

Programación Orientada a Objetos en Java

Algoritmos y Estructuras de Datos

Cualidades del software

Fundamentales:

- ▶ Correcto con respecto a una especificación.

Más o menos importantes, dependiendo del **contexto de uso**:

- ▶ Eficiente (tiempo, memoria, consumo de energía, ...).
- ▶ Reutilizable.
- ▶ Extensible / modificable.
- ▶ Usable.
- ▶ Legible.
- ▶ Predecible.
- ▶ ...

El **diseño** consiste en organizar el programa de tal manera que cumpla con las cualidades requeridas, en algún contexto de uso.

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- ▶ Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- ▶ Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- ▶ Describe el “qué” y no el “cómo”

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- ▶ Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- ▶ Describe el “qué” y no el “cómo”
- ▶ Son una forma de modularizar a nivel de los datos

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- ▶ Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- ▶ Describe el “qué” y no el “cómo”
- ▶ Son una forma de modularizar a nivel de los datos
- ▶ ¿Y para qué sirven?

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- ▶ Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- ▶ Describe el “qué” y no el “cómo”
- ▶ Son una forma de modularizar a nivel de los datos
- ▶ ¿Y para qué sirven?
- ▶ ¡Modelar la realidad!

Mmm, conocemos TADS..

Según la teórica...

- ▶ TAD quiere decir Tipo Abstracto de Datos
- ▶ ¿Qué es un Tipo Abstracto de Datos?
- ▶ Es un tipo de datos porque define un conjunto de valores y las operaciones que se pueden realizar sobre ellos
- ▶ Es abstracto ya que para utilizarlos, no se necesita conocer los detalles de la representación interna ni cómo están implementadas sus operaciones.
- ▶ Describe el “qué” y no el “cómo”
- ▶ Son una forma de modularizar a nivel de los datos
- ▶ ¿Y para qué sirven?
- ▶ ¡Modelar la realidad!
- ▶ ¿Qué TADs recuerdan de IP?

¿Y entonces?

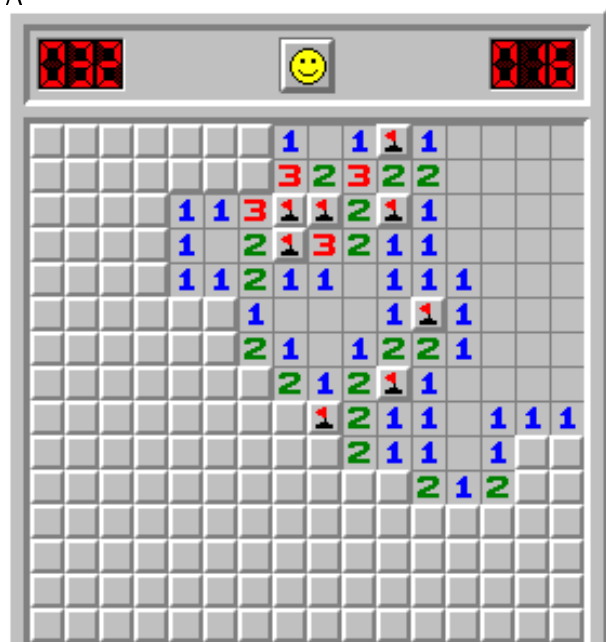
Quiero pasar de

Ejemplo: TAD Buscaminas

```
TAD Juego {  
    obs tablero: Tablero  
    obs jugadas: seq<Pos>  
  
    proc nuevoJuego(in t: Tablero): Juego  
    asegura res.tablero == t  
  
    proc jugar(inout j: Juego, in p: Pos)  
    requiere !j.perdio() && !j.gano()  
    requiere !(p in j.jugadas)  
    asegura p in j.jugadas  
  
    pred perdio(j: Juego)  
    ...  
  
    pred gano(j: Juego)  
    ...  
}
```

¿Y entonces?

A



Java....

Y vamos a utilizar a nuestro amigo **Java** al rescate



al menos eso es esta imagen para openai

¿Qué utilizaremos?

- ▶ Vamos a utilizar una pequeña parte de Programación Orientada a Objetos

¿Qué utilizaremos?

- ▶ Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- ▶ Utilizaremos clases de “Java”

¿Qué utilizaremos?

- ▶ Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- ▶ Utilizaremos clases de “Java”
- ▶ Un poco más profundo que lo que vienen utilizando

¿Qué utilizaremos?

