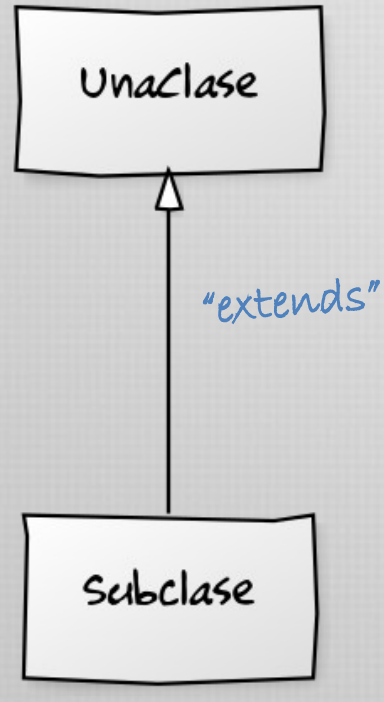
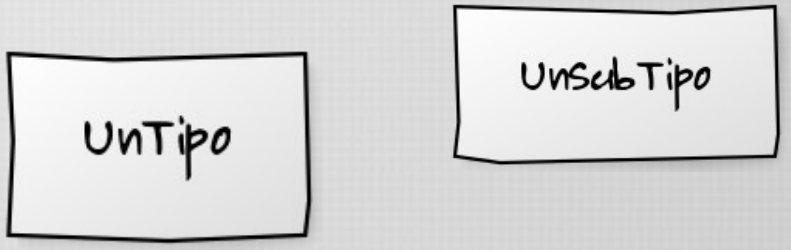
Tecnología de Programación

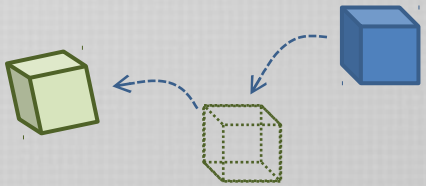
Martín L. Larrea

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Subclases vs Subtipos







Subclases vs Subtipos

Subclase $\neq$ Subtipo

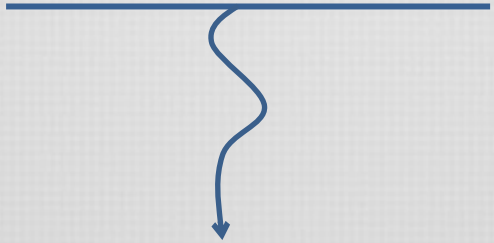
Subclase $\not\Leftrightarrow$ Subtipo

¿Cuándo coinciden?

¿ Subclase $=$ Subtipo ?

Especificaciones semánticas

En términos generales, una clase es la implementación total o parcial de un tipo de dato abstracto

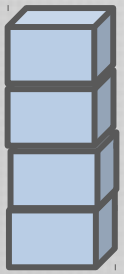


lista de operaciones nombradas
(sintaxis de la operación)

+

significado preciso
(semántica de la operación)

Sintaxis y semántica



```
TDA Pila
 apilar(e:E)
 desapilar():E
 tope():E
 pilaVacia():boolean
```

¿qué hace?
¿se requiere algo especial para invocarla?
¿cuál es el estado de la pila luego de su ejecución?

Es necesario especificar la semántica de la operación

Recordemos que
la especificación correcta y completa
de los requerimientos del software
influye en alcanzar un buen grado de correctitud...

Correctitud del software

¿Qué significa que un programa sea correcto?

Necesitamos saber la descripción precisa de
lo que se supone que debe hacer el programa (la *especificación*)

Lo mismo ocurre con el código, independientemente del tamaño:
¿es correcta la siguiente sentencia?

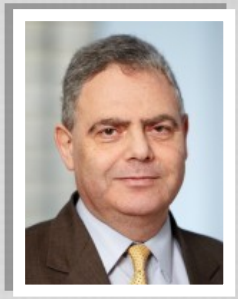
```
x := x + 1
```

La correctitud es subjetiva!

Algunos autores afirman que no debemos
preguntar
si es *correcto* un programa, sino
si es *consistente* con su especificación

Existen mecanismos para definir precisamente las
especificaciones del sistema. Entre ellos, las aserciones.

Diseño por contrato



Bertrand Meyer

Las aserciones forman parte del concepto de
Diseño Por Contrato

El Diseño por Contrato entiende
las relaciones entre una clase y sus clientes como
un *acuerdo formal* expresando las obligaciones y derechos
de cada uno de los participantes

obligación
Proveer materiales e insumos

beneficio
Orden y Limpieza

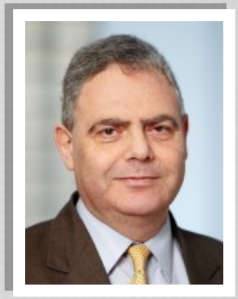
requiere un servicio

provee un servicio

requiere
Materiales e Insumos

garantiza
Orden y Limpieza

Diseño por contrato



Bertrand Meyer

Las aserciones forman parte del concepto de
Diseño Por Contrato

El Diseño por Contrato entiende
las relaciones entre una clase y sus clientes como
un *acuerdo formal* expresando las obligaciones y derechos
de cada uno de los participantes

Definiendo con precisión las responsabilidades
de cada módulo esperamos obtener...

...un buen grado de *confiabilidad* en sistemas de gran complejidad,
...definiendo SW que sea *correcto* desde el principio

¡Será necesario más adelante definir qué sucede cuando el contrato no se cumple (las excepciones)!

Fórmulas de correctitud

Un concepto importante para razonar sobre la correctitud de los elementos de SW son las fórmulas de correctitud.

Sea A una operación y P y Q dos fórmulas lógicas.
Una *fórmula de correctitud* es una expresión de la forma:

{P} A {Q}

Cuyo significado es:

*Cualquier ejecución de **A**,*
*comenzando en un estado en donde se cumple **P**,*
*terminará en un estado donde se cumple **Q***

Estas fórmulas se denominan también *ternas de Hoare*.

Fórmulas de correctitud

{P} A {Q}

precondición

poscondición

La precondición establece las propiedades que se tienen que cumplir cada vez que se llame a la operación.

La postcondición establece las propiedades que debe garantizar la operación cuando retorne.

