Exception Handling

- What are exceptions
- Try … catch
- Layers of exceptions

What do they have in common?

- The bigfoot
- Software developer providing error free code
- Little green alien from Mars

What do they have in common?

- The bigfoot
- Software developer providing error free code
- Little green alien from Mars

They all have fans believing in their existence

What’s an exception?

- Exception is an unexpected behavior
- Exceptions are not necessarily errors (bugs), they are more about forgetting to handle errors
- Called exceptions because they deal with exceptional circumstances that may arise during runtime
- Exceptions are raised when there is no logical way for a method to continue its execution
- Object oriented way of implementing “error codes” used in plain C language

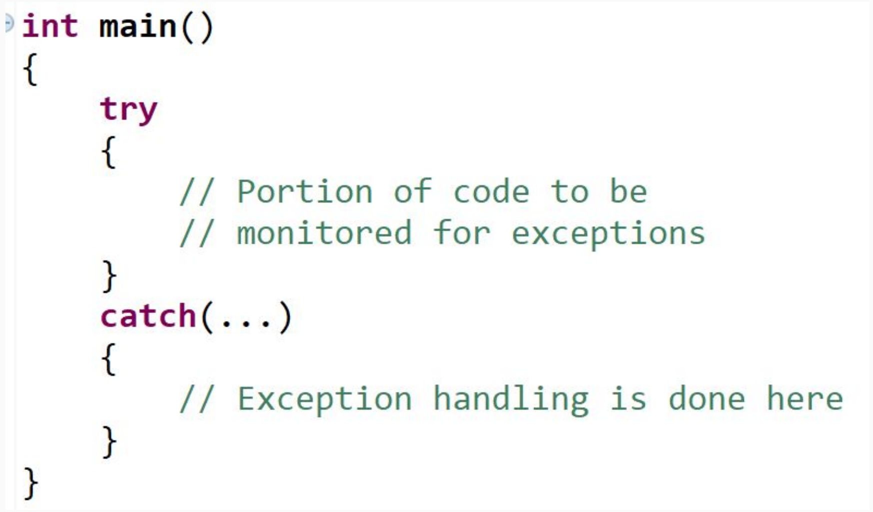
Examples

- What if malloc or new failed to provide the demanded memory block?
    - Remember that a system has a limited memory that may run out
- What if your code reads a file that is supposed to be present (and you as a programmer are taking for granted that it is always present). One sunny day, someone deleted that file …
- What if your code reads a feed from a website, then one rainy day the site's server went down

Try … catch … throw

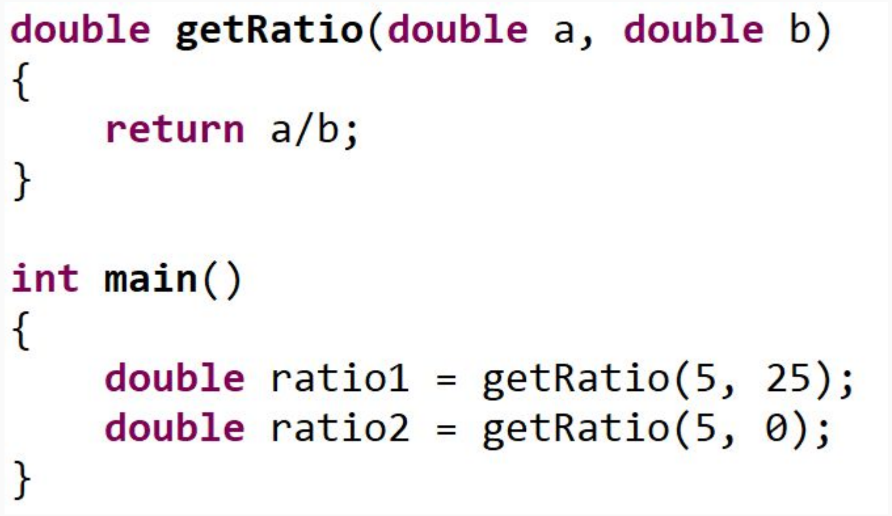
- Exceptions should be caught when they occur
  - Using a **try** and **catch** blocks
  - Portion of code to be monitored for exceptions should be enclosed within the try block (using **try** keyword)
  - Exception handling is done within the catch block (using the **catch** keyword)
  - To signal an exception or to propagate it to an outer code level, we use the **throw** keyword
  - Unlike Java, C++ does NOT support a "finally" block. Whatever code that "finally" must have, should be done in the destructor.

