

Clasificación elementos de una clase

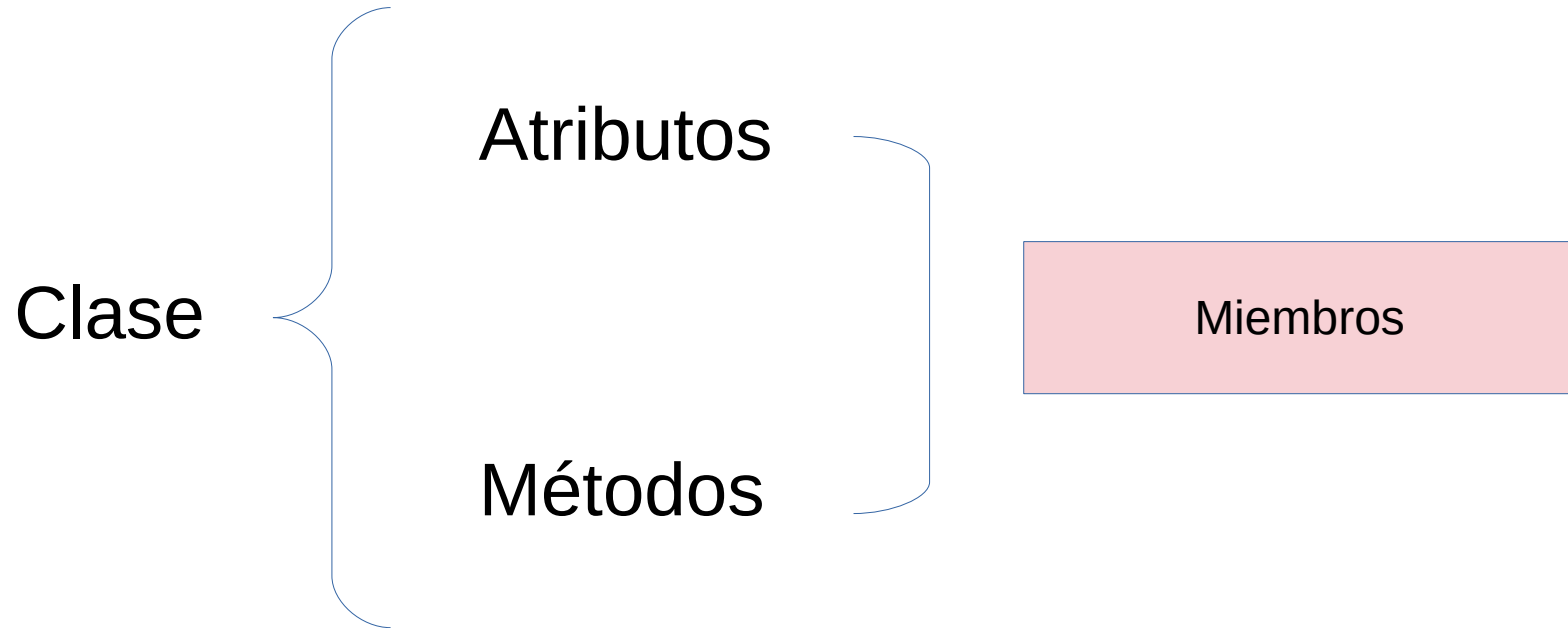
Jorge I. Meza

jimezam@autonoma.edu.co

Contenidos

- Tipos de miembros
 - De instancia
 - De clase
 - *Setters y getters*
 - Otros métodos
- Tipos de métodos
 - Principal o `main`
 - Constructores y destructores
- Pseudoobjeto `this`
- Sobrecarga de métodos

Partes de una clase



Tipos de miembros

- Miembros de instancia
- Miembros de clase

Miembros de instancia

- Sus valores y manejo, son independientes para cada instancia (objeto individual).
- La modificación de uno específico de ellos, no representa ninguna modificación para los demás.
- Son el tipo de miembros que se han utilizado hasta el momento en el curso.

Miembros de instancia

```
class Persona {  
    String nombre;  
    int edad;  
    String telefono;  
    String cedula;  
  
    void cumplirAños() {  
        edad++;  
    }  
}
```



Miembros de instancia

```
Persona rosa = new  
    Persona();  
rosa.nombre = "Rosa";  
rosa.edad = 20;
```

```
Persona juan = new  
    Persona();  
juan.nombre = "Juan";  
juan.edad = 25;
```



Rosa

20

Juan

25

Miembros de instancia



```
rosa.nombre = "Roxana";
```

Roxana

20

Juan

25

Miembros de instancia



```
juan.cumplirAños();
```

Roxana

20

Juan

26

Miembros de clase

- Sus valores y manejo, son **el mismo para todas las instancias** (objetos) creadas con la clase.
- La modificación de un atributo de clase es vista desde los demás objetos de la misma clase.
- Se determinan con el modificador `static`.
- Se utiliza en casos muy específicos.

