

DISEÑANDO MI SOLUCIÓN EN POO

.UBAfiuba 
FACULTAD DE INGENIERÍA

75.07 Algoritmos y programación III

Idea original de **Eugenio Yolis**
rediseño por **Pablo Rodríguez Massuh**

v1.1 | 1c2022

**PRESENTACIÓN DEL
PROBLEMA**

01

SOLUCIÓN “RÁPIDA”

02

LIMITACIONES

03

AGENDA

04

BUSCANDO OBJETOS

05

**MODIFICACIONES A
LA SOLUCIÓN**

06

CONCLUSIONES



01

EL PROBLEMA

API de funciones

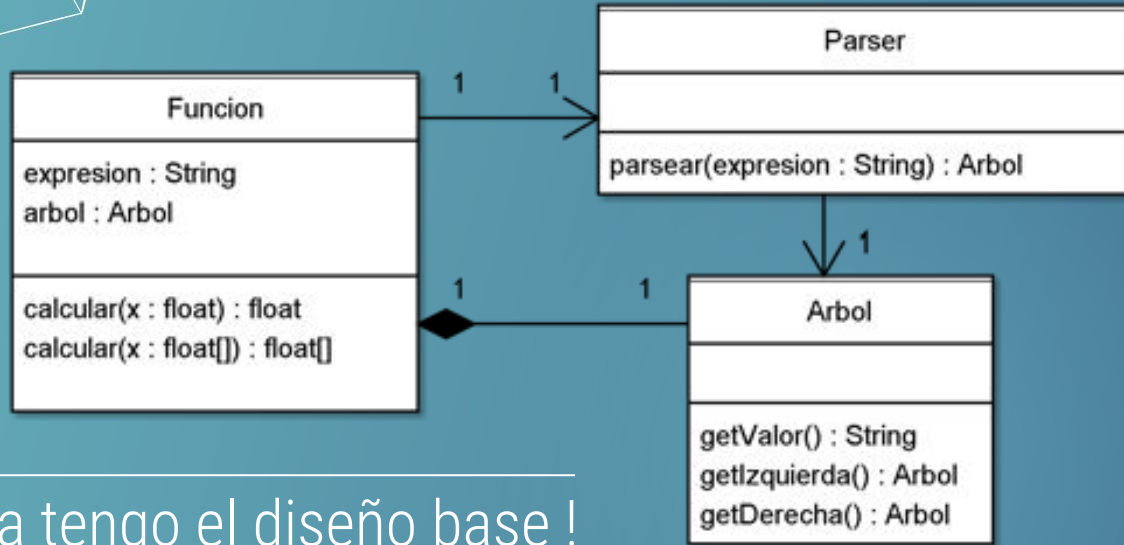
Requerimientos de una API de funciones para un graficador

- » Evaluación de funciones matemáticas de una variable [$f(x)$]
- » Soporte de funciones básicas: suma, resta, división, producto, potencia y logaritmo
- » Soporte de funciones compuestas [$f(g(x))$]
- » Evaluación (*por aproximación*) de la función derivada e integral
- » Indicar los puntos de intersección entre funciones en un determinado intervalo
- » Indicar los máximos y mínimo de una función en un determinado intervalo



02 SOLUCIÓN RÁPIDA

DIAGRAMA DE CLASES



Creo que ya tengo el diseño base !
Empiezo a programar y después le voy
agregando la funcionalidad pedida

Comenzamos a tirar código

```
public class Funcion {  
  
    private String expression = "";  
    private Arbol arbol = "";  
  
    public Funcion(String expression) {  
        this.expression = expression;  
        this.arbol = Parser.parsear(expression);  
    }  
  
    public float calcular(float x) {  
        return calcular(arbol, x);  
    }  
}
```

Seguimos tirando código ...

```
private float calcular(Arbol a, float x) {
    resultado = 0;
    String s = a.getValor();
    if (s == "+") {
        resultado = calcular(a.getIzquierda())
            + calcular(a.getDerecha());
    } else if (s == "*") {
        resultado = calcular(a.getIzquierda())
            * calcular(a.getDerecha());

    // ... Idem para el resto de las operaciones...

    } else if (s == "x") {
        resultado = x;
    } else {
        // constante
        resultado = Float.parseFloat(s);
    }
    return resultado;
}
```




03

LIMITACIONES

Del modelo presentado

LIMITACIONES

Solo se crean funciones por medio de un string. Sería bueno crearlas en forma *programática*

Hay alto acoplamiento entre el Parser y la clase Funcion

La clase Funcion tiene muchas responsabilidades

MÁS LIMITACIONES...

