Diseño por contratos

Roni Guillermo Apaza Aceituno

Universidad Nacional de San Agustin

rapazaac@unsa.edu.pe

October 17, 2018

Introducción

- Tiene raíces en los métodos formales del software.
- Requieren esfuerzo extra.
- Pero garantizan un software mas confiable.

Introducción

- "Yo creo que el uso de contratos a la Eiffel entre módulos es la no-práctica más importante en el mundo del software hoy. Por esto yo quiero decir que no hay ninguna otra práctica siendo impulsada actualmente que tenga mayor capacidad de mejorar la calidad del software producido." (DeMarco 1997).

Introducción

- El contrato puede compararse con el contrato entre dos personas.
- Con obligaciones y beneficios.
- El contrato especifica las relacioes entre las rutinas y los que hacen uso de las mismas.

Introducción

- El contrato puede compararse con el contrato entre dos personas.
- Con obligaciones y beneficios.
- El contrato especifica las relacioes entre las rutinas y los que hacen uso de las mismas.

Introducción

- Un sistema puede ser visto como un conjunto de partes.
- Cada parte satisface una necesidad, este es el objetivo.
- Los objetivos pueden ser vistos como contratos.

Introducción

- Los sistemas juegan el rol de cliente y proveedores.
- Esta relación establece beneficios y obligaciones.
- La especificación de estos beneficios y obligaciones son los contratos.

Introducción

- Un contrato es sinonimo de protección.
- Protege al cliente sabiendo *cuanto* debe recibir.
- Protege al proveedor especificando el *mínimo* servicio aceptable.

Introducción

- El ejemplo del envió de un paquete.
- Enviarlo uno mismo.
- Usar una empresa de envió.

Introducción

- Una separación de la realidad.
- **NO CLAUSULAS OCULTAS**.
- Deben existir regulaciónes externas.

Introducción

- Las expresiones lógicas que expresan un contrato son llamadas de aserciones.
- Existen diferentes tipos de aserciones.
- En el diseño de contrato se usan tres:

Introducción

- Precondiciones.
- Postcondiciones.
- Invariantes de clase.

Introducción

- Tripleta de Hoare.
- $\{P\}A\{Q\}$
- P y Q son aserciones y A es cualquier código.

Introducción

- Cualquier ejecución de A que comience en un estado cumplido **P**.
- Dara como resultado un estado.
- Este estado debe satisface **Q**

Precondiciones

- {P} Representaría las precondiciones.
- Describe una rutina en particular.

Postcondiciones

- $\{Q\}$ Representa las postcondiciones.
- Describe una rutina en particular.

Invariantes de clase

- Regulaciónes externas de un contrato.
- Sirven para expresar propiedades globales de las instancias de una clas.
- Ejemplo del no negativo elemento en el array.

Ejemplo

- $\{P\}A\{Q\}$
- Representa la formula de corrección.
- $\{$ x ¿ 10 $\}$ x := x /2 $\{$ x ¿= 5$\}$

Ejemplo

- Si se parte de un estado en el cual x es mayor que 10
- Se aplica la computación $x := x/2$
- x será mayor o igual a 5

Ejemplo

- La precondición es x ¿ 10
- la poscondición x ¿=5

principio de no redundancia

- Bajo ninguna circunstancia el cuerpo de una rutina deberá chequear el cumplimiento de la precondición.
- La precondición compromete al cliente, ya que define las condiciones por las cuales una llamada a la rutina es válida.
- Las poscondiciones comprometen a la clase (donde se implementa la rutina) ya que establecen las obligaciones de la rutina.

principio de no redundancia

- Es muy importante notar que la precondición y la poscondición que definen el contrato forman parte del elemento de software en sí.
- El diseño y la programación por contratos es en cierto sentido opuesto a lo que se denomina programación defensiva.
- La programación defensiva incita a los programadores a realizar todas las verificaciones posibles en sus rutinas de tal forma de prevenir que una llamada pueda producir errores.

Programación defensiva

- Resultando rutinas muy generales (excesivamente generales).
- Esto es razonable.
- Pero aumenta la complejidad del código.

Programación defensiva

- Los chequeos ciegos (hechos por seguridad excesiva) aumenta la cantidad de software.
- Al ser software esto agrega mas lugares propensos a errores.
- Lo cual genera mas métodos de verificación.

Programación defensiva

- La programación defensiva no tiene como objetivo tener especificaciones bien conocidas.
- Es permitir que codigo fuente se ejecute en condiciones arbitrarias.
- No define que hacer con valores incorrectos.