Miembros de clase

```
class Persona {  
    String nombre;  
    static String ciudad;  
  
    static void todosAManizales() {  
        ciudad = "Manizales";  
    }  
}
```



Miembros de clase

```
Persona rosa = new  
    Persona();  
rosa.nombre = "Rosa";  
rosa.ciudad = "Pereira";
```

```
Persona juan = new  
    Persona();  
juan.nombre = "Juan";
```



Rosa

Juan

Pereira

Miembros de clase

```
System.out.println  
(juan.ciudad) ;
```



Rosa

Juan

Pereira

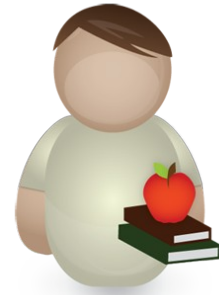
Miembros de clase

```
juan.ciudad = "Medellín";
```

```
System.out.println  
(rosa.ciudad);
```



Rosa



Juan

Medellín

Miembros de clase

```
juan.ciudad = "Medellín";
```



```
Persona.ciudad = "Pasto";
```



Miembros de clase

- Desde un método de clase no se pueden acceder a los atributos de instancia.

```
class Persona {  
    String nombre;  
    static String ciudad;  
  
    static void miMetodo() {  
        nombre = "Pericles";  
        ciudad = "Puerto Inírida";  
    }  
}
```





Miembros de clase

- Igual sucede de manera inversa, desde un método de instancia no se pueden acceder a los atributos de clase.

```
class Persona {  
    String nombre;  
    static String ciudad;  
  
    void miMetodo() {  
        nombre = "Pericles";  
        ciudad = "Puerto Inírida";  
    }  
}
```



Tipos de métodos

- Los métodos se clasifican de acuerdo con su misión en la clase.
- Los métodos se clasifican en:
 - Principal (`main`)
 - Constructores
 - Destruyores 
 - Analizadores o *setters*
 - Modificadores o *getters*
 - Definidos por el usuario 

¡Poco conocidos en Java!

Otros métodos que no quepan en esta clasificación

Método principal o `main`

- Este método es muy particular, no pertenece formalmente a la clase en la cual se escribe.
- Su misión es la de realizar las tareas mínimas iniciales para que el programa inicie su ejecución.
- No debería tener código complejo.
- En general, se utiliza para instanciar la lógica de negocio y la lógica de presentación, y ponerlas a trabajar.

Método principal o `main`

- Generalmente se ubica en la clase control o controladora o se crea una clase ex profeso para incluir este método.
- La firma de este método debe ser la siguiente:

```
public static void main(String[] args)
```

Métodos constructores

- Los métodos constructores son útiles para **definir el código que se debe ejecutar justo en el momento en que se crea un nuevo objeto.**
- Esto permite definir un código que está garantizado ya se ejecutó cuando se recibe una instancia.
- Se utilizan comúnmente para reservar recursos que el objeto requiere (archivos, conexiones, bases de datos, etc.) y dar valores iniciales a sus atributos.

Métodos constructores

- Un error común es el considerar que los métodos constructores "sirven para crear o construir los objetos". Esto es falso ya que la creación de objetos es realizada por el operador `new`.



Métodos constructores

- Los métodos constructores tienen las siguientes características.
 - Se deben llamar exactamente igual que la clase.
 - No deben tener tipo de retorno (como si lo tienen los demás métodos).

Métodos constructores

```
class Botella {  
    boolean estado;  
    int contenido;  
  
    Botella() {  
        estado = true;  
        contenido = 0;  
    }  
}
```

```
Botella b = new  
    Botella();
```


Métodos constructores

```
class Botella {  
    boolean estado;  
    int contenido;  
  
    Botella() {  
        estado = true;  
        contenido = 0;  
    }  
}
```

- Este constructor da valores estáticos iniciales a sus atributos.
- Podemos estar seguros que toda botella, inicialmente estará abierta (`true`) y vacía (`0`).

Métodos constructores

```
class Botella {  
    boolean estado;  
    int contenido;  
  
    Botella(boolean e)  
{  
    estado = e;  
    contenido = 0;  
}  
}
```

```
Botella b1 = new  
    Botella(false);  
  
Botella b2 = new  
    Botella(true);
```

Métodos constructores

```
class Botella {  
    boolean estado;  
    int contenido;  
  
    Botella(boolean e)  
    {  
        estado = e;  
        contenido = 0;  
    }  
}
```

- Este constructor asigna un valor inicial al estado de acuerdo con el parámetro enviado.
- Podemos estar seguros que toda botella recién creada, estará *abierta* o *cerrada* de acuerdo con la necesidad del desarrollador.

