

Sesión 05: Terminología OO

Programación 2

1. Introducción a la programación orientada a objetos (POO).
 2. Clases y objetos.
-

Ángel Herranz

2020-2021

Universidad Politécnica de Madrid

En capítulos anteriores...

- Clases y objetos (intro)
- Objetos, referencias y variables (y primitivos)
- Clases: plantilla para crear objetos

En capítulos anteriores...

- Clases y objetos (intro)
- Objetos, referencias y variables (y primitivos)
- Clases: plantilla para crear objetos

Encapsular

datos y comportamiento

- Racionales

Objetos, referencias y variables

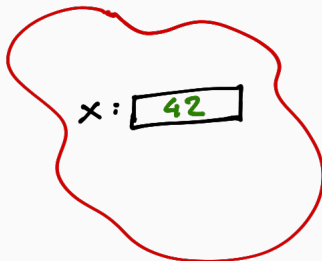
```
public class Sesion02 {  
    public static void main(String args[]) {  
        A a;  
        a = new A();  
    }  
}
```

a: null

Objetos, referencias y variables

```
public class Sesion02 {  
    public static void main(String args[]) {  
        A a;  
        a = new A();  
    }  
}
```

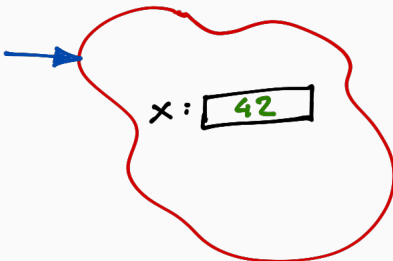
a: null



Objetos, referencias y variables

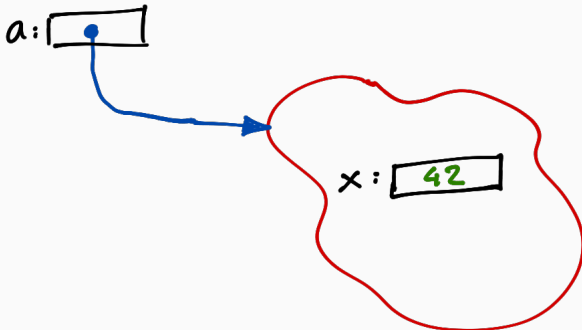
```
public class Sesion02 {  
    public static void main(String args[]) {  
        A a;  
        a = new A();  
    }  
}
```

a: null



Objetos, referencias y variables

```
public class Sesion02 {  
    public static void main(String args[]) {  
        A a;  
        a = new A();  
    }  
}
```



Encapsular: datos + comportamiento

```
class Cancion {
    String titulo;
    String interprete;
    int duracion;
    ...
    Cancion (String titulo,
            String interprete,
            int duracion) {
        this.titulo = titulo;
        this.interprete = interprete;
        this.duracion = duracion;
    }
    ...
    String duracion() {
        String minutos =
            "" + duracion / 60;
        String segundos =
            "" + duracion % 60;
        if (segundos.length() == 1)
            segundos
                = "0" + segundos;
        }
        return
            minutos + ":" + segundos;
    }
}
```


Encapsular: datos + comportamiento

```
class Cancion {
    String titulo;
    String interprete;
    int duracion;
    ...
    Cancion (String titulo,
             String interprete,
             int duracion) {
        this.titulo = titulo;
        this.interprete = interprete;
        this.duracion = duracion;
    }
    ...
    String duracion() {
        String minutos =
            "" + duracion / 60;
        String segundos =
            "" + duracion % 60;
        if (segundos.length() == 1)
            segundos
                = "0" + segundos;
        }
        return
            minutos + ":" + segundos;
    }
}
```

Encapsular: datos + comportamiento

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

```
String duracion() {  
    String minutos =  
        "" + duracion / 60;  
    String segundos =  
        "" + duracion % 60;  
    if (segundos.length() == 1)  
        segundos  
            = "0" + segundos;  
    }  
    return  
        minutos + ":" + segundos;  
    }  
}
```

Encapsular: datos + comportamiento

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
    String duracion() {  
        String minutos =  
            "" + duracion / 60;  
        String segundos =  
            "" + duracion % 60;  
        if (segundos.length() == 1)  
            segundos  
                = "0" + segundos;  
        }  
        return  
            minutos + ":" + segundos;  
    }  
}
```

Encapsular: datos + comportamiento

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