Asserciones

- Es posible establecer aserciones más fuertes que otras.
- Como se establece esto.
- Dadas dos aserciones P y Q.

Asserciones

- P es más fuerte o igual que Q si P implica Q.
- El concepto de fortificar o debilitar aserciones es usado en la herencia cuando es necesario redefinir rutinas.
- El lenguaje para soportar aserciones tiene algunas diferencias con el cálculo de predicados.

Asserciones

- No hay cuantificadores, aunque los agentes pueden suplir esta deficiencia.
- Soporta llamadas a funciones.
- En Eiffel se puede usar la palabra reservada old.

Asserciones

- Old es usada en postcondiciones.
- Utilizada para indicar el valor anterior a la ejecución de la rutina de algún elemento.

Asserciones

- Toda función que aparezca en la precondición de una rutina r debe estar disponible a todo cliente al cual esté disponible dicha rutina r.
- Si esto no fuera así, podría ser imposible para el cliente garantizar que se cumple la precondición antes de llamar a la rutina.

Asserciones

- Una aserción permite probar la exactitud de cualquier condicion que pueda haber en el programa.
- Esto en java es lograda con la sentencia assert.
- En caso de error aparece un mensaje.

Asserciones

- Este mensaje es utilizado en el desarrollo de software.
- Es usado con una expresión Booleana.
- Puede ser escrito por ejemplo:

Asserciones

- assert expresión;
- assert expression1 : expression2;

Asserciones

- // Java program to demonstrate syntax of assertion
- import java.util.Scanner;
- class Test
- {
- public static void main(String args[])
- {
- int value = 15;
- assert value ¿= 20 : " Underweight";
- System.out.println("value is "+value);
- }
- }

Asserciones

- Por defecto las aserciones estan desabilitadas.
- Se tiene que escribir una sentencia especial.
- java –ea Test
- java –enableassertions Test

Asserciones

- Desabilitar las aserciones necesita las siguientes opciones.
- java –da Test
- java –disableassertions Test

Asserciones

- En c/c++ también existen aserciones.
- La siguiente sentencia para poner una aserción es:
- void assert(int expression);

Asserciones

- Si es falsa la expresión o cero.
- Es enviado un error con el nombre de archivo y número de línea.
- La función abort() es llamada.

Asserciones

- #include < *stdio.h* >
- #include < *assert.h* >
- int main()
- {
- int x = 7;
-
- /* Some big code in between and let's say x
- is accidentally changed to 9 */
- x = 9;
- // Programmer assumes x to be 7 in rest of the code
- assert(x==7);
-
- /* Rest of the code */
- return 0;
- }

Asserciones

- En c/c++ nosotros podemos remover las aserciones en tiempo de compilación.
- Se puede usar el preprocesador NODEBUG.

Invariantes de clase

- Sirven para expresar propiedades globales de las instancias de una clase.
- Las precondiciones y las postcondiciones expresan propiedades de una rutina en particular.
- Utilizando la metafora de los contratos podemos decir.

Invariantes de clase

- Establecen regulaciones generales aplicables a todos los contratos.
- Las invariantes de clase proveen el concepto de invariante de datos de Hoare.
- Los invarientes deben satisfacer las condiciones luego de la creación de un objeto o la llamada de una rutina de un cliente.

Invariantes de clase

- class Fecha {
- int dia;
- int hora;
- invariant() {
- assert(1 ¡= dia && dia ¡= 31);
- assert(0 ¡= hora && hora ¡ 24);
- }
- }

Corrección de una clase

- Una clase es correcto respecto a sus aserciones si cumplen las siguientes condicones.
- Para toda rutina de creación rc: pre rc do rc post rc and INV
- Para toda rutina exportada r: INV and pre r do r post r and INV

La calidad de usar contratos

- El significado de la violación de un contrato significa un defecto de software.
- La aserción sirve para identificar el tipo de defecto.
- Si hay una violación de la precondición, el cliente esta equivocado.

La calidad de usar contratos

- Si hay una violación del tipo de proveedor, el proveedor esta equivocado.
- Las acciones de prueba de un programa siempre serán en tiempo de ejecución.
- Las aserciones pueden monitorear un programa en tiempo de ejecución.

La calidad de usar contratos

- La calidad es una parte importante de la creación de un programa.
- Estas afirmaciones de calidad cobran fuerza con lo previsto en contratos.
- Las afirmaciones de depuración también tienen un beneficio con los contratos.

La calidad de usar contratos

- No solamente es una herramienta de documentación.
- No solamente es una herramienta de especificación.
- Cumplen un rol importante durante todas las etapas de desarrollo de software.