Try … catch … throw:



```
int main()
{
    try
    {
        // Portion of code to be
        // monitored for exceptions
    }
    catch(...)
    {
        // Exception handling is done here
    }
}
```

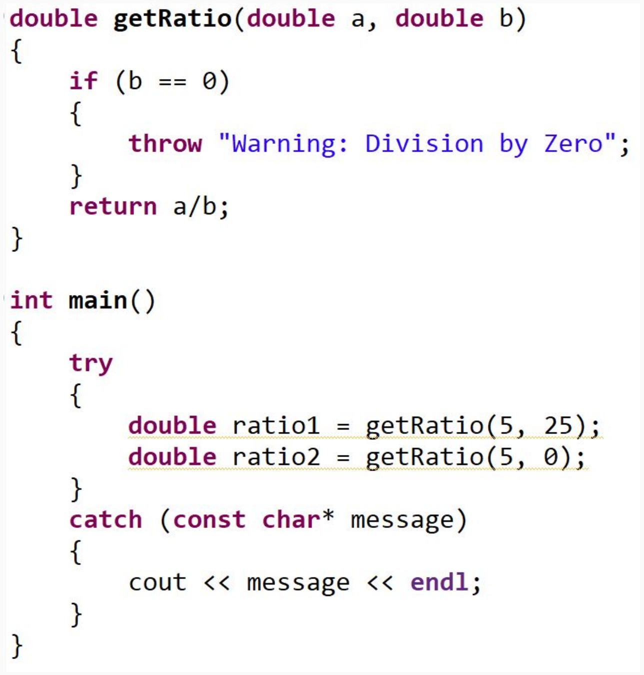
Try … catch … throw: Example



```
double getRatio(double a, double b)
{
    return a/b;
}

int main()
{
    double ratio1 = getRatio(5, 25);
    double ratio2 = getRatio(5, 0);
}
```

- **getRatio should *throw* an exception if provided with zero value for *b***

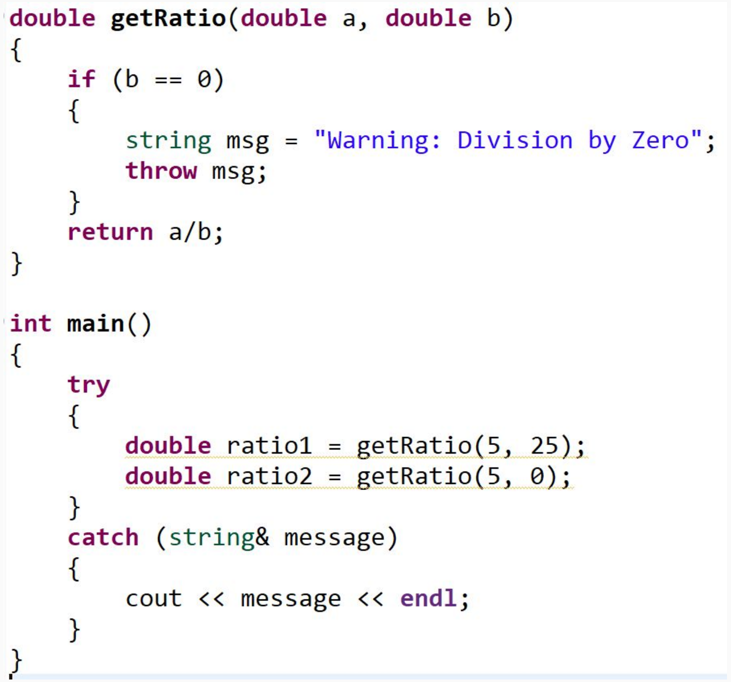
Try … catch … throw: Example



```
double getRatio(double a, double b)
{
    if (b == 0)
    {
        throw "Warning: Division by Zero";
    }
    return a/b;
}

int main()
{
    try
    {
        double ratio1 = getRatio(5, 25);
        double ratio2 = getRatio(5, 0);
    }
    catch (const char* message)
    {
        cout << message << endl;
    }
}
```

- **Catch argument type should match the throw argument type**
- **In our example, the type is const char* because this is exactly the type of the message that we sent using throw**

Try … catch … throw: Example



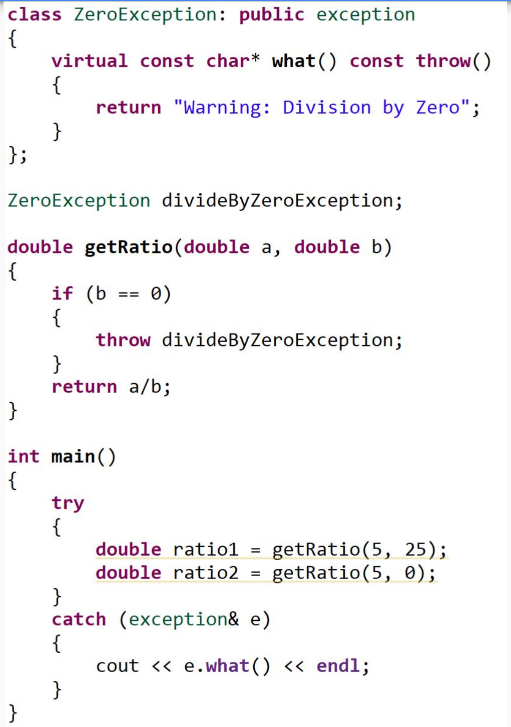
```
double getRatio(double a, double b)
{
    if (b == 0)
    {
        string msg = "Warning: Division by Zero";
        throw msg;
    }
    return a/b;
}

int main()
{
    try
    {
        double ratio1 = getRatio(5, 25);
        double ratio2 = getRatio(5, 0);
    }
    catch (string& message)
    {
        cout << message << endl;
    }
}
```