```
String duracion() {  
    String minutos =  
        "" + duracion / 60;  
    String segundos =  
        "" + duracion % 60;  
    if (segundos.length() == 1)  
        segundos  
            = "0" + segundos;  
    }  
    return  
        minutos + ":" + segundos;  
    }  
}
```

Terminología OO y Java¹

atributo, clase, constructor,
declaración, inicialización, instancia,
invocación, mensaje, miembro,
método, modificador, objeto,
observador, parámetro, puntero,
primitivo, referencia, tipo, variable

¹Sólo para empezar, porque por ejemplo Timothy Budd tiene un glosario de más casi 200 términos en su libro “Object Oriented Programming”.

Terminología OO y Java¹

*attribute, class, constructor,
declaration, initialization, instance,
invocation, message, member,
method, modifier, object,
observer, parameter, pointer,
primitive, reference, type, variable*

¹Sólo para empezar, porque por ejemplo Timothy Budd tiene un glosario de más casi 200 términos en su libro “Object Oriented Programming”.

Atributos y métodos

- Nadie usa los términos *datos* y *comportamiento*
- Datos: **atributos** (*attribute*)
- Comportamiento: **métodos** (*method*)
 - **Constructor**: mismo nombre que la clase
 - **Observador**: observa, devuelve algo, no modifica
 - **Modificador** modifica, no devuelve nada (**void**)
- Ámbos (datos y métodos): **miembros** (*member*)³

³Terminología muy Java, en concreto, *instance members*

Atributos y métodos

- Nadie usa los términos *datos* y *comportamiento*
- Datos: **atributos** (*attribute*)
- Comportamiento: **métodos** (*method*)
 - **Constructor**: mismo nombre que la clase
 - **Observador**: observa, devuelve algo, no modifica
 - **Modificador** modifica, no devuelve nada (**void**)
-  ¿Puede haber métodos *observamodificadores*?²
- Ámbos (datos y métodos): **miembros** (*member*)³

²Sí, pero hagamos caso a Bertrand Meyer y a su principio CQS (*Command Query Separation Principle*) y evitémoslos

³Terminología muy Java, en concreto, *instance members*

Terminología OO i

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

- **Clase:** la plantilla para crear instancias (objetos) de esa clase

Terminología OO i

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

- **Clase**: la plantilla para crear instancias (objetos) de esa clase
- **Atributos**: datos internos de cada una de las instancias de esa clase

Terminología OO i

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

- **Clase**: la plantilla para crear instancias (objetos) de esa clase
- **Atributos**: datos internos de cada una de las instancias de esa clase
- **Constructor** (sin tipo)

Terminología OO i

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

- **Clase**: la plantilla para crear instancias (objetos) de esa clase
- **Atributos**: datos internos de cada una de las instancias de esa clase
- **Constructor** (sin tipo)
- **Parámetros formales**

Terminología OO i

```
class Cancion {  
    String titulo;  
    String interprete;  
    int duracion;  
    ...  
    Cancion (String titulo,  
             String interprete,  
             int duracion) {  
        this.titulo = titulo;  
        this.interprete = interprete;  
        this.duracion = duracion;  
    }  
    ...  
}
```

- **Clase**: la plantilla para crear instancias (objetos) de esa clase
- **Atributos**: datos internos de cada una de las instancias de esa clase
- **Constructor** (sin tipo)
- **Parámetros formales**
- **this**: variable con la referencia al objeto que se está creando

Terminología OO ii

```
class Cancion {  
    ...  
    String duracion() {  
        String minutos =  
            "" + duracion / 60;  
        String segundos =  
            "" + this.duracion % 60;  
        if (segundos.length() == 1)  
            segundos  
                = "0" + segundos;  
        }  
        return  
            minutos + ":" + segundos;  
    }  
}
```

- **Método (observador)**: al invocarlo devuelve algo (i.e. observa una propiedad del objeto sobre el que se invoca)

Terminología OO ii