La calidad de usar contratos

- El desarrollador debe escoger que aserciones monitorear en tiempo de ejecución.
- Es importante la elección que haga el desarrollador.
- Luego de la depuración y el testeo es posible entrar en producción sin las aserciones.

La calidad de usar contratos

- Una recomendación es tener habilitadas las precondiciones.
- La violación de una aserción en tiempo de ejecución produce una excepción.
- La excepción puede ser manejada en tiempo de ejecución.

La calidad de usar contratos

- La comunicación es la principal característica de los contratos al aplicarse sobre la interacción entre personas como gerentes y desarrolladores.
- En la confección de manuales.
- Otras actividades.

La calidad de usar contratos

- Los contratos permiten un reuso del software.
- Estos proveen información de que se esta reusando.
- También de las condiciones en las cuales se puede reusar.

La herencia en los contratos

- Al generar una clase de una subclase existente es posible la redefinición de algunas características.
- ¿qué pasa con los contratos?
- ¿qué ocurre con el invariante de clase?

La herencia en los contratos

- En el caso de invariantes de una clase.
- Existe la regla de la herencia del invariante.

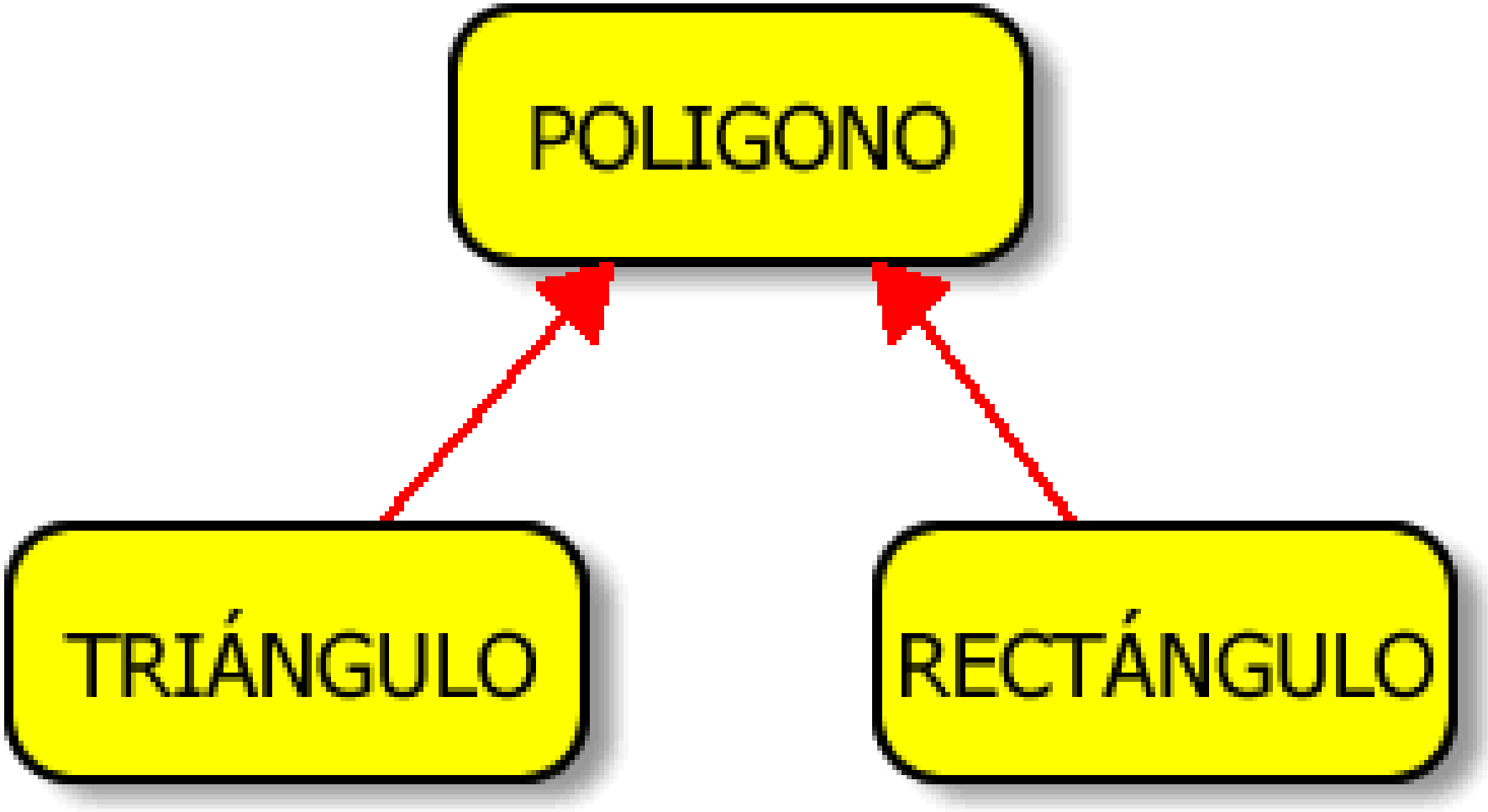
La herencia en los contratos

- La propiedad de una invariante de clase es una operación de conjunción.
- La operación de la conjunción se da entre las aserciones presentes en su cláusula invariante.
- Y las propiedades invariantes de sus padres.

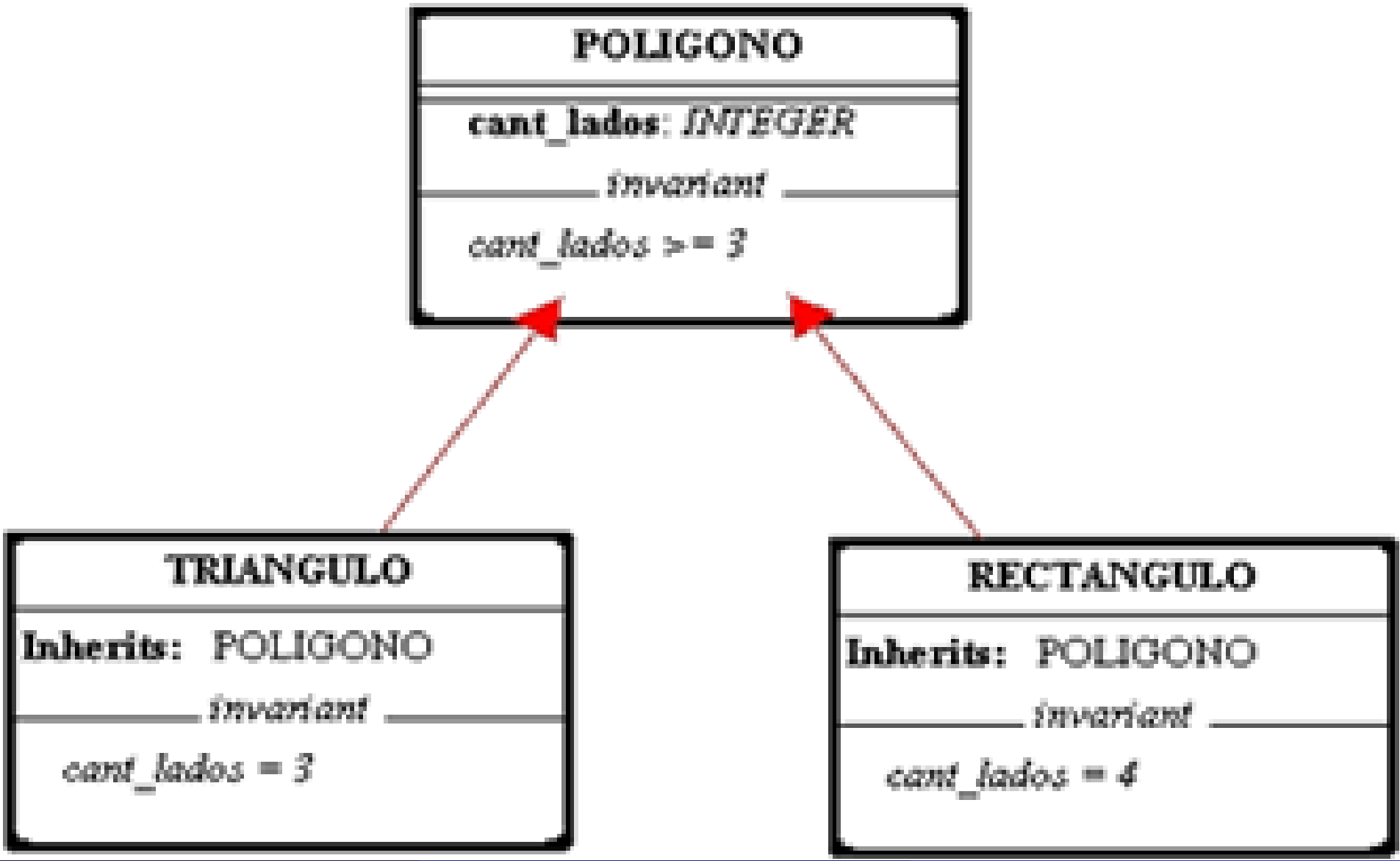
La herencia en los contratos

- Esta regla es recursiva comprendiendo todos los invariantes de los ancestros de una clase.
- Esto surge de la naturaleza propia de la herencia.
- Las instancias de un clase tambien son las instancias de un ancestro.

