

CS 485

Advanced Object Oriented Design

Enum

Spring 2017

<http://www.netobjectives.com/PatternRepository/index.php?title=PatternsByAlphabet>

<http://www.netobjectives.com/files/books/dpe/design-patterns-matrix.pdf>

Enumerated Data Types are ADTs

- An enumerated data type is a programmer-defined data type

General Format

```
enum TypeName {One or more enumerators};
```

Example

```
enum Day {MON, TUE, WED, THU, FRI, SAT, SUN};  
Day day;  
day = MON;
```

- The enumerators are integer constants the compiler assigns starting with 0 unless otherwise specified

Enumerated Data Types

```
Day day;
```

```
int whatDay, indx;
```

- `day = 3; // illegal`
- `whatDay = TUE; // legal`
- `if (day > WED) // legal`
- `for (indx = MON; indx <= SUN; ++indx) // legal`
- `day = static_cast<Day> (day + 1); // legal`

Enumerated Data Types

```
switch (day)
{
    case MON:      cout << "Monday";
                  break;

    case TUE:      cout << "Tuesday";
                  break;

    ...

}
```

- **Anonymous** Enumerator Data Types

```
enum {FREEZING = 32, BOILING = 212};
```

The logical conclusion...

```
enum Day { MON, TUE, WED, THU, FRI, SAT, SUN };
```

```
int foo (char param);
```

```
int bar (bool bParam);
```

```
Day today = MON;
```

```
if (4.2 > today) // legal
```

```
{
```

```
    std::cout << foo (MON); // legal
```

```
    std::cout << bar (MON); // legal
```

```
}
```

C++11 Scoped Enum

- Goals
 - better type checking
 - less name pollution

```
enum class Day { MON, TUE, WED, THU, FRI, SAT, SUN };

enum UnscopedDay {mon, tue, wed, thur, fri, sat, sun};

int foo (char param);
int bar (bool bParam);

int main ()
{
    Day today = Day::MON;

    if (4.2 > today) // illegal
    {
        std::cout << foo (today); // illegal
        std::cout << bar (today); // illegal
        today = TUE; // illegal
    }

    UnscopedDay tomorrow = tue;
    if (4.2 > tomorrow) // legal
    {
        std::cout << foo (tomorrow); // legal
        std::cout << bar (tomorrow); // legal
    }

    return EXIT_SUCCESS;
}
```

Declaration

```
enum class Day : char { MON, TUE, WED, THU, FRI, SAT, SUN };
```

```
enum class NAME : TYPESPECIFIER { LIST };
```

Still no easy way to print an enum value :(

<http://en.cppreference.com/w/cpp/language/enum>

Namespaces

- Keep your declared names `using namespace std;` inside a restricted scope
- Reduce name collisions
 - what if two libraries both provide the function

```
int *quartiles(int *paData, int size);
```


Example

```
namespace CS485
{
    const std::string objects = "have data and responsibilities";
}

namespace CS250
{
    const std::string objects = "have data and functionality";
}

void outputDef ()
{
    //std::cout << objects; // illegal!

    std::cout << CS250::objects << std::endl;

    std::cout << CS485::objects << std::endl;
}
```

using

```
using namespace std;

using CS485::objects;

void outputDef ()
{
    std::cout << objects; // legal!

    std::cout << CS250::objects << std::endl;

    std::cout << CS485::objects << std::endl;
}
```

Notes

- The same namespace can be spread across many header files
- namespace `std` is defined in:
 - `iostream`
 - `vector`
 - `memory`
- This allows you to only include the piece of the namespace you need.

Useful to package classes

- *Prefer non-member, non-friend functions to member functions**
- If a non-member, non-friend function can do the work, what does that tell me about the class's interface?

```
namespace Example
{
    class BigResponsibilities { ... } ;

    void helpfulFunction(BigResponsibilities &);
}
```

Explore Computer Science Careers Conference

Hosted by Women in Computer Science
& Career Development Center

Join us March 18th,
1pm - 4pm
MPR

Panelist Discussion
Career Fair
Networking

register: tinyurl.com/WICSConference

Exceptions

Exceptions

- Signal an error has occurred
 - throw - signal the error
 - immediately leave the function
 - call destructors as the stack is unwound***
 - try/catch - look for and handle exceptions
 - can rethrow exception to keep propogating
- Predefined exceptions
 - #include <exception>
 - std::exception (parent class)
 - oddity: you can throw any object.

<http://en.cppreference.com/w/cpp/error/exception>

<http://en.cppreference.com/w/cpp/language/exceptions>

Class `std::exception`

```
class exception {  
public:  
    exception() noexcept;  
    exception(const exception&) noexcept;  
    exception& operator=(const exception&) noexcept;  
    virtual ~exception();  
    virtual const char* what() const noexcept;  
};
```


Oh no

- `std::terminate`
 - called when exception handling fails
 - uncaught
 - exception thrown during exception handling
 - many, many, many others *
 - default action is `std::abort()`
 - can be set via `std::set_terminate()`
- `std::unexpected` (*until c++17*)