- ▶ Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- ▶ Utilizaremos clases de “Java”
- ▶ Un poco más profundo que lo que vienen utilizando
- ▶ ⚠ Usamos una partecita y algunos conceptos....

¿Qué utilizaremos?

- ▶ Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- ▶ Utilizaremos clases de “Java”
- ▶ Un poco más profundo que lo que vienen utilizando
- ▶ ⚠ Usamos una partecita y algunos conceptos....
- ▶ Más en ingeniería de software

¿Qué utilizaremos?

- ▶ Vamos a utilizar una pequeña parte de Programación Orientada a Objetos
- ▶ Utilizaremos clases de “Java”
- ▶ Un poco más profundo que lo que vienen utilizando
- ▶ ⚠ Usamos una partecita y algunos conceptos....
- ▶ Más en ingeniería de software

Según wikipedia

La programación orientada a objetos (POO, en español); es un paradigma de programación que parte del concepto de objetos como base, los cuales contienen información en forma de campos (a veces también referidos como atributos o propiedades) y código en forma de métodos.

Los objetos son capaces de interactuar y modificar los valores contenidos en sus campos o atributos (estado) a través de sus métodos (comportamiento).

Algunas características clave de la programación orientada a objetos son herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Clases

Vamos a utilizar clases !

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*.
2. Vamos a declarar los *atributos privados* de la clase.
3. Vamos a implementar los métodos públicos.
4. En particular, el *constructor* de la clase (o los constructores).

Definiendo clases

Vamos a definir una clase en varias etapas:

1. Vamos a declarar los *métodos públicos*.
 2. Vamos a declarar los *atributos privados* de la clase.
 3. Vamos a implementar los métodos públicos.
 4. En particular, el *constructor* de la clase (o los constructores).
- Se puede programar con clases pero no modularmente.

Ejemplo

Modelar un contador de puntos del juego de truco para dos jugadores.

Necesitamos:

- ▶ Conocer el puntaje de ambos jugadores.
- ▶ Saber si un jugador está en las buenas.
- ▶ Poder sumar puntos a cada jugador.

Este comportamiento podemos expresarlo a través de una interfaz.


```
public class Truco {  
  
    public void sumarPunto(int jugador) {  
        /* TODO */  
    }  
  
    public int puntaje(int jugador) {  
        /* TODO */  
    }  
  
    public boolean enLasBuenas(int jugador) {  
        /* TODO */  
    }  
}
```

Si tuviéramos esta clase en Java, ¿sabríamos cómo usarla?

- ▶ ¿Cuántos puntos suma `sumarPunto`?
- ▶ ¿Cuántos puntos tiene cada jugador al principio?
- ▶ ¿Cuál es la información que nos da `estaEnLasBuenas`?

```

TAD Truco {
  obs tantos(jugador: nat): nat

  proc nuevaPartida(): Truco
  asegura res.tantos(0) == 0  && res.tantos(1) == 0

  proc sumarPunto(inout t: Truco, in j: nat)
  requiere 0 <= j < 2 && t.tantos(j) < 30
  asegura t.tantos(j) = old(t).tantos(j) + 1
  asegura t.tantos((1+j)%2) = old(t).tantos((1+j)%2)

  proc puntaje(in t: Truco, in jugador: nat) : nat
  requiere 0 <= jugador < 2
  asegura res = t.tantos(jugador) % 15

  proc estaEnLasBuenas(in t: Truco, in jugador: nat) : bool
  requiere 0 <= jugador < 2
  asegura res = true <=> t.tantos(jugador) >= 15
}

```

Clases e instancias

```
Truco t1 = new Truco();
```

```
t1.sumarPunto(1);
```

```
t1.sumarPunto(1);
```

```
t1.sumarPunto(1);
```

```
t1.sumarPunto(2);
```

```
t1.sumarPunto(2);
```

```
t1.puntaje(1); // 3
```

```
t1.puntaje(2); // 2
```

```
Truco t2 = new Truco();
```

```
t2.sumarPunto(2);
```

```
t2.sumarPunto(2);
```

```
t2.sumarPunto(1);
```

```
t2.sumarPunto(2);
```

```
t2.sumarPunto(2);
```

```
t2.puntaje(1); // 1
```

```
t2.puntaje(2); // 4
```

