

TEMA 4

**Corrección y Robustez**  
(2ª parte)

Facultad de Informática  
Universidad de Murcia

**Contenido**

1. Introducción
2. Comparativa de asertos
3. Comparativa de mecanismo de excepciones

# 1.- Introducción

- **Corrección:**
  - Capacidad de los sistemas software de ajustarse a la especificación.
  - Los **asertos** establecen las condiciones que se deben cumplir.
- **Robustez:**
  - Capacidad de los sistemas software de reaccionar ante circunstancias inesperadas.
  - El **mecanismo de excepciones** proporciona un mecanismo para manejar estas situaciones excepcionales durante la ejecución de un programa.

# 2.- Aserciones en Eiffel

- Una aserción es una expresión que establece una propiedad que debe satisfacer alguna de las entidades de un programa, en algún punto de la ejecución del software.
- El código define “**el cómo**”, las aserciones “**el qué**”
- Tipos de asertos:
  - Precondición
  - Postcondición
  - Invariante

# Lenguaje de Aserciones

- El lenguaje de aserciones de Eiffel es **muy simple**:

- Expresiones booleanas con unas pocas extensiones

(old, ‘;’)

- Una aserción puede incluir funciones.

```
Positivo: n>0;
autor = not void
not vacia; conta = old conta + 1
saldo = old saldo - cantidad
```

# Ejemplo: Pre y Postcondiciones

```
put (elemento: T; key: STRING) is
  -- Insertar el elemento con clave key
  require                -- precondición
    not_full: count < capacity
    not key.empty
  do
    ... "algoritmo de inserción"

  ensure                -- postcondiciones
    count <= capacity;
    item (key) = elemento;
    count = old count + 1;
end - put
```

## Ejemplo: Invariante de clase

```
class STACK [G]
feature
  capacity: Integer;          -- tamaño de la pila
  count: Integer;            -- número de elementos

feature {None}
  representation: Array[G]
  ...
invariant                    -- invariante
  0<=count; count <=capacity;
  capacity = representation.capacity;
  (count>0) implies representation.item(count)=item
end
```

## Invariante de representación Lista activa

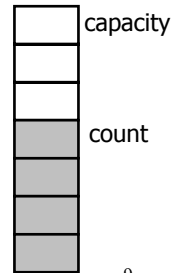
- Importante basarse en aserciones para expresar las propiedades precisas de un diseño.

```
0<=index; index <=count    --count=n° de elementos
before = (index=0);
after = (index=count+1);
is_first = ((not empty) and (index=1));
is_last = ((not empty) and (index=count));
empty implies (before or after);
not (before and after)
```

- Invariante es la mejor manera de comprender una clase

# Eiffel y el Diseño por Contrato

```
class Stack feature
  ...
  remove is
    --quita el elemento de la cima
    require
      no_vacio: not empty  --es decir count>0
    do
      count:=count-1
    ensure
      no_llena: not full
      uno_menos: count = old count -1
    end
end
```



Corrección y Robustez

9

# Herencia y Aserciones

## A) Invariante de clase

**A** {INV<sub>A</sub>}



**B** {INV<sub>B</sub> = INV<sub>A</sub> and  
"restricciones para B"}

"Siempre que espero una instancia de la clase A puedo recibir un objeto de la clase B"

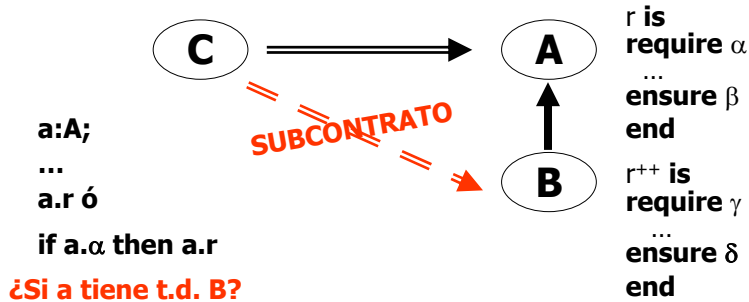
## B) PRE Y POST Condiciones. Subcontrato

La redefinición debe cambiar la implementación de una rutina, no su semántica

Corrección y Robustez

10

### B) PRE Y POST Condiciones. Subcontrato



La nueva versión ( $r^{++}$ ):

- **acepta todas** las llamadas que aceptaba la original (pre igual o más débil)
- garantiza **al menos** lo que garantizaba la otra (post igual o más fuerte)

## assert de Java $\neq$ Diseño por Contrato

- En DxC existen distintos tipos de asertos.
  - en Java todos son `assert`
- Los asertos en Eiffel forman parte de la interfaz de los métodos
  - El `javadoc` no genera la información relativa a los asertos
- El DxC especifica el mecanismo para heredar los asertos en las subclases.
  - en Java los asertos son independientes de la herencia.

