

# CS-1004 Object Oriented programming

## Week 15

---

**Instructor:**

ATIYA

LECTURER

(COMPUTER SCIENCE DEPARTMENT)

NATIONAL UNIVERSITY- FAST (KHI CAMPUS)

EMAIL: ATIYA.JOKHIO@NU.EDU.PK

ROOM: (LECTURER ROOM#19, CS DEPT BASEMENT-2 )

# Errors!

---

- In software industrial programming most of the programs contain bugs. Errors can be broadly categorized into two types. We will discuss them one by one.
  1. Compile Time Errors
  2. Run Time Errors
- **Compile Time Errors** – Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import. Syntax errors Semantic errors
- **Run Time Errors** - They are also known as exceptions. An exception caught during run time creates serious issues.

# Output?

---

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator; // 1 and 0
    result = zeroDivision(numerator, denominator);
    cout << "result is " << result << endl;
    cout <<"divide by zero" << endl << endl;
    return 0;}
```

# Output?

---

## Output

```
/tmp/FOJhImVBIQ.o
enter numerator and denominator:
2
0
Floating point exception

=== Code Exited With Errors ===|
```

```
enter numerator and denominator:
1
0

-----
Process exited after 7.731 seconds with return value 3221225620
Press any key to continue . . .
```

# Exception!

---

- An exception is a situation, which occurred by the runtime error. In other words, an exception is a runtime error. An exception may result in loss of data or an abnormal execution of program.
- Errors hinder normal execution of program
- Exception handling is the process of handling errors and exceptions in such a way that they do not hinder normal execution of the system
- It is a new feature that ANSI C++ included in it. Now almost all C++ compilers support this feature.

# Exceptions!

---

- **Exceptions are different**, however. You can't eliminate exceptional circumstances; you can only prepare for them. Your users will run out of memory from time to time, and the only question is what you will do. Your choices are limited to these:
  - Crash the program.
  - Inform the user and exit gracefully.
  - Inform the user and allow the user to try to recover and continue.
  - Take corrective action and continue without disturbing the user.

# Exception

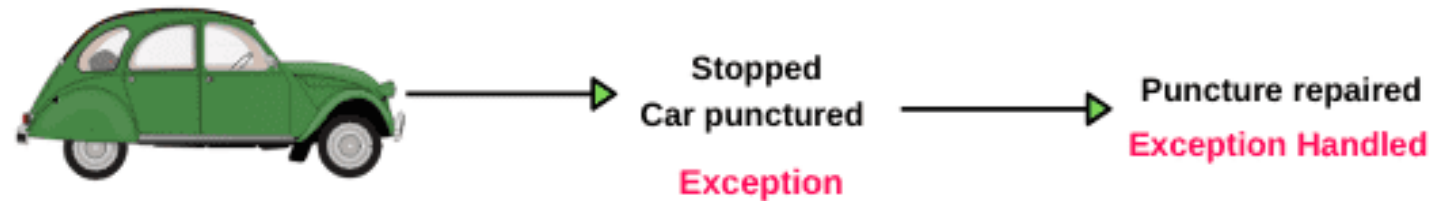
---

1. For example,  
User divides a number by zero, this will compile successfully but an exception or run time error will occur due to which our applications will be crashed. In order to avoid this we'll introduce exception handling technics in our code.

# Exception

---

- For example,
- Suppose a person is traveling by car from karachi to lahore. After traveling mid-distance, the tire of his car is punctured. This unexpected or unwanted event is nothing but an exception.
- The car owner always keeps an extra tire as an alternative on a long-distance journey. He changes the punctured tire by a new tire. After changing the tire, he continues the rest of the journey. This alternative way is called exception handling.





# Exceptions Handlers

---

- Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try**, **catch**, and **throw**.
- ❑ **throw** – A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- ❑ **catch** – A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
- ❑ **try** – A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

# Catching Exceptions

---

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

```
try {  
    // protected code  
} catch( ExceptionName exception1 ) {  
    // code to handle ExceptionName exception  
}
```

# Throwing Exceptions

---

Exceptions can be thrown anywhere within a code block using **throw** statement. The operand of the throw statement determines a type for the exception and can be any expression and the type of the result of the expression determines the type of exception thrown.

```
double division(int a, int b) {  
    if( b == 0 ) {  
        throw "Division by zero condition!"; //throw b;  
    }  
    return (a/b);  
}
```

# try

---

A block of code which may cause an exception is typically placed inside the try block. It's followed by one or more catch blocks. If an exception occurs, it is thrown from the try block.