```
class Cancion {  
    ...  
    String duracion() {  
        String minutos =  
            "" + duracion / 60;  
        String segundos =  
            "" + this.duracion % 60;  
        if (segundos.length() == 1)  
            segundos  
                = "0" + segundos;  
        }  
        return  
            minutos + ":" + segundos;  
    }  
}
```

- **Método (observador)**: al invocarlo devuelve algo (i.e. observa una propiedad del objeto sobre el que se invoca)
- **Variable local**: variables que sólo viven mientras se ejecuta el método

Terminología OO ii

```
class Cancion {  
    ...  
    String duracion() {  
        String minutos =  
            "" + duracion / 60;  
        String segundos =  
            "" + this.duracion % 60;  
        if (segundos.length() == 1)  
            segundos  
                = "0" + segundos;  
        }  
        return  
            minutos + ":" + segundos;  
    }  
}
```

Herranz

- **Método (observador)**: al invocarlo devuelve algo (i.e. observa una propiedad del objeto sobre el que se invoca)
- **Variable local**: variables que sólo viven mientras se ejecuta el método
- **Atributo**: si no está entre las variables locales, ni entre los parámetros formales, entonces debe ser un atributo (en este caso el uso de **this** es opcional)

Terminología OO iii

```
class Racional {  
    int n;  
    int d;  
    ...  
    void sum(Racional r) {  
        int den = this.d;  
        this.d = den * r.d;  
        this.n =  
            den * r.n  
            + r.d * this.n;  
    }  
}
```

- **Método (modificador)**: al invocarlo no devuelve nada pero modifica los atributos e invoca otros métodos con **this** u otros objetos)

Terminología OO iii

```
class Racional {  
    int n;  
    int d;  
    ...  
    void sum(Racional r) {  
        int den = this.d;  
        this.d = den * r.d;  
        this.n =  
            den * r.n  
            + r.d * this.n;  
    }  
}
```

- **Método (modificador)**: al invocarlo no devuelve nada pero modifica los atributos e invoca otros métodos con **this** u otros objetos)
- **Parámetros formales**: pueden ser de la misma clase!

Terminología OO iii

```
class Racional {  
    int n;  
    int d;  
    ...  
    void sum(Racional r) {  
        int den = this.d;  
        this.d = den * r.d;  
        this.n =  
            den * r.n  
            + r.d * this.n;  
    }  
}
```

- **Método (modificador)**: al invocarlo no devuelve nada pero modifica los atributos e invoca otros métodos con **this** u otros objetos)
- **Parámetros formales**: pueden ser de la misma clase!
- **Variables locale**: sólo viven mientras se ejecuta el método

Terminología OO iii

```
class Racional {  
    int n;  
    int d;  
    ...  
    void sum(Racional r) {  
        int den = this.d;  
        this.d = den * r.d;  
        this.n =  
            den * r.n  
            + r.d * this.n;  
    }  
}
```

- **Método (modificador)**: al invocarlo no devuelve nada pero modifica los atributos e invoca otros métodos con **this** u otros objetos)
- **Parámetros formales**: pueden ser de la misma clase!
- **Variables locale**: sólo viven mientras se ejecuta el método
- **Atributo**: se pueden modificar

Terminología OO iv

- Variable local

```
class Sesion03 {  
    public static void main(String...) {  
        String tiempo;  
        Cancion c;  
        c = new Cancion(  
            "Wish you were here",  
            "Pink Floyd",  
            354  
        );  
        tiempo = c.duracion();  
        ...  
    }  
}
```

Terminología OO iv

```
class Sesion03 {  
    public static void main(String...) {  
        String tiempo;  
        Cancion c;  
        c = new Cancion(  
            "Wish you were here",  
            "Pink Floyd",  
            354  
        );  
        tiempo = c.duracion();  
        ...  
    }  
}
```

- Variable local
- Invocación de constructor:
con **new**, crea una instancia de la clase y devuelve una referencia a la misma

Terminología OO iv

```
class Sesion03 {  
    public static void main(String...) {  
        String tiempo;  
        Cancion c;  
        c = new Cancion(  
            "Wish you were here",  
            "Pink Floyd",  
            354  
        );  
        tiempo = c.duracion();  
        ...  
    }  
}
```

- **Variable local**
- **Invocación de constructor:** con **new**, crea una instancia de la clase y devuelve una referencia a la misma
- **Parámetros:** son los datos que van a ser los parámetros formales titulo, interprete y duracion

Terminología OO iv