## Asertos en C++

- Ofrece la macro `assert()` en `<assert.h>`
- `assert()` evalúa su parámetro y llama a `abort()` si el resultado es cero (`false`).
- **Ejemplo:**

```
void f(int *p){  
    assert(p != 0); //si p==0 abort()  
    //cuerpo de la función  
}
```
- Si se define la macro `NDEBUG`, se desactiva la comprobación de todos los asertos definidos.
- Antes de abortar `assert` escribe en la salida el nombre del archivo fuente y la línea donde aparece.
- Ayuda útil para la depuración.

Corrección y Robustez

13

## assert de C++ para pre y postcondiciones

```
funcion (parametros)  
{  
    assert (precondicion);  
  
    // Cuerpo de la función  
  
    assert (postcondicion);  
}
```

- **Problema:** se aborta la ejecución del programa siempre que se produce un fallo en la precondición o postcondición.

Corrección y Robustez

14

## Alternativa a assert en C++

```
class FalloPrecondicion {};  
class FalloPostcondicion {};  
  
void precondition (bool condicion)  
{  
    #ifndef NDEBUG  
        if (!condicion)  
            throw FalloPrecondicion();  
    #endif  
}  
void postcondicion (bool condicion)  
{  
    #ifndef NDEBUG  
        if (!condicion)  
            throw FalloPostcondicion();  
    #endif  
}
```

Corrección y Robustez ©apuntes de AED Ginés García Mqteos 15

## Asertos en C#

- **Ejemplo 1:**

```
// Create an index for an array.  
protected int index;  
protected void Method() {  
    // Perform some action that sets the index.  
    // Test that the index value is valid.  
    Trace.Assert(index > -1);  
}
```

- **Ejemplo 2:**

```
public static void MyMethod(Type type, Type baseType){  
    Trace.Assert(type!=null, "Type parameter is null");  
    // Perform some processing.  
}
```



### 3.- Tratamiento de excepciones en Eiffel

- Un **fracaso** de una **rutina** (= termina sin cumplir el contrato) causa una **excepción** en quien la llama.
- Una **excepción** es un suceso en tiempo de ejecución que puede causar que una rutina fracase.
- Una **llamada fracasa** **sii** ocurre una excepción durante la ejecución de la rutina y **no se puede recuperar**
- Se debe prevenir el fracaso **capturando** la excepción y tratando de restaurar un estado a partir del cual el cómputo pueda proseguir

### Casos de excepción en Eiffel

Una excepción puede ocurrir durante la ejecución de una rutina como consecuencia de alguna de las situaciones siguiente:

- Intentar hacer la llamada `a . f` siendo `a=void`
- Intentar hacer `x:=y` siendo `x` expandido e `y=void`
- Ejecutar una operación que produce una condición anormal detectada por el hardware o el sistema operativo
- Llamada a una rutina que fracasa
- No cumplimiento de los asertos (`pre`, `post`, `inv`)
- Ejecutar una instrucción que pida explícitamente que se eleve una excepción

## Tratamiento de excepciones en Eiffel

- Hay sólo **dos respuestas** legítimas a una excepción que ocurra durante la ejecución de una rutina:
  - Reintento: hacer que se cumpla el INV y PRE y volver a ejecutar
  - Fracaso o pánico organizado: restaurar INV e informar del error a la rutina que hizo la llamada.

```
rutina is
  require      --precondición
  local       --definición de variables locales
  do          -- cuerpo de la rutina
  ensure      -- postcondición
  rescue
    -- clausula de rescate
    [retry]
end
```

Corrección y Robustez

19

## Excepciones en Eiffel

- ¿Podemos saber cuál fue la última excepción?
- La clase que necesite un ajuste más fino debe heredar de la biblioteca **EXCEPTIONS**.
- Cada excepción es un número entero.
- Una cláusula de rescate puede tratar distintas clases de excepciones. Ej. en una clase que herede de EXCEPTIONS

```
rescue
  if is_assertion_violation then
    "Procesar la violación de la aserción"
  else if is_signal then
    "Procesar el caso de la señal"
  else
    ...
  end
```