```
try {  
    // protected code  
}
```

# throw

---

Possible to declare throw keyword anywhere in the try block. throw keyword trigger/execute a matched type of the catch block. The try block exists When a throw statement is reached.

# Catching Exceptions

---

The `ExceptionName` is the name of the exception to be caught.

The `exception1` are your defined names for referring to the exceptions.

# Example

---

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {
    if (y == 0) {
        throw y;}
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator;
    try {
        result = zeroDivision(numerator, denominator);
        cout << "result is " << result << endl;    }
    catch (int ex) {
        cout <<"divide by zero" << endl << ex << endl;    }
    return 0;}
```

# Example

---

```
enter numerator and denominator:
```

```
2
```

```
0
```

```
divide by zero
```

```
0
```

```
-----
```

```
Process exited after 4.039 seconds with return value 0
```

```
Press any key to continue . . .
```



# Exception Handling Checklist

---

- Try block has the main code, which code may throw an exception.
- Multiple throws are possible to declare in try block based different situations.
- The catch block contains exception handling code that is executed when thrown by the Try Block.
- Possible to declare one or more catch block for different type of exception
- No code can be between try block and the first catch block.
- Catch blocks declare directly after the try block.

# flow of execution of try/catch blocks.

---

```
#include <iostream>
#include <exception>
using namespace std;
double zeroDivision(int x, int y) {
    if (y == 0) {
        throw y;
    }
    cout << "After throw (Never executed) \n";
    return (x / y);
}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin >> numerator >> denominator;
    cout << "Before try \n";
```

# flow of execution of try/catch blocks.

---

```
try {  
    result = zeroDivision(numerator, denominator);  
    cout << "result is " << result << endl;  
}  
  
catch (int e) {  
    cout << "divide by zero" << endl;    }  
    cout << "After catch (Will be executed) \n";  
  
return 0;}
```

# flow of execution of try/catch blocks.

---

```
enter numerator and denominator:
2
0
Before try
divide by zero
After catch (Will be executed)

-----
Process exited after 3.571 seconds with return value 0
Press any key to continue . . . |
```

# Without catch

---

```
#include <iostream>
#include <exception>
using namespace std;
double zeroDivision(int x, int y) {
    if (y == 0) {
        throw y;}
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator;

    try {
        result = zeroDivision(numerator, denominator);
        cout << "result is " << result << endl;
    }

    return 0;}
```

# Without catch

---

Compiler (6) <input type="checkbox"/> Resources <input type="checkbox"/> Compile Log <input type="checkbox"/> Debug <input type="checkbox"/> Find Results <input type="checkbox"/> Console <input type="checkbox"/> Close			
Line	Col	File	Message
		C:\SPRING 2024\Object Oriented Program...	In function 'int main()':
78	9	C:\SPRING 2024\Object Oriented Programming\...	[Error] expected 'catch' before numeric constant
78	9	C:\SPRING 2024\Object Oriented Programming\...	[Error] expected '(' before numeric constant
78	9	C:\SPRING 2024\Object Oriented Programming\...	[Error] expected type-specifier before numeric constant
78	8	C:\SPRING 2024\Object Oriented Programming\...	[Error] expected ')' before numeric constant
78	9	C:\SPRING 2024\Object Oriented Programming\...	[Error] expected '{' before numeric constant

# Multiple catch Clauses

---

- In some cases, more than one exception could be raised by a single piece of code
- To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception
- After one catch statement executes, the others are bypassed

# Multiple catch Clauses

---

You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.

```
try { // protected code }  
catch( ExceptionName e1 )  
{ // catch block }  
catch( ExceptionName e2 ) { // catch block }  
catch( ExceptionName eN ) { // catch block }
```



# Multiple catch Clauses

---

```
#include <iostream>
#include <exception>
using namespace std;
double zeroDivision(int x, int y) {
    if (y == 0) {
        throw y;}
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator;
    try {
        result = zeroDivision(numerator, denominator);
        cout << "result is " << result << endl;    }
```

# Multiple catch Clauses

---

```
catch (int e) {  
    cout <<"divide by zero" << endl << e << endl;    }  
    catch (int e) {  
        cout <<"another exception" << endl << e << endl;    }  
        catch (int e) {  
            cout <<"another exception" << endl << e << endl;    }  
    return 0;}
```

# catch all block

---

If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the exception declaration as follows –

```
try { // protected code }  
  
catch(...)  
  
{ // code to handle any exception }
```

# catch all block

---

```
#include <iostream>
using namespace std;
double zeroDivision(int x, int y) {
    if (y == 0) {
        throw y;}
    return (x / y);}
int main() {
    int numerator;
    int denominator;
    double result;
    cout << "enter numerator and denominator: " << endl;
    cin>>numerator>>denominator;
    try {
        result = zeroDivision(numerator, denominator);
        cout << "result is " << result << endl;    }
    catch(...){
        cout <<"divide by zero" << endl << endl;        }
    return 0;}
```

---

If an exception is thrown and not caught anywhere, the program terminates abnormally.