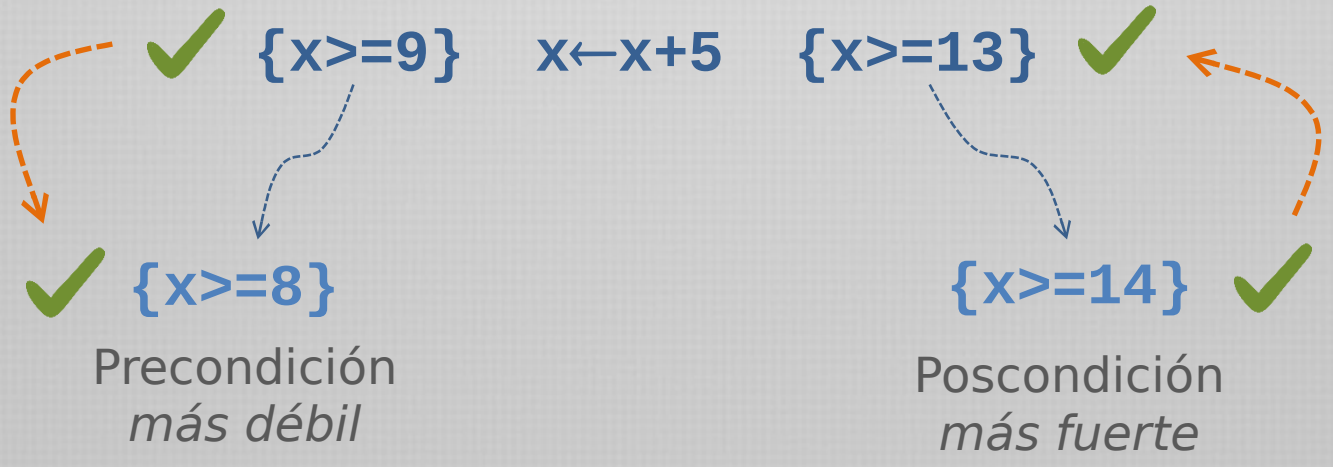
Las precondiciones y postcondiciones pueden tener distintos niveles de "*fortaleza*", o "*exigencia*".

Fórmulas de correctitud

Una fórmula P_1 *es más fuerte* que una fórmula P_2 si
P_1 implica P_2 y
las fórmulas no son iguales.

x>10 *es más fuerte que* **x>8**

Ejemplo



Aserciones

Las pre y postcondiciones son denominadas también aserciones

Las aserciones son expresiones que involucran algunas entidades del software y establecen algunas propiedades que éstas entidades deben cumplir
en un momento de la ejecución.

Usaremos la sintaxis de las expresiones booleanas para la especificación de las aserciones.

Por simplicidad podemos reemplazar el **y (and)** por el **punto y coma**

A y B y C ≡ A;B;C

Precondiciones

Las pre y postcondiciones son utilizadas para
la especificación semántica de las operaciones de las clases.

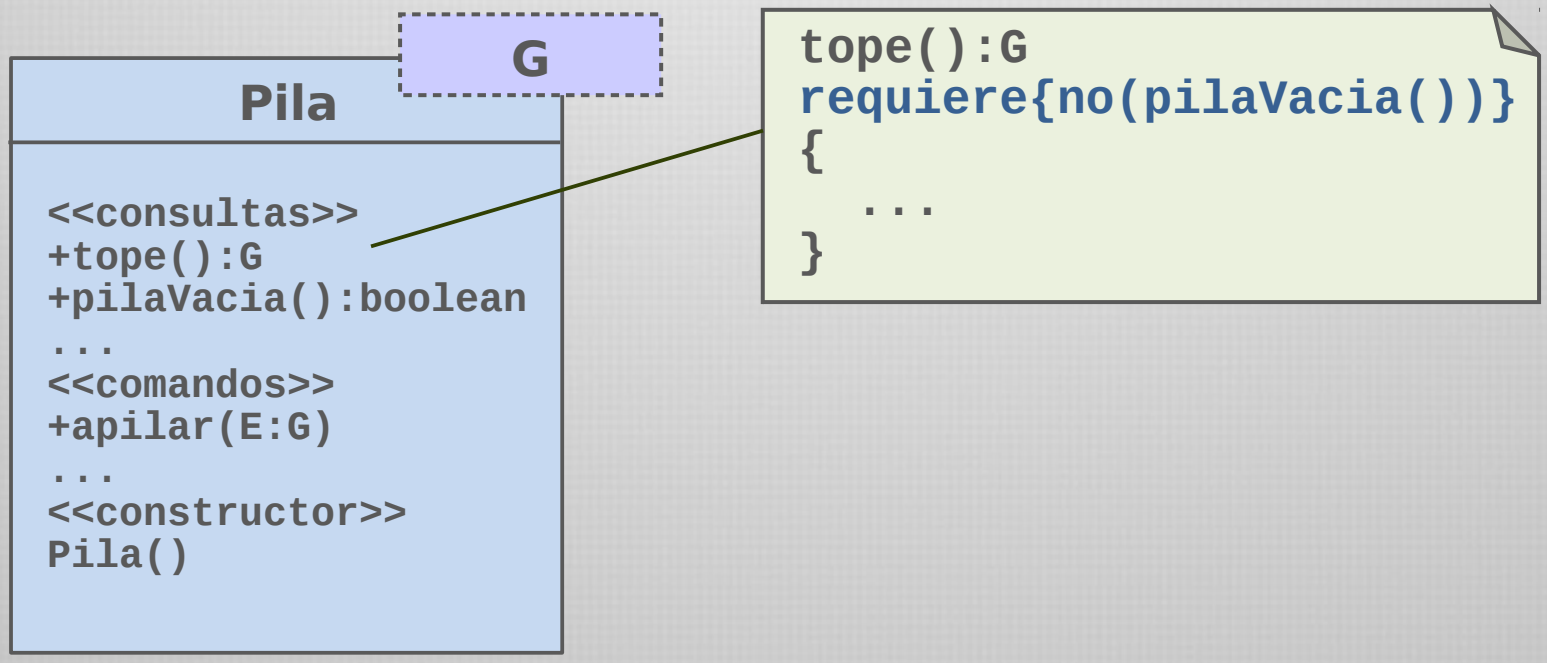
Pre

- La precondición establece las propiedades que se tienen que cumplir cada vez que se invoque a la operación
- Expresa las restricciones necesarias para que la operación funcione correctamente.