**El método “calcular”
va a crecer en
complejidad a medida
que agreguemos
funcionalidad**

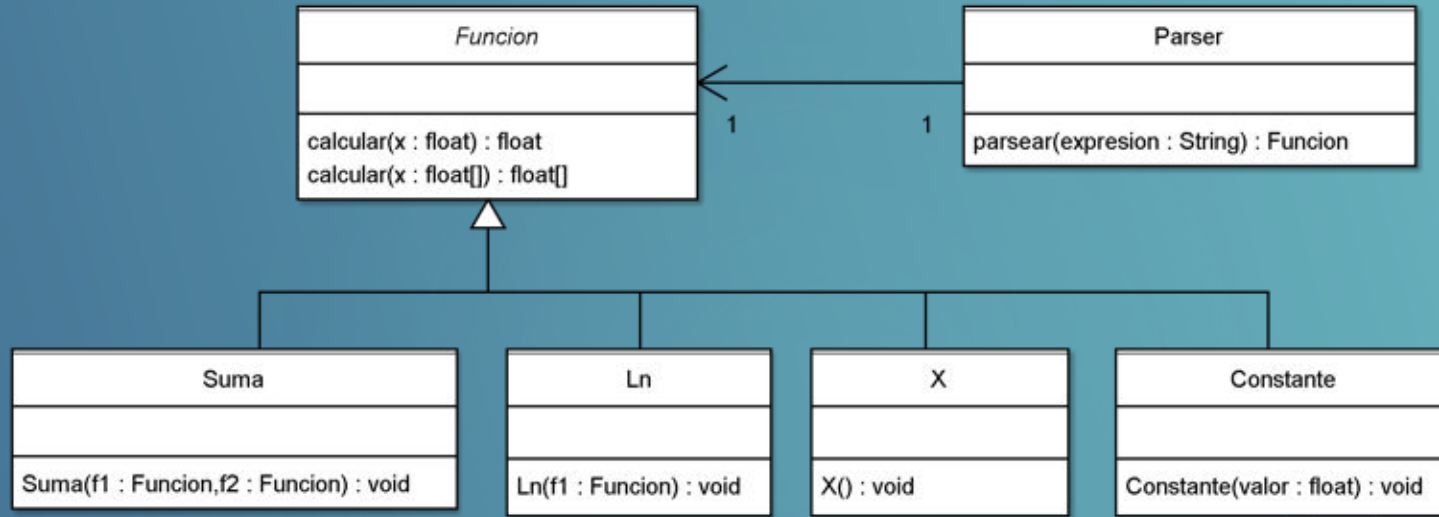
**Es poco extensible,
agregar un nuevo
operador es *complicado***

04



BUSCANDO OBJETOS

Diagrama de clases



- » Cada función encapsula la forma de calcular y la cantidad de operandos
- » El “árbol” de evaluación se arma solo
- » Permite crear funciones en forma programática

Código 1/2

```
void test() {  
    // " 3x + 5 "  
    Funcion f1 = new Multiplicacion(new Constante(3), new X());  
    Funcion f = new Suma(f1, new Constante(5));  
}  
  
public class Suma extends Funcion {  
    private Funcion f1, f2;  
    public Suma(Funcion f1, Funcion f2) {  
        this.f1 = f1;  
        this.f2 = f2;  
    }  
    public float calcular(float x) {  
        return f1.calcular(x) + f2.calcular(x);  
    }  
}
```

