Campos de datos:**

```
private String part;           // "marco", "cristal"  
private double unitCost;  
private int production;
```

- **Constructor:**

```
Plant(String p, double c, int prod) {  
    part= p;  
    unitCost= c;  
    production= prod;  
}
```

## Métodos de la clase Plant

- **No escriba métodos "set".** Los datos de la planta los fija el constructor, y nosotros no vamos a modificarlos en este problema.
- **Métodos "Get", para cada campo private:**

```
_____  
_____  
_____
```

- **Método computacional**

```
_____
```

## Métodos de la clase Plant

- **No escriba métodos "set".** Los datos de la planta los fija el constructor, y nosotros no vamos a modificarlos en este problema.
- **Métodos "Get":**

```
public String getPart() { return part;}  
public double getUnitCost() { return cost; }  
public int getProduction() {return production;
```
- **Método computacional**

```
public double dailyCost() {  
    return cost*production;  
}
```

## Clase de la planta de montaje (Assembly)

- **Montamos un producto a partir de dos partes**
- **Campos de datos:**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

- **Constructor**

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## Clase de la planta de montaje (Assembly)

- Tenemos un producto fabricado a partir de 2 partes

- Campos de datos:

```
private String product;  
private Plant plant1;  
private Plant plant2;  
private double assemblyCost;
```

- Constructor

```
Assembly(String p, Plant p1, Plant p2, double ac) {  
    product= p;  
    plant1= p1;  
    plant2= p2;  
    assemblyCost= ac;  
}
```

## Métodos de la clase Assembly

- No escriba métodos “set” y “get”.
- Métodos computacionales (coste, producción)

---

---

---

---

---

---

---

---

---

---

---

## Métodos de la clase Assembly

- No escriba métodos "set" y "get"
- Métodos computacionales

```
public double getTotalCost() {  
    double c1= plant1.getUnitCost();  
    double c2= plant2.getUnitCost();  
    return c1 + c2 + assemblyCost;  
}
```

```
public double getProduction() {  
    int p1= plant1.getProduction();  
    int p2= plant2.getProduction();  
    return Math.min(p1, p2);  
}
```

## Main()

- Escriba un método main() para crear las tres plantas y mostrar el coste por producto y la tasa de producción

---

---

---

---

---

---

---

---

---

---

## Main()

- **Escriba un método main() para crear las tres plantas y mostrar el coste por producto y la tasa de producción**

```
public class GlassExample {
    public static void main(String[] args) {
        Plant A= new Plant("marcos", 25.00, 200);
        Plant B= new Plant("cristal", 5.00, 200);
        Assembly C= new Assembly("ventana", A, B, 12.00);
        double tc= C.getTotalCost();
        int pr= C.getProduction();
        System.out.println("Coste total: " + tc);
        System.out.println("Producción: " + pr);
        System.exit(0);
    }
}
```