```
class Sesion03 {  
    public static void main(String...) {  
        String tiempo;  
        Cancion c;  
        c = new Cancion(  
            "Wish you were here",  
            "Pink Floyd",  
            354  
        );  
        tiempo = c.duracion();  
        ...  
    }  
}
```

- **Variable local**
- **Invocación de constructor:** con **new**, crea una instancia de la clase y devuelve una referencia a la misma
- **Parámetros:** son los datos que van a ser los parámetros formales titulo, interprete y duracion
- **Invocación de método:** se ejecuta el método duración sobre c (c será **this** mientras se ejecuta), devuelve lo que diga **return**

Terminología OO v

Objeto = Instancia

Puntero = Referencia

Método = Mensaje

Invocar = Ejecutar

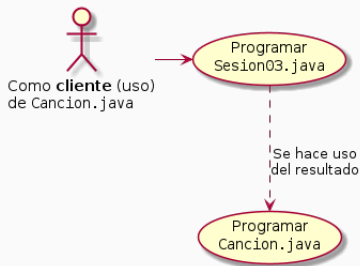
Enviar mensaje

Clase $\subseteq$ Tipo



Cuando usas código *de otro*

- No quieres saber *cómo* está hecho
- Sólo quieres saber *qué* hace
- No quieres *repetir* trabajo



Cuando *el otro* cambia el código

- *Estás usando* `Racional`, por ejemplo en `main`:

```
Racional r = new Racional(1,3);  
System.out.println(  
    "El valor decimal es " + r.n / r.d  
);
```
- El desarrollador de `Racional` renombra atributos `n` y `d` por `num` y `den`

Cuando *el otro* cambia el código

¿Qué va a pasar?

- *Estás usando* `Racional`, por ejemplo en `main`:

```
Racional r = new Racional(1,3);  
System.out.println(  
    "El valor decimal es " + r.n / r.d  
);
```

- *El desarrollador de* `Racional` *renombr*a atributos `n` y `d` por `num` y `den`

Cuando tú eres *el otro*

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {  
    double x;  
    double y;  
    ...  
    void rotar(double rad) { ... // ¡complejo! }  
}
```

- Y ahora sabes que el método *rotar* va a ser usado masivamente por código cliente

Cuando tú eres *el otro*

¿Qué vas a hacer?

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {  
    double x;  
    double y;  
    ...  
    void rotar(double rad) { ... // ¡complejo! }  
}
```

- Y ahora sabes que el **método rotar** va a ser **usado masivamente** por *código cliente*

Cuando tú eres *el otro*

¿Qué vas a hacer?

¿Qué va a pasar?

- Has desarrollado una clase para representar puntos en 2D

```
class Punto2D {  
    double x;  
    double y;  
    ...  
    void rotar(double rad) { ... // ¡complejo! }  
}
```

- Y ahora sabes que el **método rotar** va a ser **usado masivamente** por *código cliente*

¿Ideas?



¿Qué vas a hacer?

1. Utilizar **coordenadas polares** para representar los puntos

```
class Punto2D {  
    double r; // radio  
    double a; // ángulo
```

2. Implementar rotar con muy **poco esfuerzo**:

```
    void rotar(double rad) {  
        a = a + rad;  
    }  
}
```

¿Qué va a pasar?

- Cualquier otro método implementado en la clase
dejará de funcionar 👍

¿Qué va a pasar?

- Cualquier otro método implementado en la clase dejará de funcionar 👍
- Al modificar los atributos, cualquier código cliente de Punto2D dejará de compilar



¿Qué va a pasar?

- Cualquier otro método implementado en la clase dejará de funcionar 👍
- Al modificar los atributos, cualquier código cliente de Punto2D dejará de compilar



 ¡Vamos a sufrirlo!