Código 2/2

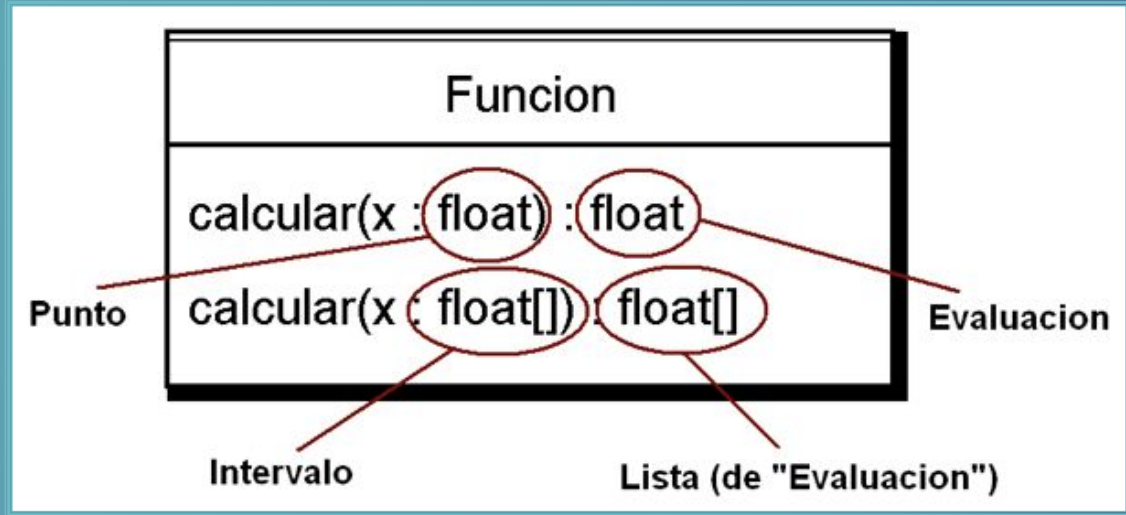
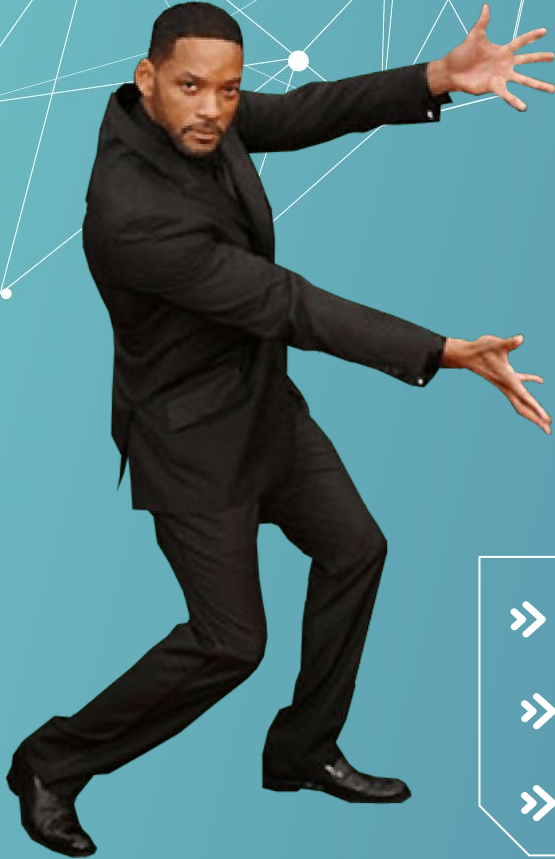
```
public class Constante extends Funcion {  
    private float valor;  
    public Constante(float valor) {  
        this.valor = valor;  
    }  
    public float calcular(float x) {  
        return valor;  
    }  
}  
  
public class X extends Funcion {  
    public X() {  
    }  
    public float calcular(float x) {  
        return x;  
    }  
}
```