- **We can use string as well ...**

C++ standard exceptions

- #include <exception>
- C++ standard library offers a list of exceptions
  - std::bad_alloc
  - std::out_of_range
  - …
- C++ offers also a base class to create user defined “object” exceptions
  - Define new exceptions by inheriting from the base class std::exception

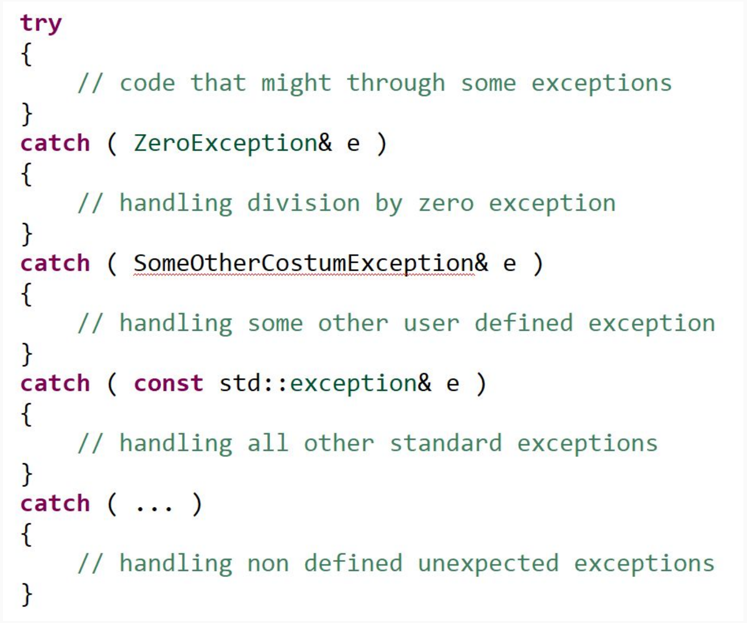
Inheriting from class exception: Example



```cpp
class ZeroException: public exception
{
    virtual const char* what() const throw()
    {
        return "Warning: Division by Zero";
    }
};

ZeroException divideByZeroException;

double getRatio(double a, double b)
{
    if (b == 0)
    {
        throw divideByZeroException;
    }
    return a/b;
}

int main()
{
    try
    {
        double ratio1 = getRatio(5, 25);
        double ratio2 = getRatio(5, 0);
    }
    catch (exception& e)
    {
        cout << e.what() << endl;
    }
}
```

- std::exception has a virtual member method called **what()** that can be reimplemented in derived classes to personalize a user message about the cause of the exception
- **const throw()** part of the declaration:
  - **const**: the method will not alter the state of the class
  - **throw()**: for C++98 means that this method is guaranteed not to throw exceptions
    - Replaced by **noexcept** since C++11

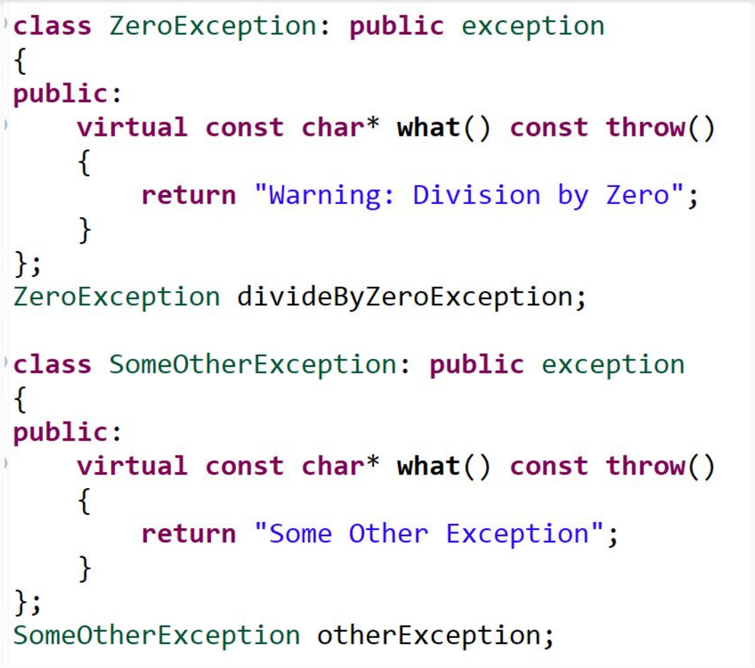
Catching multiple exceptions



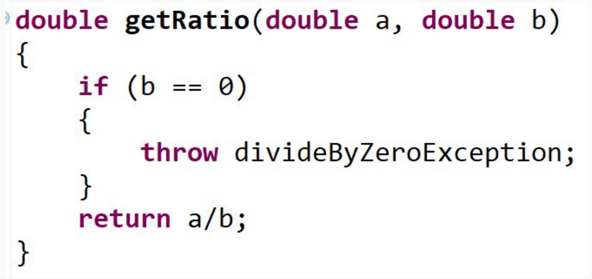
```cpp
try
{
    // code that might through some exceptions
}
catch ( ZeroException& e )
{
    // handling division by zero exception
}
catch ( SomeOtherCostumException& e )
{
    // handling some other user defined exception
}
catch ( const std::exception& e )
{
    // handling all other standard exceptions
}
catch ( ... )
{
    // handling non defined unexpected exceptions
}
```

