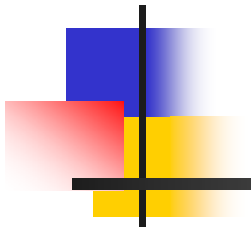


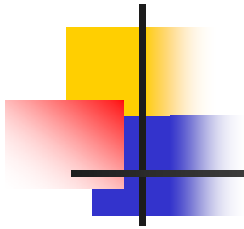
Tema 4: Corrección y Robustez en C++

Programación Orientada a Objetos

Curso 2008/2009

Begoña Moros Valle

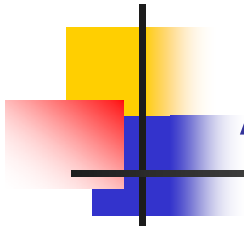
DIS
Departamento de Informática y Sistemas



Contenido

- Asertos en C++

- Mecanismo de excepciones:
  - Declaración de excepciones
  - Lanzar excepciones
  - Manejo de excepciones
  - Definición de excepciones
  - Excepciones de la librería estándar



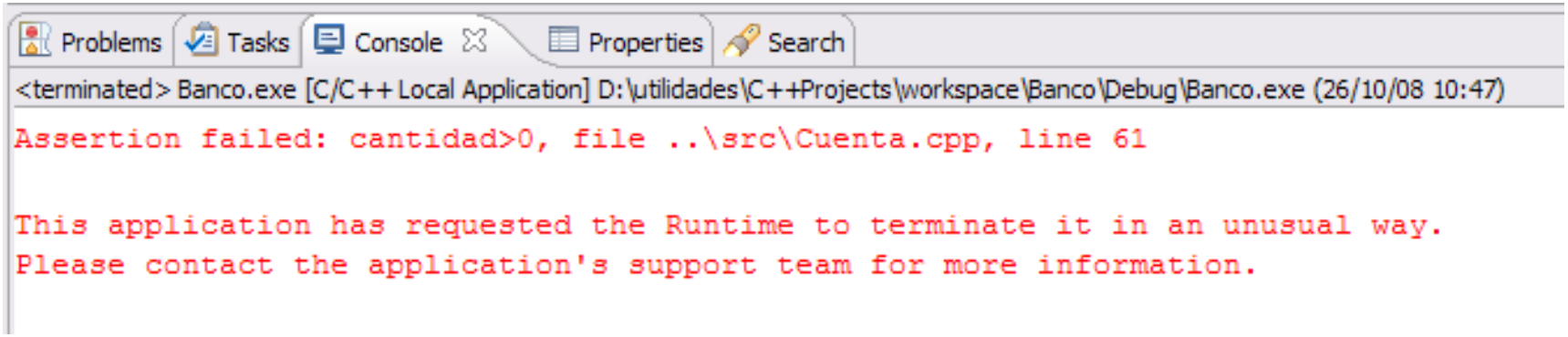
Asertos en C++

- Es posible definir puntos de chequeo en el código utilizando los asertos
  - Ayuda útil para la depuración
- Ofrece la macro `assert()` en `<assert.h>`
- `assert()` evalúa su parámetro y llama a `abort()` si el resultado es cero (`false`).

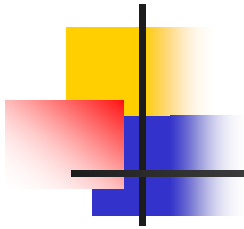
```
void Cuenta::ingreso(double cantidad){
  assert(cantidad>0);
  assert(estado == OPERATIVA);
  saldo = saldo + cantidad;
}
```

Asertos en C++

- Antes de abortar `assert` escribe en la salida de error el nombre del archivo fuente y la línea donde se produjo el error.

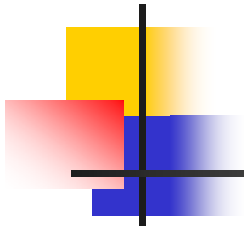


- Si se define la macro `NDEBUG`, se desactiva la comprobación de todos los asertos definidos.