La herencia en los contratos



La herencia en los contratos



La herencia en los contratos

- La invariante de un poligono es el número de lados.
- Esta restricción permite un número de lados mayor a 3.
- Se puede reescribir el número de lados incrementando este número.

La herencia en los contratos

- Una regla de correción de clases y jerarquia de herencia definida.
- Si hubiera una subclase poligono con un invariante indicando un número de lados igual a 2.
- Dicha propiedad invalidaria la clase.

La herencia en los contratos

- Al declarar rutinas, es necesario respetar las aserciones de la rutina original.
- El uso de una rutina no implica la ruptura de un contrato.
- Las clases heredan los contratos de sus ancestros y deben ser respetados.

La herencia en los contratos

- Para redefinir un contrato se puede usar la metafora del subcontrato.
- No siempre el que acuerda el contrato lleva a cabo el contrato.
- También es posible subcontratar.

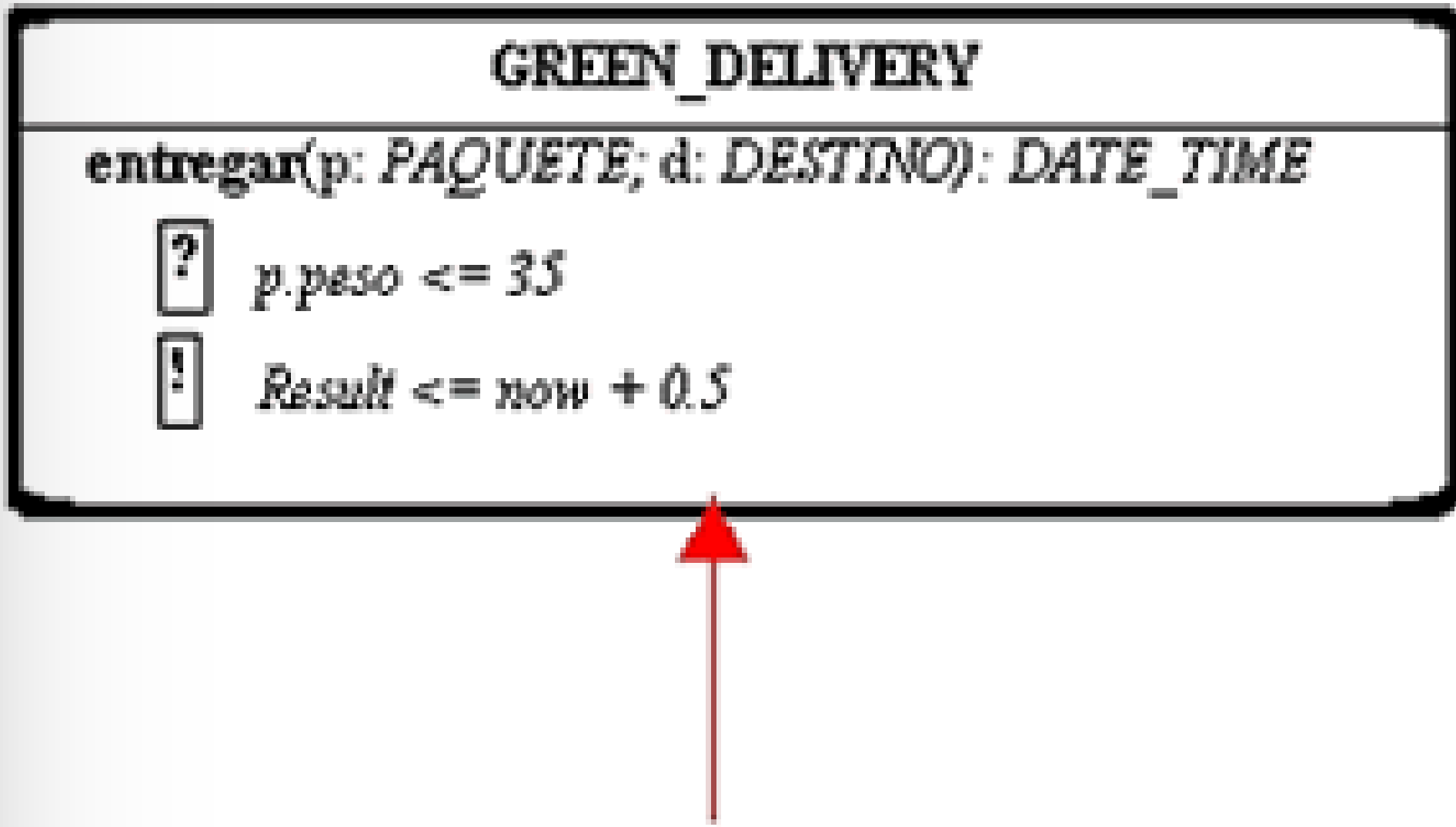
La herencia en los contratos

- En tiempo de ejecución es posible hacer esa versión redefinida del contrato de la rutina.
- Se debe respetar el contrato.
- También se respeta la tipificación.

La herencia en los contratos

- La nueva rutina debe respetar el contrato.
- En el siguiente ejemplo vamos a suponer que la precondición usa el simbolo ?
- La postcondición usa el simbolo !

La herencia en los contratos



La herencia en los contratos

- En el caso de poner un limite de menor a 10 Kg en el ejemplo se incurriria en error.
- Pero si ponemos el limite en 50 Kg no habría ningún error.
- Esto es debido a que la precondición es débil.

La herencia en los contratos

- Por lo tanto podemos decir que hay una regla para definir postcondiciones para una rutina.
- La nueva rutina debe matener o debilitar la precondición de la rutina original.
- El uso de la disyunción lógica implementa este debilitamiento de la precondición de la rutina original y la nueva precondición de la rutina redeclarada.