- Se debe aplicar a la invocación de todos los mensajes
en los que aparece la operación
- Involucra propiedades de los estados del objeto receptor y/o de los parámetros.

Precondiciones

Para especificar las precondiciones en el código del programa utilizaremos la palabra reservada **requiere**, seguida de la expresión booleana entre llaves.



Poscondiciones

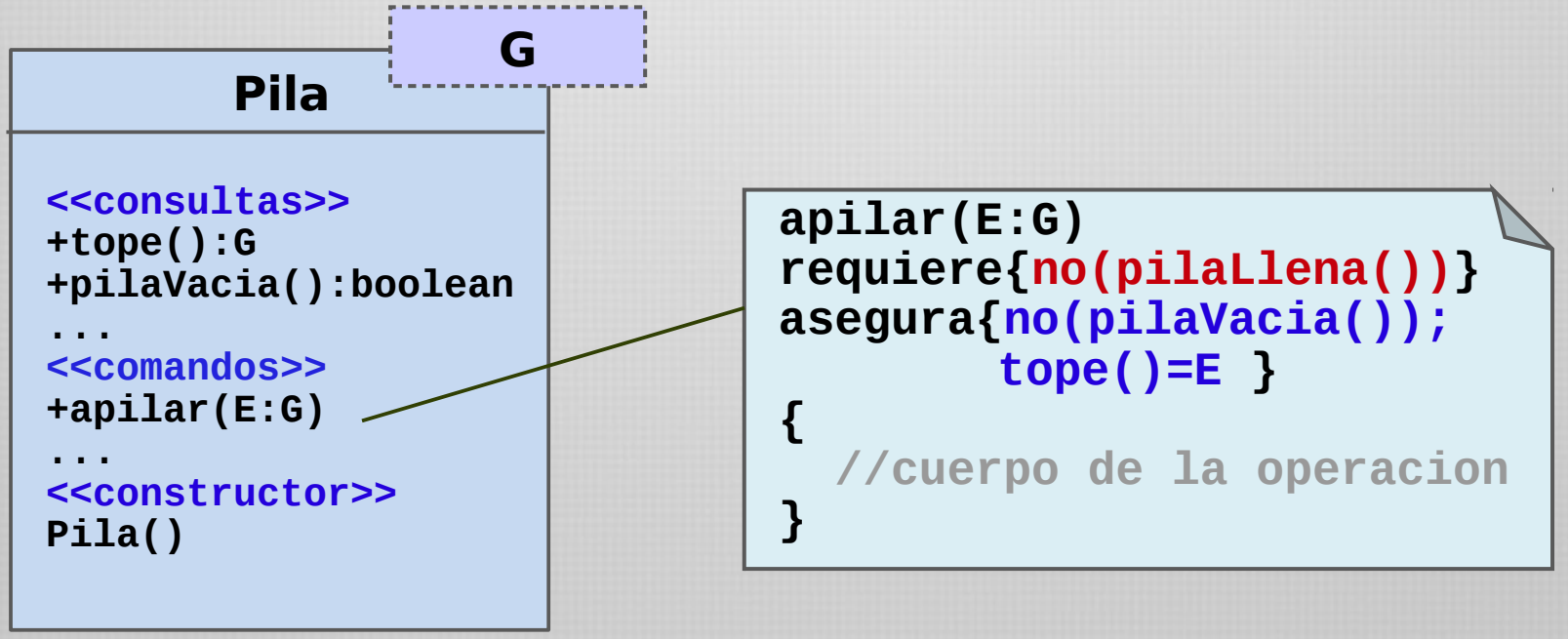
La poscondición de un servicio define las propiedades que deben ser satisfechas al finalizar cada ejecución del servicio.

Post

- Debe expresar las propiedades resultantes de la ejecución de un servicio, siempre que la ejecución funcione correctamente.
- Generalmente involucra el estado del objeto receptor (constructores o comandos) o el estado del resultado. Para este último utilizaremos la palabra **`Resultado`**
- *Eventualmente*, puede referirse al estado inicial de una entidad al comienzo de la ejecución anteponiendo la palabra **`old`**

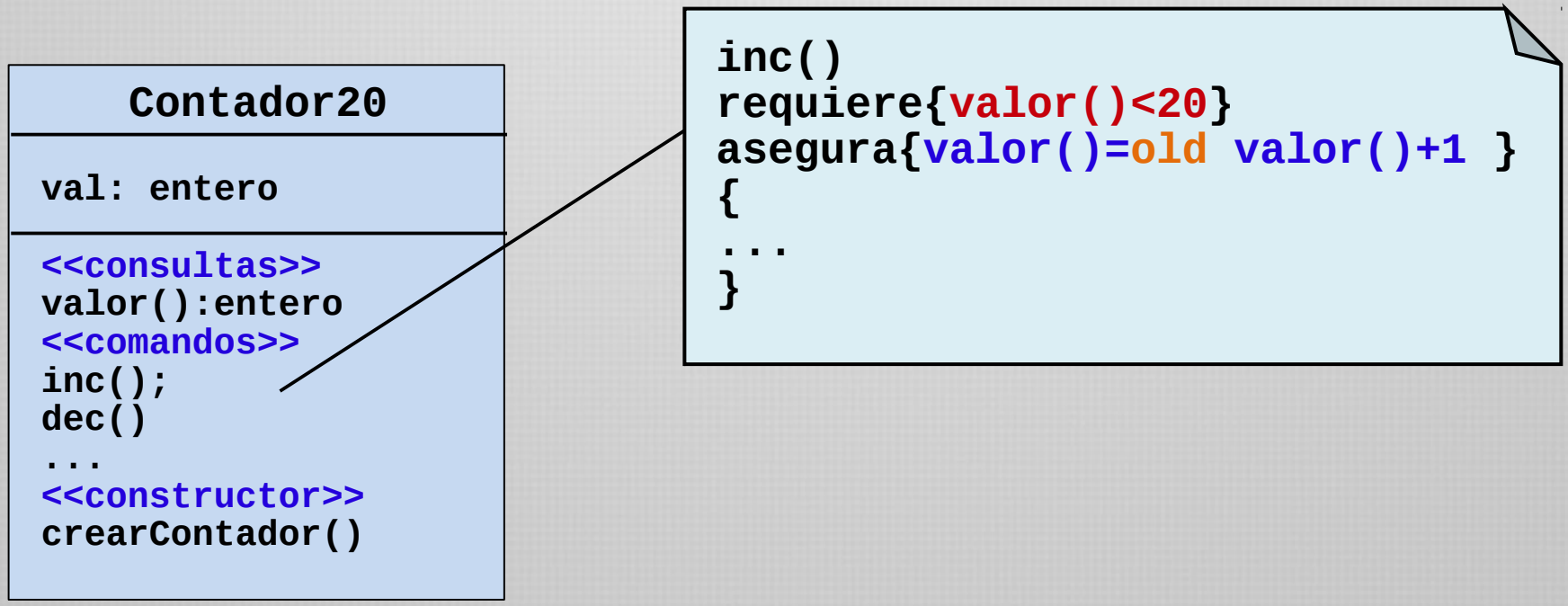
Poscondiciones

Para especificar las postcondiciones en el código del programa utilizaremos la palabra reservada **asegura**, seguida de la expresión booleana correspondiente