Clases e instancias

```
Truco t1 = new Truco();
```

```
t1.sumarPunto(1);
```

```
t1.sumarPunto(1);
```

```
t1.sumarPunto(1);
```

```
t1.sumarPunto(2);
```

```
t1.sumarPunto(2);
```

```
t1.puntaje(1); // 3
```

```
t1.puntaje(2); // 2
```

`class Truco` es la clase.

t1 y t2 son instancias de la clase.

```
Truco t2 = new Truco();
```

```
t2.sumarPunto(2);
```

```
t2.sumarPunto(2);
```

```
t2.sumarPunto(1);
```

```
t2.sumarPunto(2);
```

```
t2.sumarPunto(2);
```

```
t2.puntaje(1); // 1
```

```
t2.puntaje(2); // 4
```

Para que el comportamiento de la clase pueda llevarse a cabo, hay que implementarla.

La **implementación** está dada por:

- ▶ La **representación** interna: un conjunto de variables que determinan el estado interno de la instancia.
- ▶ Un conjunto de **algoritmos** que implementan cada una de las operaciones de la interfaz, consultando y modificando las variables de la representación interna.

Declaración de atributos privados

```
class Truco {  
    /* ... */  
    private int _puntaje0;  
    private int _puntaje1;  
    private boolean _buenas0;  
    private boolean _buenas1;  
};
```

Estados internos

```
Truco t1 = new Truco();  
t1.sumarPunto(1);  
t1.sumarPunto(1);  
t1.sumarPunto(1);  
t1.sumarPunto(2);  
t1.sumarPunto(2);
```

```
// Estado interno t1  
t1._puntaje0 == 3;  
t1._puntaje1 == 2;  
t1._buenas0  == false;  
t1._buenas1  == false;
```

```
Truco t2 = new Truco();  
t2.sumarPunto(2);  
t2.sumarPunto(2);  
t2.sumarPunto(1);  
t2.sumarPunto(2);  
t2.sumarPunto(2);
```

```
// Estado interno t2  
t2._puntaje0 == 1;  
t2._puntaje1 == 4;  
t2._buenas0  == false;  
t2._buenas1  == false;
```


Comportamiento

¿Cómo definimos comportamiento para las instancias?

Comportamiento

¿Cómo definimos comportamiento para las instancias?

A través de **métodos**:

```
public void sumarPunto(int jugador) {  
    if (jugador == 1) {  
        _puntaje1++;  
        if (_puntaje1 == 16) {  
            _puntaje1 = 0;  
            _buenas1 = true;  
        }  
    } else {  
        _puntaje2++;  
        if (_puntaje2 == 16) {  
            _puntaje2 = 0;  
            _buenas2 = true;  
        }  
    }  
}
```

Ejemplo

```
int main() {  
    Truco t1 = new Truco();  
    Truco t2 = new Truco();  
                                     // <---  
    t1.sumarPunto(1);  
    t1.sumarPunto(1);  
    t2.sumarPunto(2);  
}
```

Contexto

t1._puntaje1	0
t1._puntaje2	0
t1._buenas1	false
t1._buenas2	false
t2._puntaje1	0
t2._puntaje2	0
t2._buenas1	false
t2._buenas2	false

Ejemplo

```
int main() {  
    Truco t1 = new Truco();  
    Truco t2 = new Truco();  
    t1.sumarPunto(1);  
                                     // <---  
    t1.sumarPunto(1);  
    t2.sumarPunto(2);  
}
```

Contexto

t1._puntaje1	1
t1._puntaje2	0
t1._buenas1	false
t1._buenas2	false
t2._puntaje1	0
t2._puntaje2	0
t2._buenas1	false
t2._buenas2	false

Ejemplo

```
int main() {  
    Truco t1 = new Truco();  
    Truco t2 = new Truco();  
    t1.sumar_punto(1);  
    t1.sumar_punto(1);  
                                     // <---  
    t2.sumar_punto(2);  
}
```

Contexto

t1._puntaje1	2
t1._puntaje2	0
t1._buenas1	false
t1._buenas2	false
t2._puntaje1	0
t2._puntaje2	0
t2._buenas1	false
t2._buenas2	false

Ejemplo

```
int main() {  
    Truco t1 = new Truco();  
    Truco t2 = new Truco();  
    t1.sumar_punto(1);  
    t1.sumar_punto(1);  
    t2.sumar_punto(2);  
                                     // <---  
}
```

Contexto

t1._puntaje1	2
t1._puntaje2	0
t1._buenas1	false
t1._buenas2	false
t2._puntaje1	0
t2._puntaje2	1
t2._buenas1	false
t2._buenas2	false