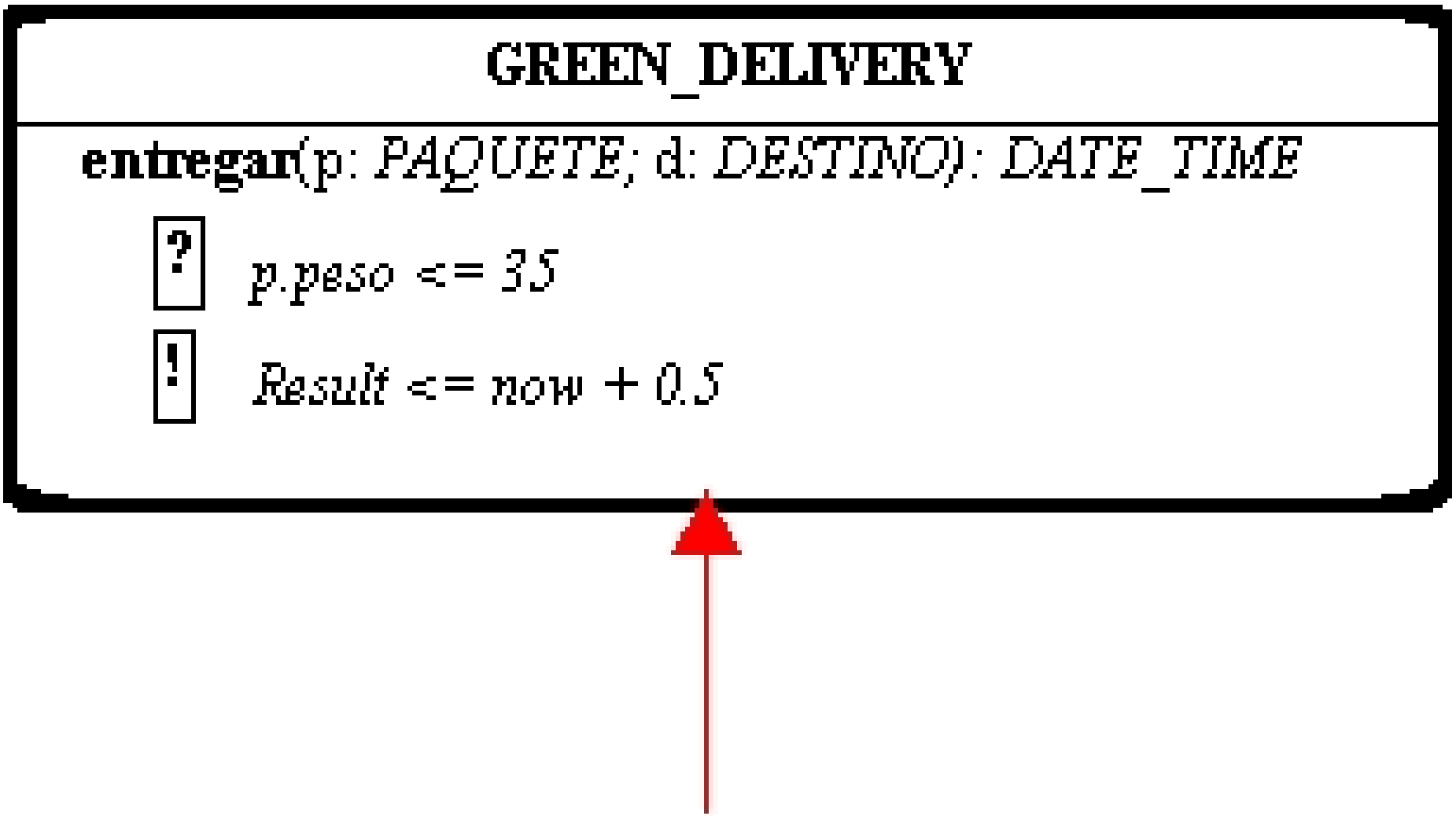
La herencia en los contratos

- Para la postcondición se define de forma diferente.
- No debe exigirse mas del contrato establecido.
- Pero puede hacerse un servicio mayor.

La herencia en los contratos

- Esta situación nos lleva a la siguiente regla de las postcondiciones.
- La nueva rutina debe mantener la nueva postcondición o hacerla mas fuerte.
- Podemos ver un ejemplo a continuación.

La herencia en los contratos



La herencia en los contratos

- Podrá existir una precondición fuerte?
- Podrá existir una postcondición débil?
- La semántica del lenguaje puede impedir esto.

La herencia en los contratos

- La herencia permite la construcción de clases abstractas con fuertes restricciones semánticas.
- Esto es aplicable a sus subclases.
- Puede existir mecanismos para garantizar las poscondiciones.

La herencia en los contratos

- La relación de precondiciones y postcondiciones pueden llegar a ser extremas.
- Si la precondicione es muy relajada.
- La postcondición puede resultar dificil o imposible de realizar.

La herencia en los contratos

- Casos excepcionales pueden utilizar precondiciones abstractas.
- Esto quiere decir sin especificar completamente la aserción.

Algunas recomendaciones

- Separar consultas de comandos.
- Separar consultas básicas de consultas derivadas.
- Para cada consulta derivada escribir un poscondición especificando su resultado en términos de una o más consultas básicas.

Algunas recomendaciones

- Para cada comando escribir una precondición que especifique el valor de cada consulta básica.
- Para toda consulta o comando decidir una precondición adecuada.
- Escribir el invariante para definir propiedades de los objetos.

Separar consultas de comandos

- Las rutinas de una clase pueden ser comandos o consultas pero no ambas cosas a la vez.