Poscondiciones

Para especificar las postcondiciones en el código del programa utilizaremos la palabra reservada **asegura**, seguida de la expresión booleana correspondiente



Obligaciones y derechos

Desde el punto de vista del contrato:

- las precondiciones obligan al objeto cliente y benefician al objeto servidor
- las postcondiciones obligan al objeto servidor y benefician al objeto cliente

	Obligaciones	Beneficios
Cliente	Satisfacer la precondición	Garantía de cumplimiento de poscondiciones
Servidor	Satisfacer las postcondiciones	Suponer precondiciones

Simplificación de código

El diseño por contrato simplifica el estilo de programación!

Con aserciones

```
desapilar():G
requiere{no(pilaVacia())}
{
  ...
}
```

Sin aserciones

```
desapilar():G
{
  Si no(pilaVacia())
  entonces
     ...
  si no
        ...
}
```

Bajo ninguna circunstancia el cuerpo del servicio debe testear las precondiciones de ese servicio

La *programación defensiva* complica el código y la definición de las responsabilidades. Las aserciones simplifican la codificación.

Precondiciones

Las precondiciones no son mecanismos de chequeo de entrada al usuario.
Sólo hacen referencia a la comunicación SW-SW,
no SW-humanos o SW-mundo exterior.

```
leer_entero():G
requiere{read()>0}
{
  ...
}
```

INCORRECTO

Las aserciones tampoco son reemplazos a las estructuras de control!

Violación de aserciones

Si se viola una precondición, se trata de un error en el cliente.

Si se viola una poscondición, se trata de un error en el servidor.

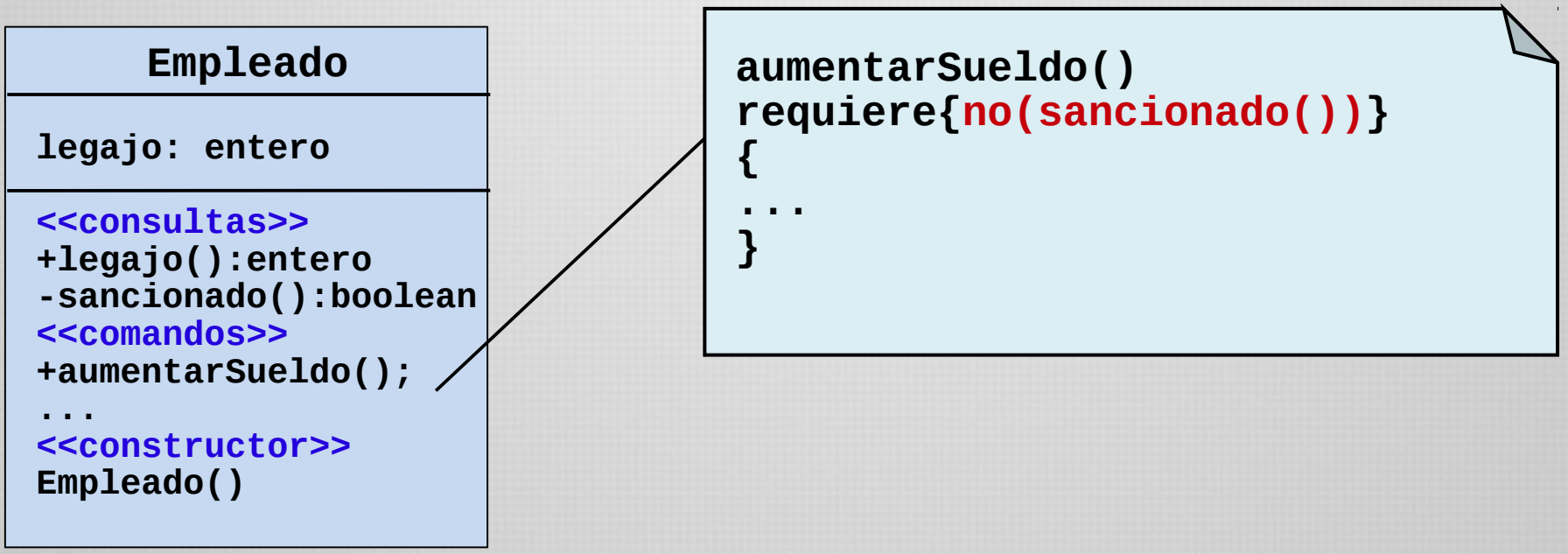
Un error es una decisión equivocada
hecha durante el desarrollo del software

Un defecto es una característica del software que
puede provocar una desviación de sus objetivos
pretendidos.

Una falla es el hecho de que el software
se desvíe de sus objetivos pretendidos.

Las aserciones pretenden evitar software con defectos y
favorecer la confiabilidad del software.

Visibilidad



Dado que la precondición debe necesariamente ser conocida por el cliente, toda entidad referenciada en una precondición debe ser visible para los usuarios del servicio.

La poscondición, sin embargo, puede especificar restricciones sobre entidades privadas.

Invariante de clase

Las precondiciones y poscondiciones describen propiedades de rutinas individuales.

Es necesario también expresar propiedades globales de todas las instancias de una clase, que deben ser preservadas por todas las rutinas.

Un invariante de clase es una aserción que expresa restricciones generales que se aplican a toda la clase.

Al no depender de ningún servicio, refieren únicamente a atributos y servicios de la clase.

