

Exception Handling

Overview

- | Clients And Servers
- | Raising an Exception
- | Client Block
- | Fishing Net
- | Throw Specification

Client-Server Programming

- | Concept of client (consumer) and server (supplier)
- | Client sends requests to server
- | Server carries out requests (action or a result)
- | Contract defined between parties

3

What is a Contract?

- | Statement of how client should behave
- | Each party has its rights and responsibilities
- | Exceptional situation arises if contract is broken

4

What is an Exception?

- | An error situation in a program
- | Action must be taken when situation arises
- | Server knows when the exception occurs
- | Client knows what to do with exception

5

Implementing Contracts in C++

- | Request corresponds to server member function
- | Server checks if client supplies good values etc.
- | If input incorrect ==> exception object is born and thrown
- | Client must catch the exception in order to recover

6

Examples of Exceptions

- | Bad user input (range error, invalid input, ...)
- | Numerical errors (overflow, underflow, division, ...)
- | Storage errors (file access, ...)
- | Process errors (process killed, stale, ...)
- | Domain-specific errors (e.g. BadAccount)
- | Free store errors (memory allocation)

7

Syntax and Keywords

- | Keywords: try, throw and catch
- | Client defines a try block
- | Server throws an exception object
- | Exception handling performed in client catch block

8

Raising an Exception

- | Use throw
- | Throw can accept different arguments
 - | `throw -1;` // For handler with int type
 - | `throw "help";` // For handler with char*
 - | `throw OVERFLOW();` // For handler with OVERFLOW object
- | Create a client block with try and catch
- | Client block calls the server

9

Example Server Function

```
void Account::Withdraw(double amount)
{
    // Precondition: There is enough balance
    if (amount > m_balance) throw -1; // NoFunds error code

    // Postcondition: OK now
    m_balance -= amount;
}
```

10

Client Side

- | The client needs to identify a block
- | Inside the block functions are called that might throw exceptions
- | Block identified using the try keyword

11

Try Block

```
try
{
    Account acnt(1234, 500.0); // Balance 500.0

    acnt.Withdraw(250.00);
    acnt.Withdraw(300.00);    // Exception !!!
}
// HANDLER SECTION AFTER TRY BLOCK
```

12

Handling Errors

- | After try block section is needed to handle errors
- | Catch is used to handle specific type of errors, directly after try block

```
| catch(int) {}           // Catch for int exceptions
| catch(char* err) {}     // Catch for char*
```
- | If no catch match is found default catch is called

```
| catch(...) {}
```

13

Example Catch

```
try
{
    Account acct(1234, 500.0); // Balance 500.0

    acct.Withdraw(250.00);
    acct.Withdraw(300.00);    // Exception !!!
}
catch(int err)  If the value server throws out is int type
{
    if (err == -1) cout << "Not enough funds available" << endl;
}
catch(...)
{
    cout << "An unhandled exception has occurred" << endl;
}
```

14

What happens when an Exception occurs in a try Block?

1. Program searches for a matching handler
2. If handler found ==> unwind stack to that point
3. Program control transferred to handler

15

Special Cases

- | No handler found ==> terminate function called
- | No exceptions thrown ==> program executes in normal fashion

16

Creating an Exception Hierarchy

I Use polymorphism for handler

```
class AccountError
{
public:
    virtual void Handle() = 0;
};

class NoFunds: public AccountError
{
private:
    // NoFunds specific data members
public:
    void Handle();
};
```

17

Server Using Error Objects

```
void Account::Withdraw(double amount)
{
    // Precondition: There is enough balance
    if (amount > m_balance) throw NoFunds; // NoFunds exception

    // Postcondition: OK now
    m_balance -= amount;
}
```

18

Catching Exception Objects

- | Possible to define 'explicit' and 'polymorphic' nets
- | Default net for the uncaught exceptions

19

Explicit Nets

```
try
{
    Account acnt(1234, 500.0); // Balance 500.0
    acnt.Withdraw(250.00);
    acnt.Withdraw(300.00);      // Exception !!!
}
catch(NoFunds& ex)
{
    cout << "Not enough funds available" << endl;
}
catch(...)
{
    cout << "An unhandled exception has occurred" << endl;
}
```

20

Polymorphic Nets

```
try
{
    Account acnt(1234, 500.0); // Balance 500.0
    acnt.Withdraw(250.00);
    acnt.Withdraw(300.00); // Exception !!!
}
catch(AccountError& ex)
{
    cout << "An AccountError has occurred" << endl;
    ex.Handle(); // Polymorphic: NoFund::Handle() called
}
catch(...)
{
    cout << "An unhandled exception has occurred" << endl;
}
```

21

All Together

```
try
{
    Account acnt(1234, 500.0); // Balance 500.0
    acnt.Withdraw(250.00);
    acnt.Withdraw(300.00); // Exception !!!
}
catch(NoFunds& ex) // Looking for specific stuff
{ // Explicit catch block
    cout << "Not enough funds available" << endl;
}
catch(AccountError& ex)
{
    cout << "An AccountError has occurred" << endl;
    ex.Handle();
}
catch(...)
{
    cout << "An unhandled exception has occurred" << endl;
}
```

22

Program Continuation

- | Program is continued after try block, if allowed to continue

```
try
{
    Account acnt(1234, 500.0); // Balance 500.0
    acnt.Withdraw(250.00);
    acnt.Withdraw(300.00);      // Exception !!!
}
catch(NoFunds& ex) {}
catch(AccountError& ex){}
catch(...){}
// Execution continues here
```

23

What is supported in C++

- | Synchronous exceptions only
- | Cause of failure generated from within the program
- | Thus, events such as Ctrl-C are not synchronous exceptions

24

Exceptions in Real Life

- | Concentrate on user logic errors, invalid input etc.
- | It is not necessary to define your exception hierarchies
- | Trade-off between robustness and efficiency
- | Place try/catch block at 'strategic' points only

25