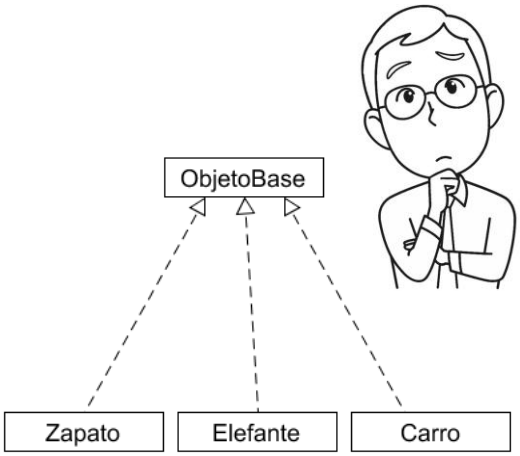
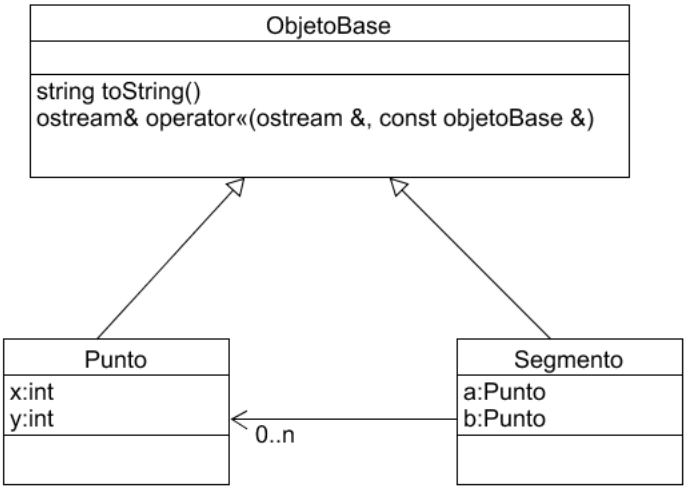
# Objeto Base




Resulta práctico que todas las clases hereden de una clase muy general, lo cual permitiría tener un puntero de esta clase capaz de señalar a cualquier objeto del programa, siendo esto de gran potencial en términos de polimorfismo (principalmente si aún no se conocen las plantillas o programación genérica).

El objetivo de la clase **ObjetoBase** es exigir una interfaz básica la cual incluya métodos básicos como **toString**, así como la implementación de la sobrecarga de operadores tales como **<<** ( con esta implementación ya no habría que implementar este método en las diferentes clases derivadas).

Como referencia se indica que algunos otros lenguajes como Java, incluyen en forma automática esta clase(**Objetc en java**) como padre de cualquier clase que cree el programador.



---

Veamos un ejemplo:

**Ejemplo 1**

```
//-------------------------------------OBJETOBASE.H---------------------------------

#ifndef OBJETOBASE_H
#define OBJETOBASE_H

#include <string>
#include <iostream>

class objetoBase {
public:
	virtual ~objetoBase();
	virtual std::string toString() const = 0;
};

std::ostream& operator<<(std::ostream&, const objetoBase&);

#endif /* OBJETOBASE_H */


//-------------------------------------OBJETOBASE.CPP-------------------------------
#include "objetoBase.h"

objetoBase::~objetoBase() {
}

std::ostream& operator<<(std::ostream &salida, const objetoBase &b) {
	salida << b.toString();
	return salida;
}


//-------------------------------------PUNTO.H---------------------------------

#ifndef PUNTO_H
#define PUNTO_H

#include <string>
#include "objetoBase.h"

class punto : public objetoBase {
private:
	double _x;
	double _y;
public:
	explicit punto(double = 0.0, double = 0.0);
	virtual std::string toString() const;
};

#endif /* PUNTO_H */
```

```
//-------------------------------------PUNTO.CPP---------------------------------

#include <sstream>
#include <iostream>
using namespace std;

#include "punto.h"

punto::punto(double x, double y) : _x(x), _y(y) {
}

string punto::toString() const {
	stringstream r;
	r << "(" << _x << ", " << _y << ")";
	return r.str();
}
//-------------------------------------SEGMENTO.H---------------------------------

#ifndef SEGMENTO_H
#define SEGMENTO_H

#include <string>
#include "objetoBase.h"
#include "punto.h"

class segmento : public objetoBase {
private:
	punto* _a;
	punto* _b;
public:
	segmento(const punto&, const punto&);
	virtual ~segmento();
	virtual std::string toString() const;
};

#endif /* SEGMENTO_H */
//-------------------------------------SEGMENTO.CPP---------------------------------

#include <iostream>
#include <sstream>

#include "segmento.h"
#include "punto.h"
using namespace std;


segmento::segmento(const punto &p1, const punto &p2)
	: _a(new punto(p1)), _b(new punto(p2)) {
}

segmento::~segmento() {
	delete _a;
```

```
        delete _b;
}


string segmento::toString() const {
        stringstream r;
        r << "[" << *_a << ", " << *_b << "]";
        return r.str();
}

//-------------------------------------MAIN.CPP---------------------------------
#include <iostream>
using namespace std;
#include "punto.h"
#include "segmento.h"

int main(int, char**) {
        punto p1(8, 3);
        punto* p2 = new punto(5, -6);
        cout << p1 << endl;
        cout << *p2 << endl;
        cout << endl;
        segmento* s1 = new segmento(p1, *p2);
        segmento* s2 = new segmento(punto(3, 4), punto(-2, 1));
        cout << *s1 << endl;
        cout << *s2 << endl;
        delete s1;
        delete s2;

        cin.get();
        return 0;
}
```