Tema 5

Tratamiento de errores con excepciones

UPSAM 2010-2011

Contenido

1. Concepto de Excepción 3
2. Manejo de excepciones 3
3. Tratamiento de Excepciones 4
4. Lanzamiento de Excepciones 5
5. Manejadores de excepciones 6
6. Declaración de Excepciones 8

1. Concepto de Excepción

Las ***Excepciones*** **son anomalías** (condiciones de error no esperadas) producidas durante la ejecución de un programa. Por ejemplo división por cero, punteros a zonas de memoria ilegales, punteros no inicializados correctamente, acceso a una zona de memoria no permitida, índices de arrays fuera de rango, errores de E/S,etc.

Normalmente las excepciones suspenden la ejecución del programa, emitiendo en el sistema el mensaje de error correspondiente. Los lenguajes orientado a objetos como C++ y Java tienen construcciones específicas para tratar con excepciones. Cuando un programa detecta una excepción, se notifica al resto del sistema **elevando** (raising) o **lanzando** (throwing) un excepción. En alguna parte existe un código especifico que maneja y **captura** (catch) la excepción.

De la misma forma que el concepto de Clase ha cambiado la forma de organizar y programar, las excepciones han cambiado la forma de manejar las condiciones de error en el flujo de control de los programas.

2. Manejo de excepciones.

Muchos lenguajes de programación como C, incorporan un tratamiento de errores basado en el valor de retorno de las funciones. Así el valor devuelto por las funciones se utiliza para determinar si una función se ha ejecutado satisfactoriamente (Por ejemplo si devuelve -1 indica error y en otro caso es interpretado como ejecución correcta).

Ejemplo. Fragmento de código y tratamiento de errores tradicional.

```
int codError=0;

codError=leerArchivo(archivo);

if (codigoError!=0){

switch(codigoError)

}

case -1: //No se encontró el archivo

             //…

            break;

case -2: //El archivo está corrupto

            break;

case -3://El dispositivo no está listo

break;

default://…
```

```
}

}

else{

//procesar los datos leídos

}
```

El problema de esta forma de tratamiento es que el código de tratamiento está mezclado junto con el flujo normal. El tratamiento de errores con excepciones es un método que clasifica los tipos de errores y estructura el código de una manera más clara de cara a realizar este tratamiento.

3. Tratamiento de Excepciones

El modelo de excepciones en la mayoría de los lenguajes de programación orientados a objetos se basa en los siguientes mecanismos: **lanzar (throw) y capturar (catch):**

1. Cuando **se detecta una excepción**, el **programa lanza (throw) una excepción**. El lanzar una excepción hace que el programa realice un salto antes de continuar la ejecución.
2. La **excepción se captura (catch)** mediante un **manejador de excepciones (handler)**. Los manejadores de excepciones comienzan con la palabra **catch** y se declaran al final de un bloque **try (en C++ y Java)**.

En resumen un código bajo este mecanismo puede lanzar una excepción directamente o indirectamente, en un bloque **try** utilizando la expresión **throw.** La excepción se maneja invocando un manejador adecuado seleccionado de una lista de manejadores encontrados inmediatamente después del bloque **try .** De forma general un bloque de código con tratamiento de excepciones tiene la siguiente forma general.

```
try{

//El código de este bloque puede producir alguna excepción

}

catch( TipoExcepción1 e){

//Código de tratamiento de la excepción

}…

catch (TipoExcepcionN e){

}
```

La estructura en C++ y Java exige que cualquier excepción que ocurra, debe ser lanzada dentro de un bloque **try**. **El bloque try es una sentencia que especifica el conjunto de sentencias que al ejecutarse podrían lanzar una excepción y si es así que manejadores tratarían las excepciones.**

El manejador de excepciones es el bloque de código diseñado para manejar una condición anómala y excepcional. Debería haber al menos un manejador por cada tipo de excepción que se pueda lanzar.