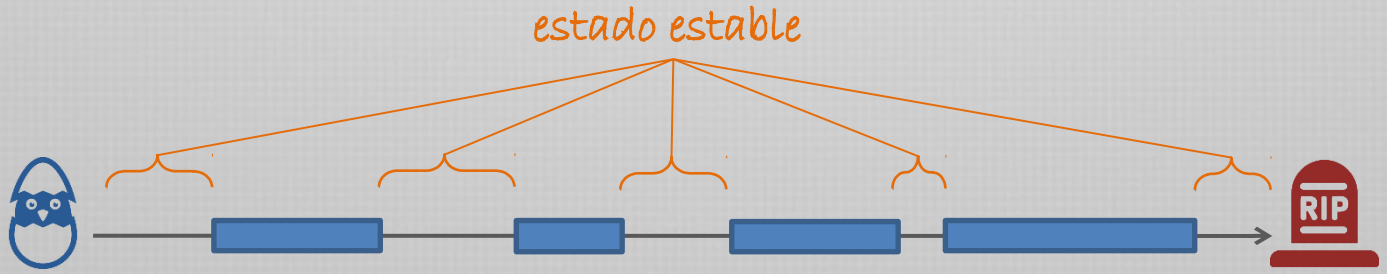
Invariante de clase

Sintácticamente, un invariante es una expresión booleana al igual que las precondiciones y poscondiciones.

Semánticamente, el invariante expresa una propiedad que debe cumplirse en todo *estado estable* de una instancia de la clase.

Estado estable de un objeto:

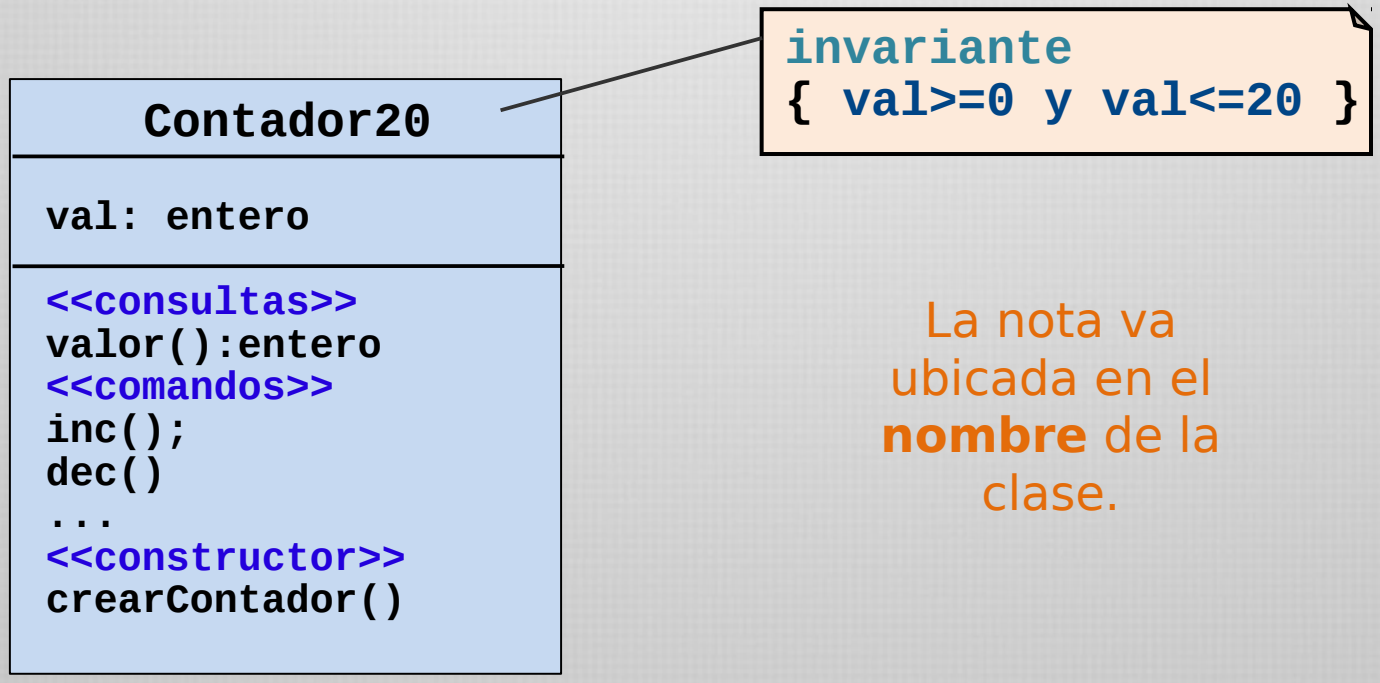
- Inmediatamente después de ejecutarse el constructor.
- Inmediatamente antes y después de responder a cualquier comando o consulta



¡Un invariante puede violarse temporalmente durante la ejecución de una operación!

Notación

Los invariantes de clase serán indicados por medio de una nota en el diagrama de clases, junto con la palabra **invariante**.



La nota va ubicada en el **nombre** de la clase.

Los invariantes describen propiedades específicas de una clase.
¿cuándo un invariante es correcto?

Invariante correcto

Una aserción I es un correcto invariante de una clase C si:

- Todo procedimiento de creación de C, satisfaciendo su respectiva precondición, concluye en un estado que satisface I
- Toda rutina exportada de la clase, aplicada a argumentos en un estado que satisface su propia precondición y la condición I, concluye en un estado donde se satisface I

Aserciones en Java

El lenguaje Java provee la sentencia **assert** que permite al programador testear condiciones de ejecución de su programa.

Esta sentencia no soporta completamente el modelo de contrato cliente/servidor visto anteriormente, pero ayuda a construir una estrategia informal de diseño por contrato.

Sintaxis:

```
assert Expresion1;
assert Expresión1:Expresión2;
```

Expresion1 tiene que ser una expresión booleana.

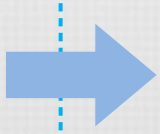
Expresion2 debe producir un texto con un mensaje de error apropiado. Puede ser la invocación a un método pero no de tipo void.

Expresion1 es una expresión que se cree verdadera en un momento de la ejecución y que es evaluada por el assert.
Si se cumple, continúa la ejecución. Si no, produce una *excepción*

Cuidado con los efectos colaterales de las funciones!