05

BUSCANDO MAS OBJETOS

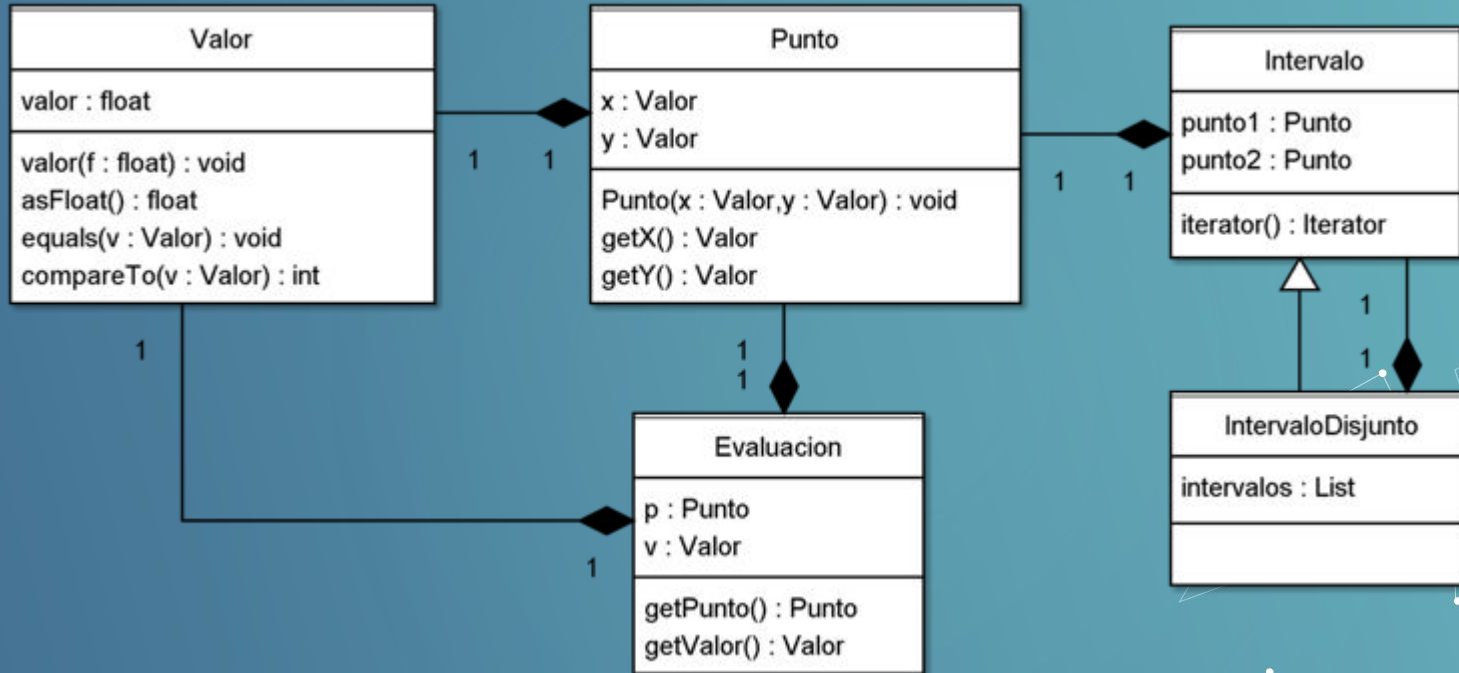


Aún más objetos



- » Facilita el cambio a funciones de 2 variables
- » Permite implementar intervalos disjuntos: [0..1] U [5..10]
- » Simplifica métodos de búsqueda de mínimos y máximos

Diagrama de clases



Código 1/2

```
public List calcular(Intervalo i) {
    List resultado = new ArrayList();
    Iterator it = i.iterator();
    while (it.hasNext()) {
        Punto p = it.next();
        resultado.add(new Evaluacion(p, evaluar(p)));
    }
    return resultado;
}

public Evaluacion getMaximo(Intervalo i) {
    Evaluacion max = null;
    List evals = calcular(i);
    Iterator it = evals.iterator();
    if (it.hasNext()) {
        max = new Evaluacion(it.next());
    }
    while (it.hasNext()) {
        Evaluacion ev = it.next();
        if (max.getValor().compareTo(ev.getValor()) < 0) {
            max = ev;
        }
    }
    return max;
}
```

Código 2/2

```
public class Constante extends Funcion {
    private Valor valor;
    public Constante(Valor valor) {
        this.valor = valor;
    }
    public Evaluacion calcular(Punto p) {
        return new Evaluacion(p, valor);
    }
}

public class X extends Funcion {
    public Evaluacion calcular(Punto p) {
        return new Evaluacion(p, p.getX());
    }
}

public class Y extends Funcion {
    public Evaluacion calcular(Punto p) {
        return new Evaluacion(p, p.getY());
    }
}
```

PENDIENTES

- » Cómo quedarían las implementaciones de las funciones que realizan los cálculos (Suma, División, Potencia, etc..) ?
- » Cómo se implementarían las funciones compuestas:
 $f(g(x))$?
- » Cómo se implementarían las derivadas e integrales ?