Métodos constructores

```
class Botella {  
    boolean estado;  
    int contenido;  
  
    Botella(boolean e,  
int c) {  
        estado = e;  
        contenido = c;  
    }  
}
```

```
Botella b1 = new  
    Botella(false,  
20);
```

```
Botella b2 = new  
    Botella(false,  
100);
```

```
Botella b3 = new  
    Botella(true,  
50);
```

Métodos constructores

```
class Botella {  
    boolean estado;  
    int contenido;  
  
    Botella(boolean e,  
int c) {  
        estado = e;  
        contenido = c;  
    }  
}
```

- Este constructor asigna un valor inicial al estado y otro al contenido, de acuerdo con el parámetro enviado.
- Podemos estar seguros que toda botella recién creada, estará *abierta* o *cerrada* y con el contenido requerido de acuerdo con la necesidad del desarrollador.

Constructor "fantasma"

- Cuando una clase no define ningún constructor, la Máquina Virtual de Java (JVM) "agrega" automáticamente un constructor vacío.
- Esto es necesario para permitir la creación de objetos a partir de la clase (sin constructor).

Constructor "fantasma"

```
class Camion {  
    String marca;  
    String placa;  
}
```



```
class Camion {  
    String marca;  
    String placa;  
    Camion() {}  
}
```

```
Camion c = new Camion() ;
```

Métodos destructores

- Los métodos destructores son útiles para **definir el código que se debe ejecutar justo antes del momento en que un objeto sea destruido.**
- Esto permite definir un código que está garantizado se ejecute antes de perder el control sobre la instancia.
- Se utilizan comúnmente para liberar los recursos reservados por el objeto durante su vida útil.

Métodos destructores

- En Java no son muy conocidos y probablemente hoy estén obsoletos (*deprecated*).
- Esto se debe a que en Java, el manejo de la memoria es automático (*garbage collector*), esto significa que el desarrollador no debe destruir explícitamente los objetos que él crea.
- Por este motivo, el momento justo de la destrucción del objeto (y la ejecución del método destructor) no son determinísticos.

Métodos de acceso o *mutadores*

- Analizadores (***getters***)

Permiten **obtener** el valor de un atributo del objeto

- Modificadores (***setters***)

Permiten **modificar** el valor de un atributo del objeto

Métodos analizadores o *getters*

```
class Camion {  
    int capacidad;  
    String placa;  
  
    int getCapacidad() {  
        return capacidad;  
    }  
  
    String getPlaca() {  
        return placa;  
    }  
}
```

Métodos modificadores o *setters*

```
class Camion {  
    int capacidad;  
    String placa;  
  
    void setCapacidad(int c) {  
        capacidad = c;  
    }  
  
    void setPlaca(String p) {  
        placa = p;  
    }  
}
```

Métodos de acceso o *mutadores*

- La importancia de implementar estos métodos *setters* y *getters*, se verá más adelante en el capítulo de **Ocultamiento de Información**.



Pseudoobjeto `this`

- Se conoce como *pseudoobjeto* ya que se comporta como un objeto (tiene atributos y métodos) pero no es necesario crearlo, como a los demás objetos regulares.
- Está disponible "automáticamente" en cada clase.
- Hace referencia a la instancia de la clase (objeto) en la cual es utilizado.

Pseudoobjeto `this`

- Similar a la frase "yo mismo", se refiere a diferentes sujetos según quien la mencione.
- En general se puede utilizar con tres objetivos distintos.
 - 1) Permitir al objeto referenciarse a si mismo en los casos en que se requiera.
 - 2) Resolver ambigüedades entre atributos y parámetros.**
 - 3) Llamar a un constructor desde otro constructor de la misma clase (reutilización).

Resolver ambigüedades entre atributos y parámetros

```
class Revista {  
    String nombre;  
    int edicion;  
    Revista(String n, int e) {  
        nombre = n;  
        edicion = e;  
    }  
}
```

¿Hay alguna ambigüedad?

Resolver ambigüedades entre atributos y parámetros

```
class Revista {  
  
    String nombre;  
    int edicion;  
  
    Revista(String nombre,  
            int edicion) {  
  
        nombre = nombre;  
        edicion = edicion;  
  
    }  
  
}
```

¿Qué tal ahora?

¿Hay alguna ambigüedad?

Resolver ambigüedades entre atributos y parámetros

```
class Revista {  
    String nombre;  
    int edicion;  
    Revista(String nombre,  
             int edicion) {  
        nombre = nombre;  
        edicion = edicion;  
    }  
}
```

¿nombre = nombre?