- El programador puede lanzar excepciones:

```
trigger(code:INTEGER; message:STRING)
```

Corrección y Robustez

20

## Lanzamiento de excepciones Eiffel vs. Java. Ejemplo Eiffel.

```
leerEnteroTeclado:INTEGER is
LOCAL
    fallos:INTEGER;
DO
    Result:=getInt;
RESCUE
    fallos=fallos+1;
    IF (fallo<5) THEN
        message("Un entero!")
    RETRY
END ELSE informar a quien lo invocó
END
```

Ejecución normal

Tratamiento de situación anormal

ELSE informar a quien lo invocó

Corrección y Robustez

21

## Lanzamiento de excepciones Eiffel vs. Java. Ejemplo Java.

```
public int LeerEnteroTeclado() throws IOException {
    boolean fin= false;
    int fallos;
    int enteroLeido=-1;
    for (fallos=0; ((fallos<=5) && (fin==false));fallos++){
        try{
            enteroLeido=getInt();
            fin = true;
        } catch (NumberFormatException e){
            System.out.println("Un entero!!");
        }
    }
    if (!fin) throw (new IOException());
    else return enteroLeido;
}
```

RETRY

Corrección y Robustez

22

## Excepciones en C++

- Una excepción es un objeto de una clase que representa un suceso excepcional
  - Puede ser útil definir una jerarquía de excepciones

```
class ErrorMat{};
class Desbordamiento_superior : ErrorMat{};
class Desbordamiento_inferior : ErrorMat{};
class Division_cero : public ErrorMat{};
```

- En realidad se puede lanzar “cualquier cosa” (un entero, una cadena de texto,...)

## Lanzar una excepción en C++

```
double divide (int a, int b)
{
    if (b==0)
        throw Division_cero();
    return double(a)/double(b);
}
```

## Manejo de excepciones en C++

```
void f()
{
    try{
        //código que puede lanzar una excepción
        // matemática
    }
    catch (Division_cero){
        //tratar el error de la división por cero
    }
    catch (ErrorMat){
        //tratar cualquier error matemático distinto
    }
    catch (...){
        //tratar el resto de excepciones
    }
}
```

**El orden es importante!!**

## Manejo de excepciones C++

```
class ErrorMat{
    //...
    virtual void imprime_depuracion() const{
        cerr<<"Error Matemático";}
};

void f()
{
    try{
        //código que puede lanzar un error matemático
    }catch (ErrorMat &e){
        e.imprime_depuracion();
        throw; //relanza la excepción que ha ocurrido
    }
}
```

## Especificación de excepciones en C++

- Especificar el conjunto de excepciones que puede lanzar
  - `void f(int a) throw (X2, X3);`
  - f puede lanzar sólo X2 y X3
- Si no se dice nada puede lanzar cualquier excepción
  - `int f();`
- Puede indicar que no lanzará ninguna excepción
  - `int f() throw();`
  - Lista vacía
- El compilador ignora la especificación de las excepciones.
- Un método redefinido no puede lanzar más excepciones que las especificadas en el método de la clase base.

## Excepciones en C#

- Subclase de **System.Exception**
  - No admite excepciones comprobadas como en Java
  - El método no tiene que declarar las excepciones que lanza
- Se pueden crear nuevos tipos de excepciones:
  - lo normal es que no añadan atributos ni métodos
  - se hace para diferenciar una excepciones de otras
  - Por convenio el nombre termina en **Exception**
- Lanzar excepción:
  - igual que Java: `throw new NombreException();`
  - igual que C++: `throw;` para relanzar una excepción

## Manejo de excepciones en C#

```
try{
    <instrucciones>
}
catch (<excepción1>){
    <tratamiento1>
}
catch (<excepción2>){
    <tratamiento2>
}
...
catch (...){
    <trata cualquier excepción>
}
finally{
    <instruccionesFinally>
}
```

```
<excepcion> = (TipoExcepcion) |
(TipoExcepcion nombreVariable)
```

## Ejemplo excepciones en C#

```
static public void G()
{
    try
    {
        int c = 0;
        int d = 2/c;
    }
    catch (DivideByZeroException)
    { d=0; }
}
```