El resto de los ingredientes

La interfaz de Truco tiene métodos para ver el puntaje de los jugadores:

```
public class Truco {
    public Truco() { /* ... */}
    public void sumarPunto(int jugador) { /* ... */}
    public int puntaje(int jugador) { /* ... */}
    public boolean enLasBuenas(int jugador) { /* ... */}

    private int _puntaje0;
    private int _puntaje1;
    private boolean _buenas0;
    private boolean _buenas1;
}
```

Pero los miembros privados de una clase no son accesibles desde afuera:

```
void ejemplo() {
    Truco t = new Truco();
    System.out.println(t._puntaje1);
    // error: The field t._puntaje1 is not visible.
}
```

```
public int puntaje(int jugador) {  
    if (jugador == 0) {  
        return _puntaje0;  
    } else {  
        return _puntaje1;  
    }  
}  
  
void ejemplo() {  
    Truco t = new Truco() ;  
    System.out.println(t.puntaje(1));  
}
```



```
public boolean buenas(int jugador) {  
    if (jugador == 0) {  
        return _buenas0;  
    } else {  
        return _buenas1;  
    }  
}
```

```
void ejemplo() {  
    Truco t = new Truco();  
    for (uint i = 0; i < 15; i++) {  
        t.sumarPunto(1);  
        t.sumarPunto(2);  
    }  
    t.sumarPunto(1);  
    cout << t.buenas(1) << endl; // true  
    cout << t.buenas(2) << endl; // false  
}
```

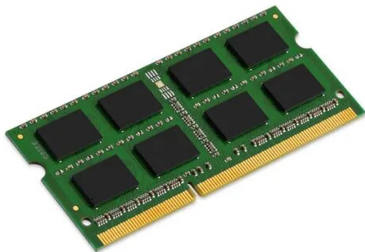
Constructor

- ▶ Los constructores son funciones especiales para inicializar una nueva instancia de un tipo.
- ▶ Se escriben con el nombre del tipo.
- ▶ No tienen tipo de retorno (está implícito ya que es el tipo de lo que estamos contruyendo).

```
public Truco() {  
    _puntaje0 = 0;  
    _puntaje1 = 0;  
    _buenas0 = false;  
    _buenas1 = false;  
}  
  
void ejemplo() {  
    Truco t = new Truco();  
}
```

Memoria

Esto es una memoria:



Y esto para qué me sirve ?

Memoria

Todo el contenido de las variables ocupa “espacio” en la “memoria”. Hay tres regiones principales (a nivel sistema operativo):

Global (estática) \Rightarrow en el ejecutable

La memoria estática se encuentra incrustada en el ejecutable. Allí se guardan constantes.

Contexto local \Rightarrow en la pila (*stack*)

Las variables locales viven únicamente dentro del scope local. Aquí se guardan tipos primitivos y referencias a arreglos y objetos.

Dinámica (manual) \Rightarrow en el *heap*

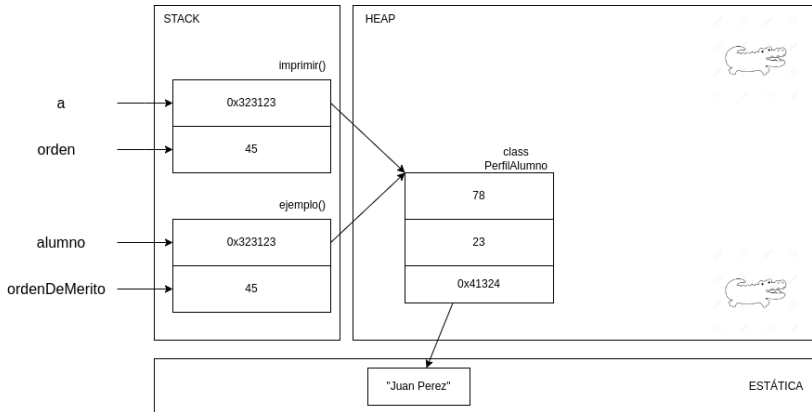
En el heap se almacenan los objetos. La memoria de un objeto se libera cuando se vuelve inalcanzable.