Aserciones en Java - ejemplos

```
if (a==1){
...
} else if (a==2){
...
} else { //cuando a vale 3
...
}
```



```
if (a==1){
...
} else if (a==2){
...
} else {
  assert (a==3)
  ...
}
```

```
assert ref != null;
assert ref.m1(parametro);
assert valor>0 : “argumento negativo”;
```

Aserciones y herencia

La convivencia de la herencia con las aserciones puede dar lugar a situaciones controversiales.

En particular, debe observarse el hecho de que las aserciones (invariantes, precondiciones y poscondiciones de operaciones) deben respetarse en las clases descendientes.

Forman parte del contrato original

Aquí entra en juego la vista de herencia como subtipo, y el hecho de que un objeto tiene varios tipos de datos y por ende, la posibilidad de sustituir un objeto por otro.

Veamos las particularidades de las aserciones con herencia de por medio...

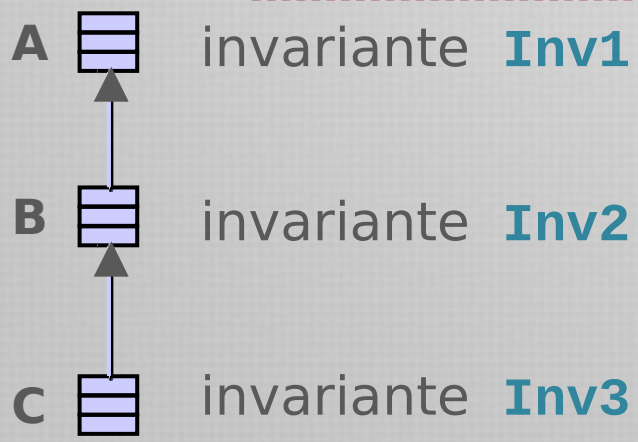
Invariantes

Un invariante de clase es una aserción que expresa restricciones generales que se aplican a toda la clase.

El invariante de la clase Polígono habla del estado estable de objetos que son instancia de Polígono...

... y por lo tanto también de las instancias de Triángulo o Cuadrado, que son instancias de Polígono.

El invariante de todos los ancestros de una clase C se aplican también a la clase C

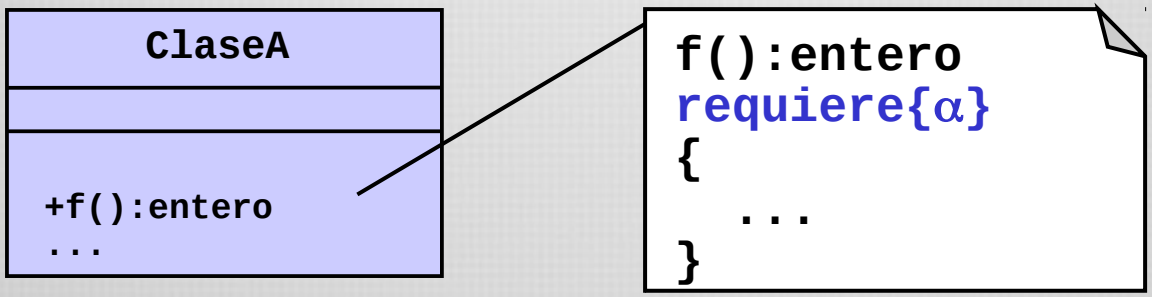


Para la clase C debe cumplirse

Inv3 and Inv2 and Inv1

Precondiciones y poscondiciones

El caso de las precondiciones y las poscondiciones de las operaciones es un poco más delicado.



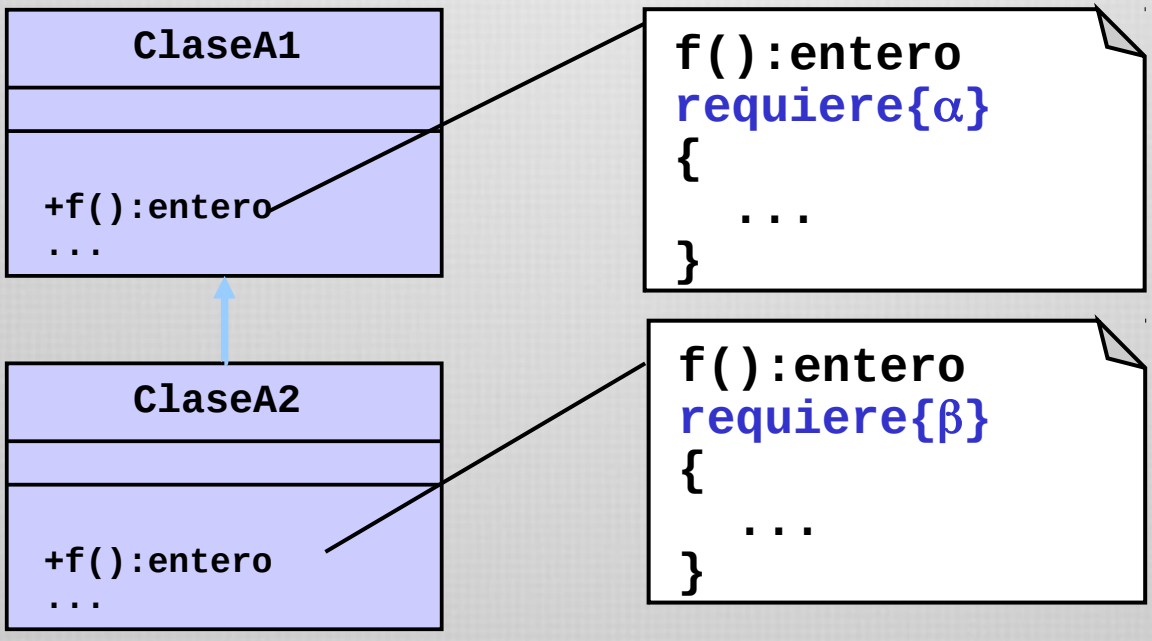

```
objA:ClaseA;
 ...
objA.f();
```

Como vimos anteriormente, quien solicita el servicio **f()** de **objA** debe cumplir con sus requerimientos (precondiciones).

Debe asegurarse de que se cumpla α .

Precondiciones

El problema surge cuando se redefinen operaciones con precondiciones y poscondiciones...




```
objA:ClaseA;
 ...
objA.f();
```

El cliente intentará cumplir la precondición α para invocar `f()`.

Sin embargo, por polimorfismo puede que **`objA`** esté referenciando a un objeto de tipo **`ClaseA2`**, cuya precondición es β...