El manejo de excepciones como hemos dicho antes ofrece una **forma de separar explícitamente el código** que trata las condiciones de error y el código de control de flujo normal de la aplicación haciéndola más legible. El siguiente fragmento de código muestra el tratamiento de los errores del ejemplo anterior con el mecanismo de excepciones.

```
try{
  //Código de acceso y tratamiento de archivo
  leerArchivo(archivo);
} catch(ArchivoDesconocido e){
  //Tratamiento si el archivo es desconocido
}catch(DispositivoError e){
  //Tratamiento si el dispositivo da algún tipo de error
}catch(ArchivoCorrupto e){
  //Tratamiento si el archivo esta dañado
}
```

4. Lanzar una Excepción (throw)

Las excepciones se lanzan utilizando la sentencia **throw**

```
throw tipoexcepcion;
```

Donde *tipoexcepción* es una expresión que determina el tipo de la excepción.Este tipo es cualquier tipo válido en el lenguaje .Este objeto se utiliza para inicializar la variable del manejador de expresiones *(clausula catch)*.

Ejemplo:

```
// exceptions_trycatchandthrowstatements.cpp
// compile with: /EHsc
#include <iostream>
using namespace std;
int main() {
   char *buf;
   try {
      buf = new char[512];
      if( buf == 0 )
```

```
         throw "Memory allocation failure!";

   }

   catch( char * str ) {

      cout << "Exception raised: " << str << '\n';

   }

}
```

El operando **throw** es sintácticamente similar a una sentencia **return**

En esencia los pasos que se siguen son los siguientes:

1. Se lanza la excepción
2. Se busca un manejador adecuado en función de su orden de aparición y se comprueba su tipo (cada manejador tiene un tipo diferente)
3. Si el manejador se encuentra, el control del programa se transfiere al manejador
4. Si no se encuentra ningún manejador, se elevará esta al bloque try superior si lo hubiere si no el programa terminará la ejecución.

5. Manejadores de excepciones

Las excepciones lanzadas dentro de un bloque son capturadas usando clausulas **catch asociadas** a un bloque try.

5.1. Bloques try-catch

Un bloque try -catch tiene la siguiente sintaxis:

```
try{

//Sentencias

}catch(Exceppcion1 e){}

catch(Exceppcion2 e){}

catch(ExceppcionN e){}
```

Un **bloque try es un contexto para indicar cual son la instrucciones a considerar y los manejadores que se invocarían** si se produjesen determinadas anomalías en esas instrucciones.

El **orden** en el que se especifican los manejadores determina el orden en que se prueban los mismos cuando se lanza una excepción .Siempre existe un bloque try junto a un bloque de manejadores **catch.**

La **clausula catch** es muy **similar a una declaración de función** de un argumento sin tipo de retorno.

Una **excepción se captura cuando su tipo coincide con el tipo de la sentencia *catch***. Una vez que existe correspondencia de tipos, el control del programa se transfiere al manejador. El manejador especifica que acciones deben realizarse para tratar la anomalía del programa.

Ejemplo en C++

```
class desbordamiento{

public:

desbordamiento();

};

float división(float a ,float b)

{

  if (b==0) throw "division por 0";

  if (a>LIMITE1 && b<LIMITE2) throw  new desbordamiento();

 return a/b;

}

void main()

{

Try{

division(5,3);

}catch( char *e){

Cout<<e<<endl;

}catch(desbordamiento *e){

}

}
```

Ejemplo 2

```
//Recibe cierto valor numérico

try{

if (n % 2 == 1){
// n es impar
     throw new ArithmeticException();
}else
  // Código normal para valores pares
}catch (ArithmeticException e){

// Tratar el caso especial de valores impares
}
```

6. Declaración de Excepciones

Sintácticamente una especificación de excepciones es parte de la declaración de funciones y tiene el siguiente formato.

tipo_ retorno **nombre_funcion**(*parámetros*) **throw** *lista tipos de excepciones*

Consideraciones:

- La lista de excepciones es la lista de tipos que una expresión **throw** puede tener en la función.
- Cualquier función o bloque de código que invoque esta función **deberá obligatoriamente capturar** estos tipos de excepciones.

Ejemplo1 .C++

```
void escrituraDisco throw (DeviceError,WritError);

…

void enviarMail throw (direcc_incorrecta,fallo_sistema);
```

Ejemplo 2.Java

```
public  escribirArchivo( String nombre_fichero) throws IOException
```

```
public class Conexion{

public Conexion(Stringurl) throws RedNoDisponible, ServidorNoEncontrado,
RecursoNoDisponible
{ …
}

publicStringleerLinea() throws RedNoDisponible
{
 …}

public void cerrar()
{ …}
}
```