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    try {
```

```
        throw 'a';    }
```

```
    catch (int x) {
```

```
        cout << "Caught ";    }
```

```
    return 0;}
```

---

Implicit type conversion doesn't happen for primitive types. For example, in the following program 'a' is not implicitly converted to int

```
terminate called after throwing an instance of 'char'
-----
Process exited after 2.722 seconds with return value 3
Press any key to continue . . . |
```

# Correct version

---

```
#include <iostream>
using namespace std;

int main()
{
    try {
        throw 'a';
    }
    catch (int x) {
        cout << "Caught " << x;
    }
    catch (...) {
        cout << "Default Exception\n";
    }
    return 0;
}
```

# output

---

```
Default Exception
```

```
-----
```

```
Process exited after 0.833 seconds with return value 0
```

```
Press any key to continue . . .
```



# C++ Standard Exceptions

---

- C++ provides a list of standard exceptions defined in header **<exception>** in namespace `std` where “exception” is the base class for all standard exceptions.
- All exceptions like [bad\\_alloc](#), [bad\\_cast](#), [runtime\\_error](#), etc generated by the standard library inherit from **`std::exception`**. Therefore, all standard exceptions can be caught by reference.

# C++ Standard Exceptions

---

<b>std::exception</b>	This is an exception and the parent class of all standard C++ exceptions.
<b>std::bad_alloc</b>	This exception is thrown by a new keyword.
<b>std::bad_cast</b>	This is an exception thrown by dynamic_cast.
<b>std::bad_exception</b>	A useful device for handling unexpected exceptions in C++ programs.
<b>std::bad_typeid</b>	An exception thrown by typeid.
<b>std::logic_error</b>	This exception is theoretically detectable by reading code.
<b>std::domain_error</b>	This is an exception thrown after using a mathematically invalid domain.
<b>std::invalid_argument</b>	An exception thrown for using invalid arguments.
<b>std::length_error</b>	An exception thrown after creating a big std::string.
<b>std::out_of_range</b>	Thrown by at method.
<b>std::runtime_error</b>	This is an exception that cannot be detected via reading the code.
<b>std::overflow_error</b>	This exception is thrown after the occurrence of a mathematical overflow.
<b>std::range_error</b>	This exception is thrown when you attempt to store an out-of-range value.
<b>std::underflow_error</b>	An exception thrown after the occurrence of mathematical underflow.

# what()

---

- In C++, the `what()` function is a **virtual member function** of the base exception class:
- `what()` is used to **retrieve a human-readable message** about what caused the exception. It's commonly used inside a catch block
- Here, `what()` is a public method provided by exception class and it has been overridden by all the child exception classes. This returns the cause of an exception.

```
catch (exception& ex)
{
    cout << ex.what() << endl;
}
```

# what()

ctures OOP\Week 15\Exception class.cpp - [Executing] - Embarcadero Dev-C++ 6.3

Project Execute Tools AStyle Window Help

TDM-GCC 9.2.0 64-bit Release

ls)

exception class.cpp

```
1
2 // what() function
3 #include <iostream>
4 #include <stdexcept>
5 using namespace std;
6 int main() {
7     try {
8         throw runtime_error("Something went wrong!");
9     } catch (const exception& e) {
10         cout << "Caught exception: " << e.what() << '\n';
11     }
12 }
```

D:\SPRING 2025\OOP\Lecture

Caught exception: Something went wrong!

-----  
Process exited after 0.2166 seconds with return value 0  
Press any key to continue . . .

# C++ Standard Exceptions

---

Examples in Dev File

# User-Defined Exceptions

---

The C++ `std::exception` class allows us to define objects that can be thrown as exceptions. This class has been defined in the `<exception>` header. The class provides us with a virtual member function named `what`.

This function returns a null-terminated character sequence of type `char *`. We can overwrite it in derived classes to have an exception description.

# User-Defined Exceptions

---

**Example 1:** Program to implement exception handling with single class

```
#include <iostream>
using namespace std;

class demo {
};