Limitaciones asertos

- Aunque los asertos se pueden utilizar para controlar la corrección del código (precondiciones, postcondiciones, invariantes) tienen las mismas limitaciones que los `assert` de Java.
  - Las precondiciones no se deben evaluar con `assert`
  - No se debe abortar un programa como consecuencia de un fallo en la postcondición
- Para ayudar a la verificación de los programas C++ también existen entornos para la definición de pruebas unitarias
  - **`CPPUnit`** es uno de los entornos más utilizado



Excepciones en C++

- En C++ se utiliza el mecanismo de excepciones para notificar que se ha producido una situación excepcional.

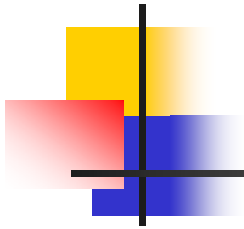
- A diferencia de Java, una excepción no tiene por qué ser un objeto de una clase:
  - Se puede lanzar “cualquier cosa” (un entero, una cadena de texto, ...)
  - No existen distintas categorías de excepciones
  - Puede ser útil definir una jerarquía de excepciones, aunque no es obligatorio

Lanzamiento de excepciones

- Se utiliza la palabra reservada **`throw`**
- Podemos lanzar un número o una cadena de texto para informar de un fallo en la precondición

→ Se desaconseja esta práctica en un programa OO

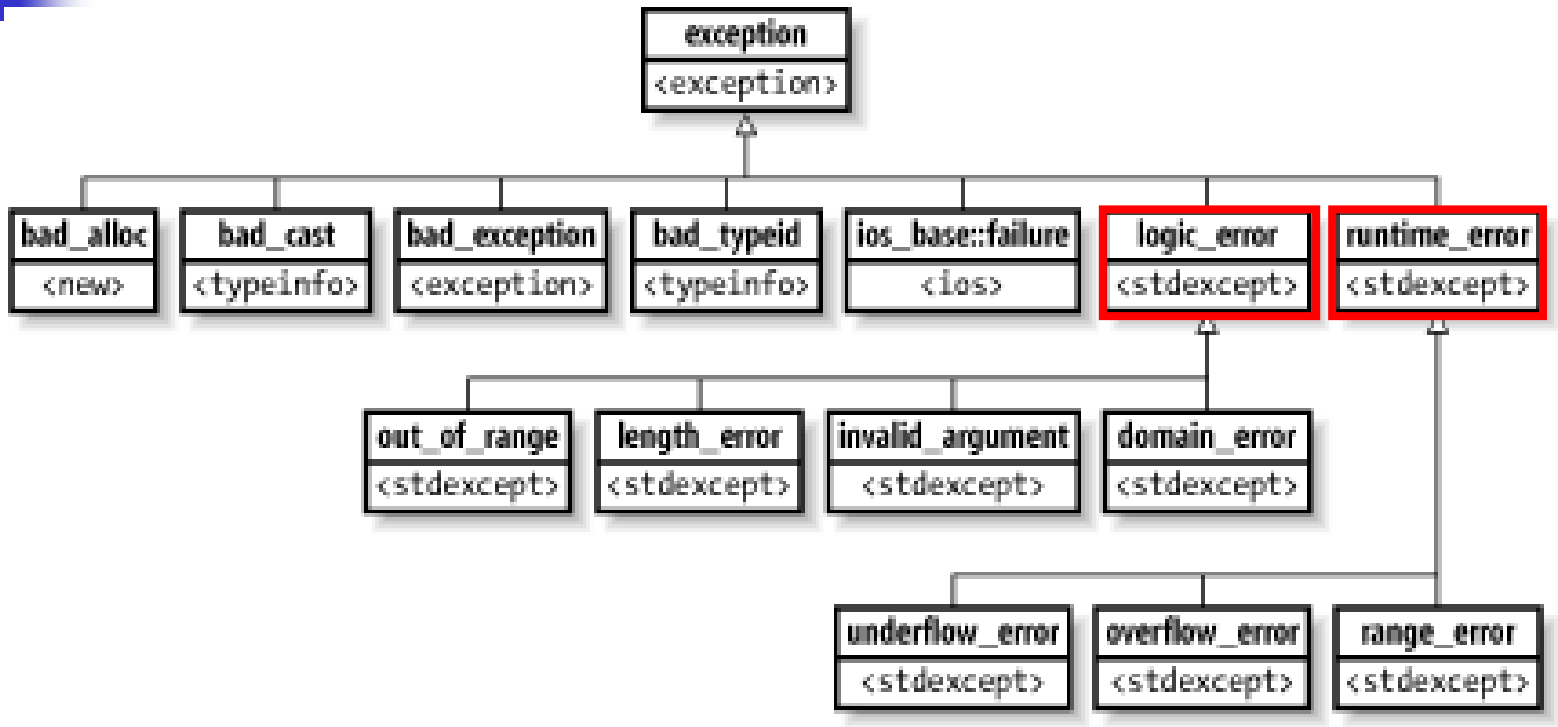
```
void Cuenta::ingreso(double cantidad){
  if (cantidad<0)
       throw cantidad;
  if (estado!= OPERATIVA)
       throw "Fallo pre. Estado incorrecto";
  saldo = saldo + cantidad;
}
```