```
class Punto2D {  
    double x;  
    double y;  
    void rotar(double rad) {  
        double x2, y2;  
        x2 = x * Math.cos(rad) - y * Math.sin(rad);  
        y2 = y * Math.cos(rad) - x * Math.sin(rad);  
        x = x2;  
        y = y2;  
    }  
}
```

Punto2D.java (usa la terminología)

clase, atributo, método (modificador), parámetro, variable

```
class Punto2D {  
    double x;  
    double y;  
    void rotar(double rad) {  
        double x2, y2;  
        x2 = x * Math.cos(rad) - y * Math.sin(rad);  
        y2 = y * Math.cos(rad) - x * Math.sin(rad);  
        x = x2;  
        y = y2;  
    }  
}
```



```
class Sesion05 {  
    public static void main(String[] args) {  
        Punto2D p = new Punto2D();  
        p.x = 0.0;  
        p.y = 1.0;  
        System.out.format("( %.2f, %.2f)\n", p.x, p.y);  
        p.rotar(Math.PI / 4.0);  
        System.out.format("( %.2f, %.2f)\n", p.x, p.y);  
    }  
}
```

Sesion05.java (usa la terminología)

variable, objeto (instancia), método (constructor), atributo, método (modificador), parámetro

```
class Sesion05 {  
    public static void main(String[] args) {  
        Punto2D p = new Punto2D();  
        p.x = 0.0;  
        p.y = 1.0;  
        System.out.format("( %.2f,  %.2f)\n", p.x, p.y);  
        p.rotar(Math.PI / 4.0);  
        System.out.format("( %.2f,  %.2f)\n", p.x, p.y);  
    }  
}
```


Punto2D en coordenadas polares

1. Reescribir Punto2D usando coordenadas polares
2. Compilar y ejecutar

private i

Para evitar que una modificación en la representación interna de una clase afecta a los *códigos cliente*, es muy importante ocultar dicha representación

private i

Para evitar que una modificación en la representación interna de una clase afecta a los *códigos cliente*, es muy importante ocultar dicha representación

```
class Punto2D {      public static void
    double x;         main(String[] args) {
    double y;         Punto2D p =
    ...               new Punto2D();
}                     p.x = 1.0;
                     p.y = 0.0;
                     }
```

private i

Para evitar que una modificación en la representación interna de una clase afecta a los *códigos cliente*, es muy importante ocultar dicha representación

```
class Punto2D {  
    private double x;  
    private double y;  
    ...  
}  
  
public static void  
    main(String[] args) {  
    Punto2D p =  
        new Punto2D();  
    p.x = 1.0; 🙅 No compila  
    p.y = 0.0; 🙅 No compila  
}
```

private ii

- Aparece **antes del tipo** en **atributos y métodos**
- Impide que el código cliente tenga acceso al atributo o al método
- Hay otros **modificadores**⁴ que iremos viendo:

private, public, protected, o **nada**

⁴Cuidado con la palabra “modificador”, aquí no se refiere a un *método modificador*, los modificadores (*modifiers*) son palabras clave de Java que cambian el significado de lo que *etiquetan*

Modificar Sesión05

- Crear dos instancias de Punto2D a y b :

$$a = (0, 0)$$

$$b = (1, 1)$$

- Imprimir los dos puntos tras cada cambio
- Modificar la ordenada (eje X) de a : ponerla a 1
- Rotar 45 el punto a
- Imprimir la distancia entre a y b

Buenas prácticas i

En lugar de exponer los atributos se define un **API**⁵ de acceso a los objetos

- `Punto2D()`: crea un punto $(0,0)$ en coordenadas cartesianas
- `Punto2D(x,y)`: crea un punto (x,y) en coordenadas cartesianas (abscisa y ordenada)
- `x()`: devuelve la abscisa
- `y()`: devuelve la ordenada

⁵*Application Public Interface*

Buenas prácticas ii

- `cambiarX(double x)`: modifica la abscisa
- `cambiarY(double y)`: modifica la ordenada
- `distancia(Punto2D b)`: devuelve un `double` que representa la distancia desde **this** a `b`
- `rotar(double a)`: rota el punto alrededor del $(0,0)$ la cantidad de `a` radianes

Buenas prácticas ii

- `cambiarX(double x)`: modifica la abscisa
- `cambiarY(double y)`: modifica la ordenada
- `distancia(Punto2D b)`: devuelve un `double` que representa la distancia desde **this** a `b`
- `rotar(double a)`: rota el punto alrededor del `(0,0)` la cantidad de `a` radianes

¡Usad primero un sistema de coordenadas y luego cambiadlo!