int main()
{
    try {
        throw demo();
    }

    catch (demo d) {
        cout << "Caught exception of demo class \n";
    }
}
```

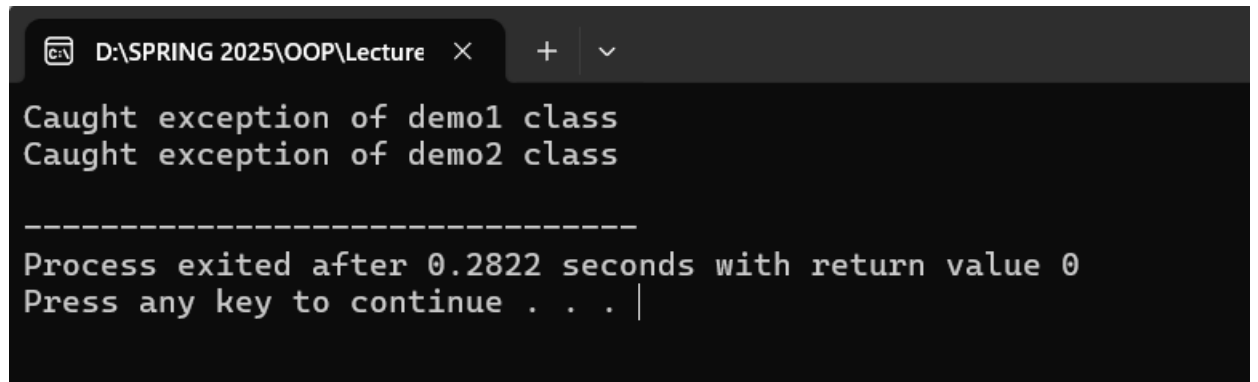
# User-Defined Exceptions

Example 2: Program to implement exception handling with two class

```
#include <iostream>
using namespace std;
class demo1 {
};
class demo2 {
};
int main()
{
    for (int i = 1; i <= 2; i++) {
        try {
            if (i == 1)
                throw demo1();

            else if (i == 2)
                throw demo2();
        } catch (demo1 d1) {
            cout << "Caught exception of demo1 class \n";
        }

        catch (demo2 d2) {
            cout << "Caught exception of demo2 class \n";
        }
    }
}
```



The screenshot shows a terminal window with the title bar "D:\SPRING 2025\OOP\Lecture". The output of the program is as follows:

```
Caught exception of demo1 class
Caught exception of demo2 class

-----
Process exited after 0.2822 seconds with return value 0
Press any key to continue . . . |
```

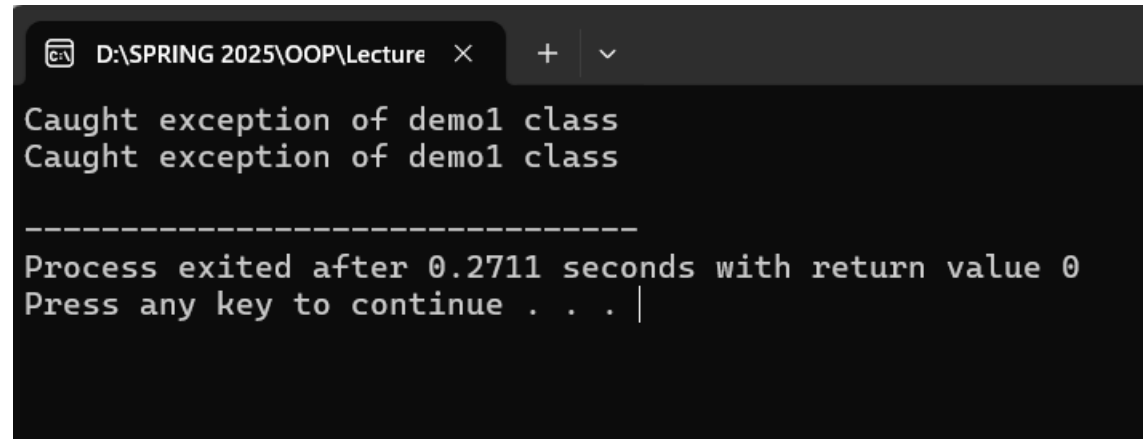


## User-Defined Exceptions:

Exception handling can also be implemented with the help of inheritance. In case of inheritance object thrown by derived class is caught by the first catch block.

```
#include <iostream>
using namespace std;

class demo1 {
};
class demo2 : public demo1 {
};
int main(){
    for (int i = 1; i <= 2; i++) {
        try {
            if (i == 1)
                throw demo1();
            else if (i == 2)
                throw demo2();
        }
        catch (demo1 d1) {
            cout << "Caught exception of demo1 class \n";
        }
        catch (demo2 d2) {
            cout << "Caught exception of demo2 class \n";
        }
    }
}
```



```
D:\SPRING 2025\OOP\Lecture x + v
Caught exception of demo1 class
Caught exception of demo1 class
-----
Process exited after 0.2711 seconds with return value 0
Press any key to continue . . . |
```

# Exception Handling in Filing

---

```
int main(int argc, char** argv)
{
    const string fileName = "test66.txt";

    try {
        readIntegerFile(fileName);
    } catch (const exception& e) {
        cerr << "Unable to open file " << fileName << endl;
        exit (1);
    }
    cout << endl;

    return (0);
}
```

```
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <exception>
using namespace std;

void readIntegerFile(const string& fileName)
{
    ifstream istr;
    int temp;
    char buffer[256];
    istr.open(fileName.c_str());
    if (istr.fail()) {
        throw exception();
    }
    while (istr)
    {
        istr.getline (buffer,100);
        cout << buffer << endl;
    }
}
```