01'

OTRO PROBLEMA

Juego Galaga

1UP

60



Implementar el modelo del “Galaga”



Las naves se mueven por el espacio.



El jugador maneja su nave.



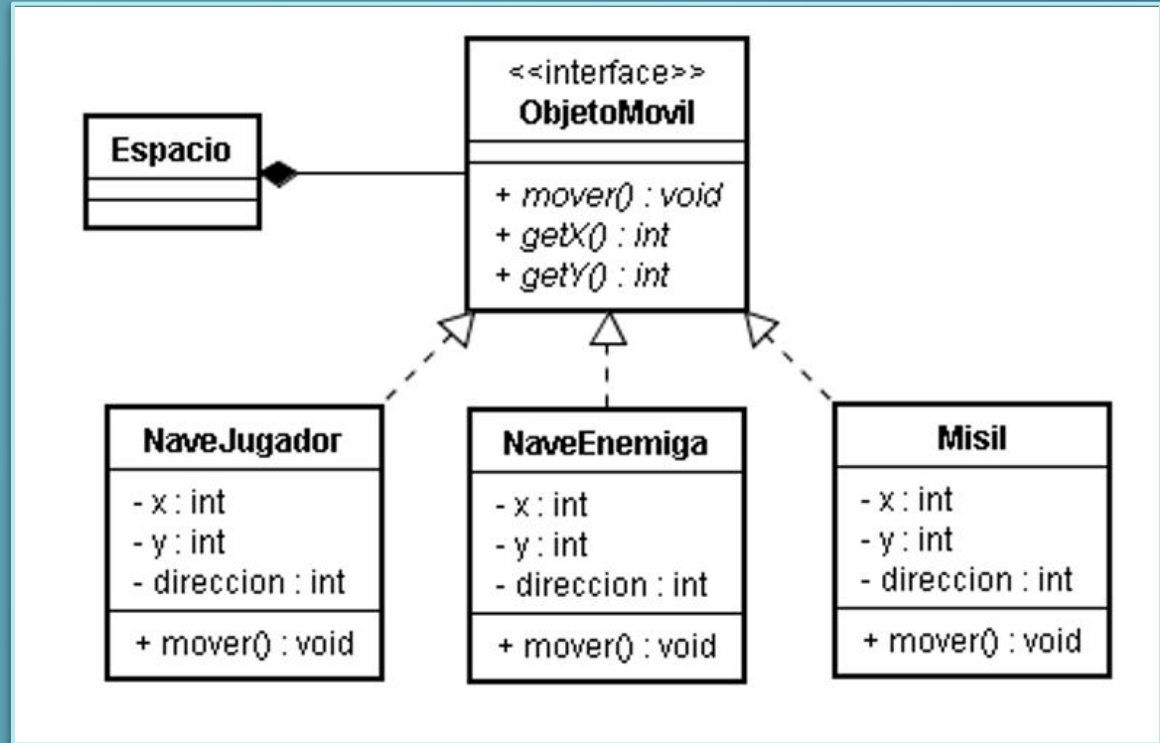
La nave del usuario dispara a las otras y las otras le disparan a él.



02' SOLUCIÓN RÁPIDA

DIAGRAMA DE CLASES

Creo que ya tengo el
diseño base.
Empiezo a programar y
después le voy
agregando la
funcionalidad pedida.



Ejemplos de código

```
int incX, incY;

if (this.direccion == 1) {

    incX = 0; incY = 1;    // arriba

} else if ... {    .... }

this.x = this.x + incX;


this.y = this.y + incY;
```

Ejemplos de código II

```
if (this.y > getEspacio().getY()) {  
    // choque contra el borde superior  
    this.y = this.y - 1;  
}
```

Ejemplos de código III

```
// voy a "chocar" al jugador  
  
int posJugY = espacio.getNaveJug().getY();  
  
int incY = posJugY / Math.abs(posJugY);  
  
this.y = this.y + incY;
```

03'

LIMITACIONES

Del modelo presentado

LIMITACIONES

Repetición de código:

Una mejora es hacer un “ObjetoMovil” abstracto, pero igual es un problema.