- We can manage different type of exceptions separately
- Respect the order of catching
  - Derived classes first, then base class
  - (...) at the end

Try catch blocks can be nested - 1

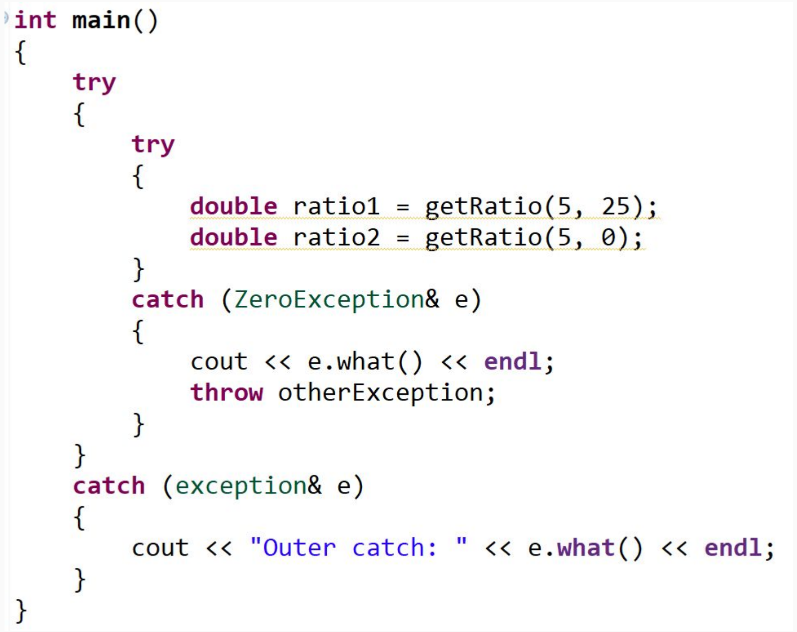


```cpp
class ZeroException: public exception
{
public:
    virtual const char* what() const throw()
    {
        return "Warning: Division by Zero";
    }
};
ZeroException divideByZeroException;

class SomeOtherException: public exception
{
public:
    virtual const char* what() const throw()
    {
        return "Some Other Exception";
    }
};
SomeOtherException otherException;
```

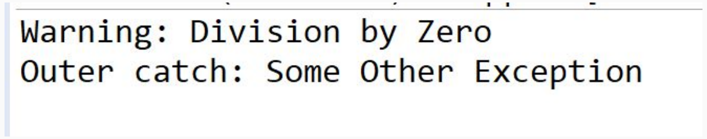


```cpp
double getRatio(double a, double b)
{
    if (b == 0)
    {
        throw divideByZeroException;
    }
    return a/b;
}
```

Try catch blocks can be nested - 2

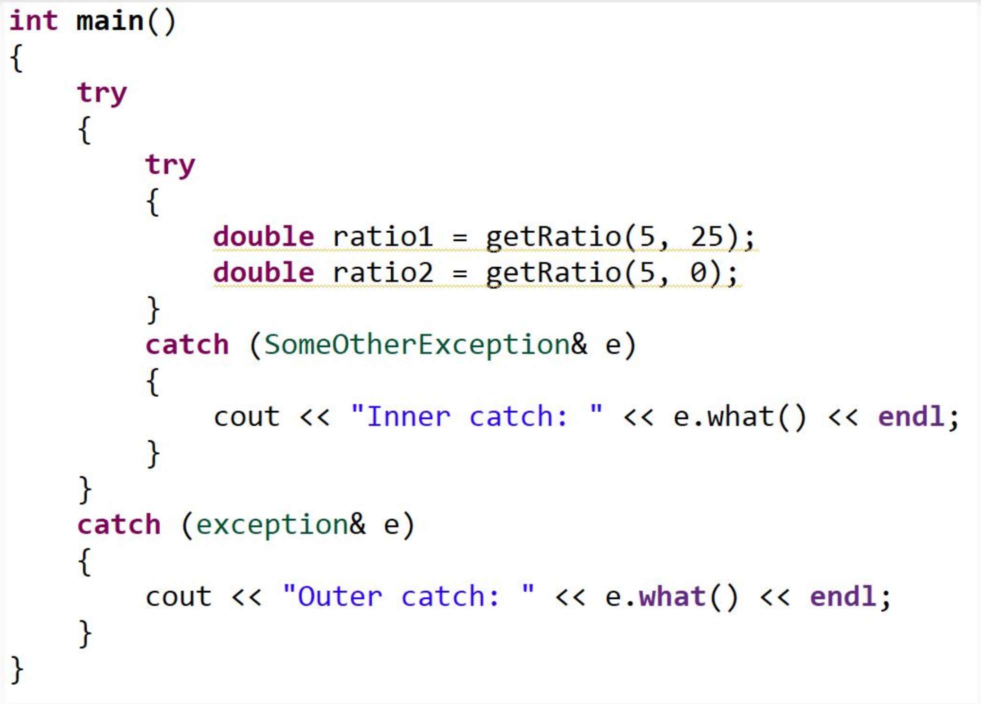


```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
            double ratio2 = getRatio(5, 0);
        }
        catch (ZeroException& e)
        {
            cout << e.what() << endl;
            throw otherException;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```



