Esquema explicativo de lo que ocurre en las versiones COMPLEJO 1 y COMPLEJO 3

Version COMPLEJO 1

-constructor con 2 parametros obligatorios
-La sobrecarga de operadores hacerla con metodos cuando sea possible
-Cuando no sea posible con métodos entonces hacerlo con funcion no miembro

```
complejo a(2,3),b(2,3);      complejo(int r, int i)

a+b--> a.operator+(b);       complejo complejo::operator+(complejo c)
   --> operator+(a,b);       complejo operator+(complejo c1, complejo c2)

a+3--> a.operator+(3);       complejo complejo::operator+(int i)
   --> operator+(a,3);       complejo operator+(complejo c, int i)

3+b--> operator+(3,b);       complejo operator+(int i, complejo c)
                             No es posible hacerla con metodo
```

Version COMPLEJO 3

-Debe permitir crear complejos con 2 o 1 parametro
-Implementar el mínimo número de constructores posible (usa argumentos implícitos).
-Los operadores + y = deben sobrecargarse el menor número de veces posible

```
complejo a(2,3),b(2);        complejo(int r, int i) --> complejo(int r, int i=0)
                             complejo(int r)        -->


a+b--> a.operator+(b);       complejo complejo::operator+(complejo c)   *(1)*
3+b--> operator+(3,b);       complejo operator+(int i, complejo c)      *(2)*
```

La suma de un complejo con un entero la podemos omitir ya que al tener un constructor que admite 1 parametro el compilador puede hacer un casting y convertir el entero en un complejo y entonces se suman 2 complejos como en el caso *(1)*

```
a+3--> a.operator+(3);       complejo complejo::operator+(int i)
   --> a+complejo(3);    *(1)*
```

Version COMPLEJO 3 con operator int()

Ahora bien, cuando se implemente el metodo operator int():

```
int t = (int)a;   -->   operator int()
```

Si el operator+(int) lo eliminamos, el compilador va a encontrar una ambigüedad en la instrucción a+3 porque va a poder hacer 2 cosas diferentes:

```
a+3??? --> a.operator+(3);   complejo complejo::operator+(int i)
       --> a+complejo(3);    puede convertir el entero 3 a complejo, (suma 2 complejos)
       --> (int)a+3          puede convertir el complejo a int (suma int + int)
```

Al poder hacer 2 cosas diferentes, el compilador se bloquea

En este caso la UNICA solución es evitar que el compilador tenga que hacer casting por no encontrar un método que encaje exactamente con la instrucción, es decir, la SOLUCION ES NO ELIMINAR el método operator+(int i)

```
a+3--> a.operator+(3);       complejo complejo::operator+(int i)
   --> a+complejo(3);    no necesita hacer casting
   --> (int)a+3          no necesita hacer casting
```

De esta forma, al encontrar un método que hace eso, no necesita hacer casting y no se produce ambigüedad.

MORALEJA:
Hay que tener cuidado a la hora de eliminar métodos (como ocurre en Versión COMPLEJO 3), ya que puede que a posteriori, si se implementan otros métodos (como ocurre en Version COMPLEJO 3 con operator int()) tengamos que deshacer lo eliminado y volver a implementar el método eliminado para evitar nuevas ambigüedades.