Constantes magicas:

```
if (this.direccion == 1)
```

LIMITACIONES II

Cálculos y condiciones difíciles de entender:

Esto va empeorando cada vez que se agrega un nuevo caso a contemplar.

Poco encapsulamiento:

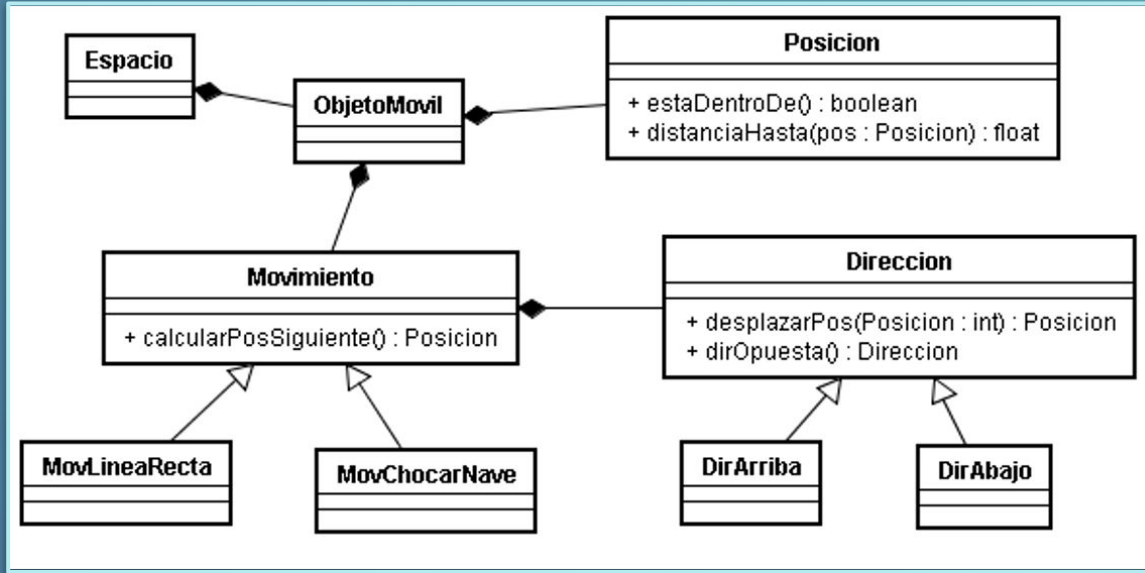
Diferentes métodos de la clase van a manipular el "X" y el "Y" directamente.

04'



BUSCANDO OBJETOS

Diagrama de clases



- » Las naves delegan en movimiento la lógica del "mover".
- » Dirección y Posición encapsulan cálculos.
- » Diferentes naves pueden usar las mismas estrategias de movimiento.

Ejemplos de código

```
// en Nave.mover
```

```
this.posicion = this.movimiento.calcularPosSiguiente(this.posicion)
```

```
// en Movimiento.calcularPosSiguiente
```

```
posicion = this.direccion.desplazarPos(posicion)
```

```
// en Movimiento.calcularPosSiguiente
```

```
if (!posicion.dentroDe(espacio.getLimites())) {
```

```
    // choque contra algun borde del espacio
```

```
    this.direccion = direccion.dirOpuesta();
```

```
}
```


06

CONCLUSIONES

RESUMEN

Encapsular todo lo posible

1

Pensar de quién es la responsabilidad

2

Revisar métodos **largos**

3

Revisar clases con **getters** y **setters** para todos sus atributos

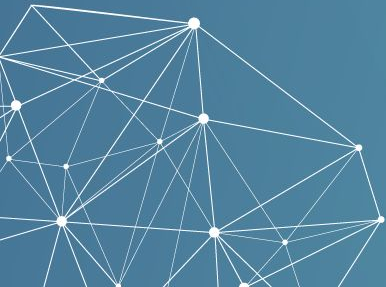
4

Revisar uso de condicionales y estructuras **case**

5

Revisar el uso de tipos **standard** (*primitivos*, *String*, *List*) como parámetros de entrada y de retorno

6





Gracias Posho !