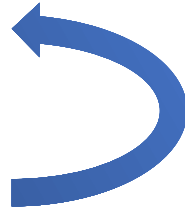
¿Quién se asigna con
quién?

¿Quién se debería
asignar a quién?



Resolver ambigüedades entre atributos y parámetros

```
class Revista {  
    String nombre;  
    int edicion;  
    Revista(String nombre,  
             int edicion) {  
        // ...  
    }  
}
```



¿Quién se debería
asignar a quién? 🤔

Resolver ambigüedades entre atributos y parámetros

```
class Revista {  
    String nombre;  
    int edicion;  
    Revista(String nombre,  
             int edicion) {  
        this.nombre = nombre;  
        this.edicion = edicion;  
    }  
}
```

this.nombre

Mi atributo nombre

nombre

Parámetro nombre

Sobrecarga de métodos

- Recordar que los métodos son las acciones que puede realizar un objeto mediante la manipulación de sus atributos y parámetros.
- Algunas veces un objeto necesita realizar una misma acción sobre un conjunto de parámetros diferente.

Ejemplo

- Una Persona puede comer Hamburguesa.
- Pero también puede requerir comer Pizza o Carne u otros deliciosos alimentos.

Ejemplo



```
class Persona {  
    void comer1 (Hamburguesa h)  
    { ... }  
  
    void comer2 (Pizza p)  
    { ... }  
  
    void comer3 (Carne c)  
    { ... }  
}
```

Las acciones `comer1`,
`comer2`, `comer3` realmente
no existen.

```
class Persona {  
    void  
    comerHamburguesa (Hamburguesa h)  
    { ... }  
  
    void comerPizza (Pizza p)  
    { ... }  
  
    void comerCarne (Carne c)  
    { ... }  
}
```

La acción es la misma en
todos los métodos, se
deberían llamar `comer`.

Ejemplo



```
class Persona {  
    void comer (Hamburguesa h)  
    { ... }  
  
    void comer (Pizza p)  
    { ... }  
  
    void comer (Carne c)  
    { ... }  
}
```

¿Será posible declarar
varios métodos con igual
nombre en la misma
clase?

Si, gracias a la
**sobrecarga de
métodos.**

Sobrecarga de métodos

- La sobrecarga de métodos le permite al desarrollador implementar **varios métodos con el mismo nombre en la misma clase, pero con diferentes parámetros.**
- Permite modelar diferentes versiones de una misma acción dependiendo de los datos de entrada de la misma.
- En inglés se conoce como ***overloading***.

Ejemplo

```
class Tanque {  
    int volumen;  
  
    Tanque() {  
        volumen = 0;  
    }  
  
    void llenar() {  
        this.llenar(1);  
    }  
  
    void llenar(int veces) {  
        for(int i=0; i<veces; i++)  
            volumen ++;  
    }  
}
```

- El Tanque se puede llenar de dos maneras: *individual* o *varias unidades*.
- El método `llenar()` reutiliza `llenar(int)` para no repetir código.

Ejemplo

```
class Tanque {  
    int volumen;
```

```
    Tanque() {  
        volumen = 0;  
    }
```

```
    void llenar() {  
        this.llenar(1);  
    }
```

```
    void llenar(int veces) {  
        for(int i=0; i<veces; i++)  
            volumen ++;  
    }
```

```
Tanque miTanque = new  
    Tanque();
```

```
miTanque.llenar();
```

```
miTanque.llenar(5);
```



Ejemplo



```
public class CalculadoraAreas {  
  
    private double calcularAreaRectangulo(double base, double altura) {  
        return base * altura;  
    }  
  
    public double calcularArea(double lado) {  
        return lado * lado;  
    }  
  
    public double calcularArea(double lado1, double lado2) {  
        return lado1 * lado2;  
    }  
  
    public double calcularArea(double base, double altura) {  
        return base * altura / 2;  
    }  
}
```



Ejemplo

```
class Bus {  
    Persona[]  
    sillas;  
  
    Bus() {  
        sillas = new  
Persona[10];  
    }  
  
    // ...
```

```
        void sentar(Persona p, int  
lugar) {  
            sillas[lugar] = p;  
        }  
  
        void sentar(String nombre,  
String cedula, int edad, int  
lugar) {  
            Persona tmp = new  
Persona(nombre, cedula, edad);  
  
            sentar(tmp, lugar);  
        }  
    }
```

Ejemplo

```
class Anfibio {  
    void respirar(Agua a) {  
        // Usando branquias  
    }  
  
    void respirar(Aire a) {  
        // Usando pulmones  
    }  
}
```

¿Alguna



pregunta?