Ejemplo de código

```
public class PerfilAlumno {  
    public PerfilAlumno(int nroLibreta,  
                        int anioLibreta,  
                        String nombre)  
  
    public int numeroLibreta;  
    public int anioLibreta;  
    public String nombre;  
}
```

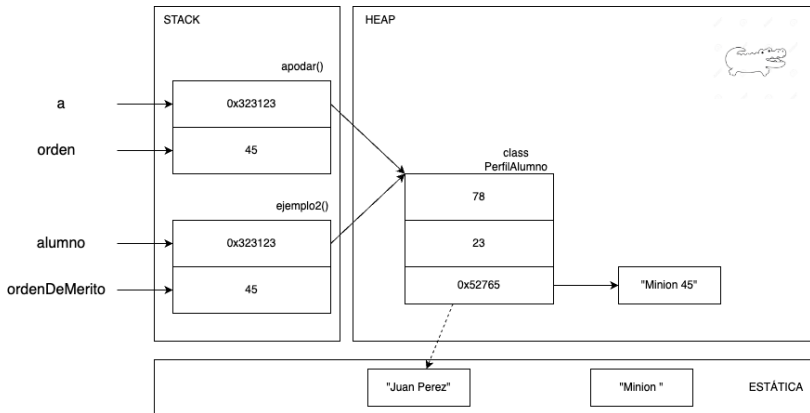
Ejemplo de código

```
void ejemplo() {  
    int ordenDeMerito = 45;  
    PerfilAlumno alumno = new PerfilAlumno(78, 23, "Juan Perez");  
  
    imprimir(ordenDeMerito, alumno);  
}  
  
void imprimir(int orden, PerfilAlumno a) {  
    System.out.println(orden);  
    System.out.println(a.nombre);  
}
```



Ejemplo de código

```
void ejemplo2() {  
    int ordenDeMerito = 45;  
    PerfilAlumno alumno = new PerfilAlumno2(78, 23, "Juan Perez");  
    apodar(ordenDeMerito, alumno);  
    System.out.println(alumno.nombre)  
}  
  
void apodar(int orden, PerfilAlumno a) {  
    a.nombre = "Minion " + orden.toString();  
}
```

Aliasing

¿Qué pasa con el siguiente código ?

```
void ejemplo3() {  
    int a = 2;  
    int b = 2;  
}
```

¿Qué hay en la memoria ?

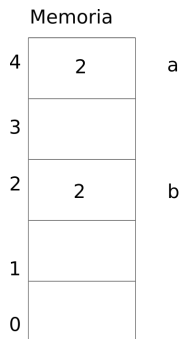
Aliasing

¿Qué pasa con el siguiente código ?

```
void ejemplo3() {  
    int a = 2;  
    int b = 2;  
}
```

¿Qué hay en la memoria ? ¿Es $a = b$?

Memoria



¿Y si cambio alguno, qué pasa con el otro ?
¿Y si ocupa mucho lugar? ¿copio todo ?

Aliasing

- ▶ Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.

Aliasing

- ▶ Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.
- ▶ Lo podemos verificar con `==`.

Aliasing

- ▶ Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.
- ▶ Lo podemos verificar con `==`.
- ▶ El aliasing es un problema cuando se trata de objetos mutables (modificables).

Aliasing

- ▶ Cuando dos variables referencian al mismo valor, decimos que hay *aliasing* entre ellas.
- ▶ Lo podemos verificar con `==`.
- ▶ El aliasing es un problema cuando se trata de objetos mutables (modificables).
- ▶ Si exponemos referencias a los atributos de nuestra clase, nos exponemos a que el usuario de la misma nos modifique sin que nos demos cuenta.

Memoria

4	2	a
3		
2	4	b
1		
0		

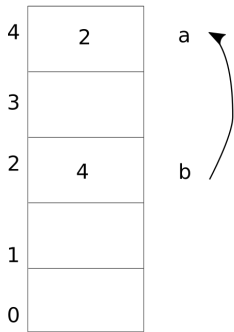
Memoria

4	2	a
3		
2	4	b
1		
0		

¿Cómo lo interpreto?

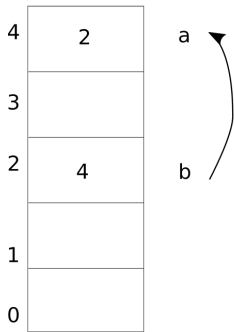
Podría interpretarlo así:

Podría interpretarlo así:
Memoria



Podría interpretarlo así:

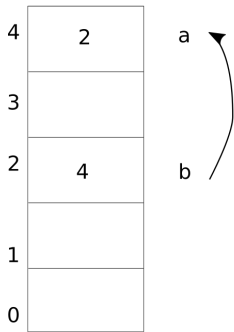
Memoria



¿Quién lo interpreta?

