

Sesión 06: Modelización

Programación 2

9. Resolución de problemas en POO.

Ángel Herranz

2021-2022

Universidad Politécnica de Madrid

En capítulos anteriores

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

Encapsular

datos y comportamiento

- Ejercicio práctico: *Racionales*
- Terminología y **ocultación**



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”.



Ocultación

Para evitar que una modificación en la representación interna de una clase afecta a quien la usa, es muy importante ocultar dicha representación



Ocultación

Para evitar que una modificación en la representación interna de una clase afecta a quien la usa, es muy importante ocultar dicha representación

```
class Punto2D {  
    double x;  
    double y;  
    ...  
}  
  
public static void  
    main(String[] args) {  
    Punto2D p =  
        new Punto2D();  
    p.x = 1;  
    p.y = -1;  
}
```



Ocultación

Para evitar que una modificación en la representación interna de una clase afecta a quien la usa, 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; 👎 No compila  
    p.y = -1; 👎 No compila  
}
```



Buenas prácticas i

En lugar de exponer los atributos se define un **API**² 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

² *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

¡Y podemos cambiar la representación interna sin impactar el *código cliente*!

The Cincinnati Kid

³Basta con visitar el artículo sobre las jugadas del Póquer en Wikipedia

The Cincinnati Kid

☞ Modelizamos una parte del juego del Póquer³

³Basta con visitar el artículo sobre las jugadas del Póquer en Wikipedia

The Cincinnati Kid

- ☞ Modelizamos una parte del juego del Póquer³
 - Hacemos un programa que *reparta* dos manos aleatorias de Póquer,
 - una para *Player 1*,
 - otra para *Player 2*,
 - y diga quien es el ganador.

³Basta con visitar el artículo sobre las jugadas del Póquer en Wikipedia

 *“Es más difícil dar nombres que programar”*

¡Tormenta de nombres y luego descartamos!

 “*Es más difícil dar nombres que programar*”

¡Tormenta de nombres y luego descartamos!

*carta, palo, valor, naipe, baraja,
jugador, apostar, apuesta, foldearse,
holdearse, recuento, ganador, dinero,
monto, mazo, repartir, aleatorio, ...*

🗨 Naipes de la baraja francesa



- Cada naipе tiene un palo:    
- Y un *valor*: A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K
- Ignoremos el comodín

 Pequeño programa principal que usa los naipes:
simplemente crear y mostrar naipes

Modelización de *enumeraciones*

```
public enum Palo {  
    TREBOLES, DIAMANTES, CORAZONES, PICAS;  
}
```

```
public enum Valor {  
    AS, DOS, TRES, CUATRO, CINCO,  
    SEIS, SIETE, OCHO, NUEVE, DIEZ,  
    VALET, DAMA, REY;  
}
```

 Explorar el método `ordinal()` y la
función `values()`