*<http://en.cppreference.com/w/cpp/error/terminate>

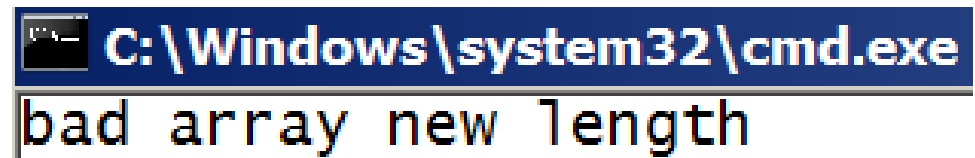
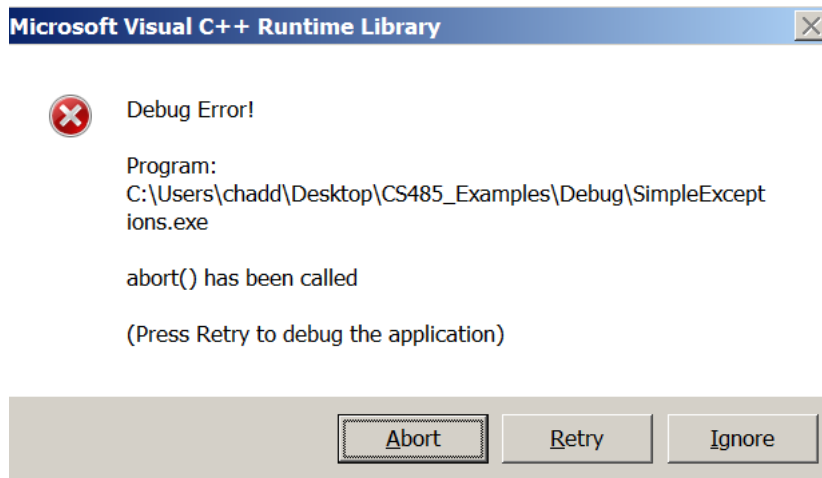
<http://en.cppreference.com/w/cpp/error/unexpected>

Example

```
int *paValues;  
int size = -1;  
  
try  
{  
    paValues = new int[size]; // try to allocate array of size -1  
}  
catch (const std::bad_alloc& e)  
{  
    std::cout << e.what () << std::endl;  
}  
catch (...)  
{  
    std::cout << "Unknown exception" << std::endl;  
}
```

Uncaught Exception

Caught Exception



```
class CS485Exception : public std::exception
{
public:

    CS485Exception (int value=0);
    CS485Exception (const CS485Exception & rcOther);

    ~CS485Exception ();

    CS485Exception& operator= (CS485Exception cOther);

    virtual const char* what () const override;

private:
    int mValue;

    char *mpsZMessage = nullptr;
};
```

Throw

```
if (0 > param)
{
    throw std::range_error ("Negative value!");
}

if (0 > param)
{
    throw CS485Exception (-1);
}
```

Empty throw; just rethrows the current exception (must be in try/catch block)

dynamic runtime check

```
// deprecated in C++11  
void foo () throw();  
void bar () throw(CS485Exception);  
void rab () throw(...);
```

noexcept specifier

- Mark whether or not your function can throw an exception -- *or allow an exception to be propagated from any other function that it invokes either directly or indirectly.*
- Violation of this leads to termination, `std::terminate`

non-throwing functions*

- marked noexcept
- destructors
- default constructors, copy constructors, move constructors
- copy assignment operators, move assignment operators
- deallocation functions (delete)

```
int cantThrowException (char data) noexcept;  
unsigned int *riskyException (int param) noexcept(false)
```

→ except that there are many caveats and qualifications

References for the previous slide

<http://en.cppreference.com/w/cpp/error/terminate>

http://en.cppreference.com/w/cpp/language/noexcept_spec

<https://akrzemi1.wordpress.com/2014/04/24/noexcept-what-for/>

<http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2009/n2855.html#problem>

<http://www.stroustrup.com/C++11FAQ.html#noexcept>

Exception Safety guarantees

- No-throw/No-fail
- Strong exception safety
- Basic exception safety
- No exception safety

Risks

- Throwing from a destructor
 - C++ cannot handle two exceptions active at the same time - the program will abort immediately
 - destructors may be invoked by throwing an exception (as the stack unwinds)
- Options
 - swallow exceptions in destructors
 - call `std::abort()` in the destructor

*Effective C++, Third Edition, Meyers, Item 11
More Effective C++, Third Edition, Meyers, p45-80

Risks

```
unsigned int *riskyException (int param) noexcept(false)
{
    CS485Exception cException;

    unsigned int *pRetVal = new unsigned int;

    if (0 > param)
    {
        throw std::range_error ("Negative value!");
    }

    *pRetVal = param;

    return pRetVal;
}
```

Which destructors are called?

Risks

```
unsigned int *unknownException (int param) noexcept(false)
{
    CS485Exception cException;

    unsigned int *pRetVal = new unsigned int;

    mightThrowException (pRetVal);

    return pRetVal;
}
```

Which destructors are called?

Mitigation

```
std::shared_ptr<unsigned int> safeUnknownException (int param) noexcept(false)
{
    CS485Exception cException;

    auto pRetVal (std::make_shared<unsigned int> ());

    mightThrowExceptionSmart (pRetVal);

    return pRetVal;
}
```

Which destructors are called?

Constructors

```
bigData::bigData(int data)
{
    mID = getID();
    mpHugeData = new int;
    *mpHugeData = data;
    mpSmallData = new int;
    *mpSmallData = lookupMightThrowException(data);
    //std::cout << "ctor(int) " << *this << std::endl;
}
```

Solution?

Exception Lab

- Download ExceptionLab.zip
- Add a new project, ExceptionLab, to Shapes Solution
- Add code from ExceptionLab.zip to your new project
- Finish the TODOs and make a release on GitHub, ExceptionLab
- Due: Friday, 11:59 pm.