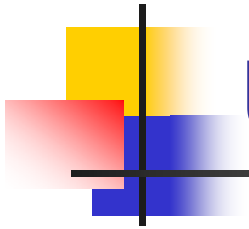
Excepciones en la librería estándar

- Existe un conjunto de excepciones predefinidas en el espacio de nombres `std` (`<stdexcept>`)
- Todas ellas heredan de la clase `std::exception` (`<exception>`)
  - Disponen del método `what()` que devuelve la cadena de texto con el informe del error
- La clase `exception` se puede utilizar como raíz de una jerarquía de excepciones definidas por el programador, aunque no es obligatorio
  - Las nuevas excepciones deben redefinir el método `what` para que lancen el mensaje de error más conveniente.

Excepciones en la librería estándar



- **`logic_error`**: fallo en las precondiciones
- **`runtime_error`**: fallo en las postcondiciones



Uso de excepciones estándar

- Para el **control de precondiciones** se lanzan excepciones compatibles con **`logic_error`**

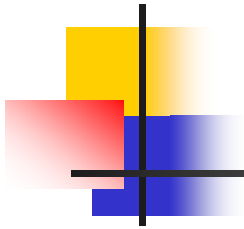
```
void Cuenta::ingreso(double cantidad) {
  if (cantidad < 0)
       throw invalid_argument(“Cantidad negativa");
  if (estado != OPERATIVA)
       throw logic_error("Estado incorrecto");
  saldo = saldo + cantidad;
}
```

Excepciones de usuario

- Cualquier tipo de datos puede ser lanzado en una excepción.
- Se recomienda crear una clase que herede de **`runtime_error`**:

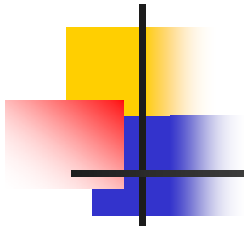
```
class RedNoDisponible: public runtime_error{
public:
 RedNoDisponible(const char* m);
};

RedNoDisponible::RedNoDisponible(const char* msg):
runtime_error(msg) {}
```

- Las excepciones que heredan de `exception` disponen del método `what()` que retorna el mensaje de error.