```
Warning: Division by Zero
Outer catch: Some Other Exception
```

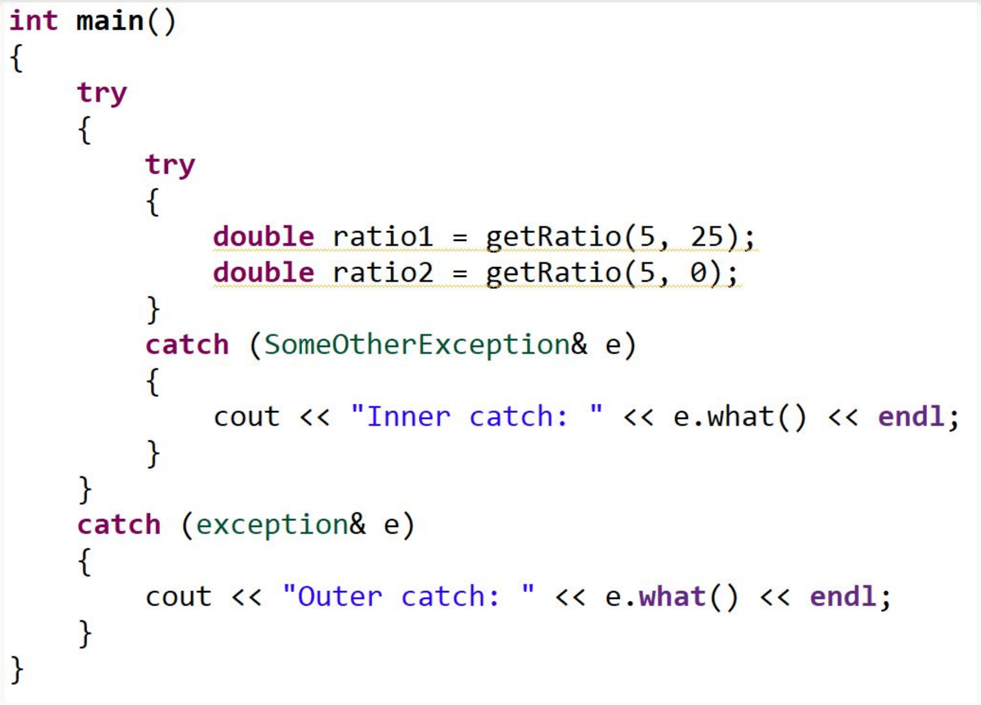
Catch me if you can ;) - take 1



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
            double ratio2 = getRatio(5, 0);
        }
        catch (SomeOtherException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```

- **Which catch will catch the ZeroException?**

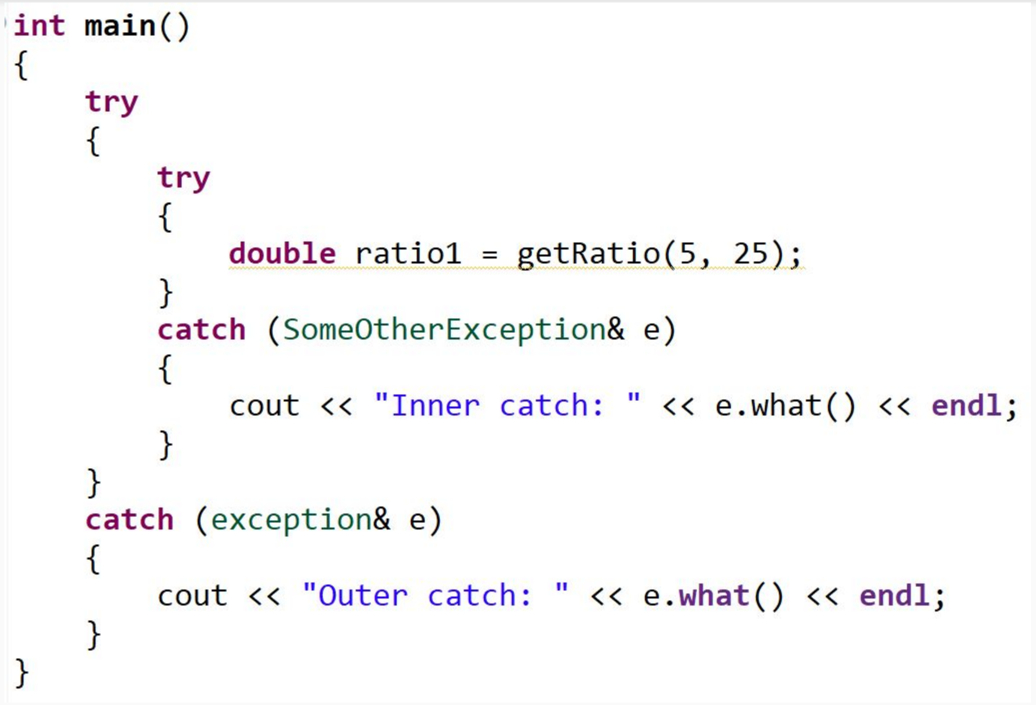
Catch me if you can ;) - take 1



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
            double ratio2 = getRatio(5, 0);