Fortaleza de precondiciones

Es necesario establecer un criterio para que el cliente no se preocupe por el polimorfismo al querer cumplir el *contrato.*

Para esto debemos tener en cuenta que nuestro cliente garantiza el cumplimiento de α para invocar a `f()`.

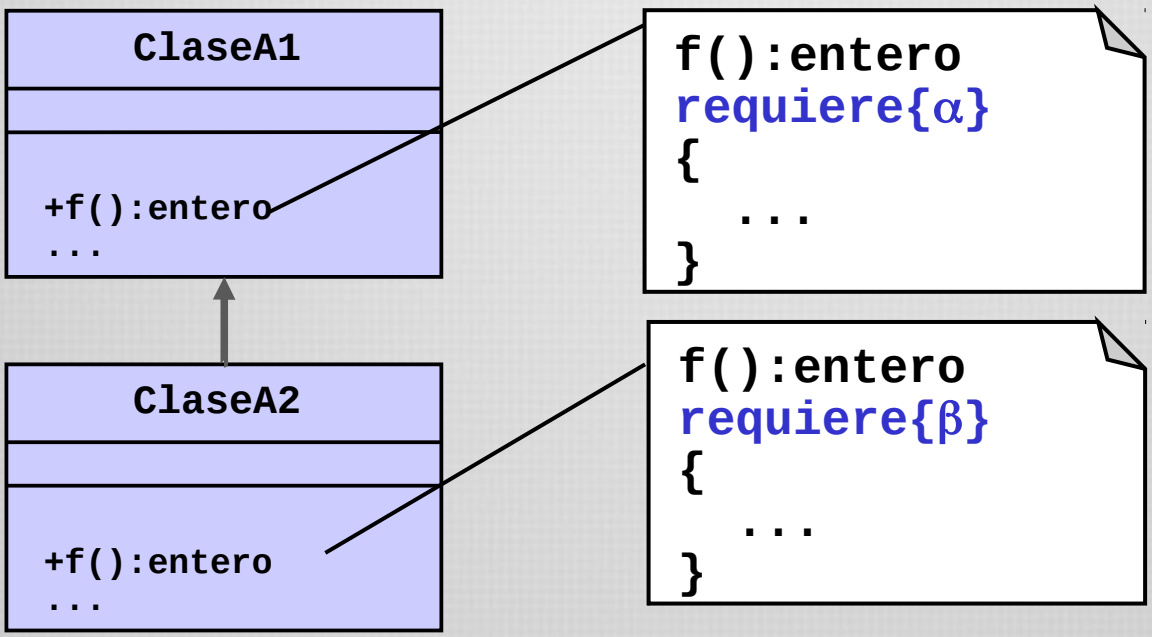
Para no interferir en este acuerdo, en la redefinición del servicio `f()` debemos incluir en la precondición un requerimiento igual o más débil que el original.

Recordemos que una fórmula P_1 *es más fuerte* que una fórmula P_2 si P_1 *implica* P_2 y las fórmulas no son iguales.

`x>10` es más fuerte que **`x>8`**

Si P_1 *es más fuerte* que P_2 entonces P_2 es *más débil* que P_1

Fortaleza de precondiciones



La precondición α debe ser más fuerte que la precondición β.

Por ejemplo:

α = g() > 20

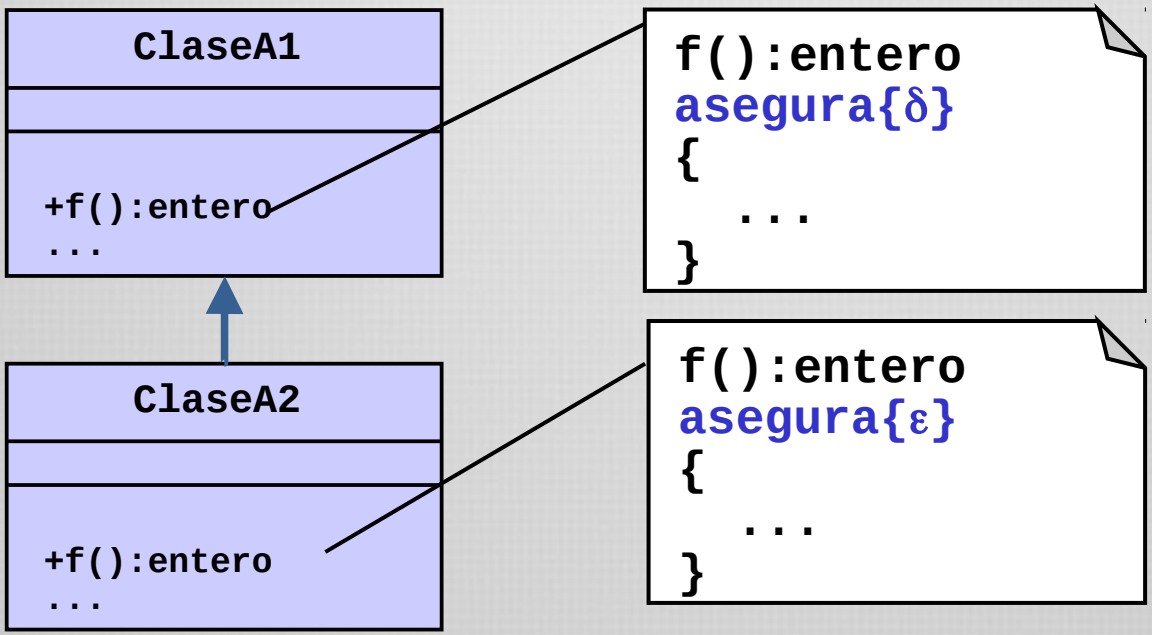
β = g() > 13

α = alumno_regular(X)

β = alumno(X)

Fortaleza de poscondiciones

Con las poscondiciones sucede algo similar...



Aquí el cliente espera que se le garantice δ.
Sin embargo, por polimorfismo, probablemente se le esté garantizando ε.

Debemos establecer nuevamente un criterio
para las poscondiciones en la redefinición de operaciones...

Fortaleza de poscondiciones

Para esto debemos tener en cuenta que nuestro cliente espera que se le garantice el cumplimiento de δ al invocar a `f()`.