- Los comandos alteran el estado interno del objeto.
- Las consultas devuelven un valor.

Separar consultas básicas de consultas derivadas.

- La intención es conseguir un conjunto de especificación formado por consultadas llamadas básicas.
- Las llamamos asi por que de ellas derivaremos otras consultas.
- Las cuales llamaremos consultas derivadas.

Para cada consulta derivada escribir un poscondición especificando su resultado en términos de una o más consultas básicas.

- Nos da la capacidad de conocer el valor de la consulta derivada.
- Una vez conocida el valor de las consultas básicas.

Para cada comando escribir una precondición que especifique el valor de cada consulta básica.

- Debido a que se conocen los resultados de todas las consultas atravez de la visualización de consultas básicas.
- Con esto se garantizan los efectos visibles de cada comando.

Para toda consulta o comando decidir una precondición adecuada.

- Facil de explicar.
- Define claramente el contrato de cada rutina.

Escribir el invariante para definir propiedades de los objetos.

- Es necesario ayudar al lector a construir un modelo conceptual apropiado.
- Esto es importante para la clase.

Mecanismos de excepciones

- Existen casos en los que las técnicas de contratos pueden fallar.
- Pueden fallar por causa del sistema operativo.
- Esto debe ser independiente de la precondición.

Mecanismos de excepciones

- Debe existir tolerancia a los fallos del software.
- Para esto existen las técnicas basadas en excepciones.
- Hay que recordar lo importante en el caso de ruptura de un contrato.

Mecanismos de excepciones

- El equilibrio debe darse entre la corrección y la claridad.
- La corrección lo entendemos como la comprobación de todos los errores.
- La claridad lo entendemos por no desordenar todo el código en su flujo con las excesivas comprobaciones.