        }
        catch (SomeOtherException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```

- **Which catch will catch the ZeroException?**
    - **The outer catch. Exceptions propagate to the outer blocks.**

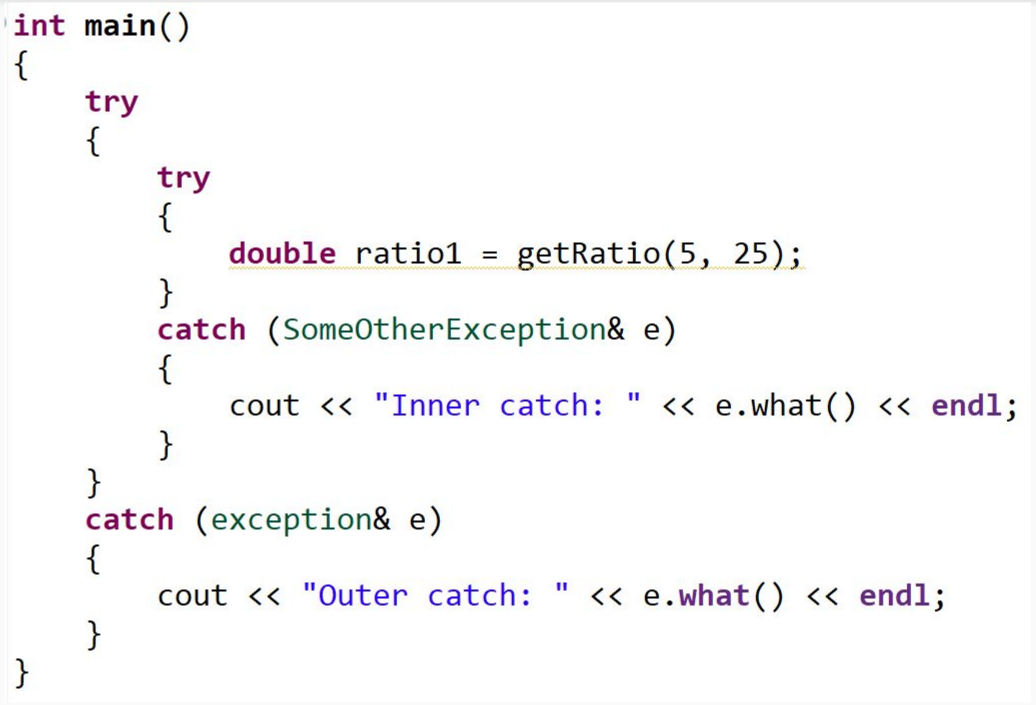
Catch me if you can ;) - take 2



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
        }
        catch (SomeOtherException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```

- **Which catch will catch the ZeroException?**

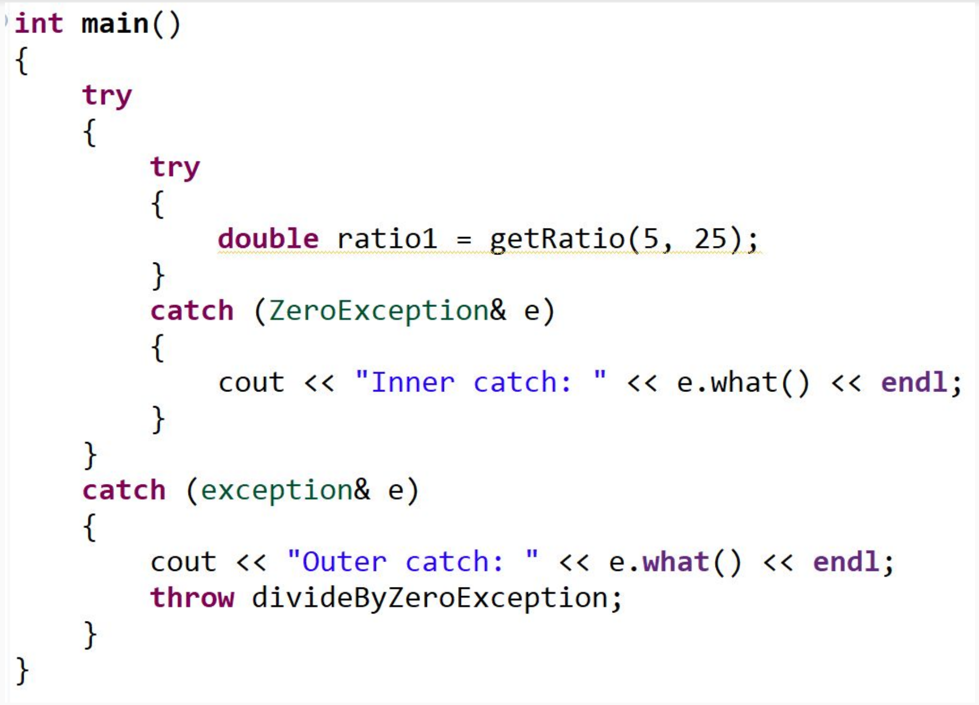
Catch me if you can ;) - take 2