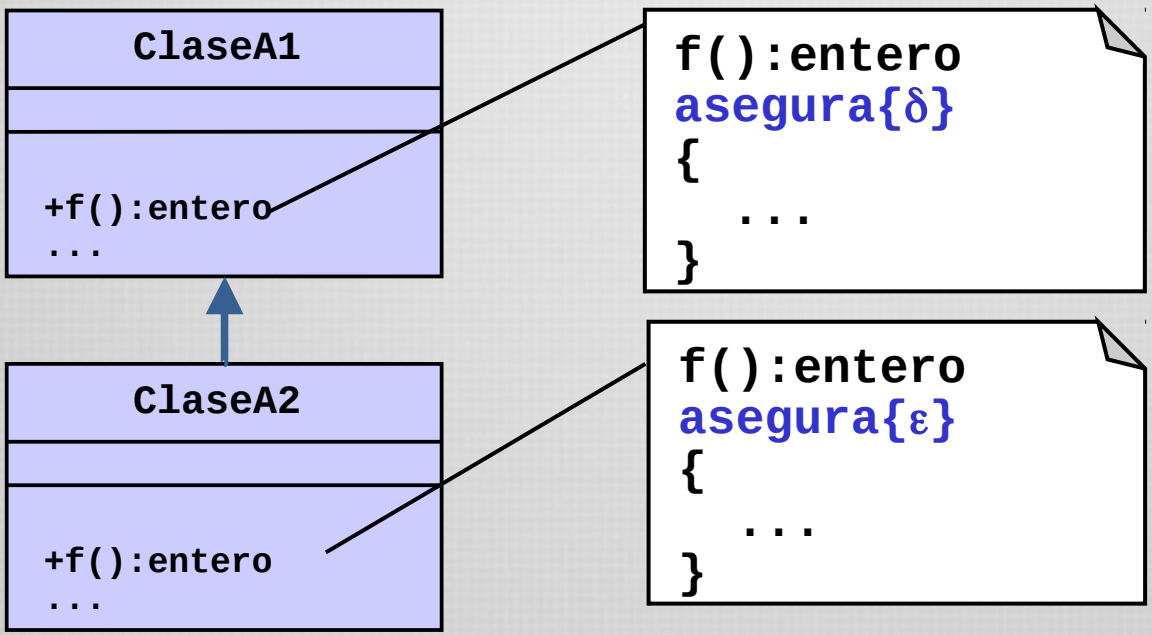
Para no interferir en este acuerdo, en la redefinición del servicio `f()` debemos incluir en la poscondición un requerimiento igual o más fuerte que el original.

De esta forma, le aseguramos que se cumpla δ, y probablemente algo más...

> La poscondición δ debe ser más débil que la poscondición ε.

Esto es, si se cumple ε entonces se cumple también δ.
De esta manera no violamos el acuerdo previo.

Fortaleza de poscondiciones



Por ejemplo:

δ = h() > 0

ε = h() > 10

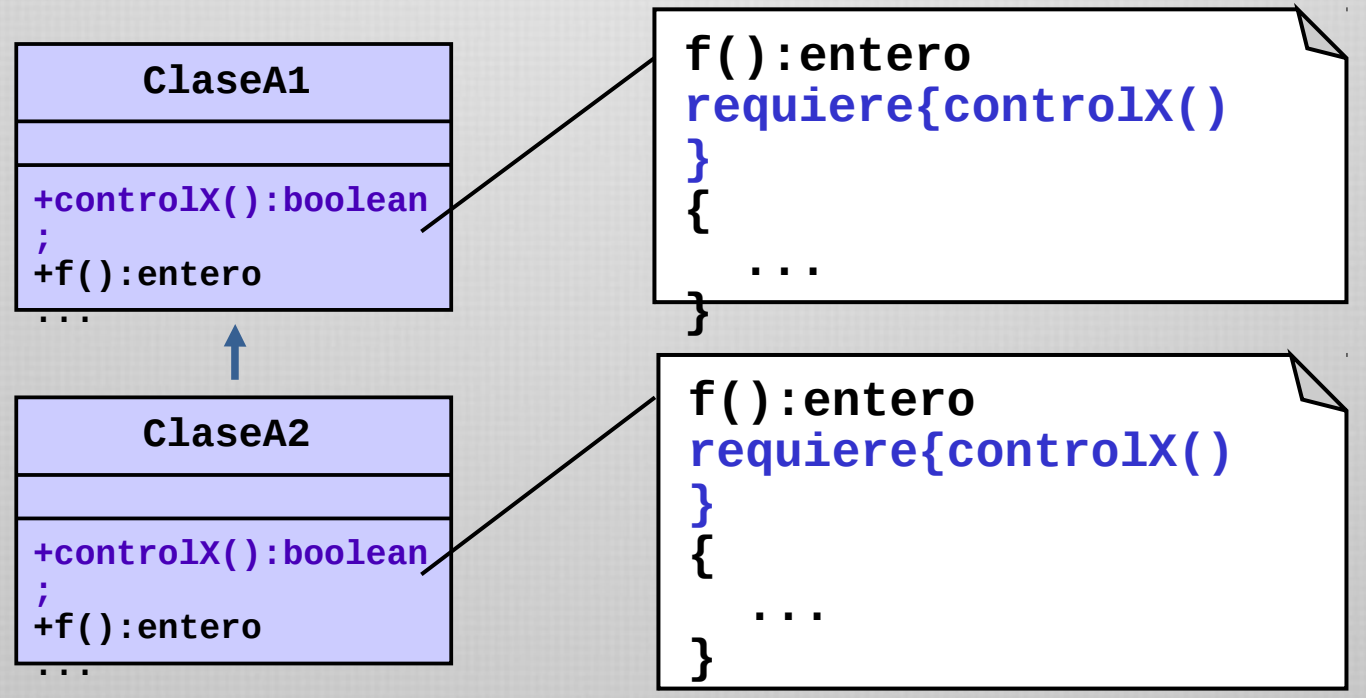
δ = conectado_Internet()

ε = chat_establecido()

Precondiciones abstractas

Una forma de abstraerse de estas consideraciones es utilizar precondiciones “encapsuladas” en operaciones.

Meyer las denomina precondiciones abstractas.
(por la abstracción de la precondición, no porque no estén implementadas).



De esta manera la precondición puede redefinirse incluso por una más fuerte, lo que no traiciona el contrato previamente establecido.

El cliente, al fin y al cabo, no conoce la implementación de **`controlX()`**