Mecanismos de excepciones

- Hay que recordar que la excepción es un suceso no deseado o inesperado.
- La necesidad de separar el correcto funcionamiento de la situación de error.
- El manejo de excepciones nos permite un manejo elegante de estas situaciones.

Mecanismos de excepciones

- La fuente de la excepción puede ser la violación de un contrato.
- Llamar a una rutina sin cumplir la precondición.
- Terminar la rutina sin cumplir la postcondición o el invariante.

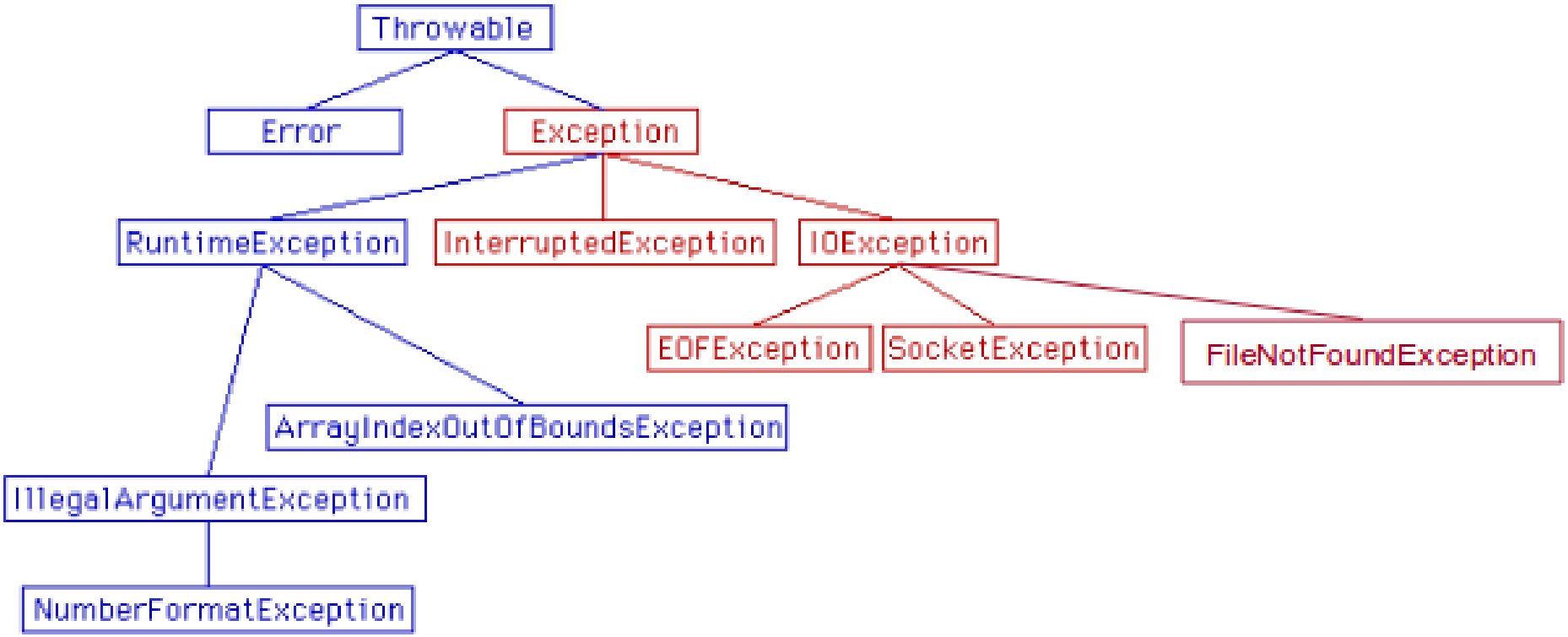
Mecanismos de excepciones

- El origen de la exceptción puede ser por una condición anormal.
- Situaciones anormales de hardware.
- Situaciones anormales del sistema operativo.

Mecanismos de excepciones

- Cuando una rutina falla se llama a una excepción.
- Cuando una excepción ocurre durante la ejecución de una rutina y no se puede recuperar.
- Existe una llamada fracasada.

Mecanismos de excepciones



Mecanismos de excepciones

- Java separa el codigo de trabajo del código de error.
- Permite la propagación de errores.
- Esto quiere decir que en caso de no poder manejar un error se puede lanzar una excepción y tratarlo el método que lo llamo.

Mecanismos de excepciones

- Registro de situaciones excepcionales.
- Esto ocurre de manera anticipada.
- El compilador tiene que estar seguro que se trate.

Preguntas