```cpp
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
        }
        catch (SomeOtherException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```

- **Which catch will catch the ZeroException?**
    - **None of them ;)**

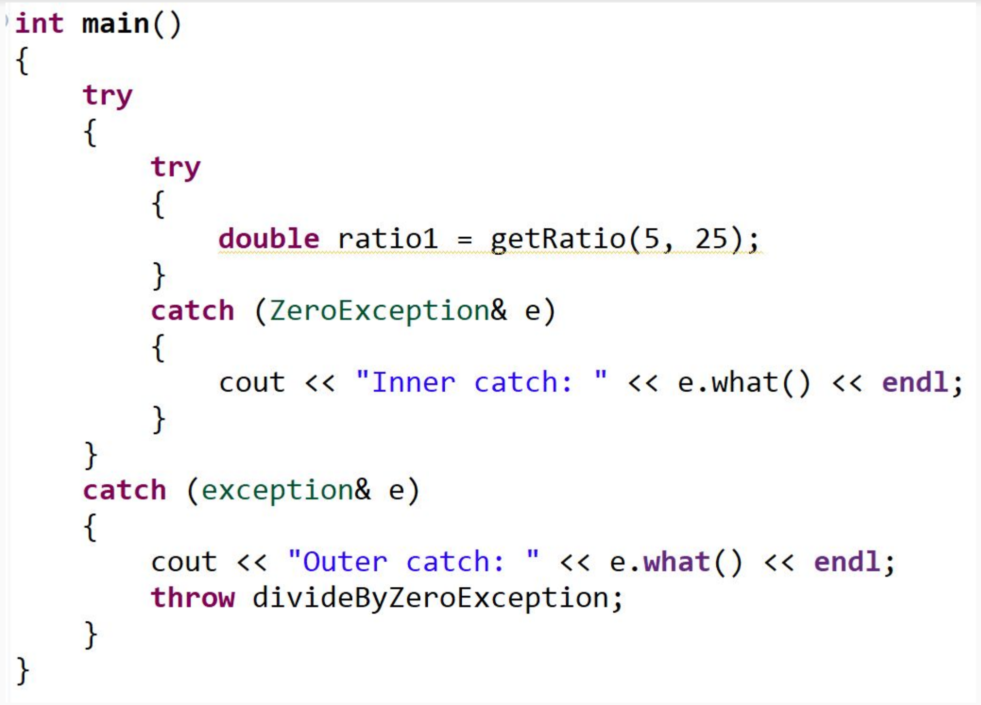
Catch me if you can ;) - take 3



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
        }
        catch (ZeroException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
        throw divideByZeroException;
    }
}
```

- **Which catch will catch the ZeroException? (assuming that the outer catch caught an exception and it threw divideByZeroException)**

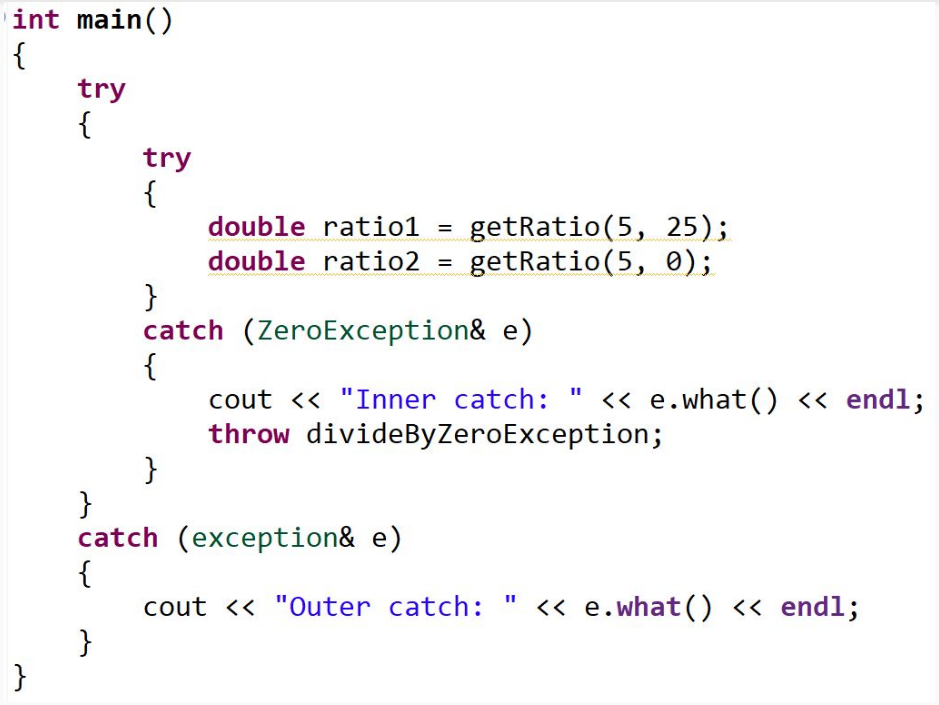
Catch me if you can ;) - take 3



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
        }
        catch (ZeroException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
        throw divideByZeroException;
    }
}
```

- **Which catch will catch the ZeroException? (assuming that the outer catch caught an exception and it threw divideByZeroException)**
    - **None of them. Exceptions do not propagate to the inner blocks.**

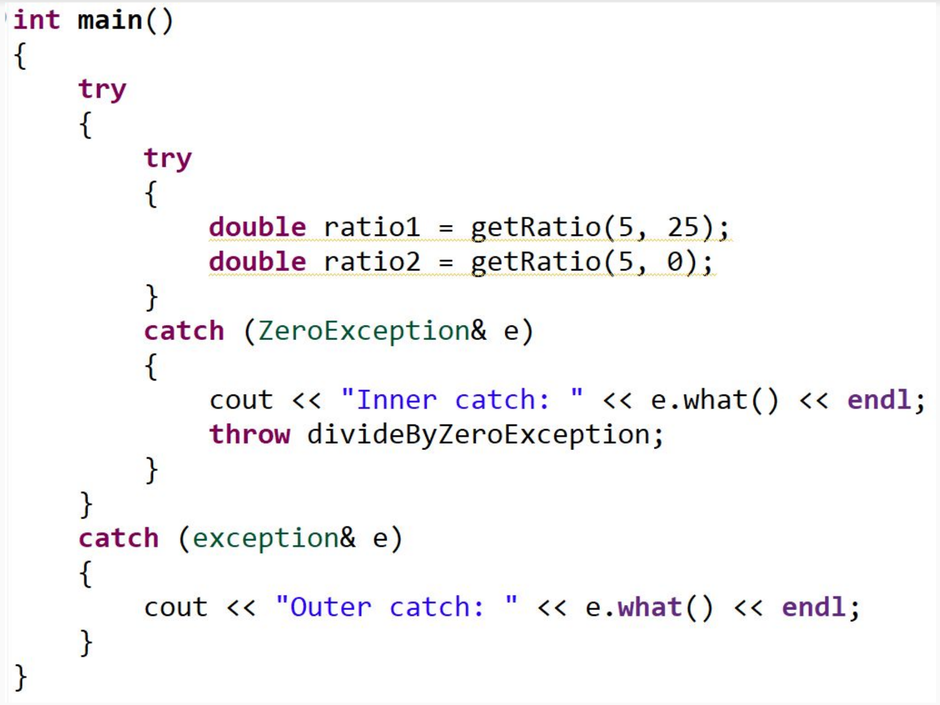
Catch me if you can ;) - take 4



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
            double ratio2 = getRatio(5, 0);
        }
        catch (ZeroException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
            throw divideByZeroException;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```

- **Which catch will catch the ZeroException?**

Catch me if you can ;) - take 4



```
int main()
{
    try
    {
        try
        {
            double ratio1 = getRatio(5, 25);
            double ratio2 = getRatio(5, 0);
        }
        catch (ZeroException& e)
        {
            cout << "Inner catch: " << e.what() << endl;
            throw divideByZeroException;
        }
    }
    catch (exception& e)
    {
        cout << "Outer catch: " << e.what() << endl;
    }
}
```

- **Which catch will catch the ZeroException?**
  - **getRatio(5, 0) will trigger the inner catch.**
  - **throw divideByZeroException will trigger the outer catch**

Exception handling and control transfer

- When a program throws an exception the execution control is transferred to the catch block and never returns to the block that threw the exception
- If an exception occur and the program does not provide exception handlers or if it does provide one but the catch block exception declaration is not of the same type as the thrown object, the program will abort.
- When the control is transferred from a throw-point to a handler, destructors are invoked for all automatic objects constructed since the try block was entered

Reading assignment for next week

- **Generic programming in C++**