Podría interpretarlo así:

Memoria



¿Quién lo interpreta?

Más en la clase siguiente...

interface

interface

¿Alguien jugó a la escoba del quince?

interface

¿Alguien jugó a la escoba del quince?

```
class EscobaDelQuince {  
    public void sumarPunto(int jugador) {  
        /* TODO */  
    }  
  
    public int puntaje(int jugador) {  
        /* TODO */  
    }  
}
```

Cómo haríamos si queremos sumar muchos puntos para truco ?

Cómo haríamos si queremos sumar muchos puntos para truco ?

```
class SumadorMuchosPuntosDeTruco {  
    public void sumarMuchosPuntos(Truco truco,  
                                   int jugador,  
                                   int totalDePuntos) {  
  
        for (int i = 0; i < totalDePuntos; i++ ) {  
            truco.sumarPunto(jugador);  
        }  
    }  
}
```

Cómo haríamos si queremos sumar muchos puntos para truco ?

```
class SumadorMuchosPuntosDeTruco {  
    public void sumarMuchosPuntos(Truco truco,  
                                   int jugador,  
                                   int totalDePuntos) {  
  
        for (int i = 0; i < totalDePuntos; i++ ) {  
            truco.sumarPunto(jugador);  
        }  
    }  
}
```

Y si lo quiero hacer para la Escoba ?

Cómo haríamos si queremos sumar muchos puntos para truco ?

```
class SumadorMuchosPuntosDeTruco {  
    public void sumarMuchosPuntos(Truco truco,  
                                   int jugador,  
                                   int totalDePuntos) {  
  
        for (int i = 0; i < totalDePuntos; i++ ) {  
            truco.sumarPunto(jugador);  
        }  
    }  
}
```

Y si lo quiero hacer para la Escoba ?

```
class SumadorMuchosPuntosEscoba {  
    public void sumarMuchosPuntos(EscobaDelQuince escoba,  
                                   int jugador,  
                                   int totalDePuntos) {  
  
        for (int i = 0; i < totalDePuntos; i++ ) {  
            escoba.sumarPunto(jugador);  
        }  
    }  
}
```

Cómo haríamos si queremos sumar muchos puntos para truco ?

```
class SumadorMuchosPuntosDeTruco {  
    public void sumarMuchosPuntos(Truco truco,  
                                   int jugador,  
                                   int totalDePuntos) {  
  
        for (int i = 0; i < totalDePuntos; i++ ) {  
            truco.sumarPunto(jugador);  
        }  
    }  
}
```

Y si lo quiero hacer para la Escoba ?

```
class SumadorMuchosPuntosEscoba {  
    public void sumarMuchosPuntos(EscobaDelQuince escoba,  
                                   int jugador,  
                                   int totalDePuntos) {  
  
        for (int i = 0; i < totalDePuntos; i++ ) {  
            escoba.sumarPunto(jugador);  
        }  
    }  
}
```

Entonces ?

Podría generalizarlo

```
class SumadorMuchosPuntos {
    public void sumarMuchosPuntos(Juego juego,
                                   int jugador,
                                   int totalDePuntos) {
        for (int i = 0; i < totalDePuntos; i++ ) {
            juego.sumarPunto(jugador);
        }
    }
    // ...
    void ejemplo() {
        EscobaDelQuince e = new EscobaDelQuince();
        Truco t = new Truco();
        SumadorMuchosPuntos sumador = new SumadorMuchosPuntos();

        sumador.sumarMuchosPuntos(e, 0, 2);
        sumador.sumarMuchosPuntos(t, 0, 3);
    }
}
```

Podría generalizarlo

```
class SumadorMuchosPuntos {
    public void sumarMuchosPuntos(Juego juego,
                                   int jugador,
                                   int totalDePuntos) {
        for (int i = 0; i < totalDePuntos; i++ ) {
            juego.sumarPunto(jugador);
        }
    }
    // ...
    void ejemplo() {
        EscobaDelQuince e = new EscobaDelQuince();
        Truco t = new Truco();
        SumadorMuchosPuntos sumador = new SumadorMuchosPuntos();

        sumador.sumarMuchosPuntos(e, 0, 2);
        sumador.sumarMuchosPuntos(t, 0, 3);
    }
}
```

¿Qué opinan?