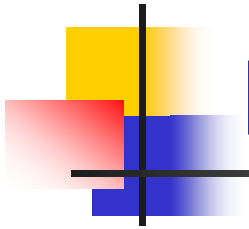
Declaración de excepciones

- Se utiliza también la palabra reservada **`throw`**
- Se puede especificar el conjunto de excepciones que puede lanzar un método
  - `void f(int a) throw (E1, E2);`
  - `f` puede lanzar excepciones de tipo `E1` y `E2` pero no otras
- Si no se dice nada, significa que podría lanzar cualquier excepción (o ninguna)
  - `int f();`
  - `f` podría lanzar cualquier excepción
- Se puede indicar que un método no lanzará ninguna excepción
  - `int f() throw();`
  - `f` no puede lanzar ninguna excepción (lista vacía)



Declaración de excepciones

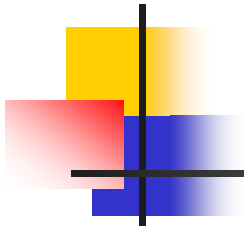
- A diferencia de Java, **el compilador ignora la declaración de las excepciones**
  - Si en tiempo de ejecución se intenta lanzar una excepción no declarada, se detecta la violación y termina la ejecución.
  - Se aplica la política “confía en el programador”:
    - El compilador no obliga al código cliente a manejar las excepciones que puede lanzar un método
    - Si ocurre una excepción y el programador no ha definido como manejarla, la excepción escaparía del método
- Un método redefinido no puede lanzar más excepciones que las especificadas en el método de la clase padre.
  - La concordancia entre las especificaciones de la clase padre e hija si es controlada por el compilador



Declaración de las excepciones

```
void Cuenta::ingreso(double cantidad) throw (invalid_argument,
logic_error){
  if (cantidad<0)
       throw invalid_argument("cantidad negativa");
  if (estado!= OPERATIVA)
       throw logic_error("Estado incorrecto");
  saldo = saldo + cantidad;
}
```

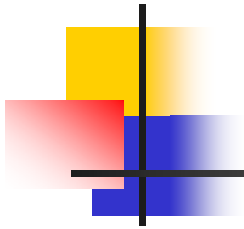
- Declaramos que el método ingreso SÓLO puede lanzar las excepciones `invalid_argument` y `logic_error`
- Si en el cuerpo del método se produjese alguna otra excepción sería incompatible con la declaración.



Manejo de excepciones

- Como en Java, se debe definir un manejador (`catch`) por cada excepción que se espera que pueda lanzar la ejecución de un bloque de código (`try`).
  - A diferencia de Java, **no es obligatorio**, el compilador no comprueba si esto se hace o no

- Las excepciones se pueden pasar al manejador por valor o por referencia
  - Cuando las excepciones son objetos **se deben pasar por referencia para asegurar el comportamiento polimórfico**



Manejo de excepciones

- Igual que en Java, cuando ocurre una excepción se evalúan los tipos definidos en los manejadores y se ejecuta el que sea compatible
  - Hay que tener en cuenta el orden en el caso de utilizar una jerarquía de excepciones
- Se puede definir un manejador para cualquier tipo de excepción
  - `catch(…)`
- Es posible relanzar la misma excepción que se está manejando
  - `throw;`

Método visualizar del navegador web

```
void Navegador::visualiza(string url){
  Conexion* conexion;
  int intentos = 0;
  while (intentos < 20) {
    try {
        conexion = new Conexion(url);
         break;
    } catch (RedNoDisponible& e) {
        intentos++;
        if (intentos == 20) throw; //relanza
    }
  }
  //Se ha abierto la conexión y se leen las líneas …
}
```

- Si al crear la conexión ocurren las excepciones `ServidorNoEncontrado` o `RecursoNoDisponible` (no manejadas) pasarán al cliente

Código cliente

```
int main(){
  Navegador navegador;
  try{
     navegador.visualiza(“http://www.poo.c++");
  }catch(ErrorConexion& e){
     cout<<"Fin del programa. "<<e.what()<<endl;
  } //maneja cualquiera de las subclases
  catch(...){  //maneja cualquier otra excepción
     cout<<"Fin por excepción no prevista"<<endl;
  }
}
```