Podría generalizarlo

```
class SumadorMuchosPuntos {
    public void sumarMuchosPuntos(Juego juego,
                                   int jugador,
                                   int totalDePuntos) {
        for (int i = 0; i < totalDePuntos; i++ ) {
            juego.sumarPunto(jugador);
        }
    }
    // ...
    void ejemplo() {
        EscobaDelQuince e = new EscobaDelQuince();
        Truco t = new Truco();
        SumadorMuchosPuntos sumador = new SumadorMuchosPuntos();

        sumador.sumarMuchosPuntos(e, 0, 2);
        sumador.sumarMuchosPuntos(t, 0, 3);
    }
}
```

¿Qué opinan? Cada clase que la usa debería ser específica para mi truco. Me faltaría una clase general...

```
class Juego {
    private EscobaDelQuince e;
    private Truco t;

    public Juego(EscobaDelQuince e) {
        this.e = e;
        this.t = null; // esta linea no hace falta
    }

    public Juego(Truco t) {
        this.e = null; // esta linea no hace falta
        this.t = t;
    }

    public void sumarPunto(int jugador) {
        if (e == null) {
            e.sumarPunto(jugador);
        } else {
            t.sumarPunto(jugador);
        }
    }
}
```

```
void ejemplo() {  
    Juego je = new Juego(new EscobaDelQuince());  
    Juego jt = new Juego(new Truco());  
    SumadorMuchosPuntos sumador = new SumadorMuchosPuntos();  
  
    sumador.sumarMuchosPuntos(je, 0, 2);  
    sumador.sumarMuchosPuntos(jt, 0, 3);  
}
```

```
void ejemplo() {  
    Juego je = new Juego(new EscobaDelQuince());  
    Juego jt = new Juego(new Truco());  
    SumadorMuchosPuntos sumador = new SumadorMuchosPuntos();  
  
    sumador.sumarMuchosPuntos(je, 0, 2);  
    sumador.sumarMuchosPuntos(jt, 0, 3);  
}
```

>Qué problema le ven?

```
interface Juego {
    public void sumarPunto(int jugador);
    public int puntaje(int jugador);
}

void ejemplo() {
    Juego je = new EscobaDelQuince();
    Juego jt = new Truco();
    SumadorMuchosPuntos sumador = new SumadorMuchosPuntos();

    sumador.sumarMuchosPuntos(je, 0, 2);
    sumador.sumarMuchosPuntos(jt, 0, 3);
}
```

```
class Truco implements Juego {
    public void sumarPunto(int jugador) {
        // Implementación
    };
    public int puntaje(int jugador) {
        // Implementación
    };
}

class EscobaDelQuince implements Juego {
    public void sumarPunto(int jugador) {
        // Implementación (posiblemente distinta).
    };
    public int puntaje(int jugador) {
        // Implementación (posiblemente distinta).
    };
}
```

Interfaces comunes

Hay métodos que tienen una implementación por defecto en todos los objetos de Java, pero se puede sobre-escribir.

Método toString

Método toString

```
class Truco {  
    @Override // ¿qué hace esto?  
    public String toString() {  
        StringBuffer sbuffer = new StringBuffer();  
  
        sbuffer.append("Jugador 0: ");  
        sbuffer.append(puntaje(0).toString());  
        sbuffer.append(puntaje(0) + " ");  
  
        // Mismo para jugador 1  
        // ...  
  
        return sbuffer.toString();  
    }  
}
```

Método equals

Método equals

```
@Override
public boolean equals(Object otro) {

    // otro es null
    boolean oen = (otro == null);

    // clase es distinta
    boolean cd = otro.getClass() != this.getClass();

    if (oen || cd) {
        return false;
    }

    Truco otroTruco = (Truco) otro; // casting.

    return _puntaje0 == otroTruco._puntaje0
        && _puntaje1 == otroTruco._puntaje1
        && _buenas0 == otroTruco._buenas0
        && _buenas1 == otroTruco._buenas1;
}
```

Recuerden:

- ▶ Si el tipo es primitivo, `a == b` dice si “a” y “b” son *iguales*.
- ▶ Si el tipo es un objeto, `a == b` dice si “a” y “b” son alias de un mismo objeto.
- ▶ Si el tipo es un objeto, `a.equals(b)` dice si “a” y “b” son *iguales*.

Fin.